Title
Practical Algorithms for Reliable Autonomy

Permalink
https://escholarship.org/uc/item/6hq5j9x0

Author
Fridovich-Keil, David

Publication Date
2020

Peer reviewed|Thesis/dissertation

Practical Algorithms for Reliable Autonomy

by

David Fridovich-Keil

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering — Electrical Engineering & Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire J. Tomlin, Chair
Professor Ruzena Bajcsy
Professor S. Shankar Sastry
Assistant Professor Koushil Sreenath

Spring 2020

Practical Algorithms for Reliable Autonomy

Abstract

Practical Algorithms for Reliable Autonomy

by

David Fridovich-Keil

Doctor of Philosophy in Engineering — Electrical Engineering & Computer Sciences

University of California, Berkeley

Professor Claire J. Tomlin, Chair

In this dissertation, I present several ideas for building *practical* algorithms for building more reliable autonomous systems. That is, each of the ideas below is, from the outset, designed to be physically implementable on real hardware and operate in real-time. In Chapter 1, I provide a high-level overview of some of the key challenges in autonomy today and how this work addresses some of those challenges. Chapter 2 presents an approach for reachability-based robust motion planning, which for the first time attains adversarial robustness for real-time motion planning problems. Chapter 3 introduces the prediction problem which arises in multi-agent situations, and describes a Bayesian approach for determining how confident one should be in a given predictive model. Chapter 4 combines elements of the prediction and motion planning problems in a multi-player differential game and presents a novel real-time solution strategy. Finally, Chapter 5 lists several open problems and discusses several exciting next steps.

To my sister, Sara, and my parents, Judy and Mark

Contents

List of Figures

List of Tables

Acknowledgments

I want to thank my advisor, Claire, for all her support through this process. Meeting with her has been the highlight of each week. I also want to thank my labmates for always being there to bounce ideas off of, for all their help both with academics and with life as well. Collaborating with them and students in other groups has been one of the joys of graduate school. Finally thanks to the rest of my committee for all your help along the way.

Last but not least, thanks to my sister and my parents. Without you, none of this would have been possible. You've been with me every step of the way!

Chapter 1

Introduction

Autonomy is more and more present these days, with familiar parts of the puzzle in everything from cruise control devices to search engines. This dissertation aims to present a generalized view of the important challenges, and will focus in on challenges which are especially important in multi-agent scenarios.

1.1   The autonomy pipeline

Commonly in autonomous systems, there is a "pipeline" that involves familiar sub-problems. Illustrated in Figure 1.1, we see that most autonomous systems involve, first and foremost, a *plant* of some kind. The plant represents the specific, often physically embodied, entity which interacts with nature and often whose precise outcome is unpredictable. We will see examples of different types of plants in this thesis, including quadrotors, cars, and airplanes, but many of the principles we develop will also apply to less embodied plants, such as social networks and financial systems.

In any case, the plant embodies the interaction between nature and whatever man-made process we are interested in. The way we measure the outcome of that interaction is with *sensors*. These sensors essentially record the relevant information for the problem at hand, but all higher-level processing of that information is deferred until the next step. The sensors give us our only look at what actually happened—any other information which we might wish to have is, unfortunately, unavailable.

Perception algorithms process the data recorded by the sensors. In the case of an autonomous car, which we will see extensively later in this dissertation, the core perception algorithms have to do with (a) localization and mapping, and (b) object detection/recognition in camera/LIDAR frames. In many instances tracking relevant variables over time is also very important, but we shall largely ignore this challenge here. Regardless, perception algorithms generate what is, in control parlance, termed a "state estimate" and are, unsurprisingly, closely related the state estimators and the study of state estimation.

Equipped with a state estimate from sensor data and perception algorithms, many autonomous systems also require some means of *predicting* the future evolution of some of the state variables in order to decide how best to react. Often, this comes down to predicting
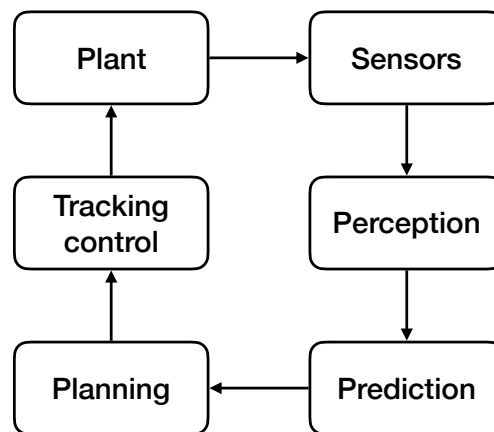
Fig. 1.1: Typical pipeline of operations involved in autonomy. Most applications, embodied or otherwise, will have some or all of these components.

the decisions made by other intelligent participants in the problem, something discussed in both Chapters 3 and 4.

Once the autonomous system has a good idea of what's going on around it (from the "perception" module) and what's likely to happen in the immediate future (from the prediction module), it must then *plan* a best response. This can be formulated in many different ways—often we'll see this posed as an optimization problem, but that is by no means the only way—but when the dust settles it should be clear what the autonomous system is "trying" to do. Executing this plan, though, is an entirely different matter, which we shall come to shortly.

The challenge of actually designing the input such that the plant executes the plan is typically called *tracking control*. Of course, planning itself may very well be posed as an optimal control problem, so it is best not to confuse or conflate *tracking control* with *control theory* writ large. The problem tracking control must solve essentially boils down to the problem that real systems don't always behave how we'd like them to, how our models say they'll behave. This will be discussed in detail later, in Chapter 2.

1.2 Core challenges with autonomy

As I see it, autonomy engineers face two core challenges, and a third is also sometimes important for some practitioners.

The first of these core issues has to do with sensing and perception, and is essentially ignored here. The bottom line is, often engineers concerned with planning and control just do not have a clear enough picture of what is going on in order to design a safe and correct set of inputs to the system. For example, in the case of an autonomous car, a regular human driver (and certainly my father) may be surprised that the LIDAR-enhanced sensing and perception capabilities of an autonomous car are routinely confused by commonplace trees and bushes that line many suburban streets. In any case, relatedly, the perception algorithms most commonly employed are not always trustworthy. Although convolutional deep neural

networks have made an indelible impact on detection problems, their output is not all that reliable. It is common knowledge, for instance, that very small perturbations of images can be classified arbitrarily badly. Fixing, or at the very least understanding, this unreliability is a major direction of research now and I expect that it will continue to be a challenge many years in the future.

The second core issue relates to how multiple agents interact, for example, on the road. Whenever multiple decision-making, intelligent agents interact, be it on the road or on a social network, an autonomous system needs to be capable of understanding and, often, influencing that interaction. This will be the main focus of both Chapters 3 and 4, and much future work discussed in Chapter 5. As we shall see in Chapter 3, it is common to employ some method of *prediction* and thereby decompose the problem into sequential prediction and planning subproblems. Chapter 4 will explore this issue further and offer a different viewpoint, namely, that multi-agent situations can often be best expressed as multi-player differential games. Both views are certainly very useful in different circumstances, and I believe they both will have a place in autonomous systems in the future.

Finally, but not insignificantly, I believe a major challenge of autonomy rests in robust tracking control. This will be the focus of Chapter 2. Here, the basic challenge is that, while abstract reasoning to do with perception, prediction, and planning may be very helpful, it is essentially useless if we can't rely on the tracking controller to execute the plans we generate. In many systems, such as cars driving on a dry road, this problem may very well be effectively solved. On the other hand, there are many devices and maneuvers, such as acrobatics on a quadrotor, for which tracking control is really not up to the same standard. Many times this may be because the plant itself is "less stable" or somehow more sensitive to our input in a "hard to control" way, but what we will focus on is an issue which also arises called *model mismatch*. Here, we may use some mathematical model of how the physical system actually evolves for higher level reasoning, control design, etc., but difference or mismatch between the physical system and a model of it can be significant. When we design robust controllers, we are essentially trying to minimize our sensitivity to this mismatch, and we may do so in a number of ways. Chapter 2 presents one particular method, based on worst-case analysis.

Chapter 2

Robust Control for Modular, Real-Time Motion Planning

2.1 Background

This Chapter deals with the final challenge mentioned in Chapter 1. Namely, this Chapter focuses on the challenges that arise from *model mismatch*.

In robotics and control theory, whenever we are faced with a real plant—like a car or a plane or even something non-embodied like a social network—we first model the time-evolution of the relevant variables, the "state variables," and how they respond to the inputs we control. Following the convention, we shall refer to the state variables collectively as a vector x, the inputs or controls u, and time t. This input-state relationship is often described by a differential equation,

$$\dot{x} = f(t, x, u) \tag{2.1}$$

and often, we shall consider time-invariant dynamics where we drop the dependence of f on t.

Example 1. *Consider a pedestrian moving on the Cartesian plan. The state variables of interest may be varied, including such things as position, heading, and speed, but this is a matter of design choice.*

For example, we shall choose to neglect heading and speed and model the state dynamics as follows:

$$x \triangleq \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} . \tag{2.2}$$

Clearly, we have a lower-order dynamical model than we otherwise might have chosen with inputs u_x and u_y directly affecting the horizontal and vertical positions p_x and p_y of the pedestrian. Still, we shall see a variety of reasons that such low order model may be useful. In any case, this choice of state $x \triangleq (p_x, p_y)$ and input $u \triangleq (u_x, u_y)$ essentially defines the dynamics f, in this case trivially.

This model of the dynamics, f, is purely mathematical in nature and is at best only an approximation of the plant's true dynamics. Denoting the true dynamics f_p, the difference between the two may be inconsequential or highly consequential, depending upon the

problem at hand. This Chapter will present an approach to quantifying this model mis-
match, called FaSTrack [18], and several recent applications. FaSTrack addresses the issue
of model mismatch specifically for the case of *motion planning*, which as we shall see is a
fundamental challenge in robotics. This Chapter proceeds to introduce motion planning and
the core mathematical concepts behind FaSTrack, namely zero-sum differential games, and
then introduce three applications.

Motion planning

Motion planning is a core problem in robotics, and like simultaneous localization and map-
ping it is probably one of the great success stories the field. In motion planning, we essentially
try to design a path which an autonomous system—in this literature, typically a *robot*—can
follow to satisfy some constraints—typically avoiding obstacles. As the reader might expect,
this is a fundamental challenge for robotics, and it is especially relevant for robots that *move*,
or mobile robotics. For example, consider the following:

Example 2. *A quadrotor has dynamics*

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ g \tan u_x \\ -g \tan u_y \\ g - u_z \end{bmatrix}. \tag{2.3}$$

*This is, of course, a drastic simplification that does not even track things like angular
velocities as state variables or account for complicated effects like air friction, but as we will
see in this Chapter it is still useful for designing a robust controller. The state variables
in this model are position and speed, but notably* not *angles, whereas the control variables
are the pitch u_x, roll u_y, and thrust acceleration u_z, and the acceleration due to gravity is
$g = 9.81 \, \mathrm{m \, s^{-2}}$. A fact which we shall ignore in this Chapter, yet is certainly worth noticing,
is that this model is actually* linear *with a slight transformation of the control variables u_x
and u_y.*

To design a trajectory for the quadrotor of Example 2, we must specify a sequence of
control inputs $u \equiv (u_x, u_y, u_z)$ that correspond to motion of the given system along that path
through the state space. Doing so may be simple—in this case, since a simple transformation
renders the system *linear* (multiple double integrators), it is actually straightforward in the
absence of obstacles—or complicated, depending upon the structure of the model.

Although it certainly has a long history, much of the modern interest in motion planning
falls into two categories: sampling-based (global) and optimization-based (local) methods.
We will see examples of both methods in this dissertation, but at least in this Chapter all of
the core ideas will apply equally well to both. This Chapter will *not* attempt to provide a
full summary of these types of methods; rather, the interested reader is encouraged to refer
standard texts on robotics and model predictive control such as [25] or [5]. One fact which
we note here is that [5] focuses on linear systems although actually model predictive control

also applies and is commonly used for nonlinear systems. The reader is encouraged to refer to [6] for a good description of the general principles.

Regardless, a common problem faced in motion planning is that practical algorithms need to be *real-time*, or operate at around 10 Hz. This requirement is simple to understand; when a robot is operating in the wild, it should always be following a plan generated based on the latest sensor information about the world around it and where obstacles may be. There has been a significant amount of research on this problem since it is so fundamental, and as we have already mentioned, there are a number of approaches prevalent in the literature which we shall not dwell upon here. Rather, we shall emphasize that the vast majority of these approaches *do not* account for a common problem familiar to anyone working with a physical robot: physical systems often differ in significant ways from our best attempts to model them mathematically. Section 2.1 will present the main ideas behind a class of robust control algorithms which we will use to address this key challenge.

Robust control as a zero-sum game

There are many different approaches to robust control, and for a more complete overview, the reader should refer to a classic textbook such as [56] or (for linear control) [17]. Here, we consider one particular interpretation of robust control in which we imagine the presence of a (typically bounded) *disturbance* to the dynamical model we would otherwise use as an abstraction of the physical system. For example, in Example 3 we present a version of the dynamics from Example 2.

Example 3. *Recall the dynamics presented earlier, in Example 2. We can account for some error in these dynamics, e.g. due to a wind blowing on the quadrotor at a certain speed, by rewriting the dynamics as*

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{p_z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ v_y + d_y \\ v_z + d_z \\ g \tan u_x \\ -g \tan u_y \\ g - u_z \end{bmatrix}. \tag{2.4}$$

Here, the disturbance is given by $d \equiv (d_x, d_y, d_z)$ and is typically taken to be bounded in a box such that $|d_x| < \bar{d}_x, |d_y| < \bar{d}_y$, and $|d_z| < \bar{d}_z$. Our model essentially encodes the restriction that "nature" can only affect the velocity of the vehicle and not its acceleration, and it cannot impart any angular velocity. This is clearly an assumption; yet, it may be a simplification we are willing to make. In this case, since we have chosen not to model angles as state variables, it may be reasonable to neglect nature's effect on them by imparting a torque.

In general, we shall consider disturbed dynamics as a modification of (2.1), i.e.:

$$\dot{x} = f(t, x, u, d). \tag{2.5}$$

Further, we shall consider both u and d to be bounded, and although the reasons for considering primarily additive and bounded control and disturbance will be clear enough from

the technical details, we note that this choice is not strictly necessary and, at least in the case of disturbances, derived from early work by R. Isaacs, who invented the area of differential game theory which is largely the focus of this Section. His book [19] is a wonderful introduction to differential game theory and has been a great resource for the author.

In any case, we model "nature" as getting to choose the value of the disturbance in order to maximize (make worse) our control objective, whatever it may be. Thus, we have a zero-sum, two player differential game:

$$V_T(t, x_0) = \sup_{d[u](\cdot)} \inf_{u(\cdot)} \mathcal{V}_{\mathcal{T}}(t), x(t) = x_0 \,. \tag{2.6}$$

Here, $\mathcal{V}_{\mathcal{T}}(t)$ is a completely arbitrary control objective (which we wish to minimize), which is understood without loss of generality to exist on time $t \in [0, T]$ and in particular to *neglect the trajectory before time* t, and the notation $d[u](\cdot)$ is meant to capture the "non-anticipative" information pattern present. In other words, we wish to constrain nature to only choose a disturbance signal $d[u](\cdot)$ which depends *causally* on the control signal $u(\cdot)$. That is, nature gets to look at the current and prior values of the control, but crucially, does not have access to the future. The function $V_T(t, x)$ is called the *value function* corresponding to objective $\mathcal{V}_{\mathcal{T}}(t)$. Further, we have assumed that trajectories of the system begin at initial state x_0, i.e. $x(t) = x_0$.

One fact which bears mentioning and which has been known since at least the late 1960s is that the Nash equilibria of these differential games are given by a set of coupled Hamilton-Jacobi (HJ) partial differential equations (PDEs) [42]. A Nash equilibrium is a set of strategies for each player where no player has any rational incentive to deviate from his or her strategy. Nash equilibria are generally *not* unique and it is not always clear if they are the best encapsulation of the notion of equilibrium for a given problem; yet, as we shall see, Nash equilibria are appropriate for the problems considered in this Chapter, as shall be made clear in the remainder of this Section.

Depending upon the structure of the objective $\mathcal{V}_{\mathcal{T}}$, we can sometimes express the problem in (2.6) as an equivalent so-called Hamilton-Jacobi-Isaacs (HJI) PDE. In particular, we shall presume that the objective takes the form of an accumulated running cost plus a final cost, i.e.,

$$\mathcal{V}_{\mathcal{T}}(t_0, x_0) \triangleq \int_{t_0}^{T} g(s, x, u, d)ds + L(x(T))$$
$$\mathcal{V}_{\mathcal{T}}(T, x_0) = L(x_0), x(t_0) = x_0 \tag{2.7}$$

With this assumption, we can write down the corresponding HJI equation:

$$\frac{\partial V_T}{\partial t} = -\min_u \max_d \left\{ \left\langle \frac{\partial V_T}{\partial x}, f(t, x, u, d) \right\rangle + g(t, x, u, d) \right\} \,. \tag{2.8}$$

Here, we see that at each point in time, the value function V_T evolves according to a minimax version of the inner product of the its spatial gradient and the dynamics themselves, which can be understood as the directional derivative of the value function along the flow field defined by the dynamics. Of course, we may need to bound u and d.

Furthermore, note the ordering of the min and max in (2.8). This expression of the
HJI equation makes it clear that, at each moment in time, the disturbance must choose its
value *after* the control. In other words, we are giving the instantaneous advantage to the
disturbance for added robustness. In practice, however, we shall make an assumption known
as *Isaacs' condition* which, hence the name, dates from Isaacs' initial work on differential
games [19], that the ordering of the max and min do not affect the solution. This may be a
strong assumption in some cases; yet, the reader can readily see that this describes real-time
decision making in essentially any realistic setting.

Although the choice of objective $\mathcal{V}_{\mathcal{T}}$ can be arbitrary, we shall be primarily concerned
here with a type of *reachability* problem in which $\mathcal{V}_{\mathcal{T}}$ does not have a running cost and we
shall consider final costs that are the minimum over the time interval $[0, T]$ of a function,
i.e.,

$$\mathcal{V}_{\mathcal{T}}(t_0, x_0) = \min_{t \in [t_0, T]} \ell(t, x), x(t_0) = x_0 \,. \tag{2.9}$$

With this assumption, we can rewrite a HJI equation that corresponds taking care to
handle the min properly, see e.g. [34]:

$$\frac{\partial V_T}{\partial t} = -\min\left\{0, \min_u \max_d \left\langle \frac{\partial V_T}{\partial x}, f(t, x, u, d) \right\rangle\right\} \,. \tag{2.10}$$

This looks complicated, but the addition of the min with zero can be understood to
ensure that, in forward time, the value function can only get more positive. This means
that, in backward time, the value function can only get more negative and hence the value
function will never be above a value *it will later take*. Thus, the value function captures the
"min over time" character of (2.9) at each moment in time that the value function exists.
As before, we also require the value function to agree with the cost at the final time.

Example 4. *Consider a quadrotor with the dynamics from Example 3. The objective is
expressed as the minimum distance to the origin over time, i.e.:* $\mathcal{V}_{\mathcal{T}}(0, x_0) = \min_{t \in [0, T]} \|x(t)\|_2$
where $x(0) = x_0$. *The corresponding HJI equation for this reachability problem is*

$$\frac{\partial V_T}{\partial t} = -\min\left\{0, \min_u \max_d \left\langle \frac{\partial V_T}{\partial x}, f(t, x, u, d) \right\rangle\right\} \tag{2.11}$$

$$= -\min\left\{0, \min_u \max_d \left(V_T^{p_x}(v_x + d_x) + V_T^{p_y}(v_y + d_y) + V_T^{p_z}(v_z + d_z)\right.\right.$$
$$\left.\left. V_T^{v_x}(g\tan u_x) - V_T^{v_y}(g\tan u_y) + V_T^{v_z}(g - u_z)\right)\right\} \,. \tag{2.12}$$

Here, we have used the shorthand $V_T^{(\cdot)} \equiv \frac{\partial V_T}{\partial(\cdot)}$. *At this point, we note that the min and
max can be separated. This is a general result of Isaacs' condition! We continue:*

$$\frac{\partial V_T}{\partial t} = -\min\left\{0, \min_u V_T^{v_x}(g\tan u_x) - V_T^{v_y}(g\tan u_y) + V_T^{v_z}(g - u_z)\right.$$
$$\left. + \max_d V_T^{p_x}(v_x + d_x) + V_T^{p_y}(v_y + d_y) + V_T^{p_z}(v_z + d_z)\right\} \,. \tag{2.13}$$

*Of course, we cannot forget about the final condition that the value function must satisfy,
namely:* $V_T(T, x_0) = \|x_0\|_2$. *The final answer, then, meets this final value condition but also
satisfies the first order condition above.*

*One other detail which determines the solution but which we have ignored so far is the
control and disturbance bounds above. Presuming that these are an interval as in Example 3,
the min and max can be easily solved according to the sign of the partial derivative of V_T.
We omit these details in this example.*

Robust *tracking* control

Thus far we have introduced the idea of nature "playing against us;" as we shall see, this is a
powerful idea that gives us a mechanism for addressing *tracking* control problems. Tracking
control problems arise specifically when the objective encodes a notion of tracking perfor-
mance for a specific trajectory. A simple example, though by no means the only one, is of
tracking a stationary reference signal. A more complicated example is given in Example 5.
This type of control is typically used to follow a desired trajectory which may be generated
by a motion planning algorithm, as in Figure 1.1.

Example 5. *Suppose a system with dynamics $\dot{x} = f(t, x, u)$ is trying to track the motion of
another system $\dot{\hat{x}} = h(t, \hat{x}, w)$ in the same state space, i.e. we are interested in minimizing
the following control objective:*

$$\mathcal{V}_{\mathcal{T}} = \max_{t \in [0,T]} \|x - z\|_2 \,. \tag{2.14}$$

*Although this objective is expressed as max over time rather than a min, it is relatively
simple to come up with the corresponding HJI equation as before. To do so, we simply replace
the first min in (2.10) with a max, i.e.:*

$$\frac{\partial V_T}{\partial t} = -\max \left\{ 0, \min_u \max_d \left\langle \frac{\partial V_T}{\partial x}, f(t, x, u) - h(t, \hat{x}, w) \right\rangle \right\} \,. \tag{2.15}$$

*where the difference $f - h$ corresponds to the relative dynamics of $x - \hat{x}$, and the arguments
of the value function $V_T(\cdot, \cdot)$ are (respectively) time t and relative state $r = x - \hat{x}$.*

Following the core idea of Example 5 and combining it with the notion of adversarial
robustness above, we can pose the problem of tracking an *a priori* unknown reference in
Eq. 2.16, taking r to be the relative state, which is generally some function of the system's
state and the reference. We shall assume that the reference follows dynamics $\dot{\hat{x}} = h(t, \hat{x}, w)$,
In virtually all our examples and applications, we shall assume that $\hat{x}$ and x share the same
relevant dimensions (e.g. position), and that hence the computation of relative state r will
essentially amount to a simple subtraction, possibly with some zero-padding. For simplicity,
here we shall take $r = x - \hat{x}$ and as in Example 5 the objective is maximum relative distance
over $t \in [0, T]$. The problem may be expressed as

$$V_T(t_0, x_0) = \sup_{w[u](\cdot)} \sup_{d[u](\cdot)} \inf_{u(\cdot)} \left\{ \max_{t \in [0,T]} \|r\|_d \right\} \tag{2.16}$$

where the notation $\| \cdot \|_d$ indicates measuring a Euclidean norm but only in position di-
mensions (i.e., distance). The problem of finding a minimal relative state representation is
known to be extremely challenging in general [19].

As before, this zero-sum, effectively two-player differential game has a corresponding HJI equation as follows:

$$\frac{\partial V_T}{\partial t} = -\max\left\{0, \min_u \max_d \max_w \left\langle \frac{\partial V_t}{\partial x}, f(t, x, u, d) - h(t, \hat{x}, w) \right\rangle \right\}. \qquad (2.17)$$

Thus, we have arrived at a general approach which provides strong guarantees of tracking performance. That is, our formulation allows for worst-case disturbance and reference behavior, and provided we can solve the corresponding HJI equation given in (2.17), we can quantify exactly how far the disturbed system could get from the reference over time horizon T.

One fact which bears mentioning is that, once we have solved (2.17), the solution V_T is *controlled-invariant*, i.e., if the tracker applies control consistent with the arg min in (2.17) then the value of V_T will not increase along trajectories of the relative system. Accordingly, we have that the smallest nonempty sublevel set of V_T has a special meaning—it is the set of relative states that, applying the arg min control from before and provided that the relative system starts in a relative state contained in that set, the tracker can keep the relative distance (measured in the $\| \cdot \|_d$ norm) within the corresponding value of V_T. This smallest relative distance is called the *Tracking Error Bound*, or TEB. We shall call the corresponding set of states $\mathcal{S}$.

In practice, it may be useful to consider the case of $T = \infty$, i.e., infinite time horizon. Although this may not be strictly necessary in practice, it certainly makes matters simpler for guarantees of tracking performance to hold for all time and not only for an interval. Although a general theory for when relative dynamics afford such a solution remains elusive, in practice we find an infinite horizon solution by simply choosing a large value of T and halting computation backward in time whenever the "current" solution V_T is not changing very much.

2.2 Applications

The ideas of the previous Section, and especially (2.17), form a general-purpose guaranteed tracking tool known as FaSTrack [18], short for "Fast and Safe Tracking." In the remainder of this Chapter, we shall discuss some of the ways in which FaSTrack can be used in robotics applications, and we shall restrict our attention to some of the fundamental issues we face in such applications.

2.3 Multiple planning algorithms

Motion planning, as has been discussed already, is an extremely fundamental problem in robotics, and especially in mobile robotics. Correspondingly, the community has devoted significant research to it, and two major classes of motion planning algorithms have gained popularity: sample-based and optimization-based. We shall not go into detail on these approaches here, rather, the interested reader is encouraged to consult [21, 25, 16] which present common sampling methods, as well as [55, 5] which discuss optimization and model

predictive control. So far, we have presented a novel methodology for achieving *a prior*
known tracking performance despite adversarial disturbances in real-time, FaSTrack.

Still, there may be certain algorithms for planning that work especially well under some
circumstances—e.g., in cluttered spaces—and other algorithms that work well in different
circumstances—e.g., in mostly free space. For example, we may prefer an optimization-
based local method (see, e.g., [55]) in relatively open areas where we can expect there to be
minimal suboptimality issues due to the local nature of the solver, and a global sampling
method (e.g., [16]) in cluttered spaces. Hence, this Section addresses how we can modify
FaSTrack to enable the safe switching between multiple underlying planning algorithms, and
is largely adapted from work presented at ICRA 2018, titled "Planning, Fast and Slow: A
Framework for Adaptive Real-Time Safe Trajectory Planning" [15], coauthored with Sylvia
Herbert, Jaime Fisac, Sampada Deglurkar, and Claire Tomlin.

Problem formulation

We shall restrict our attention to static environments and time-invariant dynamical systems,
and operate in the situation where the different planners all use the same geometric algorithm
(in this case, [16]), yet differ in the speed with which the expect the robot to move along
planned trajectories. Let us consider planners $\{\pi_i\}_{i=1}^N$, and assume that planners are sorted
in order of strictly increasing speed. In this case, they also have strictly increasing tracking
error bound. In fact, we have that the set of relative states corresponding to the TEB for
planner π_i is contained within that of $\pi_j, j > i$. Moreover, we shall assume that transitions
are only made between π_i and π_j if $|i - j| = 1$, i.e., if the planners are adjacent. Thus, the
problem reduces to ensuring that such transitions $\pi_{i \to j}$ are *safe*, which we formalize as the
relative state ending up within the set of states $(\mathcal{S}_j)$ corresponding to the final planner π_j.

The dynamics of the robot, a small quadcopter, are modeled as in Example 3 and account
for a disturbance (e.g., wind) d bounded in magnitude by $\overline{d}$, i.e.:

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{p_z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ v_y + d_y \\ v_z + d_z \\ g \tan u_x \\ -g \tan u_y \\ g - u_z \end{bmatrix}. \tag{2.18}$$

We shall also model each geometric planner π_i as following dynamics

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{p_z} \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{2.19}$$

with control inputs $|\{p_x, p_y, p_z\}| < \{\overline{p}_{x,i}, \overline{p}_{y,i}, \overline{p}_{z,i}\}$, and increasing $\overline{p}_{.,i} > 0$ for sequential i.
Relative dynamics are straightforward: we subtract the position dimensions of the state
which align in both models and carry over the velocity dimensions which only appear in

(2.18). Thus, we have

$$
\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x - p_x \\ v_y + d_y - p_y \\ v_z + d_z - p_z \\ g \tan u_x \\ -g \tan u_y \\ g - u_z \end{bmatrix}. \tag{2.20}
$$

For fixed bounds on u and d, it is clear that increasing the planner's maximum speed $\bar{p}$ will only increase the size of the TEB. In fact, we can prove the following result:

Theorem 1. *For the relative dynamics above in* (2.20) *and a sequence of planners with increasing maximum speed $\bar{p}$ and fixed bounds on u and d, the TEB set $\mathcal{S}_i$ corresponding to π_i (with max speed $\bar{p}_i$) is included by all sets $\mathcal{S}_j, j < i$.*

Proof. Suppose relative state $r \in \mathcal{S}_j$ for $j < i$ and some fixed j. Then, there exists a sequence of optimal controls p for planner π_j, at each time within the bounds specified by $\bar{p}_j$, such that $r \in \mathcal{S}_j$, or equivalently, that from r the relative system will stay within distance $\mathcal{E}_j$ (measured by $\| \cdot \|_d$) of the origin. Holding bounds on u (for the tracker) and d (for nature) fixed, it is clear that increasing the planner's input bound $\bar{p}_j$ will not disqualify planner j's input trajectory as a means to maintain $\mathcal{E}_j$. Hence, $r \in \mathcal{S}_i$ for $i < j$, as required. $\square$

Achieving safe transitions

Safe transitions from π_i to π_j fall into two categories:

(a) $i < j$. Here, the transition is actually to a planner with a larger TEB whose corresponding set of relative states includes that of the next planner π_j, i.e., by Theorem 1 $\mathcal{S}_i \subset \mathcal{S}_j$. Hence, the transition is trivial, and we can safely transition from π_i to π_j along the planned path using the controller associated to π_j.

(b) $i > j$. In this case, the transition is to a planner with smaller TEB, and by Theorem 1, $\mathcal{S}_j \subset \mathcal{S}_i$. We handle this case by solving a separate reachability problem with the relative system's target being $\mathcal{S}_j$ and the planner's input bound $\bar{p}_j$. This idea is presented later within this Section.

Thus, we only need focus on case (b). As above, we address this challenge by again using reachability, and consider the HJI equation for the relative system, namely (2.17), and copied below and adapted to the dynamics of (2.20):

$$
\frac{\partial V_T}{\partial t} = -\max \left\{ 0, \min_u \max_d \max_w \left\langle \frac{\partial V_t}{\partial x}, f(t, x, u, d) - h(t, \hat{x}, w) \right\rangle \right\}. \tag{2.21}
$$

Crucially, we must also satisfy the final condition that the relative state be within the set corresponding to the tracking error bound for the planner we are switching *into*. That is, if we are switching from planner i to j, then we must satisfy the final condition that the relative state be within $\mathcal{S}_j$.
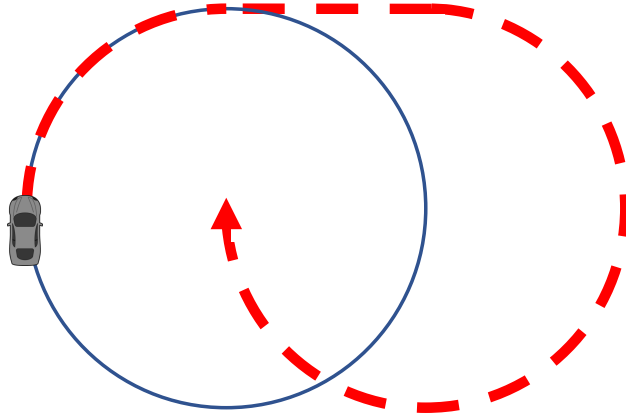
Fig. 2.1: A Dubins car attempting to track a geometric planner, shown in the limit where
the planner's maximum speed $\bar{p}$ is very close to zero. In order to track the planner more
closely, the Dubins car must first exit its initial orbit (solid blue curve), as shown by the
dashed red curve.

We must solve the PDE over an *a priori* unspecified time interval, such that at the initial
time the preimage of $\mathcal{S}_j$ includes $\mathcal{S}_i$. To encode both this property and the aforementioned
terminal condition, we solve the final value HJI PDE above, with the terminal condition that
$L(r_0)$ for some relative state r_0 be the signed distance of r_0 to $\mathcal{S}_j$ (i.e., negative if $r_0 \in \mathcal{S}_j$,
positive if $r_0 \notin \mathcal{S}$, and zero otherwise). We solve the PDE for an *a priori* unspecified time
interval until the initial condition is met. That is, we solve backward in time until the
smallest nonempty sublevel set of the solution includes $\mathcal{S}_i$. The resulting maximum relative
distance is called the *Safe Switching Bound* or SSB, with a corresponding set of states $\hat{\mathcal{S}}_j$,
and the smallest time horizon for which this holds is $T_{i \to j}$. Moreover, by construction we
know that this safe switching set includes the set of relative states corresponding to the TEB
for both planners. This property is summarized in Lemma 1.

Lemma 1. *The states corresponding to a safe switch from planner i to j include those
associated to planner j's TEB as well as those for planner i. That is,*

$$\mathcal{S}_j \subseteq \hat{\mathcal{S}}_j \text{ and } \mathcal{S}_i \subseteq \hat{\mathcal{S}}_j. \tag{2.22}$$

Proof. The proof follows by construction. The first inclusion is a consequence of final condi-
tion in the HJI PDE above, namely that relative states must terminate within the tracking
error bound set $\mathcal{S}_j$ to be included in the safe switching set $\hat{\mathcal{S}}_j$. A similar argument applies
for the second inclusion which follows from the initial condition. $\square$

This property is illustrated in Figure 2.1, where a Dubins car is seeking to track a
geometric planner and in order to attain a lower tracking error it must first increase tracking
error. Here, the Dubins car has dynamics

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cos \theta \\ v \sin \theta \\ u \end{bmatrix}, \tag{2.23}$$

(a) Exact SSB **(b)** Over-approximation
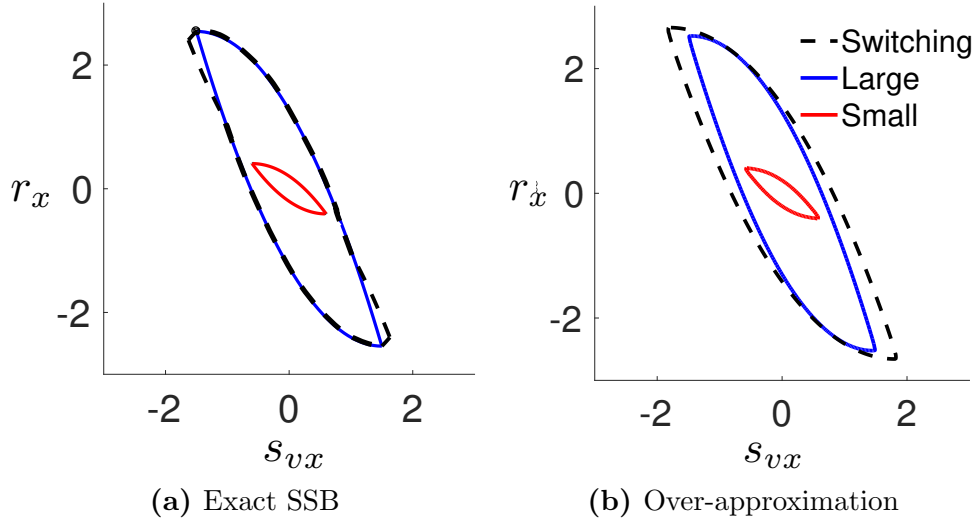
Fig. 2.2: Exact and over-approximated safe switching bounds and corresponding sets for the double integrator tracking a geometric planner, both with dynamics given in (2.25). Note that the SSB itself can be read out as the vertical extent of the dashed boundary, and that units are in meters. The blue and red sets correspond to the tracking error sets $\mathcal{S}$ for geometric planners with large and small maximum speeds $\bar{p}$, respectively.

and the geometric planner has dynamics given above in (2.2) and for clarity copied below:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \end{bmatrix} = \begin{bmatrix} w_x \\ w_y \end{bmatrix}. \tag{2.24}$$

Moreover, it should be noted that this safe switching set is really an over-approximation of the true set, since it accounts for trajectories which do not start within $\mathcal{S}_i$. The set can be computed easily for a tracker with double integrator dynamics following a geometric planner, with both dynamics given by

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \end{bmatrix} = \begin{bmatrix} v_x \\ u \end{bmatrix}, \; \dot{p}_x = w. \tag{2.25}$$

Figure 2.2 summarizes how the safe switching bound is an over-approximation in this case.

Building the meta-plan

We call a motion plan comprised of segments from a variety of planners a *meta-plan*. We shall construct a meta-plan as a route through a randomly-generated tree $\mathcal{T}$. We generate the meta-planning tree according to the following steps. Note that we shall presume that the first time this algorithm is run, planner 1 is used. There will, of course, be environments with obstacles near the start position for which this assumption means that the overall meta-planning algorithm will fail to find a solution; adapting to the more general case in which some other planner starts is also possible.

1. **Root.** Each meta-planning invocation begins a new rooted tree $\mathcal{T}$. Initially, the root is placed at the location of the tracker. In subsequent invocations, the root is placed

at the position along the existing meta-plan which corresponds to a short planning time (typically $\ll 1\,\mathrm{s}$) from now, and the current planner will be the one tried first during step 3. In the first instance, we must assume that the origin is within $\mathcal{S}_1$, and in the second, we note that—by the inclusion principle in Lemma 1 and Theorem 1— the relative state is certainly within either the safe switching set (if in the midst of a switch) or the tracking error set (otherwise). Also, we note that the *current planner* is the one that either we are currently using or, if in a switch, then it is the planner we are switching into. Finally, as shall be made clear in step 2, the root is characterized only by a geometric position. In general, the intent is that meta-plans be continuous in position.

2. **Sample.** Points are sampled uniformly at random from free space and each new sampled position is then added to $\mathcal{T}$ by connecting to its nearest neighbor in the tree. This connection process is detailed in steps 3 and 4.

3. **Plan.** Given a sampled position p_b and its nearest neighbor p_a, we first try to connect the two by running the first planner π_1, which has the largest TEB by Theorem 1. If π_1 successfully finds a trajectory that is collision-free when augmented by the TEB for π_1, then we insert the sampled point p_b into the tree $\mathcal{T}$ with associated planner π_1. Otherwise, if it does not succeed, then we must try a planner with a smaller TEB, i.e., we proceed to try with planner π_2. By assumption, all the planners have the same dynamics (2.19) and hence a single algorithm may be run only once and all that needs change is the check for collisions; in general, if planner dynamics differed then it would be necessary to generate a new trajectory from p_a to p_b. If the second planner fails, then we proceed with the third and so on until we reach the planner which follows the one used by the nearest neighbor, i.e., until we try π_{a+1}. By assumption, transitions to subsequent planners are not allowed, but if we removed that assumption than of course, we would need to try all planners.

4. **Virtual backtrack.** Suppose that we have just inserted a newly-sampled point p_b with nearest neighbor p_a, then we must ensure that using we can safely use the planner associated with p_b, π_i. If that planner has a smaller tracking error set $\mathcal{S}_i$ than the planner used to reach p_a, i.e., if $i \geq a$ by Theorem 1, then the transition is automatically safe by construction. However, we need to be careful if that is not the case ($i < a$).

First, we shall simply try to transition from π_a to π_i by rechecking that the plan generated by π_i is collision-free while augmenting it with planner i's SSB, and we recall that by Lemma 1 the corresponding safe switching set $\hat{\mathcal{S}}_i$ includes the i'th tracking error set $\mathcal{S}_i$.

If this does not work, then we shall perform a *virtual backtrack*, which is illustrated in Figure 2.3. Essentially, failure to connect under planner i's SSB indicates that we must be already using π_i *by the time we reach p_a*. Thus, we shall attempt a transition one step in $\mathcal{T}$ earlier between p_a and its parent $p_{a'}$. To do so, we use planner π_i to find a trajectory from $p_{a'}$ to p_a, and assuming that it was not already used for that segment in $\mathcal{T}$ then it is collision-checked by augmenting it with the i-th SSB and ensuring that the length of the trajectory in time is at least the SSB's associated transition
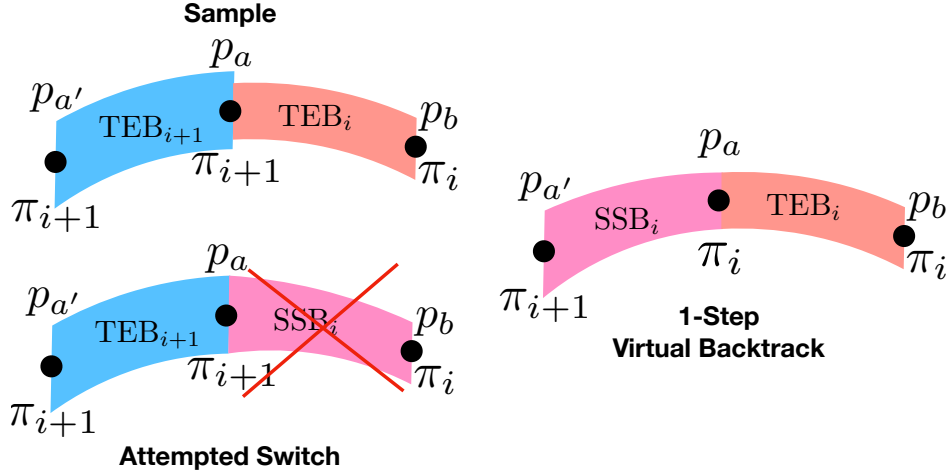
Fig. 2.3: Illustration of the virtual backtracking procedure. A newly-sampled point p_b is attached to its parent p_a with planner π_i in the top left. In the bottom left, we see that using the SSB associated with planner i to collision-check the plan from p_a to p_b fails. To the right, we execute a one-step virtual backtrack and successfully place the switch between p_a and its parent $p_{a'}$.

time $T_{i-1 \to i}$. Of course, since this SSB assumes an adjacent planner is being used at $p_{a'}$, i.e., that $\pi_j, j \in \{i-1, i, i+1\}$ is being used, then if a different planner is being used this procedure cannot work and the meta-planner will fail. If we allowed non-adjacent transitions, then SSBs would need to be computed beforehand for all possible transitions and we would not have this restriction. Regardless, if this one-step backtrack fails because the collision-check itself fails, then we could try a similar procedure recursively at each parent up to the root; in practice we terminate after a single step for simplicity.

It should be clear by construction that the above meta-planning procedure is sound, as summarized in the theorem below. The proof is omitted due to the clarity with which the result derives from the steps mentioned above.

Theorem 2. *If the meta-planner (which proceeds at each invocation according to the steps in Section 2.3) succeeds, i.e., it finds a sequence of plans connecting points in $\mathcal{T}$ between the root and the goal position, and if the physical system uses the associated TEB and SSB tracking controllers, then under the static environment and other assumptions regarding geometric planners, it will not result in collision.*

Proof. Although most of the statement is self-evident by construction, we shall draw attention to the more subtle situation in which the robot must *replan* at some random time, e.g., due to the observation of a new obstacle. Here, because the planners are all geometric, in the worst case (in which the meta-planner cannot find a suitable meta-plan to the desired goal), then it is always possible to follow the existing meta-plan in reverse back to the start position because the planners are geometric. If they were not, then this property would not hold; the more general case is discussed above. □

i	$\overline{d}_{\{x,y,z\}}$ $(\mathrm{m\,s^{-1}})$	$\overline{d}_{\{ax,ay,az\}}$ $(\mathrm{m\,s^{-2}})$	$\overline{p}_{\{x,y,z\}}$ $(\mathrm{m\,s^{-1}})$
1	0.5	0.1	1.0
2	0.4	0.1	0.8
3	0.3	0.1	0.6
4	0.2	0.1	0.4
5	0.2	0.1	0.2

Table 2.1: Disturbance bounds and maximum geometric planner speeds for software demonstration.

i	$\overline{d}_{\{x,y,z\}}$ $(\mathrm{m\,s^{-1}})$	$\overline{d}_{\{ax,ay,az\}}$ $(\mathrm{m\,s^{-2}})$	$\overline{p}_{\{x,y,z\}}$ $(\mathrm{m\,s^{-1}})$
1	0.6	0.1	0.4
2	0.6	0.1	0.3
3	0.6	0.1	0.2

Table 2.2: Disturbance bounds and maximum geometric planner speeds for hardware demonstration.

Results

To demonstrate the meta-planner in hardware, consider a quadrotor with dynamics similar to those modeled as in (2.18) following a geometric planner with the dynamics of (2.19). For clarity, these the relative dynamics are copied from (2.20) in (2.26) with minor modification to account for acceleration disturbances $\{d_{ax}, d_{ay}, d_{az}\}$:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x - p_x \\ v_y + d_y - p_y \\ v_z + d_z - p_z \\ g\tan u_x + d_{ax} \\ -g\tan u_y + d_{ay} \\ g + d_{az} - u_z \end{bmatrix}. \tag{2.26}$$

For all examples, we shall use $-0.15\,\mathrm{rad} \leq \{u_x, u_y\} \leq 0.15\,\mathrm{rad}$, set $7.81\,\mathrm{m\,s^{-2}} \leq u_z \leq 11.81\,\mathrm{m\,s^{-2}}$, and recall that $g = 9.81\,\mathrm{m\,s^{-2}}$ is the acceleration due to gravity. We also set the disturbance and planner input bounds according to Table 2.1 for all software demonstrations, and according to Table 2.2 for all hardware demonstrations. Also of note is that the dynamics of (2.26) effectively assume that the quadrotor is at zero yaw, which we maintain via an auxiliary PD controller.

Note that these dynamics (2.26) separate into three decoupled subsystems, one in each cardinal direction. For example, in the x-direction, we have the subsystem:

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \end{bmatrix} = \begin{bmatrix} v_x + d_x - p_x \\ g\tan u_x \end{bmatrix}. \tag{2.27}$$

Each of these subsystems may be solved independetly, e.g., using level set methods [32], and in fact due to their simplicity they also afford an analytic solution which is encoded in the

(a) LQR controller   **(b)** HJI controller
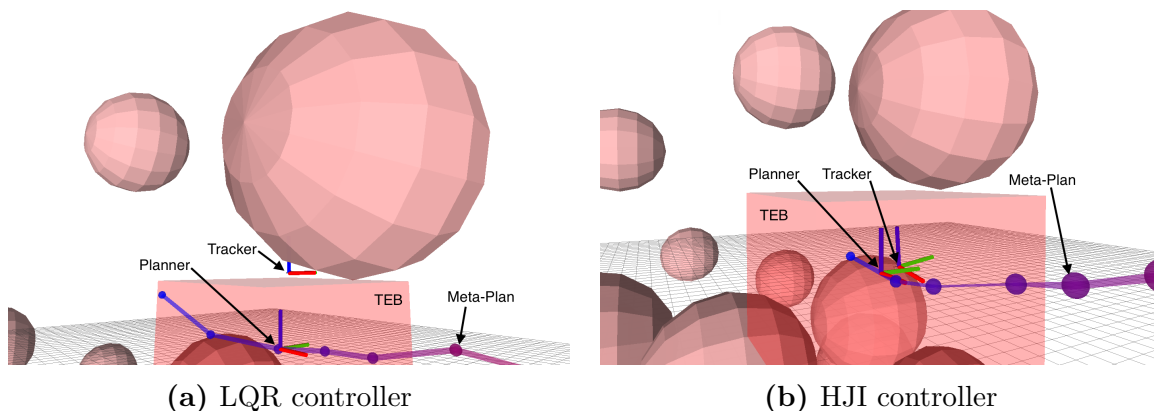
Fig. 2.4: In Figure 2.4a, using an LQR controller the tracker violates the TEB and collides
with an obstacle. In Figure 2.4b, the HJI controller from the meta-planner keeps the tracker
within the appropriate TEB and avoids obstacles.
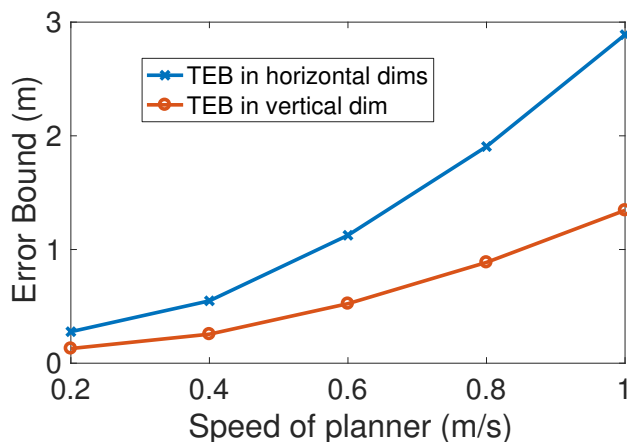
Fig. 2.5: The tracking error bound increases with the planner speed as shown and also with
increasing disturbance bounds.

corresponding software implementation of the meta-planner available at `https://github.`
`com/HJReachability/meta_fastrack`. To demonstrate the meta-planner in software, we
use the aforementioned implementation (which is entirely written in C++ and uses the
popular Robot Operating System (ROS) [36] framework for interprocess communication) and
use the disturbance and planner input bounds given in Table 2.1 as well as the BIT* geometric
planning algorithm [16] which is implemented in the Open Motion Planning Library (OMPL)
[45]. The meta-planning process typically completes in well under 1 s, and we see in Figure 2.4
that using the HJI-based controller specified during meta-planning avoids obstacles that a
simple linear feedback controller, e.g., obtained via LQR, does not. For simplicity, we operate
in a cube-shaped environment with smaller spherical obstacles as shown.

Additionally, we note that the TEB does increase in size as the disturbances get larger
and the planner gets faster. This relationship is summarized for the bounds of Table 2.1 in
Figure 2.5.

To test the meta-planner in hardware, we use a small Crazyflie 2.0 quadrotor and a motion
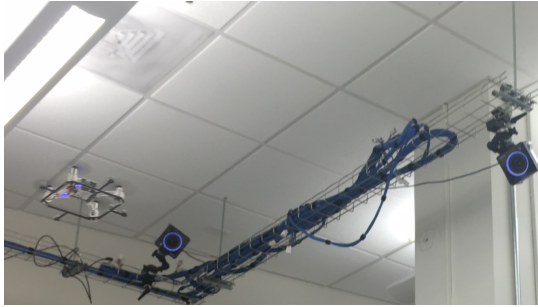
(a) Crazyflie 2.0 in motion capture room
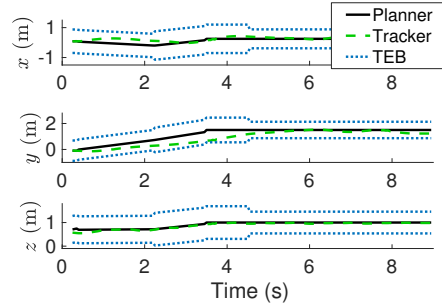
(b) Typical tracking data in hardware

Fig. 2.6: Figure 2.6a shows a Crazyflie quadrotor flying inside our motion capture arena. In Figure 2.6b, we see tracking data from a typical flight, including bounds for the quadrotor's position derived from meta-planning.

capture room for state estimation. ROS provides a convenient framework for interprocess communication across the necessary computers. Figure 2.6a shows our Crazyflie quadcopter flying in the motion capture room, and Figure 2.6b summarizes the tracking performance for a typical flight. Notice how the TEB has also changed during flight, and the HJI-based controller derived for safe switching successfully keeps the tracker within bounds at all times. All code for the project can be found at `https://github.com/HJReachability/fastrack`.

2.4 Incremental, safe exploration

The robust control approach underlying FaSTrack [18] also enables incremental exploration with strong non-collision guarantees. Incremental exploration is an important problem in mobile robotics which arises when a robot is initially unaware of its environment but must safely navigate to a pre-specified goal location. The robot is typically equipped with at least one sensor, and for the purposes of this Section we shall presume that it is a range sensor— that is, it detects whether the space within some field of view is free or occupied. The overall approach to addressing this type of problem fundamentally relies upon constructing motion plans that never sacrifice the ability to return to the starting position. This Section is based upon [11], which was coauthored with Jaime Fisac and Claire Tomlin.

Preliminaries

We consider an autonomous navigation task in a bounded *a priori* unknown static environment $\mathcal{X} \subset \mathbb{R}^{n_x}$. The autonomous system has dynamic state $x \in \mathcal{S} \subset \mathbb{R}^{n_x}$, which includes, but is in general not limited to its location x in the environment $\mathcal{X}$. We presume that for each point $x \in \mathcal{X}$, the environment representation can assign a label from the set $\{\text{OCCUPIED}, \text{FREE}, \text{UNKNOWN}\}$. The system's knowledge of the environment will be updated online according to measurements from a well-characterized sensor, with field of view $\mathcal{F} : \mathcal{S} \to 2^{\mathcal{X}}$. In this work, we will restrict our attention to deterministic sensing models, i.e. if $x \in \mathcal{X}$ is within the sensor's field of view $\mathcal{F}(s)$, it will be correctly identified as either

$\{\text{OCCUPIED}, \text{FREE}\}$. Probabilistic extensions are possible, though beyond the scope of this paper.

We assume known time-invariant system dynamics, of the form:

$$\dot{x} = f(x, u, d) \ ,$$

where $u \in \mathcal{U} \subset \mathbb{R}^{n_u}$ is the system's control input and $d \in \mathcal{D} \subset \mathbb{R}^{n_d}$ is a bounded disturbance.

In general, the dynamical model f of the physical system will be nonlinear and high-order, making it challenging to compute trajectories in real time. Instead, we can use an approximate, lower-order dynamical model for real-time trajectory computation, along with a framework which produces a known tracking controller for the full-order model allowing it to follow the trajectories of the low-order model with a guarantee on accuracy. Let the simplified state of the system for planning purposes be $p \in \mathcal{P} \subset \mathbb{R}^{n_p}$, governed by approximate *planning dynamics*:

$$\dot{p} = g(p, c) \ ,$$

with $c \in \mathcal{C} \subset \mathbb{R}^{n_c}$ the control input of the simplified system, which we will refer to as the *planning system*.

We use the FaSTrack framework [18] to provide a robust controlled invariant set in the relative state space $\mathcal{R} \subset \mathbb{R}^{n_r}$ between the planning reference and the full system. This relative state depends on the dynamical models used. A concrete example will be presented in Section 2.4, and we direct the reader to [18, 15] for further discussion. The output of this robust analysis is two-fold: the autonomous system is given an optimal tracking control law $k^* : \mathcal{R} \to \mathcal{U}$ that will keep the relative state inside of this invariant set at runtime *regardless* of the low-order trajectory proposed by the planning algorithm. In turn, the planning algorithm can use the projection of the invariant set onto the planning state space $\mathcal{P}$ as a guaranteed tracking error bound $\mathcal{E} \subset \mathcal{P}$ for the purposes of collision-checking at planning time. A feature of the FaSTrack framework is that the robust safety analysis depends only upon the relative dynamics, and *not* on the particular algorithm used for planning low-order trajectories. We inherit this modularity in our recursively feasible planning framework, which can be used with an arbitrary low-level motion planner. In Section 2.4, we demonstrate our framework with a standard third-party algorithm from the Open Motion Planning Library (OMPL) [45].

Recursive feasibility: safety and liveness

We now define several important concepts more formally, as they pertain directly to the theoretical safety guarantees of our proposed framework. Let $\xi\big(\cdot; t_0, p, c(\cdot)\big) : \mathbb{R} \to \mathcal{P}$ denote the trajectory followed by the planning system starting at state p at time t_0 under some control signal $c(\cdot)$ over time.

Given a planned state p, we refer to its *footprint* $\phi(p)$ as the set of points $x \in \mathcal{X}$ that are occupied by the system in this state. We additionally define the robust footprint $\phi_{\mathcal{E}}(p)$ as the set of points $x \in \mathcal{X}$ that are occupied by some $p' \in \{p + \mathcal{E}\}$ (with $+$ here denoting Minkowski addition). This represents the set of locations that may be occupied by the physical system while attempting to track the planned state p. We will require that the system is at all times

guaranteed to only occupy locations known to be FREE. For convenience, we will denote by $\mathcal{X}_{\text{FREE}}(t)$ the set of points $x \in \mathcal{X}$ that are labelled as FREE at time t.

We then have the following definitions.

Definition 1. *A planned trajectory $\xi\big(\cdot; t_0, p, c(\cdot)\big)$ is known at time t_0 to be safe, i.e. collision-free, if it satisfies the following criterion:*

$$\forall t \geq t_0, \phi_{\mathcal{E}}\Big(\xi\big(t; t_0, p, c(\cdot)\big)\Big) \subseteq \mathcal{X}_{FREE}(t_0) \ .$$

Observe that Definition 1 is *not* a statement about stability, as in e.g. [4]. Dynamic stability is in fact neither a necessary nor a sufficient condition for safety understood as guaranteed collision (and failure) avoidance.

Definition 2. *The safe forward reachable set Ω_F of a set of states $\mathcal{L} \subseteq \mathcal{P}$ at time t_0 is the set of states $p' \in \mathcal{P}$ that are known at t_0 to be safely reachable from $\mathcal{L}$ under some control signal $c(\cdot)$.*

$$\Omega_F(\mathcal{L}; t_0) := \Big\{ p' \mid \exists p \in \mathcal{L}, \exists t \geq t_0, \exists c(\cdot), \forall \tau \in [t_0, t] :$$
$$\phi_{\mathcal{E}}\Big(\xi\big(\tau; t_0, p, c(\cdot)\big)\Big) \subseteq \mathcal{X}_{FREE}(t_0), p' = \xi\big(t; t_0, p, c(\cdot)\big)\Big\} .$$

Analogously, the safe backward reachable set Ω_B of $\mathcal{L}$ at t_0 is the set of states $p' \in \mathcal{P}$ from which $\mathcal{L}$ is known at time t_0 to be safely reachable under some control signal (this can also be thought of as the set of states $p' \in \mathcal{P}$ that can be safely reached from $\mathcal{L}$ in backward time, hence the name backward *reachable set):*

$$\Omega_B(\mathcal{L}; t_0) := \Big\{ p' \mid \exists p \in \mathcal{L}, \exists t \geq t_0, \exists c(\cdot), \forall \tau \in [t_0, t] :$$
$$\phi_{\mathcal{E}}\Big(\xi\big(\tau; t_0, p', c(\cdot)\big)\Big) \subseteq \mathcal{X}_{FREE}(t_0), p = \xi\big(t; t_0, p', c(\cdot)\big)\Big\} .$$

We will often consider reachable sets of individual states; for conciseness, we will write $\Omega_B(p; t_0)$ rather than $\Omega_B(\{p\}, t_0)$.

We now proceed to define viability in terms of these sets.

Definition 3. *A state p is* viable *at time t_0 with respect to a goal state p_{goal} and a home state p_{home} if at t_0 it is known to be possible to reach either p_{goal} or p_{home} from p while remaining safe, i.e. $p \in \Omega_B\big(\{p_{goal}, p_{home}\}; t_0\big)$. A trajectory ξ is viable at t_0 if all states along ξ are viable at t_0. Note that a trajectory can be safe (Def. 1) but not viable.*

Definition 4. *The safely explorable set $\mathcal{P}_{SE}(p) \subset \mathcal{P}$ of a state p is the collection of states that can eventually be visited by the system through a trajectory starting at state p with no prior knowledge of $\mathcal{X}$ whose states are, at each time $t \geq 0$, viable according to the known free space $\mathcal{X}_{FREE}(t)$.*

Based on the idea of the safely explorable set we can now introduce the important notion of *liveness* for the purposes of our work.

Definition 5. *A state p is live with respect to a goal state p_{goal} if it is possible to reach p_{goal} from p while remaining in the safely explorable set for all time, i.e if $p_{goal} \in \mathcal{P}_{SE}(p)$. A trajectory ξ is live if all states in ξ are live.*

Finally, we will refer to a planning algorithm as *recursively feasible* if, given that the initial state p_0 is live, all future states p are both live and viable. We will show that our proposed framework is recursively feasible. Moreover, we will also show that it is *safely probabilistically complete*, in the sense that, if p_0 is live with respect to p_{goal}, then we will eventually reach p_{goal} through continued guaranteed safe exploration, with probability 1.

General framework

Our framework is comprised of two concurrent, asynchronous operations: building a graph of states which discretely under-approximate the forward and backward reachable sets of the initial "home" state, and traversing this graph to find recursively feasible trajectories. Namely, we define the graph $\mathcal{G}_F := \{V, E\}$ of vertices V and edges E. Vertices are individual states in $\mathcal{P}$, and directed edges are trajectories ξ between pairs of vertices. $\mathcal{G}_F$ will be a discrete under-approximation of the current safe forward reachable set of the initial state p_{home}. We also define the graph $\mathcal{G}_B \subseteq \mathcal{G}_F$ to contain only those vertices which are in the safe backward reachable set of p_{home} and p_{goal}, and the corresponding edges. We use the notation $p \in \mathcal{G}_F$ to mean that state p is a vertex in $\mathcal{G}_F$, and likewise for $\mathcal{G}_B$.

We use following two facts extensively. They follow directly from the definitions above and our assumptions on deterministic sensing and a static environment.

Remark 1. *A trajectory ξ that is safe at time t_0 will continue to be safe for all $t \geq t_0$.*

Remark 2. *A state p that is in the safe forward or backward reachable set of another state p_0 at time t_0 will continue to belong to this set for all $t \geq t_0$, i.e. $\Omega_F(p_0; t_0) \subseteq \Omega_F(p_0; t)$ and $\Omega_B(p_0; t_0) \subseteq \Omega_B(p_0; t)$.*

Building the graph

We incrementally build the graph by alternating between outbound expansion and inbound consolidation steps. In the outbound expansion step, new candidate states are sampled, and if possible, connected to $\mathcal{G}_F$. This marks them as part of the forward reachable set of p_{home}. In the inbound consolidation step, we attempt to find a safe trajectory from forward-reachable states in $\mathcal{G}_F$ back to a state in $\mathcal{G}_B$, which is known to be *viable*. Successful inbound consolidation marks a state as either able to reach p_{goal} or safely return to p_{home}.

Outbound expansion. This process incrementally expands a discrete under-approximation $\mathcal{G}_F$ of the forward reachable set of the home state, $\Omega_F(p_{\text{home}}; t)$. Note that, by Remark 2, $\Omega_F(p_{\text{home}}; t)$ can only grow as the environment $\mathcal{X}$ is gradually explored over time and therefore any state p added to $\mathcal{G}_F$ at a given time t is guaranteed to belong to $\Omega_F(p_{\text{home}}; t')$ for all $t' \geq t$.

We add states to $\mathcal{G}_F$ via a Monte Carlo sampling strategy inspired by existing graph-based kinodynamic planners [21], illustrated in Fig. 2.7a. We present a relatively simple

(a) Outbound expansion.   **(b)** Inbound consolidation.
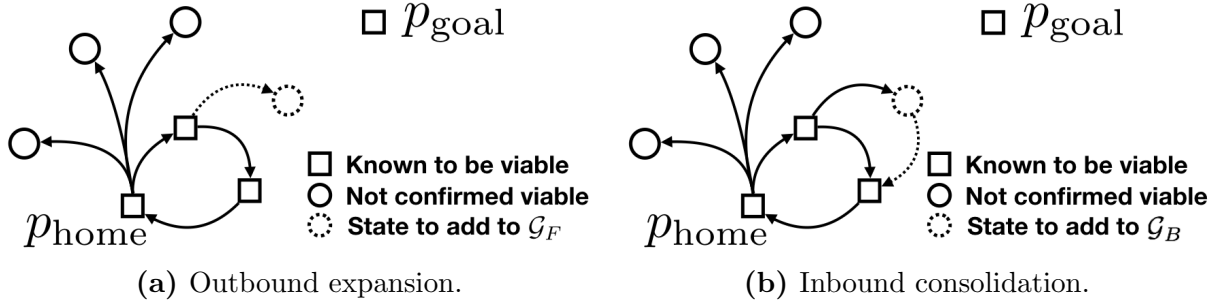
Fig. 2.7: In outbound expansion (a), a new state is sampled from $\mathcal{P}$ and added to $\mathcal{G}_F$ if safely reachable from $\mathcal{G}_F$. In inbound consolidation (b) a state in $\mathcal{G}_F$ is added to $\mathcal{G}_B$ if it can safely reach a (viable) state in $\mathcal{G}_B$.

strategy here, although more sophisticated options for sampling new states are possible, e.g. [22, 16].

Let p_{new} be sampled uniformly at random from $\mathcal{P}$ at time t such that $\phi_{\mathcal{E}}(p_{\text{new}}) \subseteq \mathcal{X}_{\text{FREE}}(t)$. We wish to establish whether or not p_{new} is in the safe forward reachable set of home at t, i.e. $p_{\text{new}} \in \Omega_F(p_{\text{home}}; t)$. This is accomplished by invoking a third-party motion planner, which will attempt to find a safe trajectory to p_{new} from any of the points already known to be in $\Omega_F(p_{\text{home}}; t)$. In Section 2.4, we use a standard kinodynamic planner from the OMPL [45] for this purpose.

We observe that repeatedly executing this procedure will, in the limit, result in a dense discrete under-approximation of $\Omega_F(p_{\text{home}}; t)$. Formally, assuming that the low-level planner will find a valid trajectory to a sampled state p if one exists, then for any $\epsilon > 0$, we have that the probability that a new sampled state $p' \in \Omega_F(p_{\text{home}}, t)$ will lie within a distance of ϵ from the nearest state $p \in \mathcal{G}_F$ goes to 1 in the limit of infinite samples. We formalize this observation below, which will be useful in proving the safe probabilistic completeness of our framework.

Lemma 2. *For all $\epsilon > 0$, assuming we sample candidate states p uniformly and independently from $\mathcal{P}$ and $\mathcal{P}$ is compact, then letting p_k be the k-th sampled state from $\mathcal{P}$ we have that $\forall t$:*

$$\lim_{k \to \infty} P\left(\min_{p \in \mathcal{G}_F} \|p_k - p\| < \epsilon \mid p_k \in \Omega_F(p_{home}; t) \right) = 1 \ .$$

Proof. This follows directly from the properties of uniform sampling from compact sets. $\square$

Inbound consolidation. This process incrementally adds states in $\mathcal{G}_F$ to a discrete approximation $\mathcal{G}_B$ of the safe *backward* reachable set of $\{p_{\text{home}}, p_{\text{goal}}\}$. By Definition 3, any state added to this set is *viable*, which means that a trajectory will always exist from it to either p_{goal} or p_{home}. This is a crucial element of our overall guarantee of recursive feasibility. We recall that $\mathcal{G}_B \subseteq \mathcal{G}_F$.

Suppose that $p \in \mathcal{G}_F \setminus \mathcal{G}_B$. We will attempt to add p to $\mathcal{G}_B$ by finding a safe trajectory from p to any of the states currently in $\mathcal{G}_B$ by invoking the low-level motion planner. If we succeed in finding such a trajectory, then by construction there exists a trajectory all the way to p_{home}, so we add p to $\mathcal{G}_B$. If p is added to $\mathcal{G}_B$, we also add all of its ancestors in $\mathcal{G}_F$

to $\mathcal{G}_B$, since there now exists a trajectory from each ancestor through p to either p_{home} or p_{goal}. This procedure is illustrated in Fig. 2.7b.

Exploring the graph

When requested, we must be able to supply a safe trajectory beginning at the current state reference $p(t)$ tracked by the system. Recall from Section 2.4 that under the robust tracking framework [18], the physical system's state $x(t)$ is guaranteed to remain within an error bound $\mathcal{E}$ of $p(t)$ measured on the planning state space $\mathcal{P}$. This property allows us to make guarantees in terms of planning model states p rather than full physical system states x.

Trajectories ξ output by our framework must guarantee future safety for all time; that is, as the system follows ξ we must always be able to find a safe trajectory starting from any future state. In addition, we require that p_{home} remains safely reachable throughout the trajectory; this ensures that *liveness* is preserved (if it was possible from p_{home} to safely explore $\mathcal{X}$ and reach p_{goal} then this possibility will not be lost by embarking on ξ). Note that liveness is an important property separate from safety: a merely safe planner may eventually trap the system in a periodic safe orbit that it cannot safely exit.

By construction, any cycle in $\mathcal{G}_B$ is safe for all future times (Remark 1). Readily, this suggests that we could guarantee perpetual recursive feasibility by always returning the same cycle. However, this naive strategy would never reach the goal. Moreover, it would not incrementally explore the environment. In order to force the system to explore unknown regions of $\mathcal{X}$, we modify this naive strategy by routing the system through a randomly selected *unvisited* state $p_{\text{new}} \in \mathcal{G}_B$, and then back to p_{home}. The trajectory always ends in a periodic safe orbit between p_{new} and p_{home}. Note that this random selection does not need to be done naively (e.g. by uniform sampling of unvisited states in $\mathcal{G}_B$), and efficient exploration strategies are certainly possible. In our examples we will use an ϵ-greedy sampling heuristic by which, with probability $1-\epsilon$, we select the unvisited $p \in \mathcal{G}_B$ closest to p_{goal}, and otherwise, with probability ϵ, we uniformly sample an unvisited state in $\mathcal{G}_B$.

Of course, if p_{goal} is ever added to $\mathcal{G}_B$, we may simply return a trajectory from the current state $p(t)$ to p_{goal}. This will always be possible because, by construction, every state in $\mathcal{G}_B$ is safely reachable from every other state in $\mathcal{G}_B$ (if necessary, looping through p_{home}).

Algorithm summary

To summarize, our framework maintains graph representations of the forward reachable set of p_{home} and the backward reachable set of $\{p_{\text{home}}, p_{\text{goal}}\}$. Over time, these graphs become increasingly dense (Lemma 2). Additionally, all output trajectories terminate at p_{goal} or in a cycle that includes p_{home}. This implies our main theoretical result:

Theorem 3. *Assuming that we are able to generate an initial viable trajectory (e.g. a loop through p_{home}), all subsequently generated trajectories will be viable and preserve the liveness of p_{home}. Thus, our framework guarantees* recursive feasibility.

Proof. By assumption, the initial trajectory ξ_0 output at t_0 is safe (Definition 1). We now proceed by induction: assume that the i-th reference trajectory ξ_i is viable for the knowledge of free space at the time t_i at which it was generated, i.e. $\forall t \geq t_i, \xi_i(t) \in \Omega_B(\{p_{\text{home}}, p_{\text{goal}}\}; t_i)$.

Assuming p_{goal} has not been reached yet at the time of the next planning request, t_{i+1}, a new trajectory will be generated from initial state $\xi_i(t_{i+1})$. The new trajectory ξ_{i+1} will be created by concatenating safe trajectories between states in $\mathcal{G}_B \subseteq \Omega_B(\{p_{\text{home}}, p_{\text{goal}}\}; t_i)$ and therefore will be a viable trajectory. Such a trajectory can always be found, because it is always possible to choose $\xi_{i+1} \equiv \xi_i$, which, by the inductive hypothesis was a viable trajectory at time t_i and, by Remark 2, continues to be viable at t_{i+1}. Therefore all planned trajectories ξ_i will retain the ability to either safely reach p_{goal} or safely return to p_{home}. In the former case, ξ_i is immediately live (and since $\forall t \geq 0, \xi_i(t) \in \Omega_F(p_{\text{home}}; t_i)$, p_{home} must have been live too); in the latter, ξ_i will inherit the liveness of p_{home}, by observing that $\forall t \geq 0, \xi_i(t) \in \Omega_B(p_{\text{home}}; t_i)$. $\qquad\square$

Corollary 1. *Given that the safety of trajectories is evaluated using the robust footprint $\phi_{\mathcal{E}}(\cdot)$, and the relative state between the dynamical system and the planning reference is guaranteed to be contained in $\mathcal{E}$, Theorem 3 implies that the dynamical system can continually execute safe trajectories in the environment.*

Moreover, we ensure that each output trajectory visits an unexplored state in $\mathcal{G}_B$, which implies that $\mathcal{G}_B$ approaches the safely explorable set $\mathcal{P}_{\text{SE}}(p_{\text{home}})$ from Definition 4. Together with Theorem 3, this implies the following completeness result:

Theorem 4. *In the limit of infinite runtime, our framework eventually finds the goal with probability 1 if it is within the safely explorable set.*

Proof. By Theorem 3, all trajectories output will be viable; hence, the autonomous system will remain safe for all time (Corollary 1). Further, since each generated trajectory visits a previously unvisited state in $\mathcal{G}_B$ with nonzero probability, by Lemma 2 it will eventually observe new regions in the safely explorable set $\mathcal{P}_{\text{SE}}(p_{\text{home}})$ if any exist. Moreover, those regions will eventually be sampled, added to $\mathcal{G}_B$, and visited by subsequent trajectories. Because we have assumed all sets of interest to be bounded, this implies that we will eventually add p_{goal} to $\mathcal{G}_B$ as long as $p_{\text{goal}} \in \mathcal{P}_{\text{SE}}(p_{\text{home}})$. $\qquad\square$

Remarks

We conclude this Section with several brief remarks regarding implementation.

In our discussion of building the graph, we specify that states should be connected to existing states in $\mathcal{G}_F$ and $\mathcal{G}_B$. In practice, we find that connecting to one of the k-nearest neighbors (measured in the Euclidean norm over $\mathcal{P}$) in the appropriate graph suffices.

In our discussion of exploring the graph, we describe traversing $\mathcal{G}_B$ to find safe trajectories between vertices. For efficiency, we recommend maintaining the following at each vertex: cost-from-home, cost-to-home, and cost-to-goal, where cost may be any consistent metric on trajectories (e.g. duration). If these quantities are maintained, then care must be taken to update them appropriately for descendants and ancestors of states that are added to $\mathcal{G}_F$ and $\mathcal{G}_B$ while building the graph.

Finally, we observe that outbound expansion, inbound consolidation, and graph exploration may all be performed in parallel and asynchronously.

Example

We demonstrate our framework in a real-time simulation, implemented within the Robot Operating System (ROS) software environment [36].

Let the high-order system dynamics be given by the following 6D model:

$$\dot{x} = \begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{z} \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x \\ g\cos u_1 \\ v_y \\ -g\sin u_2 \\ v_z \\ u_3 - g \end{bmatrix} \tag{2.28}$$

where g is acceleration due to gravity, the states are position and velocity in (x, y, z), and the controls are $u_1 = $ pitch, $u_2 = $ roll, and $u_3 = $ thrust acceleration. These dynamics are a reasonably accurate model for a lightweight quadrotor operating near a hover and at zero yaw.

We consider the following lower-order 3D dynamical model for planning:

$$\dot{p} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ c \end{bmatrix} \tag{2.29}$$

where v is a constant tangential speed in the Frenet frame, states are absolute heading θ, and (x, y) position in fixed frame, and control c is the turning rate. We interpret these dynamics as a Dubins car operating at a fixed z height z_{p}.

We take controls to be bounded in all dimensions independently by known constants: $u \in [\underline{u}_1, \overline{u}_1] \times [\underline{u}_2, \overline{u}_2] \times [\underline{u}_3, \overline{u}_3]$ and $c \in [\underline{c}, \overline{c}]$. In order to compute the FaSTrack tracking error bound $\mathcal{E}$, we must solve a Hamilton Jacobi (HJ) reachability problem for the *relative dynamics* defined by (2.28) and (2.29). In this case, the relative dynamics are given by:

$$\dot{r} = \begin{bmatrix} \dot{d} \\ \dot{\psi} \\ \dot{v}_T \\ \dot{v}_N \end{bmatrix} = \begin{bmatrix} v_T\cos\psi + v_N\sin\psi \\ -c - v_T\sin\psi + v_N\cos\psi \\ u_1\cos\theta - u_2\sin\theta + cv_T \\ -u_1\sin\theta - u_2\cos\theta - cv_T \end{bmatrix} \tag{2.30}$$

with the relative states d (distance), ψ (bearing), v_T (tangential velocity), and v_N (normal velocity) illustrated in Fig. 2.8a.

Fig. 2.8b is a 3D rendering of the FaSTrack value function [18] computed using level set methods [32]. The value function records the maximum relative distance between the high- and low-order dynamical models (i.e. d). In order to guarantee that, at run-time, the distance between the two systems does not exceed this value, the value function is computed by solving a differential game in which $c(\cdot)$ seeks to maximize the relative distance and $u(\cdot)$ seeks to minimize it. Observe in Fig. 2.8b that level sets of the value function with sufficiently high value are well-approximated by discs centered on the origin in (x, y). Thus, we approximate the TEB $\mathcal{E}$ by such a disc for rapid collision-checking during each call to the low-level motion planner. Since the high-order dynamics (2.28) do allow for variation in z,

(a) Relative states.

(b) Computed value function.
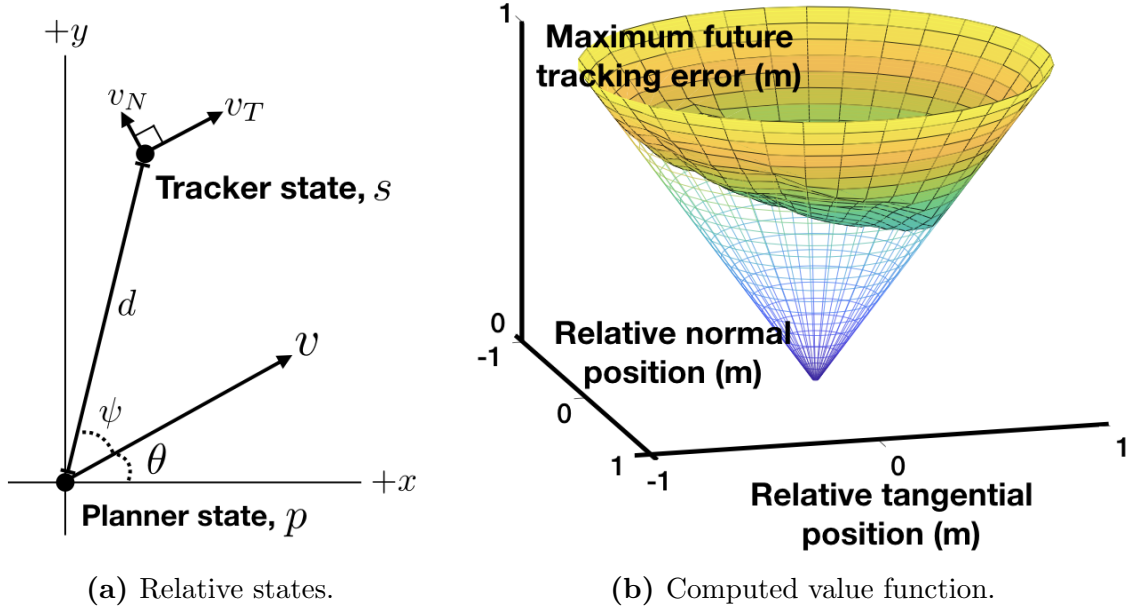
Fig. 2.8: (a) Relative states for 6D near-hover quadrotor tracking 3D Dubins car. (b) Minimum value over v_T and v_N, for each relative (x, y) position in the planner's Frenet frame. Any non-empty sublevel set can be used as a tracking error bound $\mathcal{E}$.

we also incorporate a z extent for $\mathcal{E}$ which may be obtained by solving a similar differential game in the (z, v_z) subsystem of (2.28), as in [15].

We use the KPIECE1 kinodynamic planner [44] within the Open Motion Planning Library (OMPL) [45] to plan all trajectories for the low-level dynamics while building the graphs $\mathcal{G}_F$ and $\mathcal{G}_B$. For simplicity, we model static obstacles as spheres in $\mathbb{R}^3$ and use an omnidirectional sensing model in which all obstacles within a fixed range of the vehicle are sensed exactly. We emphasize that these choices of environment and sensing models are deliberately simplified in order to more clearly showcase our framework. The framework itself is fully compatible with arbitrary representations of static obstacles and deterministic sensing models. Extensions to dynamic obstacles and probabilistic sensing are promising directions for future research.

We demonstrate our framework in a simple simulated environment, shown in Fig. 2.9, designed to illustrate the importance of maintaining recursive feasibility. This simulation is intended as a proof of concept; our central contribution is theoretical and applies to a range of planning problems.[1]

Observe in Fig. 2.9 that our method avoids collision where a non-recursively-feasible approach would likely fail. Here, the goal is directly in front of the home position and the way there *appears* to be in $\mathcal{X}_{\text{FREE}}(t)$. However, just beyond our sensor's field of view $\mathcal{F}$, there is a narrow dead end. Many standard planning techniques would either optimistically assume the unknown regions of the environment are free space, or plan in a receding horizon within known free space $\mathcal{X}_{\text{FREE}}(t)$. In both cases, the planner would tend to guide the system into the narrow dead end, leading to a crash (recall that the planner's speed v is fixed).

---

[1]Video summary available at: `https://youtu.be/GKQwFxdJWSA`
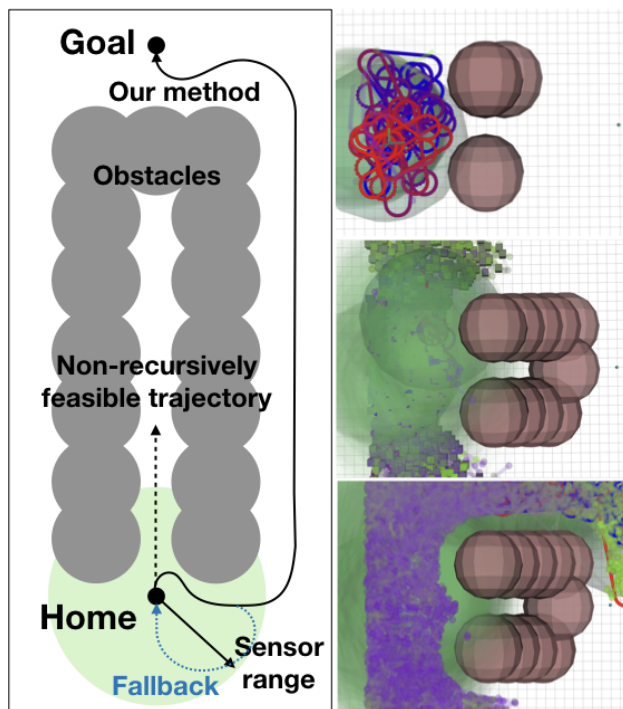
Fig. 2.9: Depiction of our framework in operation, using a Dubins car model with a fixed minimum turning radius and constant speed. *Left:* Schematic diagram of an environment in which a non-recursively feasible planning algorithm could enter a narrow dead end and fail to recover. *Right:* Snapshots of our framework over time. We build a search graph in known free space, identifying robustly viable trajectories that can safely return to the initial state or directly reach the goal. The physical system iteratively explores the environment along these recursively feasible plans and is eventually guaranteed to identify a viable trajectory to the goal, if one exists (bottom right).

By contrast, our approach eventually takes a more circuitous—but recursively feasible—route to the goal. The evolution of planned viable trajectories is shown on the right in Fig. 2.9. Initially, we plan tight loops near p_{home}, but over time we visit more of the safely explorable space $\mathcal{P}_{\text{SE}}(p_{\text{home}})$, and eventually we find p_{goal}. All code for the project can be found at `https://github.com/HJReachability/fastrack`.

2.5 A More Scalable Solution Strategy

Solving the Hamilton-Jacobi-Isaacs equation (2.10) is generally numerically challenging, although methods have been developed for highly structured systems [7] as well as which depend upon approximations [23, 24]. This Section shall develop one such approximate method based upon neural network classifiers, which is able to extend HJI methods for reachability to higher dimensional state spaces and, in some cases, maintain a guarantee of conservativeness. The following is based upon [38], which was coauthored with Vicenç Rubies-Royo, Sylvia Herbert, and Claire Tomlin.

Classification-based Approximate Reachability

In this Section we introduce our classification-based method for approximating the optimal control of HJ reachability when the dynamics are control-affine, and for simplicity we shall also assume time-invariance. Even though we will use feedforward neural networks to build the classifiers, it is possible to use other methods (e.g. SVM, decision trees). Ultimately, the choice of the classifier determines how conservative the results of the procedure will be. We leave a full investigation of classifier performance for future work.

Control- and disturbance-affine systems. A control/disturbance-affine system is a special case of (2.1) of the form

$$\dot{x} = \alpha(x) + \sum_{i=1}^{N_u} \beta_i(x)u_i + \sum_{j=1}^{N_d} \gamma_j(x)d_j \,, \tag{2.31}$$

recalling that $x \in \mathcal{S} \subset \mathbb{R}^n$ is the state, and taking $u \in \mathcal{U} \subset \mathbb{R}^{N_u}, d \in \mathcal{D} \subset \mathbb{R}^{N_d}$, and $\alpha, \beta_i, \gamma_j : \mathbb{R}^n \to \mathbb{R}^n$. We will assume that both control and disturbance are bounded by interval constraints along each dimension, i.e. $u_i \in [u_{min}^i, u_{max}^i]$ for $i = 1, \ldots, N_u$, and $d_j \in [d_{min}^j, d_{max}^j]$ for $j = 1, \ldots, N_d$. Observe that when dynamics f are of the form (2.31), the objective on the right-hand side of (2.10) is affine in the instantaneous control u' and disturbance d' at every time t. The optimal solution, therefore, lies at one of the 2^{N_u} (or 2^{N_d}) corners of the hyperbox containing u (or d). That is, the optimal control and disturbance policies are "bang-bang"[2] (we refer the reader to chapter 4 of [27]). Furthermore, the optimal values for any u_i or d_j at a certain state and time are mutually independent; therefore, for control/disturbance-affine systems, we can frame the HJI reachability problem of solving the right-hand side of (2.10) as a series of $N_u + N_d$ *binary* classification problems at each time.

Dynamic programming with binary classifiers. Algorithm 1 describes the process of learning these classifiers in detail. We begin by discretizing the time-horizon T into small (evenly spaced) intervals of size $\Delta t > 0$ in line 3, and proceed to use the dynamic programming principle backwards in time to build a sequence of approximately optimal control and disturbance policies. In total, the number of classifiers will be $\frac{T}{\Delta t}(N_u + N_d)$.

At an intermediate time $t < 0$, we will have already obtained the binary classifiers for the control and disturbance policies from $t + \Delta t$ to 0: $\Pi_{(t+\Delta t):0}^u$ and $\Pi_{(t+\Delta t):0}^d$. Here, Π_τ^u and Π_τ^d each denote a *set of classifiers* for the discrete time step τ (i.e. $|\Pi_\tau^c| = N_u$ and $|\Pi_\tau^d| = N_d$). We now define the function C, which computes the cost (2.6) if control and disturbance acted according to these pre-trained policies:

$$C(x, \Pi_{(t+\Delta t):0}^u, \Pi_{(t+\Delta t):0}^d) := \mathcal{V}(t, x, u(\cdot), d(\cdot)) \,, \tag{2.32}$$

where, due to our discretization, control and disturbance are piecewise constant over time, i.e. $u(t) = \Pi_\tau^u$ and $d(t) = \Pi_\tau^d$ for $t \in [\tau, \tau + \Delta t)$ and all discrete time steps τ.

At time t, we can determine for some arbitrary state x the optimal control and disturbance by solving the right-hand side of (2.10) as follows. First, compute the cost of applying

---

[2]For many physical systems, it is preferable to apply a smooth control signal. We note that the bang-bang control resulting from solving the right side of (2.10) need only be applied at the boundary of the reach-avoid set.

Algorithm 1: Learning policies and disturbances

---

1 **Input:** $\dot{x} = f(x, u, d), \mathcal{S}, \mathcal{U}, \mathcal{D}, T, \Delta t, C, N, Trn(\cdot, \cdot)$

2 **Initialize** $\Pi_u, \Pi_d \leftarrow \{\}$

3 **For** $k = 0, \dots, \lfloor T/\Delta t \rfloor$

4   **Initialize** $P, U^*, D^* \leftarrow \{\}$ // No training data.

5   **For** $q = 1, \dots, N$

6   **Sample** $x \sim \text{Unif}\{\mathcal{S}\}$

7   **Initialize** $u^*, d^* \leftarrow u_{min}, d_{min}$

8 $\hat{s} \leftarrow \xi(-k\Delta t; x, -(k+1)\Delta t, u_{min}, d_{min})$

9 $\hat{c} \leftarrow C(\hat{s}, \Pi_u, \Pi_d)$

10   **For** $i = 1, \dots, N_u$ // Find best control.

11 $\hat{u} \leftarrow u_{min}; \hat{u}^i \leftarrow u^i_{max}$

12 $x' \leftarrow \xi(-k\Delta t; x, -(k+1)\Delta t, \hat{u}, d_{min})$

13   **If** $(C(x', \Pi_u, \Pi_d) < \hat{c})$: $u^*_i \leftarrow u^i_{max}$

14   **For** $j = 1, \dots, N_d$ // Find best disturbance.

15 $\hat{d} \leftarrow d_{min}; \hat{d}^i \leftarrow d^i_{max}$

16 $x' \leftarrow \xi(-k\Delta t; x, -(k+1)\Delta t, u_{min}, \hat{d})$

17   **If** $(C(x', \Pi_u, \Pi_d) > \hat{c})$: $d^*_i \leftarrow d^i_{max}$

18 $U^* \leftarrow \{U^*, u^*\}$ // Record control.

19 $D^* \leftarrow \{D^*, d^*\}$ // Record disturbance.

20 $P \leftarrow \{P, x\}$ // Record state.

21   // Train new classifiers. Add them to overall policy.

22 $\Pi^u_{-(k+1)\Delta t} \leftarrow Trn(P, U^*), \Pi_u \leftarrow \{\Pi_u, \Pi^u_{-(k+1)\Delta t}\}$

23 $\Pi^d_{-(k+1)\Delta t} \leftarrow Trn(P, D^*), \Pi_d \leftarrow \{\Pi_d, \Pi^d_{-(k+1)\Delta t}\}$

24 **Return** Π_u, Π_d

$u_{min} = (u^0_{min}, \dots, u^{N_u}_{min})$ and $d_{min} = (d^0_{min}, \dots, d^{N_d}_{min})$ from t to $t + \Delta t$; that is, let $\hat{c} = C\big(\xi(t + \Delta t; x, t, u_{min}, d_{min}), \Pi^u_{(t+\Delta t):0}, \Pi^d_{(t+\Delta t):0}\big)$. Now, separately for each component i of u (and likewise for d), set $u^i(t) = u_{max}$ and compute the cost. If the cost is less than (resp. greater than, for disturbance) $\hat{c}$, then this is the optimal control (resp. disturbance) in dimension i at time t. This corresponds to lines 7-17.

Equipped with this procedure for computing *approximately optimal*[3] control and disturbance actions, we record the computed state-action pairs (lines 18-20) for N states sampled uniformly over $\mathcal{S}$[4] (lines 5-6). We then train separate binary classifiers for *each component* of u and d, and add them to their current set Π^u_τ or Π^d_τ. These are finally appended to the time-indexed control and disturbance policy sets Π_u and Π_d (lines 22-23). $Trn(\cdot, \cdot)$ denotes a training procedure given state-action pairs. The final subsection in this segment contains further details pertaining to how the classifiers were trained.

Two of the main benefits of performing approximate reachability analysis using binary

---

[3]Approximately optimal, since we compute policies at time t based on previously trained control and disturbance policies for $\tau > t$.

[4]While other distributions could be used, in this work we focus solely on uniform sampling. Different sampling strategies may result in different algorithm performance.

classifiers rather than grids are memory usage and time complexity. The memory footprint of medium-sized neural networks of the sort used in this paper can be on the order of 10^3 parameters or ~ 10 Kb, as opposed to ~ 10 Gb for dense grids of 4D systems. In our experience, Algorithm 1 typically terminates after an hour for the 6D and 7D systems presented in Section 24, whereas grid-based methods are completely intractable for coupled systems of that size.

Special case: value function convergence.

For some instances of problem (2.9) and (2.6) the value function $V(t, s)$ converges, i.e. we have: $\lim_{t \to -\infty} V(t, s) = V^*(s)$. From (2.10), the corresponding optimal control and disturbance policies also converge. While in this paper we make no claims regarding convergence of the classifiers to the true optimal policies, our empirical results do suggest convergence in practice (see Figure 2.14). When this happens, we denote $\Pi^u_{-T} = \Pi^u_{-\infty}$ (resp. $\Pi^d_{-T} = \Pi^d_{-\infty}$), for T large enough. In practice, the horizon can be progressively increased as needed. A benefit of converged policies is that when estimating $V^*(s)$ we only require the last set of binary classifiers $\Pi^u_{-\infty}$ and $\Pi^d_{-\infty}$, allowing us to store only $N_u + N_d$ classifiers.

Summary of guarantees. Algorithm 1 returns a set of approximately optimal policies for the control and the disturbance for a finite number of time steps. Recalling (2.32), in order to obtain an estimate of the value at a certain state x and time t, it suffices to simulate an entire trajectory from that state and time using the learned policies. The value $V^{\Pi_u, \Pi_d}(t, x)$ is the cost of the associated trajectory, measured according to (2.9).

A benefit of working with policy approximators rather than value function approximators is that in the case of no disturbance, the value function induced by the learned control policy will always upper-bound the true value. This means that a reach-avoid set computed via Algorithm 1 will be a subset of the true reach-avoid set.

For reachability problems involving a disturbance, if the optimal disturbance policy is known *a priori*, the same guarantee still applies. However, if the optimal disturbance is unknown and must also be learned, no guarantees can be made because the learned disturbance policy will not generally be optimal. We formalize this result with the following proposition.

Proposition 1. *If we assume (a) no disturbance, or (b) access to a worst-case optimal disturbance policy, then the computed reach-avoid set is a subset of the true set.*

Proof. First assume no disturbance. Due to the use of function approximators, the control policy Π_u will be suboptimal relative to the optimal controller $u^*(\cdot)$, meaning it is less effective at minimizing the cost functional (2.9). Therefore, $V^{\Pi_u}(t, x) \geq V(t, x)$. Denoting the neural network reach-avoid sets as $\mathcal{RA}^{\Pi_u}_t := \{x : V^{\Pi_u}(t, x) \leq 0\}$, this inequality implies that $\mathcal{RA}^{\Pi_u}_t \subseteq \mathcal{RA}_t$. $\square$

Note that this applies to *all* states x and times t, not just those that were sampled in Algorithm 1. When optimizing over both control and disturbance this guarantee does not hold because the disturbance will generally be suboptimal and therefore not worst-case. However, when provided with an optimal disturbance policy at the onset, we recover the case of optimizing over only control.
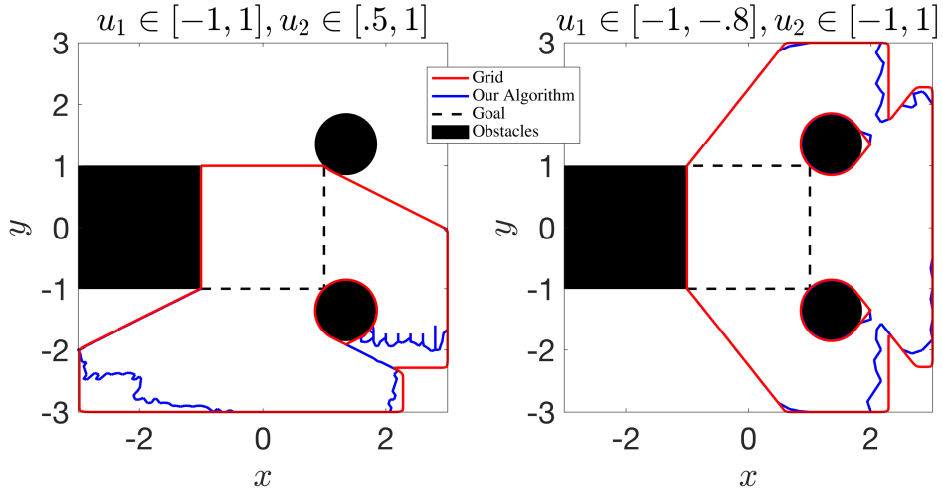
Fig. 2.10: Reach-avoid set computation for 2D dynamics in (2.33) for two different control bounds. Sets computed using our method are subsets of the true sets.

Examples

In this Section, we will present two reachability problems *without* disturbances, and compare the results of our proposed method with those obtained from a full grid-based approach [32]. In each case, we observe that our method agrees with the ground truth, with a small but expected degree of conservatism. For these examples, the set $\mathcal{L}$ is a box of side-length 2 centered at $(x, y) = (0, 0)$, and $\mathcal{G}$ consists of the outer boundaries (i.e. $\max\{|x|, |y|\} \leq 3$) and the shaded obstacles (Fig. 2.10 and Fig. 2.11).

2D point. Consider a 2D dynamical system with inputs $u_1 \in [\underline{u}_1, \overline{u}_1]$ and $u_2 \in [\underline{u}_2, \overline{u}_2]$ which evolves as follows:

$$\dot{x} = u_1, \ \dot{y} = u_2 \tag{2.33}$$

Fig. 2.10 shows the reach-avoid sets for two different control bounds. We overlay the sets computed by our method on top of that computed using a dense 121×121 grid [32]. The red set was computed using standard HJ reachability and the blue set was computed using our classification-based method. Points inside the reach-avoid sets represent states from which there exists a control sequence which reaches the target while avoiding all obstacles. As guaranteed in Proposition 1, the set computed via Algorithm 1 is always a subset of the ground truth, meaning that every state marked in Fig. 2.10 as safe is also safe using the optimal controller. The computation time for the grid-based approach was 20 seconds, while for the classification-based it was 10 minutes.

4D unicycle. Next, we consider a higher-dimensional system representing a 4D unicycle model:

$$\dot{s} = \begin{bmatrix} \dot{x} & \text{(x-position)} \\ \dot{y} & \text{(y-position)} \\ \dot{\theta} & \text{(yaw angle)} \\ \dot{v} & \text{(tangential speed)} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ u_\omega \\ u_a \end{bmatrix} \tag{2.34}$$
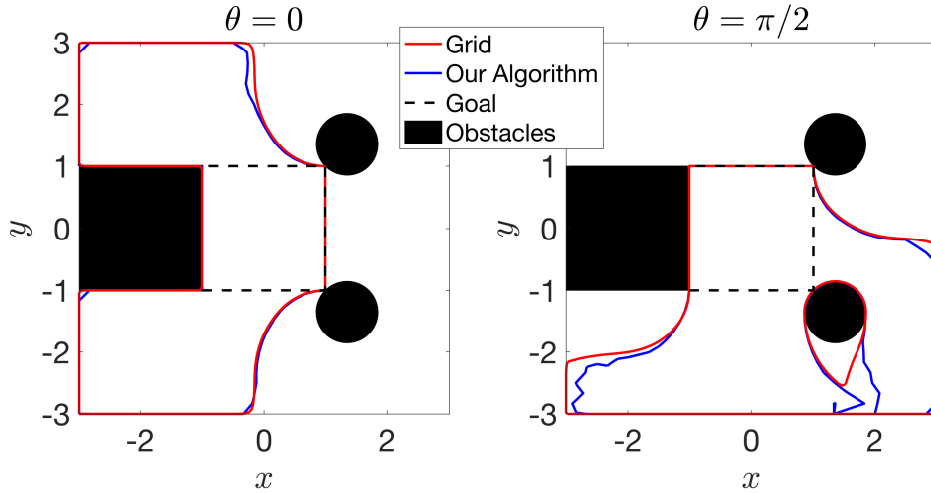
Fig. 2.11: Reach-avoid set computation for the 4D dynamics in (2.34) for two different 2D slices and tangential speed $v = 1$.

in which controls are tangential acceleration $u_a \in [0, 1]$ and yaw rate $u_\omega \in [-1, 1]$. Fig. 2.11 shows a computed a reach-avoid set for this system for different 2D slices of the 4D state space on a 121^4 grid. As expected, our approach yields a conservative subset of the true reach-avoid set. In this case, the computation time for the grid-based approach was 3 days, while for the classification-based it was 30 minutes.

FaSTrack overview and 6D quadrotor. As we have seen in Section 2.1, FaSTrack (Fast and Safe Tracking) is a recent method for safe real-time motion planning [18]. FaSTrack breaks down an autonomous system into two agents: a simple *planning model* used for real-time motion planning, and a more complicated *tracking model* used to track the generated plan. To ensure safe tracking, FaSTrack computes the largest *relative* distance between the two models (tracking error), and the planning algorithm uses this result to enlarge obstacles for collision-checking. The computation also provides an optimal feedback controller to ensure that the tracker remains within this bound during planning.

To solve for the largest tracking error in FaSTrack, we set the cost in (2.9) as the distance to the origin in relative position space. We denote relative states by $r \in \mathcal{R} \subset \mathbb{R}^{N_r}$, and solve the zero-sum differential game of (2.16).

In practice, we combine the input of the planner with that of the disturbance additively, i.e., we relative state dynamics. Henceforth, policy $\Pi^d_{-\infty}$ will represent the concatenated disturbance and planning algorithm policies.

Following Proposition 1, when the optimal converged disturbance policy $\Pi^d_{-\infty}$ is known analytically, the policies learned in Algorithm 1 will (by Proposition 1) yield a value function which *over-approximates* the optimal value function, i.e. $V^{\Pi_u, d^*}(t, r) \geq V(t, r)$. Thus, the maximum relative distance ever achieved between tracking mode and planning model, from any initial relative state, will always be *greater* when using the binary classifier policies than the optimal policy. For safe trajectory tracking, this translates into enlarging obstacles by a larger amount, meaning we still preserve safety.

We employ Algorithm 1 to find the largest tracking error for two nonlinear models of

(a) Ground truth   **(b)** Learned $\Pi^u_{-\infty}$ **(c)** Learned $\Pi^{\{u,d\}}_{-\infty}$ **(d)** Level curves
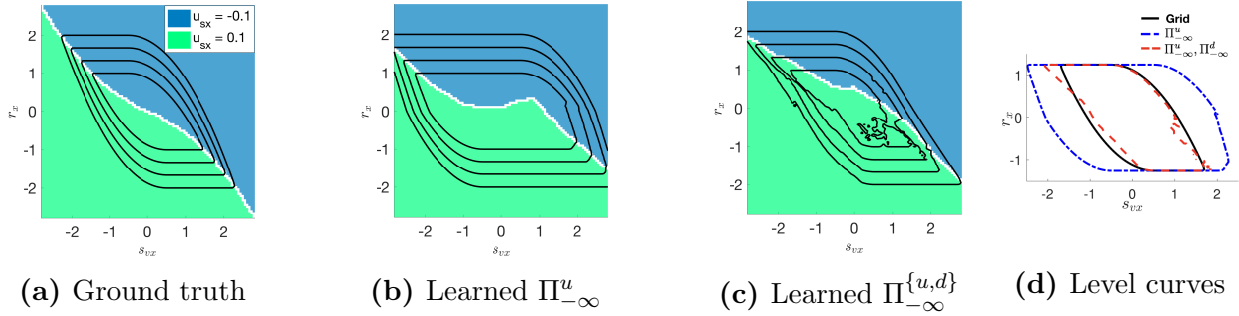
Fig. 2.12: Level sets of V^* in the (r_x, v_x) states (setting other states to zero) for (a) ground truth grid-based representation, (b) neural network $\Pi^u_{-\infty}$ trained on optimal disturbance policy $d^*(\cdot)$, and (c) neural networks $\Pi^u_{-\infty}$ and $\Pi^d_{-\infty}$ trained jointly. We encode the optimal (learned) control at each state as a different color. (d) Overlay of level sets from (a-c). Our method only yields a conservative result (a superset of the ground truth) when $\Pi^u_{-\infty}$ is trained against $d^*(\cdot)$.

the *tracking model*, which become control-affine under small angle assumptions. First, we consider a 6D near-hover model which decouples into three 2D subsystems and thus admits a comparison to grid-based methods. Then, we present results for a *fully-coupled* 7D model that cannot be solved exactly using grid-based techniques and use it for quadrotor control.

6D decoupled. We first consider a 6D quadrotor tracking model and 3D geometric planning model. Here, the quadrotor control consists of pitch (u_θ) and roll (u_ϕ) angles, and thrust acceleration (u_T), while the planning model's maximum speeds are b_x, b_y, and b_z in each dimension. All of our results assume $u_\phi, u_\theta \in [-0.1, 0.1]$ rad, $u_T - g \in [-2.0, 2.0]$ m s^{-2}, and $b_x = b_y = b_z = 0.25$ m s^{-1}. We assume a maximum velocity disturbance of 0.25 m s^{-1} in each dimension. The relative position states (r_x, r_y, r_z) and the tracker's velocity states (x_{vx}, x_{vy}, x_{vz}) adhere to the following *relative dynamics*:

$$\begin{bmatrix} \dot{r}_x \\ \dot{r}_y \\ \dot{r}_z \end{bmatrix} = \begin{bmatrix} x_{vx} - d_{vx} - b_x \\ x_{vy} - d_{vy} - b_y \\ x_{vz} - d_{vz} - b_z \end{bmatrix}, \quad \begin{bmatrix} \dot{x}_{vx} \\ \dot{x}_{vy} \\ \dot{x}_{vz} \end{bmatrix} = \begin{bmatrix} g \tan u_\theta \\ -g \tan u_\phi \\ u_T - g \end{bmatrix}. \tag{2.35}$$

Without yaw, these dynamics decouple into three 2D subsystems, $(r_x, s_{vx}), (r_y, s_{vy})$, and (r_z, s_{vz}), and we use the technique in [7] to solve 2.17 independently for each 2D subsystem using grid-based techniques.

Figure 2.12 shows the level sets of the value function V^* and corresponding optimal tracker control policies. Fig. 2.12a is the grid-based ground truth, while Figure 2.12b shows the induced value function for the neural network classifier policy $\Pi^u_{-\infty}$ trained against the optimal disturbance policy $d^*(\cdot)$, and Figure 2.12c shows the induced value function when $\Pi^u_{-\infty}$ and $\Pi^d_{-\infty}$ were trained jointly. Note that the classification-based results shown here did *not* take advantage of system decoupling. Corroborating our theoretical results, the level sets of the value function induced by our learned classifiers over-approximate the true level sets when the disturbance plays optimally (Figure 2.12d). Also, observe that using a learned (and hence, generally suboptimal) $\Pi^d_{-\infty}$, the resulting level sets in Figure 2.12c still well-approximate (though they do not include) those in 2.12a. For each level curve, the maximum tracking error x is the largest value of the level curve along the r_x axis. Observe
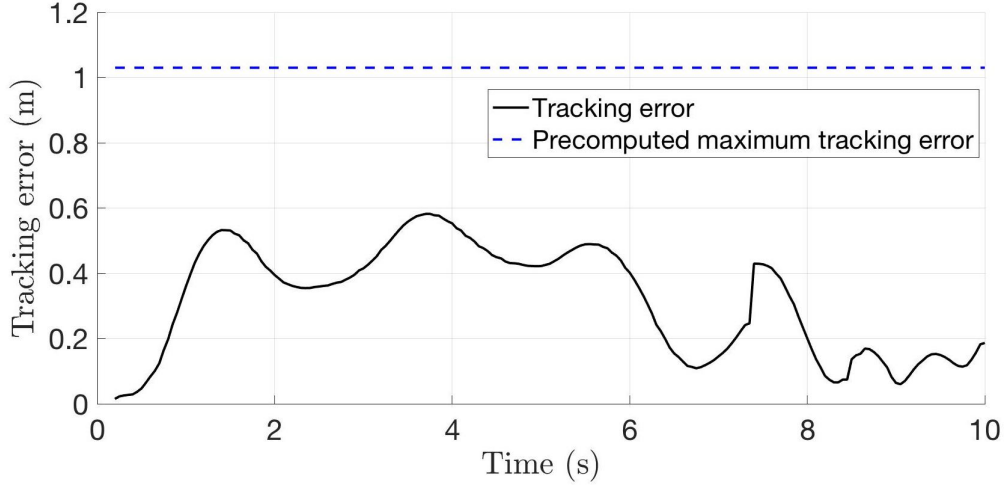
Fig. 2.13: Relative distance between the quadrotor (tracking model) and planned trajectory (planning model) over time during a hardware test wherein a Crazyflie 2.0 must navigate through a motion capture arena around spherical obstacles. The quadrotor stays well within the computed tracking error bound throughout the flight. Note that the tracking error is large because our controller accounts for adversarial disturbances, unlike many common controllers.

in Figure 2.12d that the maximum tracking error is similar in all three cases. Finally, the line that separates the colored areas in the background of each inset in Figure 2.12 denotes the decision boundary for the controller in each case.

7D coupled. In this example, we introduce yaw (x_ψ) into the model as an extra state in (2.36) and introduce yaw rate control $u_\psi \in [-1.0, 1.0] \, \text{rad s}^{-1}$. The relative position dynamics in (r_x, r_y, r_z) are identical to (2.35). The remaining states evolve as:

$$
\begin{bmatrix} \dot{x}_{vx} \\ \dot{x}_{vy} \\ \dot{x}_{vz} \\ x_\psi \end{bmatrix} = \begin{bmatrix} g(\sin u_\theta \cos x_\psi + \sin u_\phi \sin x_\psi) \\ g(-\sin u_\phi \cos x_\psi + \sin u_\theta \sin x_\psi) \\ u_T \cos u_\phi \cos u_\theta - g \\ u_\psi \end{bmatrix}. \tag{2.36}
$$

This dynamical model is now *7D*. It is too high-dimensional and coupled in the controls for current grid-based HJ reachability schemes, yet our proposed method is still able to compute a safety controller and the associated largest tracking error.

Hardware demonstration. We tested our learned controller on a Crazyflie 2.0 quadrotor in a motion capture arena. Figure 2.13 displays results for (2.36). As shown, the quadrotor stays well within the computed error bound. For this experiment Π_u was trained using a sub-optimal disturbance policy. Even though we do not have a rigorous safety guarantee in this general case because we computed the disturbance, these results corroborate our intuition from Figure 2.12 where the computed error bound remains essentially unchanged when using a learned disturbance instead of the optimum. However, by Proposition 1, with the optimal disturbance we could compute a strict guarantee. The hardware demonstration can be seen in our video: `https://youtu.be/_thXAaEJYGM`.

Implementation details. In this part of the chapter, we train each binary classifier by minimizing the cross-entropy loss between inputs and labels via stochastic gradient descent. We run the classification problem for a pre-specified number of gradient steps between each new set of policies. Since we expect policies to vary slowly over time, we initialize the weights for each new network with those from its predecessor. This serves two purposes. First, it serves as a "warm start" leading to faster stochastic gradient descent convergence. Second, it provides a practical indicator of policy convergence—i.e. if the initial classification accuracy of a new policy is almost equal to that of its predecessor, the policy has most likely converged. Figure 2.14 shows a typical learning curve when running Algorithm 1. The figure shows the progression of the validation error (against unseen state-action pairs) in each iteration.
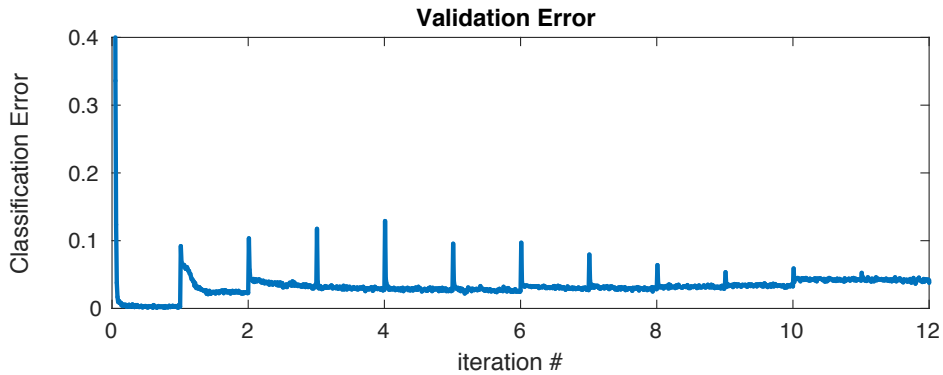
Fig. 2.14: Learning curves for a single classifier of the 6D decoupled system. Classification error decreases between spikes, which mark each new k in Algorithm 1. Spikes shrinking hints that classifiers eventually converge.

All feedforward neural network classifiers had two hidden layers of 20 neurons each, with rectified linear units (ReLUs) as the activation functions, and a final softmax output. The gradient descent algorithm employed was RMSprop with learning rate $\alpha = 0.001$ and momentum constant $\beta = 0.95$. When using function approximators, it is in general unclear how many samples should be taken as a function of the state dimension. In our case, the number of points N sampled at each iteration was $1k$ for the 2D example, and $200k$ for the 4D, 6D and 7D system. All initial weights and biases were drawn from a uniform probability distribution between $[-0.1, 0.1]$. All computations were performed on a 12 core, 64-bit machine with Intel® Core™ i7-5820K CPUs @ 3.30GHz. In our implementation we did not employ any form of parallelization. All code for the project can be found at `https://github.com/HJReachability/Classification_Based_Reachability`.

Chapter 3

Adaptive, Confidence-Aware Prediction

Unlike the previous Chapter, which dealt primarily with uncertainty regarding internal system dynamics, this Chapter will deal with uncertainty in the decisions of other agents. As we will see, that *extrinsic* uncertainty is fundamentally different and requires a different approach. In the previous Chapter, our primary outlook was that any uncertainty could be handled as though it were adversarial. We will discuss some shortcomings and present an alternative view later in Chapter 5, but regardless, to handle uncertainty in the actions of others, it is clear that an adversarial assumption is too conservative for some applications. To see this, one need only imagine worst-case behavior on the road, where avoiding someone who *wants* to collide is virtually impossible.

In this Chapter, we will also primarily concern ourselves with motion planning problems. That is, many problems in autonomy boil down to a single autonomous agent trying to decide how best to act in any particular situation. Here, the problem we shall address is: how can we predict the motion of other agents, and plan around those predictions? This Chapter is based upon work published at the International Journal of Robotics Research in 2019, titled "Confidence-aware motion prediction for real-time collision avoidance" [13], and coauthored with Andrea Bajcsy, Jaime Fisac, Sylvia Herbert, Steven Wang, Anca Dragan, and Claire Tomlin, which itself is based upon earlier work published at the Robotics: Science & Systems conference in 2018 [10].

3.1   The Inherent Difficulty with Prediction

Predicting other agents' decisions is inherently difficult because there are so many possible trajectories that any agent could follow over some time horizon into the future. To deal with this uncertainty, there are a number of approaches in the literature ranging from Partially Observable Markov Decision Processes (POMDPs) [46], to generative modeling [20], to inverse optimal control [57]. Although existing methods vary significantly, they share in common the idea that it is important to model the decisions of another agent. Whether this is done implicitly or explicitly, these models (as any do) inevitably break, often at inopportune moment. This Chapter will present an approach based on inverse optimal control which

exploits such a model when it appears to explain decisions a robot observes, and effectively reverts to a conservative, adversarial mode when it does not. For clarity, the method is demonstrated in the case of a human avoiding a quadrotor, although the method is certainly more broadly applicable. As we shall see, in these problems the best one can hope to do is to plan to avoid collision with *high probability*.

3.2 The Bayesian Solution: Adaptive Confidence-Awareness

Problem Setup

We consider a single mobile robot operating in a shared space with a single human agent (e.g. a pedestrian or human-driven car). For simplicity, we presume that the robot has full knowledge of its own state and that of the human, although both would require online estimation in practice. As we present each formal component of this problem, we will provide a concrete illustration using a running example in which a quadcopter is navigating around a pedestrian.

Dynamical system models and safety

We will model the motion of both the human and the robot as the evolution of two dynamical systems. Let the state of the human be $x_H \in \mathbb{R}^{n_H}$, where n_H is the dimension of the human state space. We similarly define the robot's state, for planning purposes, as $x_R \in \mathbb{R}^{n_R}$. In general, these states could represent the positions and velocities of a mobile robot and a human in a shared environment, the kinematic configurations of a human and a robotic manipulator in a common workspace, or the positions, orientations, and velocities of human-driven and autonomous vehicles in an intersection. We express the evolution of these states over time as a family of ordinary differential equations:

$$\dot{x}_H = f_H(x_H, u_H) , \qquad \dot{x}_R = f_R(x_R, u_R) \tag{3.1}$$

where $u_H \in \mathbb{R}^{m_H}$ and $u_R \in \mathbb{R}^{m_R}$ are the control actions of the human and robot, respectively.

Running example: We introduce a running example for illustration purposes throughout the paper. In this example we consider a small quadcopter that needs to fly to goal location $g_R \in \mathbb{R}^3$ in a room where a pedestrian is walking. For the purposes of planning, the quadcopter's 3D state is given by its position in space $x_R = [p_x, p_y, p_z]$, with velocity controls assumed decoupled in each spatial direction, up to $v_R = 0.25$ m/s. The human can only move by walking and therefore her state is given by planar coordinates $x_H = [h_x, h_y]$ evolving as $\dot{x}_H = [v_H \cos u_H, v_H \sin u_H]$. Intuitively, we model the human as moving with a fixed speed and controlling her heading angle. At any given time, the human is assumed to either move at a leisurely walking speed ($v_H \approx 1$ m/s) or remain still ($v_H \approx 0$).

Ultimately, the robot needs to plan and execute an efficient trajectory to a pre-specified goal state (g_R), without colliding with the human. We define the keep-out set $\mathcal{K} \subset \mathbb{R}^{n_H} \times \mathbb{R}^{n_R}$ as the set of joint robot-human states to be avoided (for example, because they imply physical

collisions). To avoid reaching this set, the robot must reason about the human's future motion when constructing its own motion plan.

Running example: In our quadcopter-avoiding-pedestrian example, $\mathcal{K}$ consists of joint robot-human states in which the quadcopter is flying within a square of side length $l = 0.3$ m centered around the human's location, while at any altitude, as well as any joint states in which the robot is outside the environment bounds defined as a box with a square base of side $L = 3.66$ m and height $H = 2$ m, regardless of the human's state.

Robust robot control

Provided an objective and a dynamics model, the robot must generate a motion plan which avoids the keep-out set $\mathcal{K}$. Unfortunately, this safety requirement is difficult to meet during operation for two main reasons:

1. *Model mismatch.* The dynamical system model f_R will never be a perfect representation of the real robot. This mismatch could lead to unintended collision.

2. *Disturbances.* Even with a perfect dynamics model, there may be unobserved, external "disturbance" inputs such as wind or friction. Without accounting for these disturbances, the system is not guaranteed to avoid $\mathcal{K}$, even if the planned trajectory is pointwise collision-free.

To account for modelling error and external disturbances, we could in principle design a higher fidelity dynamical model directly in a robust motion planning framework. Unfortunately, however, real-time trajectory optimization in high dimensions can be computationally burdensome, particularly when we also require some notion of robustness to external disturbance. Ideally we would like to enjoy the computational benefits of planning with a lower-fidelity model while enforcing the safety constraints induced by the higher-fidelity model. To characterize this model mismatch, we consider a higher fidelity and typically higher-order dynamical representation of the robot, with state representation $s_R \in \mathbb{R}^{n_S}$. This dynamical model will also explicitly account for external disturbances as unknown bounded inputs, distinct from control inputs. In order to map between this higher fidelity "tracking" state s_R and the lower fidelity "planning" state x_R, we shall assume a known projection operator $\pi : \mathbb{R}^{n_S} \to \mathbb{R}^{n_R}$. Fortunately, we can plan in the lower-dimensional state space at runtime, and guarantee robust collision avoidance via an offline reachability analysis that quantifies the effects of model mismatch and external disturbance. This framework, called FaSTrack and first proposed by [18], is described in further detail below and in chapter 2.

Running example: We model our quadcopter with the following flight dynamics (in the near-hover regime, at zero yaw with respect to a global coordinate frame):

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \ , \ \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} a_g \tan u_\theta \\ -a_g \tan u_\phi \\ u_T - a_g \end{bmatrix} \ , \tag{3.2}$$

where $[p_x, p_y, p_z]$ is the quadcopter's position in space and $[v_x, v_y, v_z]$ is its velocity expressed in the fixed global frame. We model its control inputs as thrust acceleration u_T and attitude

angles (roll u_ϕ and pitch u_θ), and denote the acceleration due to gravity as a_g. The quad-copter's motion planner generates nominal kinematic trajectories in the lower-dimensional $[p_x, p_y, p_z]$ position state space. Therefore we have a linear projection map $\pi(s_R) = [I_3, 0_3]s_R$, that is, x_R retains the position variables in s_R and discards the velocities.

Predictive human model

In order to predict the human's future motion, the robot uses its internal model of human dynamics, f_H. Under this modeling assumption, the human's future trajectory depends upon the choice of control input over time, $u_H(\cdot)$. Extensive work in econometrics and cognitive science, such as [50, 28, 1], has shown that human behavior—that is, u_H—can be well modeled by utility-driven optimization. Thus, the robot models the human as optimizing a reward function, $r_H(x_H, u_H; \theta)$, that depends on the human's state and action, as well as a set of parameters θ. This reward function could be a linear combination of features as in many inverse optimal control implementations (where the goal or feature weighting θ must be learned, either online or offline), or more generally learned through function approximators such as deep neural networks, where θ are the trained weights as in [9].

We assume that the robot has a suitable human reward function r_H, either learned offline from prior human demonstrations or otherwise encoded by the system designers. Thus endowed with r_H, the robot can model the human's choice of control action as a probability distribution over actions conditioned on state. Under maximum-entropy assumptions ([57]) inspired by noisy-rationality decision-making models ([1]), the robot models the human as more likely to choose (discrete) actions u_H with high expected utility, in this case the state-action value (or Q-value):

$$P(u_H \mid x_H; \beta, \theta) = \frac{e^{\beta Q_H(x_H, u_H; \theta)}}{\sum_{\tilde{u}} e^{\beta Q_H(x_H, \tilde{u}; \theta)}} \quad . \tag{3.3}$$

We use a temporally- and spatially-discretized version of human dynamics, $\tilde{f}_H$. These discrete-time dynamics may be found by integrating f_H over a fixed time step of Δt with fixed control u_H over the interval. We provide further details on this discretization for each example.

Running example: The quadcopter's model of the human assumes the human intends to reach some target location $g_H \in \mathbb{R}^2$ in a straight line. The human's reward function is given by the distance traveled over time step Δt, i.e. $r_H(x_H, u_H; g_H) = -v_H \Delta t$, and human trajectories are constrained to terminate at g_H. The state-action value, parameterized by $\theta = g_H$, captures the optimal cost of reaching g_H from x_H when initially applying u_H for a duration Δt: $Q_H(x_H, u_H; g_H) = -v_H \Delta t - \|x_H + v_H \Delta t [\cos u_H, \sin u_H]^\top - g_H\|_2$.

Often, the coefficient β is termed the *rationality coefficient*, since it quantifies the degree to which the robot expects the human's choice of control to align with its model of utility. For example, taking $\beta \downarrow 0$ yields a model of a human who appears "irrational," choosing actions uniformly at random and completely ignoring the modeled utility. At the other extreme, taking $\beta \uparrow \infty$ corresponds to a "perfectly rational" human, whose actions exactly optimize the modeled reward function. As we will see below, β can also be viewed as a measure of the robot's *confidence* in the predictive accuracy of Q_H.

Note that $Q_H(x_H, u_H; \theta)$ only depends on the human state and action and not on the robot's. Thus far, we have intentionally neglected discussion of human-robot interaction effects. These effects are notoriously difficult to model, and the community has devoted a significant effort to building and validating a variety of models, e.g. [49], [39]. In that spirit, we could have chosen to model human actions u_H as dependent upon robot state x_R in (3.3), and likewise defined Q_H to depend upon x_R. This extended formulation is sufficiently general as to encompass all possible (Markov) interaction models. However, in this work we explicitly do not model these interactions; indeed, one of the most important virtues of our approach is its robustness to precisely these sorts of modeling errors.

Probabilistically safe motion planning

Ideally, the robot's motion planner should generate trajectories that reach a desired goal state efficiently, while maintaining safety. More specifically, in this context "safety" indicates that the physical system will never enter the keep-out set $\mathcal{K}$ during operation, despite human motion and external disturbances. That is, we would like to guarantee that $(\pi(s_R), x_H) \notin \mathcal{K}$ for all time.

To make this type of strong, deterministic, *a priori* safety guarantee requires the robot to avoid the set of all human states x_H which could possibly be occupied at a particular time, i.e. the human's *forward reachable set*. If the robot can find trajectories that are safe for *any* possible human trajectory then there is no need to predict the human's next action. Unfortunately, the forward reachable set of the human often encompasses such a large volume of the workspace that it is impossible for the robot to find a guaranteed safe trajectory to the goal state. This motivates refining our notion of prediction: rather than reasoning about all the places where the human *could* be, the robot can instead reason about *how likely* the human is to be at each location. This probabilistic reasoning provides a guide for planning robot trajectories with a quantitative degree of safety assurance.

Our probabilistic model of human control input (3.3) coupled with dynamics model f_H allows us to compute a probability distribution over human states for every future time. By relaxing our conception of safety to consider only collisions which might occur with sufficient probability P_{th}, we dramatically reduce the effective volume of this set of future states to avoid. In practice, P_{th} should be chosen carefully by a system designer in order to trade off overall collision probability with conservativeness in motion planning.

The proposed approach in this paper follows two central steps to provide a quantifiable, high-confidence collision avoidance guarantee for the robot's motion around the human. First, we present our proposed Bayesian framework for reasoning about the uncertainty inherent in a model's prediction of human behavior. Based on this inference, we demonstrate how to generate a real-time probabilistic prediction of the human's motion over time. Next, we extend a state-of-the-art, provably safe, real-time robotic motion planner to incorporate our time-varying probabilistic human prediction.

Confidence-Aware Human Motion Prediction

Any approach to human motion prediction short of computing a full forward reachable set must, explicitly or implicitly, reflect a model of human decision-making. In this work, we

make that model explicit by assuming that the human chooses control actions in a Markovian fashion according to the probability distribution (3.3). Other work in the literature, such as [41], aims to learn a generative probabilistic model for human trajectories; implicitly, this training procedure distills a model of human decision making. Whether explicit or implicit, these models are by nature imperfect and liable to make inaccurate predictions eventually. One benefit of using an explicit model of human decision making, such as (3.3), is that we may reason directly and succinctly about its performance online.

In particular, the entropy of the human control distribution in (3.3) is a decreasing function of the parameter β. High values of β place more probability mass on high-utility control actions u_H, whereas low values of β spread the probability mass more evenly between different control inputs, regardless of their modeled utility Q_H. Therefore, β naturally quantifies how well the human's motion is expected to agree with the notion of optimality encoded in Q_H. The commonly used term "rationality coefficient", however, seems to imply that discrepancies between the two indicate a failure on the human's part to make the "correct" decisions, as encoded by the modeled utility. Instead, we argue that these inevitable disagreements are primarily a result of the model's inability to fully capture the human's behavior. Thus, instead of conceiving of β as a rationality measure, we believe that β can be given a more pragmatic interpretation related to the accuracy with which the robot's model of the human is able to explain the human's motion. Consistently, in this paper, we refer to β as *model confidence.*

An important related observation following from this interpretation of β is that the predictive accuracy of a human model is likely to change over time. For example, the human may change their mind unexpectedly, or react suddenly to some aspect of the environment that the robot is unaware of. Therefore, we shall model β as an unobserved, time-varying parameter. Estimating it in real-time provides us with a direct, quantitative summary of the degree to which the utility model Q_H explains the human's current motion. To do this, we maintain a Bayesian *belief* about the possible values of β. Initially, we begin with a uniform prior over β and over time this distribution evolves given measurements of the human's state and actions.

Real-time inference of model confidence

We reason about the model confidence β as a hidden state in a hidden Markov model (HMM) framework. The robot starts with a prior belief b^0_- over the initial value of β. In this work, we use a uniform prior, although that is not strictly necessary. At each discrete time step $k \in \{0, 1, 2, \dots\}$, it will have some belief about model confidence $b^k_-(\beta)$. [1] After observing a human action u^k_H, the robot will update its belief to b^k_+ by applying Bayes' rule.

The hidden state may evolve between subsequent time steps, accounting for the important fact that the predictive accuracy of the human model may change over time as unmodeled factors in the human's behavior become more or less relevant. Since by definition we do not have access to a model of these factors, we use a naive "ϵ-static" transition model: at each time k, β may, with some probability ϵ, be re-sampled from the initial distribution b^0_-, and otherwise retains its previous value. We define the belief over the next value of β (denoted by

---

[1]To avoid confusion between discrete and continuous time, we shall use superscripts to denote discrete time steps (e.g. x^k_H) and parentheticals to denote continuous time (e.g. $x_H(t)$).

β') as an expectation of the conditional probability $P(\beta' \mid \beta)$, i.e. $b^k_-(\beta') := \mathbb{E}_{\beta \sim b^{k-1}_+}[P(\beta' \mid \beta)]$. Concretely, this expectation may be computed as

$$b^k_-(\beta') = (1 - \epsilon)b^{k-1}_+(\beta') + \epsilon b^0_-(\beta') \ . \tag{3.4}$$

By measuring the evolution of the human's state x_H over time, we assume that, at every time step k, the robot is able to observe the human's control input u^k_H. This observed control may be used as evidence to update the robot's belief b^k_- about β over time via a Bayesian update:

$$b^k_+(\beta) = \frac{P(u^k_H \mid x^k_H; \beta, \theta)b^k_-(\beta)}{\sum_{\tilde{\beta}} P(u^k_H \mid x^k_H; \tilde{\beta}, \theta)b^k_-(\tilde{\beta})} \ , \tag{3.5}$$

with $b^k_+(\beta) := P(\beta \mid x^{0:k}_H, u^{0:k}_H)$ for $k \in \{0, 1, ...\}$, and $P(u^k_H \mid x^k_H; \beta, \theta)$ given by (3.3).

It is critical to be able to perform this update rapidly to facilitate real-time operation; this would be difficult in the original continuous hypothesis space $\beta \in [0, \infty)$, or even in a large discrete set. Fortunately, our software examples and hardware demonstration suggest that maintaining a Bayesian belief over a relatively small set of $N_\beta = 5$ discrete values of β distributed on a log scale achieves significant improvement relative to using a fixed value.

The "ϵ-static" transition model leads to the desirable consequence that old observations of the human's actions have a smaller influence on the current model confidence distribution than recent observations. In fact, if no new observations are made, successively applying time updates asymptotically contracts the belief towards the initial distribution, that is, $b^k_-(\cdot) \to b^0_-(\cdot)$. The choice of parameter ϵ effectively controls the rate of this contraction, with higher ϵ leading to more rapid contraction.

Human motion prediction

Equipped with a belief over β at time step k, we are now able to propagate the human's state distribution forward to any future time via the well-known Kolmogorov forward equations, recursively. In particular, suppose that we know the probability that the human is in each state x^κ_H at some future time step κ. We know that (according to our utility model) the probability of the human choosing control u^κ_H in state x^κ_H is given by (3.3). Accounting for the otherwise deterministic dynamics model $\tilde{f}_H$, we obtain the following expression for the human's state distribution at the following time step $\kappa + 1$:

$$P(x^{\kappa+1}_H; \beta, \theta) = \sum_{x^\kappa_H, u^\kappa_H} P(x^{\kappa+1}_H \mid x^\kappa_H, u^\kappa_H; \beta, \theta) \cdot \tag{3.6}$$
$$P(u^\kappa_H \mid x^\kappa_H; \beta, \theta)P(x^\kappa_H; \beta, \theta) \ ,$$

for a particular choice of β. Marginalizing over β according to our belief at the current step time k, we obtain the overall occupancy probability distribution at each future time step κ:

$$P(x^\kappa_H; \theta) = \mathbb{E}_{\beta \sim b^k} P(x^\kappa_H; \beta, \theta) \ . \tag{3.7}$$

Note that (3.6) is expressed more generally than is strictly required. Indeed, because the only randomness in dynamics model $\tilde{f}_H$ originates from the human's choice of control input u_H, we have $P(x^{\kappa+1}_H \mid x^\kappa_H, u^\kappa_H; \beta, \theta) = \mathbb{1}\{x^{\kappa+1}_H = \tilde{f}_H(x^\kappa_H, u^\kappa_H)\}$.

Model confidence with auxiliary parameter identification

Thus far, we have tacitly assumed that the only unknown parameter in the human utility model (3.3) is the model confidence, β. However, often one or more of the auxiliary parameters θ are also unknown. These auxiliary parameters could encode one or more human goal states or intents, or other characteristics of the human's utility, such as her preference for avoiding parts of the environment. Further, much like model confidence, they may change over time.

In principle, it is possible to maintain a Bayesian belief over β and θ jointly. The Bayesian update for the hidden state (β, θ) is then given by

$$b_+^k(\beta, \theta) = \frac{P(u_H^k \mid x_H^k; \beta, \theta) b_-^k(\beta, \theta)}{\sum_{\tilde{\beta}, \tilde{\theta}} P(u_H^k \mid x_H^k; \tilde{\beta}, \tilde{\theta}) b_-^k(\tilde{\beta}, \tilde{\theta})} \quad, \tag{3.8}$$

with $b_+^k(\beta, \theta) := P(\beta, \theta \mid x_H^{0:k}, u_H^{0:k})$ the running posterior and $b_-^k(\beta, \theta) := P(\beta, \theta \mid x_H^{0:k-1}, u_H^{0:k-1})$ the prior at time step k.

This approach can be practical for parameters taking finitely many values from a small, discrete set, e.g. possible distinct modes for a human driver (distracted, cautious, aggressive). However, for certain scenarios or approaches it may not be practical to maintain a full Bayesian belief on the parameters θ. In such cases, it is reasonable to replace the belief over θ with a point estimate $\bar{\theta}$, such as the maximum likelihood estimator or the mean, and substitute that estimate into (3.6). Depending on the complexity of the resulting maximum likelihood estimation problem, it may or may not be computationally feasible to update the parameter estimate $\bar{\theta}$ at each time step. Fortunately, even when it is computationally expensive to estimate $\bar{\theta}$, we can leverage our model confidence as an indicator of when re-estimating these parameters may be most useful. That is, when model confidence degrades that may indicate poor estimates of $\bar{\theta}$.

Prediction Examples

We illustrate these inference steps with two sets of examples: our running pedestrian example and a simple model of a car.

Pedestrian model (running example)

So far, we have presented a running example of a quadcopter avoiding a human. We use a deliberately simple, purely kinematic model of continuous-time human motion:

$$\dot{x}_H = \begin{bmatrix} \dot{h}_x \\ \dot{h}_y \end{bmatrix} = \begin{bmatrix} v_H \cos u_H \\ v_H \sin u_H \end{bmatrix} \quad. \tag{3.9}$$

However, as discussed above, the proposed prediction method operates in discrete time (and space). The discrete dynamics corresponding to (3.9) are given by

$$x_H^{k+1} - x_H^k \equiv x_H(t + \Delta t) - x_H(t) \tag{3.10}$$

$$= \begin{bmatrix} v_H \Delta t \cos u_H(t) \\ v_H \Delta t \sin u_H(t) \end{bmatrix} \quad,$$

for a time discretization of Δt.

Dubins car model

To emphasize the generality of our method, we present similar results for a different application domain: autonomous driving. We will model a human-driven vehicle as a dynamical system whose state x_H evolves as

$$\dot{x}_H = \begin{bmatrix} \dot{h}_x \\ \dot{h}_y \\ \dot{h}_\phi \end{bmatrix} = \begin{bmatrix} v_H \cos h_\phi \\ v_H \sin h_\phi \\ u_H \end{bmatrix} . \tag{3.11}$$

Observe that, while (3.11) appears very similar to (3.9), in this Dubins car example the angle of motion is a *state*, not a *control input*.

We discretize these dynamics by integrating (3.11) from t to $t + \Delta t$, assuming a constant control input u_H:

$$x_H^{k+1} - x_H^k \equiv x_H(t + \Delta t) - x_H(t) = \tag{3.12}$$

$$\begin{bmatrix} \frac{v_H}{u_H(t)} \left(\sin \left(h_\phi(t) + u_H(t)\Delta t \right) - \sin(h_\phi(t)) \right) \\ -\frac{v_H}{u_H(t)} \left(\cos \left(h_\phi(t) + u_H(t)\Delta t \right) - \cos(h_\phi(t)) \right) \\ u_H \Delta t \end{bmatrix}$$

For a specific goal position $g = [g_x, g_y]$, the Q-value corresponding to state-action pair (x_H, u_H) and reward function $r_H(x_H, u_H) = -v_H \Delta t$ (until the goal is reached) may be found by solving a shortest path problem offline.

Accurate model

First, we consider a scenario in which the robot has full knowledge of the human's goal, and the human moves along the shortest path from a start location to this known goal state. Thus, human motion is well-explained by Q_H.

The first row of Fig. 3.1 illustrates the probability distributions our method predicts for the pedestrian's future state at different times. Initially, the predictions generated by our Bayesian confidence-inference approach (right) appear similar to those generated by the low model confidence predictor (left). However, our method rapidly discovers that Q_H is an accurate description of the pedestrian's motion and generates predictions that match the high model confidence predictor (center). The data used in this example was collected by tracking the motion of a real person walking in a motion capture arena. See Section 3.2 for further details.

Likewise, the first row of Fig. 3.2 shows similar results for a human-driven Dubins car model (in simulation) at an intersection. Here, traffic laws provide a strong prior on the human's potential goal states. As shown, our method of Bayesian model confidence inference quickly infers the correct goal and learns that the human driver is acting in accordance with its model Q_H. The resulting predictions are substantially similar to the high-β predictor. The data used in this example was simulated by controlling a Dubins car model along a pre-specified trajectory.
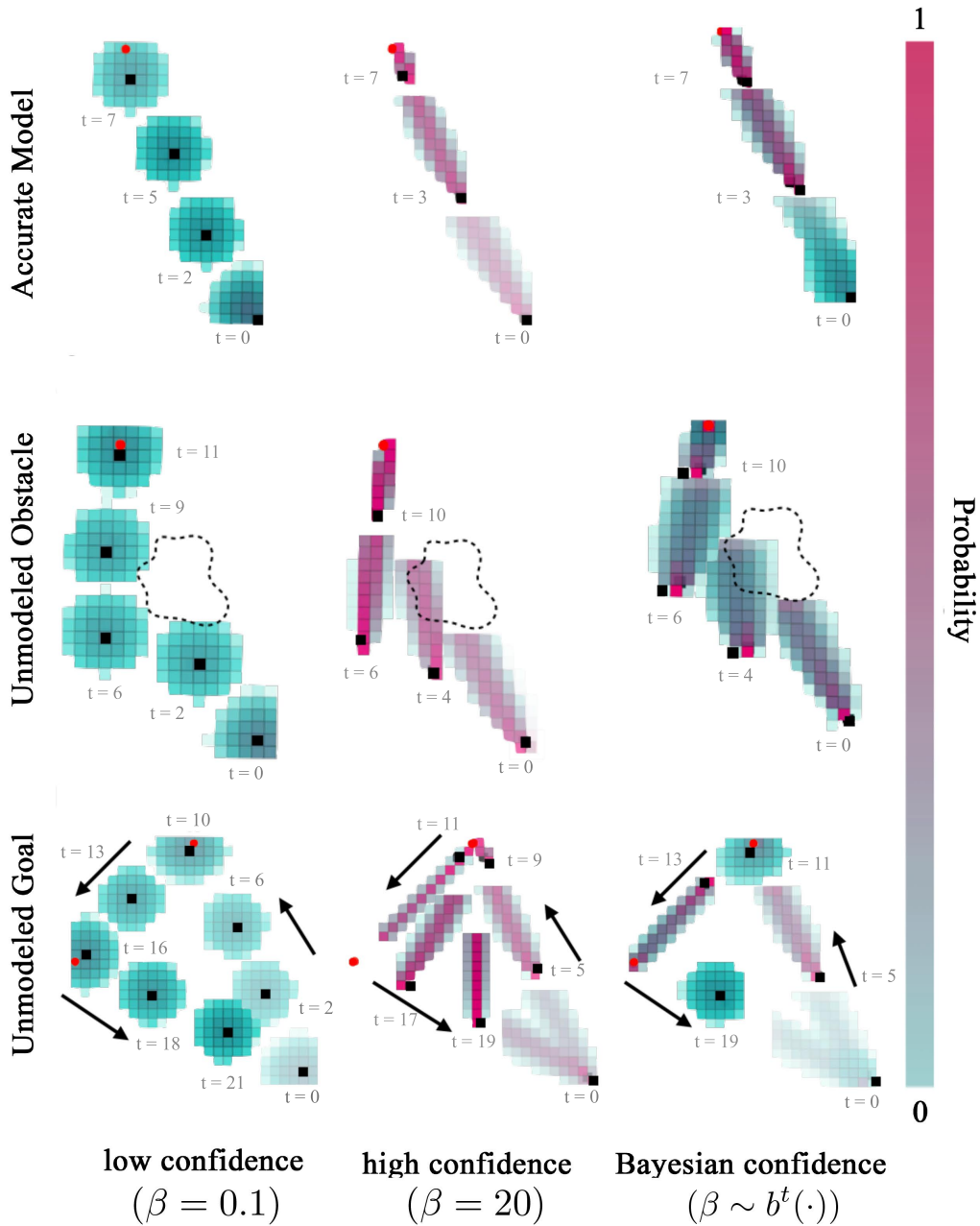
Fig. 3.1: Snapshots of pedestrian trajectory and probabilistic model predictions. Top row: Pedestrian moves from the bottom right to a goal marked as a red circle. Middle row: Pedestrian changes course to avoid a spill on the floor. Bottom row: Pedestrian moves to one known goal, then to another, then to a third which the robot has not modeled. The first two columns show predictions for low and high model confidence; the third column shows the predictions using our Bayesian model confidence. For all pedestrian videos, see: `https://youtu.be/lh_E9rW-MJo`.
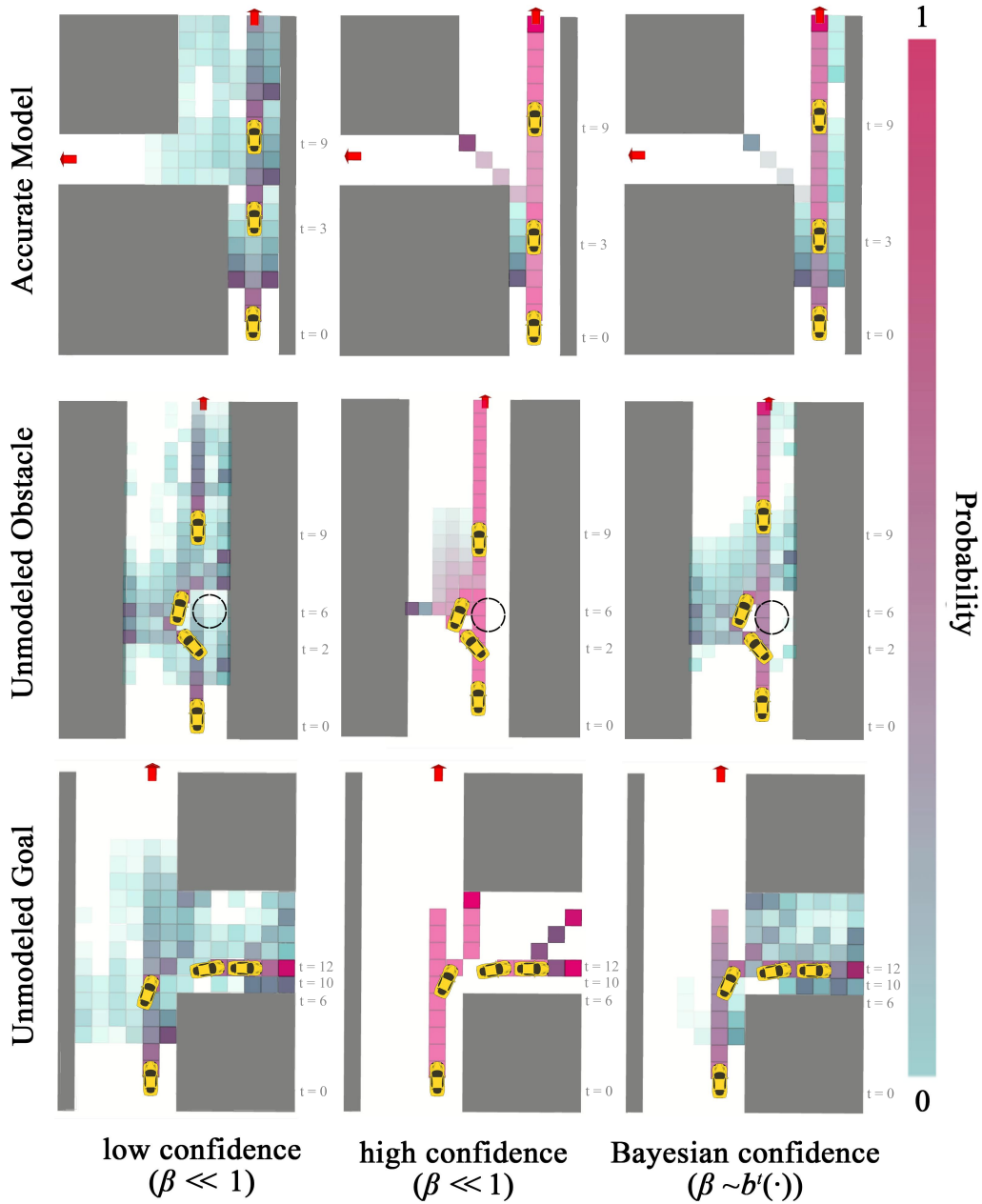
Fig. 3.2: Snapshots of Dubins car and probabilistic predictions. Top row: Car moves straight ahead toward one of two known goals (red arrows), staying in its lane. Middle row: Car suddenly swerves to the left to avoid a pothole. Bottom row: Car turns to the right, away from the only known goal. The left and center columns show results for low and high confidence predictors, respectively, and the right column shows our approach using Bayesian inferred model confidence. For all Dubins car videos, see: `https://youtu.be/sAJKNnP42fQ`.

Unmodeled obstacle

Often, robots do not have fully specified models of the environment. Here, we showcase the resilience of our approach to unmodeled obstacles which the human must avoid. In this scenario, the human has the same start and goal as in the accurate model case, except that there is an obstacle along the way. The robot is unaware of this obstacle, however, which means that in its vicinity the human's motion is not well-explained by Q_H, and $b(\beta)$ ought to place more probability mass on higher values of β.

The second rows of Fig. 3.1 and Fig. 3.2 illustrate this type of situation for the pedestrian and Dubins car, respectively. In Fig. 3.1, the pedestrian walks to an *a priori* known goal location and avoids an unmodeled spill on the ground. Analogously, in Fig. 3.2 the car swerves to avoid a large pothole. By inferring model confidence online, our approach generates higher-variance predictions of future state, but only in the vicinity of these unmodeled obstacles. At other times throughout the episode when Q_H is more accurate, our approach produces predictions more in line with the high model confidence predictor.

Unmodeled goal

In most realistic human-robot encounters, even if the robot does have an accurate environment map and observes all obstacles, it is unlikely for it to be aware of all human goals. We test our approach's resilience to unknown human goals by constructing a scenario in which the human moves between both known and unknown goals.

The third row of Fig. 3.1 illustrates this situation for the pedestrian example. Here, the pedestrian first moves to one known goal position, then to another, and finally back to the start which was not a modelled goal location. The first two legs of this trajectory are consistent with the robot's model of goal-oriented motion, though accurate prediction does require the predictor to infer *which* goal the pedestrian is walking toward. However, when the pedestrian returns to the start, her motion appears inconsistent with Q_H, skewing the robot's belief over β toward zero.

Similarly, in the third row of Fig. 3.2 we consider a situation in which a car makes an unexpected turn onto an unmapped access road. As soon as the driver initiates the turn, our predictor rapidly learns to distrust its internal model Q_H and shift its belief over β upward.

Safe probabilistic planning and tracking

Given probabilistic predictions of the human's future motion, the robot must plan efficient trajectories which avoid collision with high probability. In order to reason robustly about this probability of future collision, we must account for potential tracking errors incurred by the real system as it follows planned trajectories. To this end, we build on the recent FaSTrack framework of [18], which provides control-theoretic robust safety certificates in the presence of deterministic obstacles, and extend it to achieve approximate probabilistic collision-avoidance.

Background: fast planning, safe tracking

Recall that x_R is the robot's state for the purposes of motion planning, and that s_R encodes a higher-fidelity, potentially higher-dimensional notion of state (with associated dynamics). The recently proposed FaSTrack framework from [18] uses Hamilton-Jacobi reachability analysis to quantify the worst-case tracking performance of the s_R-system as it follows trajectories generated by the x_R-system. For further reading on reachability analysis refer to [8], [33], and [2]. A byproduct of this FaSTrack analysis is an error feedback controller that the s_R system can use to achieve this worst-case tracking error. The tracking error bound may be given to one of many off-the-shelf real-time motion planning algorithms operating in x_R-space in order to guarantee real-time collision-avoidance by the s_R-system.

Formally, FaSTrack precomputes an optimal tracking controller, as well as a corresponding compact set $\mathcal{E}$ in the robot's planning state space, such that $\big(\pi(s_R(t)) - x_{R,\text{ref}}(t)\big) \in \mathcal{E}$ for *any* reference trajectory proposed by the lower-fidelity planner. This bound $\mathcal{E}$ is a trajectory tracking certificate that can be passed to an online planning algorithm for real-time safety verification: the dynamical robot is guaranteed to *always* be somewhere within the bound relative to the current planned reference point $x_{R,\text{ref}}(t)$. This tracking error bound may sometimes be expressed analytically; otherwise, it may be computed numerically offline using level set methods, e.g. [32]. Equipped with $\mathcal{E}$, the planner can generate safe plans online by ensuring that the entire tracking error bound around the nominal state remains collision-free throughout the trajectory. Efficiently checking these $\mathcal{E}$-augmented trajectories for collisions with known obstacles is critical for real-time performance. Note that the planner only needs to know $\mathcal{E}$ (which is computed offline) and otherwise requires no explicit understanding of the high-fidelity model.

Running example: Since dynamics (3.2) *are decoupled in the three spatial directions, the bound* $\mathcal{E}$ *computed by FaSTrack is an axis-aligned box of dimensions* $\mathcal{E}_x \times \mathcal{E}_y \times \mathcal{E}_z$. *For further details refer to [15].*

Robust tracking, probabilistic safety

Unfortunately, planning algorithms for collision checking against deterministic obstacles cannot be readily applied to our problem. Instead, a trajectory's collision check should return the probability that it *might* lead to a collision. Based on this probability, the planning algorithm can discriminate between trajectories that are *sufficiently safe* and those that are not.

As discussed above, a safe online motion planner invoked at time t should continually check the probability that, at any future time τ, $(\pi(s_R(\tau)), x_H(\tau)) \in \mathcal{K}$.

The tracking error bound guarantee from FaSTrack allows us to conduct worst-case analysis on collisions given a human state x_H. Concretely, if no point in the Minkowski sum $\{x_R + \mathcal{E}\}$ is in the collision set with x_H, we can guarantee that the robot is not in collision with the human.

The probability of a collision event for any point $x_R(\tau)$ along a candidate trajectory is then

$$P_{\text{coll}}\big(x_R(\tau)\big) := P\big((x_R, x_H) \in \mathcal{K}\big). \tag{3.13}$$

Assuming worst-case tracking error bound $\mathcal{E}$, this quantity can be upper-bounded by the total probability that $x_H(\tau)$ will be in collision with *any* of the possible robot states $\tilde{x}_R \in \{x_R(\tau) + \mathcal{E}\}$. For each robot planning state $x_R \in \mathbb{R}^{n_R}$ we define the set of human states in potential collision with the robot:

$$\mathcal{H}_{\mathcal{E}}(x_R) := \{\tilde{x}_H \in \mathbb{R}^{n_H} : \tag{3.14}$$
$$\exists \tilde{x}_R \in \{x_R + \mathcal{E}\}, (\tilde{x}_R, \tilde{x}_H) \in \mathcal{K}\}.$$

Running example: Given $\mathcal{K}$ and $\mathcal{E}$, $\mathcal{H}_{\mathcal{E}}(x_R)$ is the set of human positions within the rectangle of dimensions $(l + \mathcal{E}_x) \times (l + \mathcal{E}_y)$ centered on $[p_x, p_y]$. A human anywhere in this rectangle could be in collision with the quadcopter.

The following result follows directly from the definition of the tracking error bound and a union bound.

Proposition 2. *The probability of a robot with worst case tracking error $\mathcal{E}$ colliding with the human at any trajectory point $x_R(\tau)$ is bounded above by the probability mass of $x_H(\tau)$ contained within $\mathcal{H}_{\mathcal{E}}(x_R(\tau))$.*

We shall consider *discrete-time* motion plans. The probability of collision along any such trajectory from current time step k to final step $k + K$ is upper-bounded by:

$$P_{\text{coll}}^{k:k+K} \leq \overline{P_{\text{coll}}^{k:k+K}} := \tag{3.15}$$
$$1 - \prod_{\kappa=k}^{k+K} P\left(x_H^\kappa \notin \mathcal{H}_{\mathcal{E}}(x_R^\kappa) \mid x_H^\kappa \notin \mathcal{H}_{\mathcal{E}}(x_R^s), k \leq s < \kappa\right).$$

Evaluating the right hand side of (3.15) exactly requires reasoning about the joint distribution of human states over all time steps and its conditional relationship on whether collision has yet occurred. This is equivalent to maintaining a probability distribution over the exponentially large space of trajectories $x_H^{k:k+K}$ that the human might follow. As motion planning occurs in real-time, we shall resort to a heuristic approximation of (3.15).

One approach to approximating (3.15) is to assume that the event $x_H^{\kappa_1} \notin \mathcal{H}_{\mathcal{E}}(x_R^{\kappa_1})$ is independent of $x_H^{\kappa_2} \notin \mathcal{H}_{\mathcal{E}}(x_R^{\kappa_2}), \forall \kappa_1 \neq \kappa_2$. This independence assumption is equivalent to removing the conditioning in (3.15). Unfortunately, this approximation is excessively pessimistic; if there is no collision at time step κ, then collision is also unlikely at time step $\kappa+1$ because both human and robot trajectories are continuous. In fact, for sufficiently small time discretization Δt and nonzero collision probabilities at each time step, the total collision probability resulting from an independence assumption would approach 1 exponentially fast in the number of time steps K.

We shall refine this approximation by finding a tight lower bound on the right hand side of (3.15). Because collision events are correlated in time, we first consider replacing each conditional probability $P\left(x_H^\kappa \notin \mathcal{H}_{\mathcal{E}}(x_R^\kappa) \mid x_H^s \notin \mathcal{H}_{\mathcal{E}}(x_R^s), k \leq s < \kappa\right)$ by 1 for all $\kappa \in \{k+1, \ldots, k+K\}$. This effectively lower bounds $\overline{P_{\text{coll}}^{k:k+K}}$ by the worst case probability of collision at the current time step k:

$$\overline{P_{\text{coll}}^{k:k+K}} \geq 1 - P\left(x_H^k \notin \mathcal{H}_{\mathcal{E}}(x_R^k)\right) \tag{3.16}$$
$$= P\left(x_H^k \in \mathcal{H}_{\mathcal{E}}(x_R^k)\right).$$
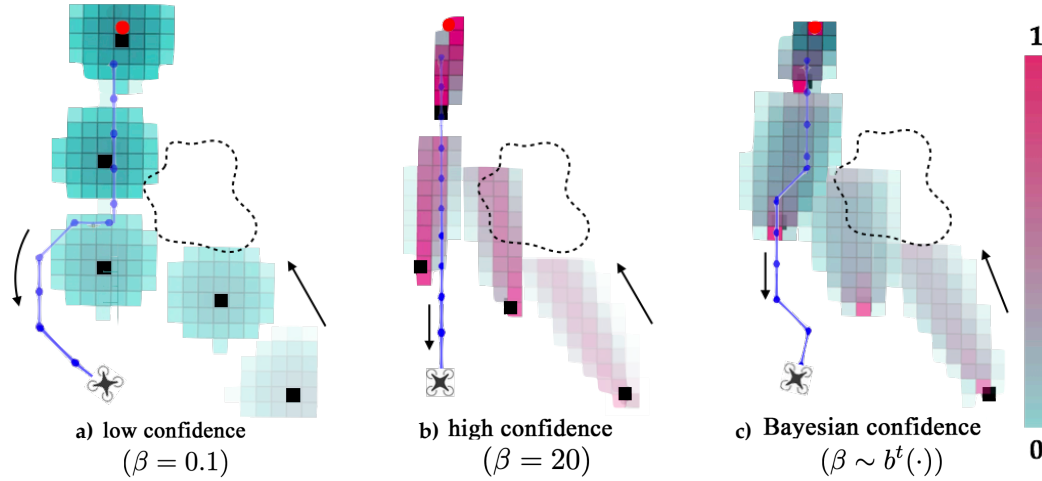
Fig. 3.3: Scenario from the middle row of Fig. 3.1 visualized with robot's trajectory. When β is low and the robot is not confident, it makes large deviations from its path to accommodate the human. When β is high, the robot refuses to change course and comes dangerously close to the human. With inferred model confidence, the robot balances safety and efficiency with a slight deviation around the human.

This bound is extremely loose in general, because it completely ignores the possibility of future collision. However, note that probabilities in the product in (3.15) may be conditioned in any particular order (not necessarily chronological). This commutativity allows us to generate $K - k + 1$ lower bounds of the form $\overline{P_{\text{coll}}^{k:k+K}} \geq P\big(x_H^\kappa \in \mathcal{H}_\mathcal{E}(x_R^\kappa)\big)$ for $\kappa \in \{k, \dots, k + K\}$. Taking the tightest of all of these bounds, we can obtain an informative, yet quickly computable, approximator for the sought probability:

$$\overline{P_{\text{coll}}^{k:k+K}} \geq \max_{\kappa \in \{k:k+K\}} P\big(x_H^\kappa \in \mathcal{H}_\mathcal{E}(x_R^\kappa)\big) \ \approx P_{\text{coll}}^{k:k+K} \tag{3.17}$$

To summarize, the left inequality in (3.17) lower-bounds $\overline{P_{\text{coll}}^{k:k+K}}$ with the greatest marginal collision probability at *any point* in the trajectory. On the right side of (3.17), we take this greatest marginal collision probability as an approximator of the actual probability of collision $P_{\text{coll}}^{k:k+K}$ over the entire trajectory. In effect, we shall approximate $P_{\text{coll}}^{k:k+K}$ with a tight lower bound *of an upper bound*. While this type of approximation may err on the side of optimism, we note that both the robot's ability to replan over time and the fact that the left side of (3.17) is an *upper bound* on total trajectory collision probability mitigate this potentially underestimated risk.

Safe online planning under uncertain human predictions

This approximation of collision probability allows the robot to discriminate between valid and invalid candidate trajectories during motion planning. Using the prediction methodology proposed in Section 3.2, we may quickly generate, at every time t, the marginal probabilities in (3.17) at each future time $\kappa \in \{k, \dots, k + K\}$, based on past observations at times $0, \dots, k$. The planner then computes the instantaneous probability of collision $P\big(x_H^\kappa \in \mathcal{H}_\mathcal{E}(x_R^\kappa)\big)$ by
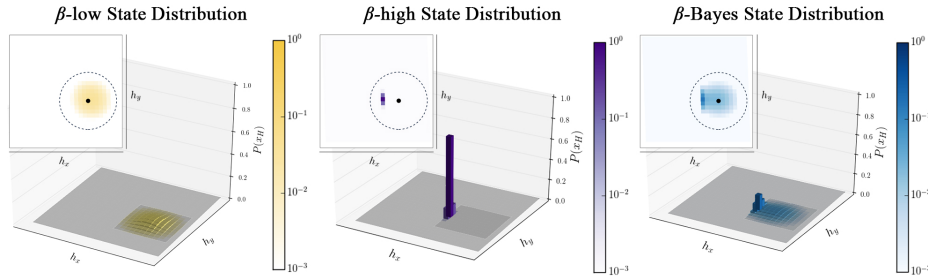
Fig. 3.4: The human (black dot) is moving west towards a goal. Visualized are the predicted state distributions for one second into the future when using low, high, and Bayesian model confidence. Higher-saturation indicates higher likelihood of occupancy. The dashed circle represents the pedestrian's 1 second forward reachable set.

integrating $P\big(x_H^\tau \mid x_H^{0:k}\big)$ over $\mathcal{H}_\mathcal{E}(x_R^\kappa)$, and rejects the candidate point x_R^κ if this probability exceeds P_{th}.

Note that for graph-based planners that consider candidate trajectories by generating a graph of time-stamped states, rejecting a candidate edge from this graph is equivalent to rejecting all further trajectories that would contain that edge. This early rejection rule is consistent with the proposed approximation (3.17) of $P_{\text{coll}}^{k:k+K}$ while preventing unnecessary exploration of candidate trajectories that would ultimately be deemed unsafe.

Throughout operation, the robot follows each planned trajectory using the error feedback controller provided by FaSTrack, which ensures that the robot's high-fidelity state representation s_R and the lower-fidelity state used for planning, x_R, differ by no more than the tracking error bound $\mathcal{E}$. This planning and tracking procedure continues until the robot reaches its desired goal state.

Running example: Our quadcopter is now required to navigate to a target position shown in Fig. 3.3 without colliding with the human. Our proposed algorithm successfully avoids collisions at all times, replanning to leave greater separation from the human whenever her motion departs from the model. In contrast, robot planning with fixed model confidence is either overly conservative at the expense of time and performance or overly aggressive at the expense of safety.

Connections to reachability analysis

In this Section, we present an alternative, complementary analysis of the overall safety properties of the proposed approach to prediction and motion planning. This discussion is grounded in the language of reachability theory and worst-case analysis of human motion.

Forward reachable set

Throughout this Section, we frequently refer to the human's time-indexed forward reachable set. We define this set formally below.

Definition 6. *For a dynamical system $\dot{x} = f(x, u)$ with state trajectories given by the function $\xi\big(x(0), t, u(\cdot)\big) =: x(t)$, the forward reachable set $\Omega_F(x, t)$ of a state x after time t*
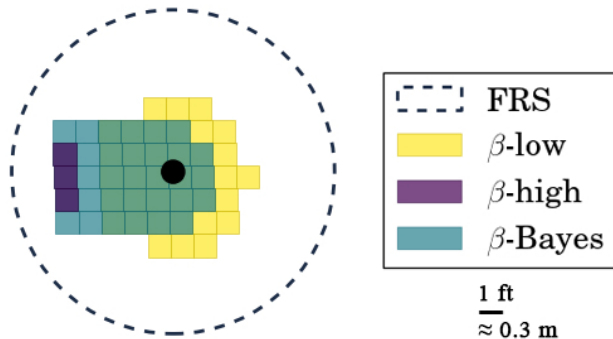
Fig. 3.5: Visualization of the states with probability greater than or equal to the collision threshold, $P_{\text{th}} = 0.01$. The human's forward reachable set includes the set of states assigned probability greater than P_{th}. We show these "high probability" predicted states for predictors with fixed low and high β, as well as our Bayesian-inferred β.

has elapsed is

$$\Omega_F(x,t) := \{x' : \exists u(\cdot), x' = \xi(x,t,u(\cdot))\} .$$

That is, a state x' is in the forward reachable set of x after time t if it is *reachable* via some applied control signal $u(\cdot)$.

Remark 3. *For $P_{th} = 0$ and any finite β, the set of states assigned probability greater than P_{th} is identical to the forward reachable set, up to discretization errors. This is visualized for low, high, and Bayesian model confidence in Fig. 3.4.*

A sufficient condition for the safety of individual trajectories

Above, we construct an approximation to the probability of collision along a trajectory, which we use during motion planning to avoid potentially dangerous states. To make this guarantee of collision-avoidance for a motion plan even stronger, it would suffice to ensure that the robot never comes too close to the human's forward reachable set. More precisely, a planned trajectory is safe if $\{x_R(t) + \mathcal{E}\} \cap \Omega_F(x_H, t) = \emptyset$, for every state $x_R(t)$ along a motion plan generated when the human was at state x_H. The proof of this statement follows directly from the properties of the tracking error bound $\mathcal{E}$ described in Section 3.2.

While this condition may seem appealing, it is in fact highly restrictive. The requirement of avoiding the full forward reachable set is not always possible in confined spaces; indeed, this was our original motivation for wanting to predict human motion. However, despite this shortcoming, the logic behind this sufficient condition for safety provides insight into the effectiveness of our framework.

Recovering the forward reachable set

Though it will not constitute a formal safety guarantee, we analyze the empirical safety properties of our approach by examining how our predicted state distributions over time relate to forward reachable sets. During operation, our belief over model confidence β evolves

to match the degree to which the utility model Q_H explains recent human motion. The "time constant" governing the speed of this evolution may be tuned by the system designer to be arbitrarily fast by choosing the parameter ϵ to be small, as discussed in above. Thus, we may safely assume that $b(\beta)$ places high probability mass on small values of β as soon as the robot observes human motion which is not well explained by Q_H.

Fig. 3.5 shows the sets of states with "high enough" ($> P_{\text{th}}$) predicted probability mass overlaid on the human's forward reachable set at time t, which is a circle of radius $v_H t$ centered on x_H for the dynamics in our running example. When β is high (10), we observe that virtually all of the probability mass is concentrated in a small number of states in the direction of motion predicted by our utility model. When β is low (0.05) we observe that the set of states assigned probability above our collision threshold P_{th} occupies a much larger fraction of the reachable set. A typical belief $b(\beta)$ recorded at a moment when the human was roughly moving according to Q_H yields an intermediate set of states.

Fig. 3.6 illustrates the evolution of these sets of states over time, for the unmodeled obstacle example of Figure 3.1 in which a pedestrian avoids a spill. Each row corresponds to the predicted state distribution at a particular point in time. Within a row, each column shows the reachable set and the set of states assigned occupancy probability greater than $P_{\text{th}} = 0.01$. The color of each set of states corresponds to the value of β used by the low confidence and high confidence predictors, and the *maximum a posteriori* value of β for the Bayesian confidence predictor. The human's known goal state is marked by a red dot.

Interestingly, as the Bayesian model confidence decreases—which occurs when the pedestrian turns to avoid the spill at $t \approx 6$ s—the predicted state distribution assigns high probability to a relatively large set of states, but unlike the low-β predictor that set of states is oriented toward the known goal. Of course, had $b(\beta)$ placed even more probability mass on lower values of β then the Bayesian confidence predictor would converge to the low confidence one.

Additionally, we observe that, within each row as the prediction horizon increases, the area contained within the forward reachable set increases and the fraction of that area contained within the predicted sets decreases. This phenomenon is a direct consequence of our choice of threshold P_{th}. Had we chosen a smaller threshold value, a larger fraction of the forward reachable set would have been occupied by the lower-β predictors.

This observation may be viewed prescriptively. Recalling the sufficient condition for safety of planned trajectories above, if the robot replans every T_{replan} seconds, we may interpret the fraction of $\Omega_F(\cdot, t + T_{\text{replan}})$ assigned occupancy probability greater than P_{th} by the low confidence predictor as a rough indicator of the safety of an individual motion plan, robust to worst-case human movement. As this fraction tends toward unity, the robot is more and more likely to be safe. However, for any $P_{\text{th}} > 0$, this fraction approaches zero for $T_{\text{replan}} \uparrow \infty$. This immediately suggests that, if we wish to replan every T_{replan} seconds, we can achieve a particular level of safety as measured by this fraction by choosing an appropriate threshold P_{th}.

In summary, confidence-aware predictions rapidly place high probability mass on low values of β whenever human motion is not well-explained by utility model Q_H. Whenever this happens, the resulting predictions encompass a larger fraction of the forward reachable set, and in the limit that $P_{\text{th}} \downarrow 0$ we recover the forward reachable set exactly. The larger this fraction, the more closely our approach satisfies the sufficient condition for safety presented
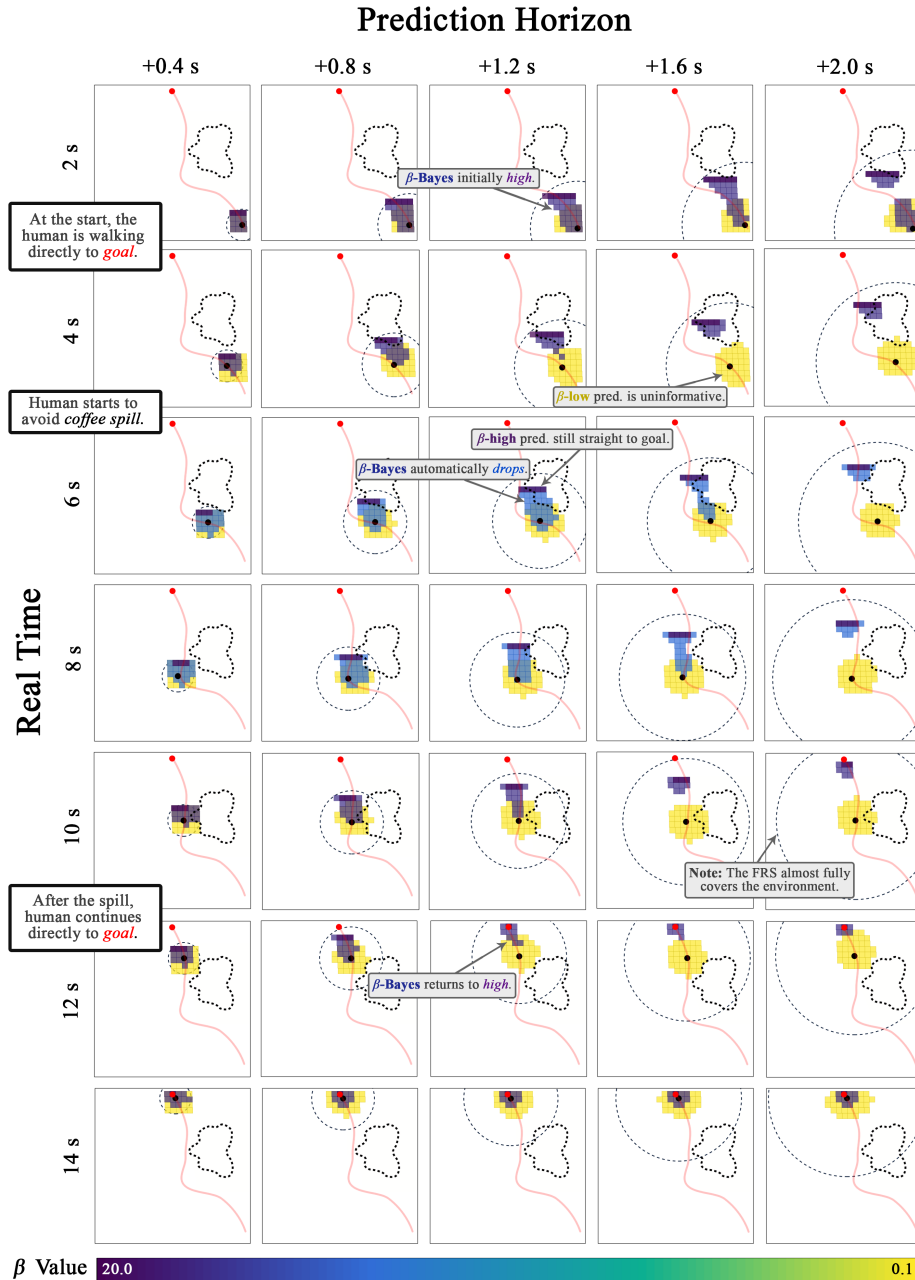
Fig. 3.6: The human (black dot) is walking towards the known goal (red dot) but has to avoid an unmodeled coffee spill on the ground. Here we show the snapshots of the predictions at various future times (columns) as the human walks around in real time (rows). The visualized states have probability greater than or equal to $P_{\text{th}} = 0.01$. Each panel displays the human prediction under low confidence (in yellow), high confidence (in dark purple), and Bayesian confidence (colored as per the most likely β value), as well as the forward reachable set. The human's actual trajectory is shown in red.
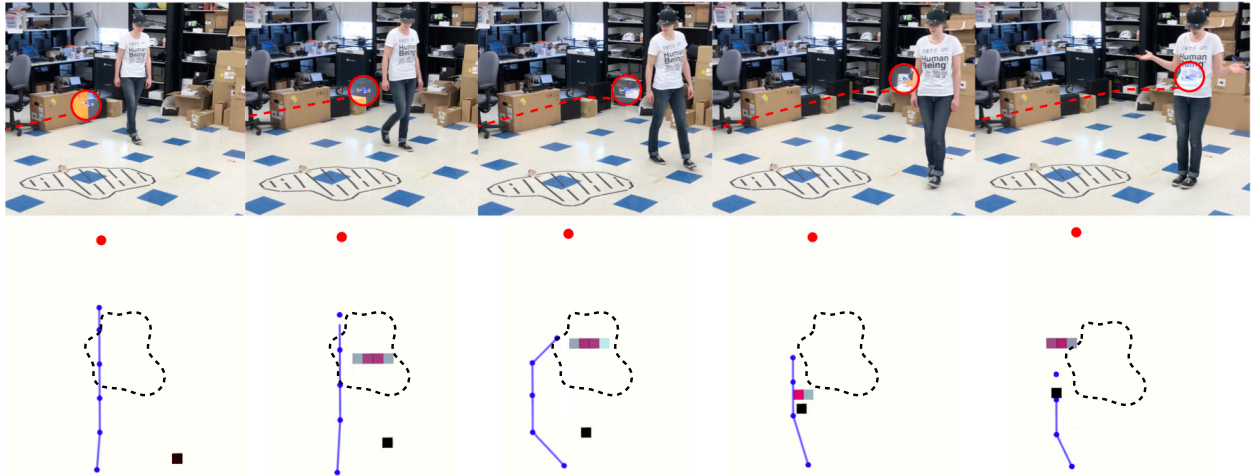
Fig. 3.7: Predicting with fixed-β (in this case, $\beta = 20$) can yield highly inaccurate predictions (and worse, confidently inaccurate ones). The subsequent motion plans may not be safe; here, poor prediction quality leads to a collision.
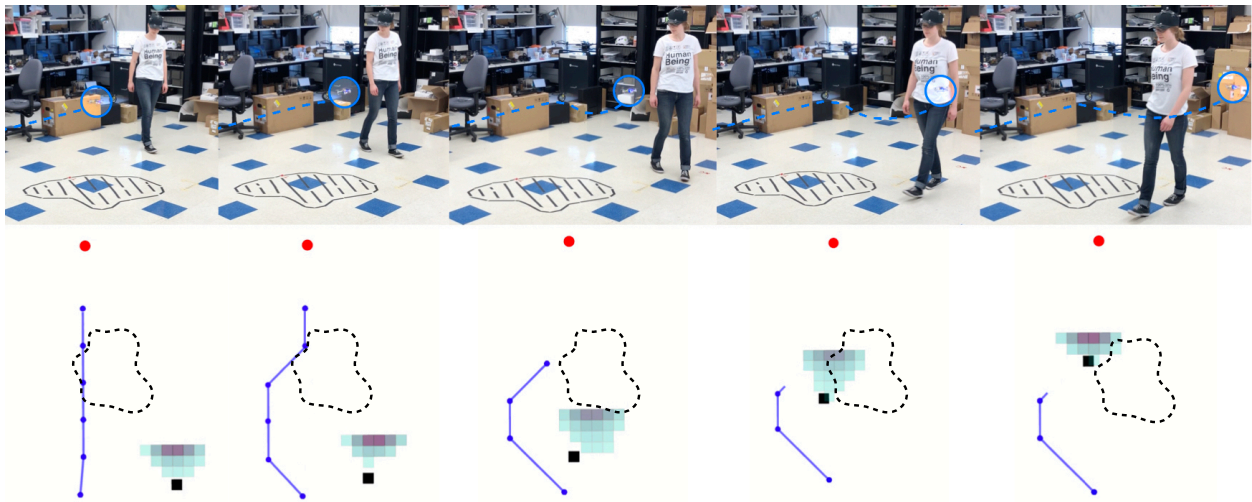
Fig. 3.8: Inferring β leads to predicted state distributions whose entropy increases whenever the utility model Q_H fails to explain observed human motion. The resulting predictions are more robust to modeling errors, resulting in safer motion plans. Here, the quadcopter successfully avoids the pedestrian even when she turns unexpectedly.

above.

Hardware Demonstration

We implemented confidence-aware human motion prediction (Section 3.2) and integrated it into a real-time, safe probabilistic motion planner (Section 3.2), all within the Robot Operating System (ROS) software framework of [36]. To demonstrate the efficacy of our methods, we tested our work for the quadcopter-avoiding-pedestrian example used for illustration throughout this paper. Human trajectories were recorded as (x, y) positions on the

ground plane at roughly 235 Hz by an OptiTrack infrared motion capture system, and we used a Crazyflie 2.0 micro-quadcopter, also tracked by the OptiTrack system.[2]

Figure 3.3 illustrates the unmodeled obstacle case from Section 3.2, in which the pedestrian turns to avoid a spill on the ground. Using a low model confidence results in motion plans that suddenly and excessively deviate from the ideal straight-line path when the pedestrian turns to avoid the spill. By contrast, the high confidence predictor consistently predicts that the pedestrian will walk in a straight line to the goal even when they turn; this almost leads to collision, as shown detail in Figure 3.7. Our proposed approach for Bayesian model confidence initially assigns high confidence and predicts that the pedestrian will walk straight to the goal, but when they turn to avoid the spill, the predictions become less confident. This causes the quadcopter to make a minor course correction, shown in further detail in Figure 3.8.

---

[2]We note that in a more realistic setting, we would require alternative methods for state estimation using other sensors, such as lidar and/or camera(s). A video recording may be found at `https://youtu.be/2ZRGxWknENg`.

Chapter 4

Coupled Planning and Prediction

So far, in Chapters 2 and 3 we have seen the problems of motion planning and probabilistic prediction of the decisions of other agents. Recalling the diagram of Figure 1.1, it is clear that these problems are typically solved in sequence with prediction *preceding* planning. In this Chapter, we shall see why this may not always be a good idea, and that coupling the two results in a fairly general form of a differential game. We shall note some preliminary facts about these games and subsequently present a new real-time, approximate solution strategy for them. Much of this Chapter is based upon work published at the International Conference on Robotics and Automation in 2020, titled "Efficient iterative linear-quadratic approximations for nonlinear multi-player general-sum differential games" [14] (coauthored with Ellis Ratner, Lasse Peters, Anca Dragan, and Claire Tomlin), and "An iterative quadratic method for general-sum differential games with feedback linearizable dynamics" [12] (coauthored with Vicenç Rubies-Royo and Claire Tomlin).

4.1   The problem with operating sequentially

Recalling the diagram of Figure 1.1, we see that it is common to *predict* the future state of the world (i.e., decisions of other agents) before *planning* an autonomous response. This sequential pipelining effectively means that planned responses do not impact the predicted state of the world. In other words, it encodes that decisions of the autonomous system cannot affect the decisions of other agents. This could be a fair assumption, for example in the case of an autonomous car which observes a pedestrian looking at his or her phone. Whether the pedestrian chooses to jaywalk or not should not depend upon the behavior of the car (unless he or she has very good hearing). In more common situations, though, and especially in cases where we can expect a certain level of situational awareness like intersections and merging, this sequential operation may is a dramatic simplification. To really capture the effect each agent has on every other agent, we should couple the problems of prediction and planning, which results in a multi-player differential game. In full generality, this should almost certainly be a *stochastic* differential game in order to account for the inherent randomness in agents' decisions. As a first step, however, we shall restrict our attention to deterministic differential games in this Chapter.

4.2 Preliminaries for differential games

We consider a N-player finite horizon general-sum differential game characterized by non-linear system dynamics

$$\dot{x} = f(t, x, u_{1:N}), \tag{4.1}$$

where $x \in \mathbb{R}^n$ is the state of the system, and $u_i \in \mathbb{R}^{m_i}, i \in [N] \equiv \{1, \ldots, N\}$ is the control input of player i, and $u_{1:N} \equiv (u_1, u_2, \ldots, u_N)$. Each player has a cost function J_i defined as an integral of running costs g_i. J_i is understood to depend implicitly upon the state trajectory $x(\cdot)$, which itself depends upon initial state $x(0)$ and control signals $u_{1:N}(\cdot)$:

$$J_i\big(u_{1:N}(\cdot)\big) \triangleq \int_0^T g_i\big(t, x(t), u_{1:N}(t)\big) dt, \forall i \in [N]. \tag{4.2}$$

We shall presume that f is continuous in t and continuously differentiable in $\{x, u_i\}$ uniformly in t. We shall also require g_i to be twice differentiable in $\{x, u_i\}, \forall t$.

Ideally, we would like to find time-varying state feedback control strategies $\gamma_i^* \in \Gamma_i$ for each player i which constitute a global Nash equilibrium for the game defined by (4.1) and (4.2). Here, the strategy space Γ_i for player i is the set of measurable functions $\gamma_i : [0, T] \times \mathbb{R}^n \to \mathbb{R}^{m_i}$ mapping time and state to player i's control input. Note that, in this formulation, player i only observes the state of the system at each time and is unaware of other players' control strategies. With a slight abuse of notation $J_i(\gamma_1; \ldots; \gamma_N) \equiv J_i\big(\gamma_1(\cdot, x(\cdot)), \ldots, \gamma_N(\cdot, x(\cdot))\big)$, the global Nash equilibrium is defined as the set of strategies $\{\gamma_i\}$ for which the following inequalities hold (see, e.g., [3, Chapter 6]):

$$\begin{aligned} J_i^* &\triangleq J_i(\gamma_1^*; \ldots; \gamma_{i-1}^*; \gamma_i^*; \gamma_{i+1}^*; \ldots \gamma_N^*) \\ &\leq J_i(\gamma_1^*; \ldots; \gamma_{i-1}^*; \gamma_i; \gamma_{i+1}^*; \ldots \gamma_N^*), \forall i \in [N]. \end{aligned} \tag{4.3}$$

In (4.3), the inequalities must hold for all $\gamma_i \in \Gamma_i, \forall i \in [N]$. Informally, a set of feedback strategies $(\gamma_1^*, \ldots, \gamma_N^*)$ is a global Nash equilibrium if no player has a unilateral incentive to deviate from their current strategy.

Since finding a global Nash equilibrium is generally computationally intractable, recent work in adversarial learning [29] and motion planning [52, 51] consider local Nash equilibria instead. Further, [52, 51] simplify the information structure of the game and consider open loop, rather than feedback, strategies. Local Nash equilibria are characterized similarly to (4.3), except that the inequalities may only hold in a local neighborhood within the strategy space [37, Definition 1]. In this paper, we shall seek a related type of equilibrium, which we describe more precisely in Section 10. Intuitively, we seek strategies which satisfy the *global* Nash conditions (4.3) for the limit of a sequence of *local* approximations to the game. Our experimental results indicate that it does yield highly interactive strategies in a variety of differential games.

4.3 Iterative linear-quadratic games

We approach the N-player general-sum game with dynamics (4.1) and costs (4.2) from the perspective of classical LQ games. It is well known that Nash equilibrium strategies

---

Algorithm 2: Iterative LQ Games

   Input: initial state $x(0)$, control strategies $\{\gamma_i^0\}_{i\in[N]}$, time horizon T, running costs
       $\{g_i\}_{i\in[N]}$
   Output: converged control strategies $\{\gamma_i^*\}_{i\in[N]}$

1 **for** *iteration* $k = 1, 2, \ldots$ **do**
2 $\xi^k \equiv \{\hat{x}(t), \hat{u}_{1:N}(t)\}_{t\in[0,t]} \leftarrow$
3   getTrajectory$\big(x(0), \{\gamma_i^{k-1}\}\big)$;
4 $\{A(t), B_i(t)\} \leftarrow$ linearizeDynamics$\big(\xi^k\big)$;
5 $\{l_i(t), Q_i(t), r_{ij}(t), R_{ij}(t)\} \leftarrow$ quadraticizeCost$\big(\xi^k\big)$;
6 $\{\tilde{\gamma}_i^k\} \leftarrow$ solveLQGame$\big($
7 $\{A(t), B_i(t), l_i(t), Q_i(t), r_{ij}(t), R_{ij}(t)\}\big)$;
8 $\{\gamma_i^k\} \leftarrow$ stepToward$\big(\{\gamma_i^{k-1}, \tilde{\gamma}_i^k\}\big)$;
9   **if** *converged* **then**
10   **return** $\{\gamma_i^k\}$

---

for finite-horizon LQ games satisfy coupled Riccati differential equations. These coupled Riccati equations may be derived by substituting linear dynamics and quadratic running costs into the generalized HJ equations [43] and analyzing the first order necessary conditions of optimality for each player [3, Chapter 6]. These coupled differential equations may be solved approximately in discrete-time using dynamic programming [3]. We will leverage the existence and computational efficiency of this discrete-time LQ solution to solve successive approximations to the original *non*linear *non*quadratic game.

Iterative LQ game algorithm

Our iterative LQ game approach proceeds in stages, as summarized in Algorithm 2. We begin with an initial state $x(0)$ and initial feedback control strategies $\{\gamma_i^0\}$ for each player i, and integrate the system forward (line 3 of Algorithm 2) to obtain the current trajectory iterate $\xi^k \equiv \{\hat{x}(t), \hat{u}_{1:N}(t)\}_{t\in[0,T]}$. Next (line 4) we obtain a Jacobian linearization of the dynamics f about trajectory ξ^k. At each time $t \in [0, T]$ and for arbitrary states $x(t)$ and controls $u_i(t)$ we define deviations from this trajectory $\delta x(t) = x(t) - \hat{x}(t)$ and $\delta u_i(t) = u_i(t) - \hat{u}_i(t)$. Thus equipped, we compute a continuous-time linear system approximation about ξ^k:

$$\dot{\delta x}(t) \approx A(t)\delta x(t) + \sum_{i\in[N]} B_i(t)\delta u_i(t), \tag{4.4}$$

where $A(t)$ is the Jacobian $D_{\hat{x}}f\big(t, \hat{x}(t), \hat{u}_{1:N}(t)\big)$ and $B_i(t)$ is likewise $D_{\hat{u}_i}f\big(t, \hat{x}(t), \hat{u}_{1:N}(t)\big)$.
   We also obtain a quadratic approximation to the running cost g_i for each player i (see

line 5 of Algorithm 2)

$$g_i\big(t, x(t), u_{1:N}(t)\big) \approx$$

$$g_i\big(t, \hat{x}(t), \hat{u}_{1:N}(t)\big) + \frac{1}{2}\delta x(t)^T \left(Q_i(t)\delta x(t) + 2l_i(t)\right) +$$

$$\frac{1}{2}\sum_{j\in[N]} \delta u_j(t)^T \left(R_{ij}(t)\delta u_j(t) + 2r_{ij}(t)\right), \tag{4.5}$$

where vector $l_i(t)$ is the gradient $D_{\hat{x}}g_i$, r_{ij} is $D_{\hat{u}_j}g_i$, and matrices Q_i and R_{ij} are Hessians $D^2_{\hat{x}\hat{x}}g_i$ and $D^2_{\hat{u}_j\hat{u}_j}g_i$, respectively. We neglect mixed partials $D^2_{\hat{u}_j\hat{u}_k}g_i$ and $D^2_{\hat{x}\hat{u}_j}g_i$ as they rarely appear in cost structures of practical interest, although they could be incorporated if needed.

Thus, we have constructed a finite-horizon continuous-time LQ game, which may be solved via coupled Riccati differential equations [3, 17]. This results in a new set of *candidate* feedback strategies $\{\tilde{\gamma}_i^k\}$ which constitute a feedback (global) Nash equilibrium of the LQ game [3]. In fact, these feedback strategies are affine maps of the form:

$$\tilde{\gamma}_i^k\big(t, x(t)\big) = \hat{u}_i(t) - P_i^k(t)\delta x(t) - \alpha_i^k(t), \tag{4.6}$$

with gains $P_i^k(t) \in \mathbb{R}^{m_i \times n}$ and affine terms $\alpha_i^k(t) \in \mathbb{R}^{m_i}$.

However, we find that choosing $\gamma_i^k = \tilde{\gamma}_i^k$ often causes Algorithm 2 to diverge because the trajectory resulting from $\{\tilde{\gamma}_i\}$ is far enough from the current trajectory iterate ξ^k that the dynamics linearizations (Algorithm 2, line 4) and cost quadraticizations (line 5) no longer hold. As in ILQR [48], to improve convergence, we take only a small step in the "direction" of $\tilde{\gamma}_i^k$.[1] More precisely, for some choice of step size $\eta \in (0, 1]$, we set

$$\gamma_i^k\big(t, x(t)\big) = \hat{u}_i(t) - P_i^k(t)\delta x(t) - \eta\alpha_i^k(t), \tag{4.7}$$

which corresponds to line 8 in Algorithm 2. Note that at $t = 0$, $\delta x(0) = 0$ and $\gamma_i^k\big(0, x(0)\big) = \hat{u}_i(0) - \eta\alpha_i^k(0)$. Thus, taking $\eta = 0$, we have $\gamma_i^k\big(t, x(t)\big) = \hat{u}_i(t)$ (which may be verified recursively). That is, when $\eta = 0$ we recover the open-loop controls from the previous iterate, and hence $x(t) = \hat{x}(t)$. Taking $\eta = 1$, we recover the LQ solution in (4.6). Similar logic implies the following lemma.

Lemma 3. *Suppose that trajectory ξ^* is a fixed point of Algorithm 2, with $\eta \neq 0$. Then, the converged affine terms $\{\alpha_i^*(t)\}$ must all be identically zero for all time.*

In ILQR, it is important to perform a line-search over step size η to ensure a sufficient decrease in cost at every iteration, and thereby improve convergence (e.g., [48]). In the context of a noncooperative game, however, line-searching to decrease "cost" does not make sense, as costs $\{J_i\}$ may conflict. For this reason, like other local methods in games (e.g., [52]), our approach is not guaranteed to converge from arbitrary initializations. In practice, however, we find that our algorithm typically converges for a fixed, small step size (e.g. $\eta = 0.01$). Heuristically decaying step size with each iteration k or line-searching until $\|\xi^k - \xi^{k-1}\|$ is smaller than a threshold are also promising alternatives. Further investigation of line-search methods in games is a rich topic of future research.

---

[1]We also note that, in practice, it is often helpful to "regularize" the problem by adding scaled identity matrices ϵI to Q_i and/or R_{ij}.

Remark 4. *Although we have presented our algorithm in continuous-time, in practice, we solve the coupled Riccati equations analytically in discrete-time via dynamic programming. Please refer to [3, Corollary 6.1] for a full derivation. To discretize time at resolution Δt, we employ Runge-Kutta integration of nonlinear dynamics (2.1) with a zero-order hold for control input over each time interval Δt.*

That is, we numerically compute:

$$\hat{x}(t + \Delta t) = \hat{x}(t) + \int_t^{t+\Delta t} f(s, \hat{x}(s), \hat{u}_{1:N}(s)) ds$$
$$\text{where } \hat{u}_i(s) = \gamma_i^{k-1}(t, \hat{x}(t)), \forall i \in [N], \text{and}$$
$$\hat{x}(0) = x(0). \tag{4.8}$$

Characterizing fixed points

Suppose Algorithm 2 converges to a fixed point $(\gamma_1^*, \ldots, \gamma_N^*)$. These strategies are the *global* Nash equilibrium of a *local* LQ approximation of the original game about the limiting operating point ξ^*. While it is tempting to presume that such fixed points are also local Nash equilibria of the original game, this is not always true because converged strategies are only optimal for a LQ *approximation* of the game at every time rather than the original game. This approximation neglects higher order coupling effects between each player's running cost g_i and other players' inputs $u_j, j \neq i$. These coupling effects arise in the game setting but *not* in the optimal control setting, where ILQR converges to local minima.

Computational complexity and runtime

The per-iteration computational complexity of our approach is comparable to that of ILQR, and scales modestly with the number of players, N. Specifically, at each iteration, we first linearize system dynamics about ξ^k. Presuming that the state dimension n is larger than the control dimension m_i for each player, linearization requires computing $\mathcal{O}(n^2)$ partial derivatives at each time step (which also holds for ILQR). We also quadraticize costs, which requires $\mathcal{O}(Nn^2)$ partial derivatives at each time step (compared to $\mathcal{O}(n^2)$ for ILQR). Finally, solving the coupled Riccati equations of the resulting LQ game at each time step has complexity $\mathcal{O}(N^3n^3)$, which may be verified by inspecting [3, Corollary 6.1] (for ILQR, this complexity is $\mathcal{O}(n^3)$).

Total algorithmic complexity depends upon the number of iterations, which we currently have no theory to bound. However, empirical results are extremely promising. For the three-player 14-state game described in below, each iteration takes $< 8\,\text{ms}$ and the entire game can be solved from a zero initialization $(P_i^0(\cdot) = 0, \alpha_i^0(\cdot) = 0)$ in $< 0.25\,\text{s}$. Moreover, receding horizon invocations in a hardware collision-avoidance test can be solved in $< 50\,\text{ms}$ (see below). All computation times are reported for single-threaded operation on a 2017 MacBook Pro with a 2.8 GHz Intel Core i7 CPU. For reference, the iterative best response scheme of [51] reports solving a receding horizon two-player zero-sum racing game at $2\,\text{Hz}$, and the method of [47] reportedly takes several minutes to converge for a different two-player zero-sum example. The dynamics and costs in both cases differ from those described below (or are not clearly reported); nonetheless, the runtime of our approach compares favorably.

Examples

In this Section, we demonstrate our algorithm experimentally in three-player noncooperative settings, both in software simulation and hardware.[2]

Monte Carlo study. We begin by presenting a Monte Carlo study of the convergence properties of Algorithm 2. As we shall see, the solution to which Algorithm 2 converges depends upon the initial strategy of each player, γ_i^0. For clarity, we study this sensitivity in a game with simplified cost structure so that differences in solution are more easily attributable to coupling between players.

Concretely, we consider a three-player "hallway navigation" game with time horizon $10\,\text{s}$ and discretization $0.1\,\text{s}$. Here, three people wish to interchange positions in a narrow hallway while maintaining at least $1\,\text{m}$ clearance between one another. We model each player i's motion as:

$$\begin{aligned} \dot{p}_{x,i} &= v_i \cos(\theta_i)\,, \quad \dot{\theta}_i = \omega_i\,, \\ \dot{p}_{y,i} &= v_i \sin(\theta_i)\,, \quad \dot{v}_i = a_i\,, \end{aligned} \tag{4.9}$$

where $p_i := (p_{x,i}, p_{y,i})$ denotes player i's position, θ_i heading angle, v_i speed, and input $u_i := (\omega_i, a_i)$ yaw rate and longitudinal acceleration. Concatenating all players' states into a global state vector $x := (p_{x,i}, p_{y,i}, \theta_i, v_i)_{i=1}^3$, the game has 12 state dimensions and six input dimensions.

We encode this problem with running costs g_i (4.2) expressed as weighted sums of the following:

$$\text{wall: } \mathbf{1}\{|p_{y,i}| > d_{\text{hall}}\}(|p_{y,i}| - d_{\text{hall}})^2 \tag{4.10}$$

$$\text{proximity: } \mathbf{1}\{\|p_i - p_j\| < d_{\text{prox}}\}(d_{\text{prox}} - \|p_i - p_j\|)^2 \tag{4.11}$$

$$\text{goal: } \mathbf{1}\{t > T - t_{\text{goal}}\}\|p_i - p_{\text{goal},i}\|^2 \tag{4.12}$$

$$\text{input: } u_i^T R_{ii} u_i \tag{4.13}$$

Here, $\mathbf{1}\{\cdot\}$ is the indicator function, i.e., it takes the value 1 if the given condition holds, and 0 otherwise. d_{hall} and d_{prox} denote threshold distances from hallway center and between players, which we set to 0.75 and $1\,\text{m}$, respectively. The goal cost is active only for the last t_{goal} seconds, and the goal position is given by $p_{\text{goal},i}$ for each player i. Control inputs are penalized quadratically, with R_{ii} a diagonal matrix. The hallway is too narrow for all players to cross simultaneously without incurring a large proximity cost; hence, this proximity cost induces strong coupling between players' strategies.

Figure 4.1 displays the results of our Monte Carlo study. We seed Algorithm 2 with 500 random sinusoidal open-loop initial strategies, which correspond to the trajectories shown in Figure 4.1(E). From each of these initializations, we run Algorithm 2 for 100 iterations and cluster the resulting trajectories by Euclidean distance. As shown in Figure 4.1(A1, B1, C1), these clusters correspond to plausible modes of interaction; in each case, one or more players incur slightly higher cost to make room for the others to pass. Beside each of these clusters in Figure 4.1(A2, B2, C2), we also report the mean and standard deviation of each player's cost at each solver iteration. As shown in Figure 4.1(F), state trajectories converge within an ℓ_∞ tolerance of 0.01 in well under 100 iterations.

---

[2]Video summary available at `https://youtu.be/KPEPk-QrkQ8`.
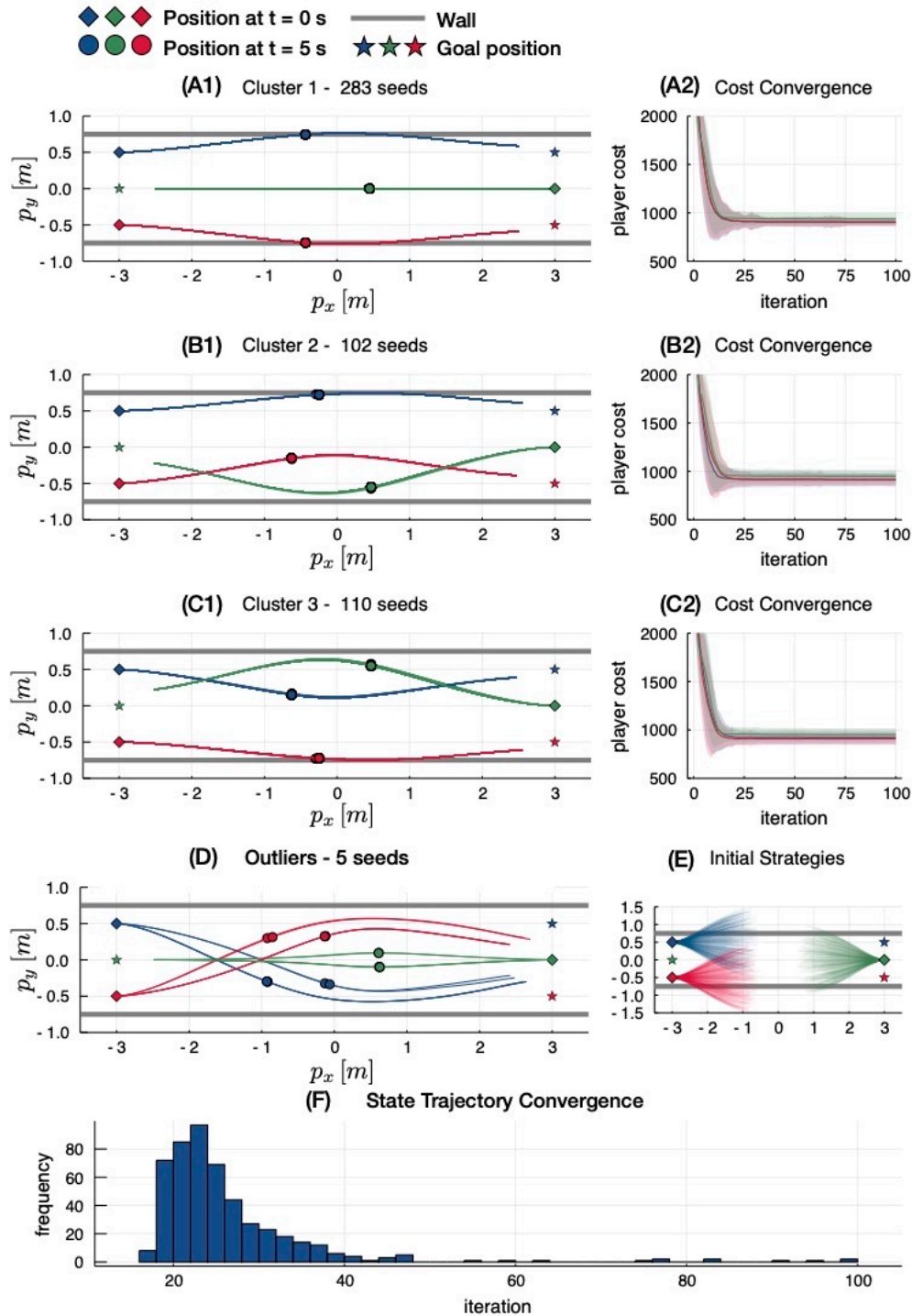
Fig. 4.1: Monte Carlo results for a three-player hallway navigation game. (A1, B1, C1) Converged trajectories clustered by total Euclidean distance; each cluster corresponds to a qualitatively distinct mode of interaction. (A2, B2, C2) Costs for each player at each solver iteration. The shaded region corresponds to one standard deviation. (D) Several converged trajectories did not match a cluster (A-C). (E) Trajectories resulting from 500 random initial strategies. (F) Histogram of iterations until state trajectory has converged.
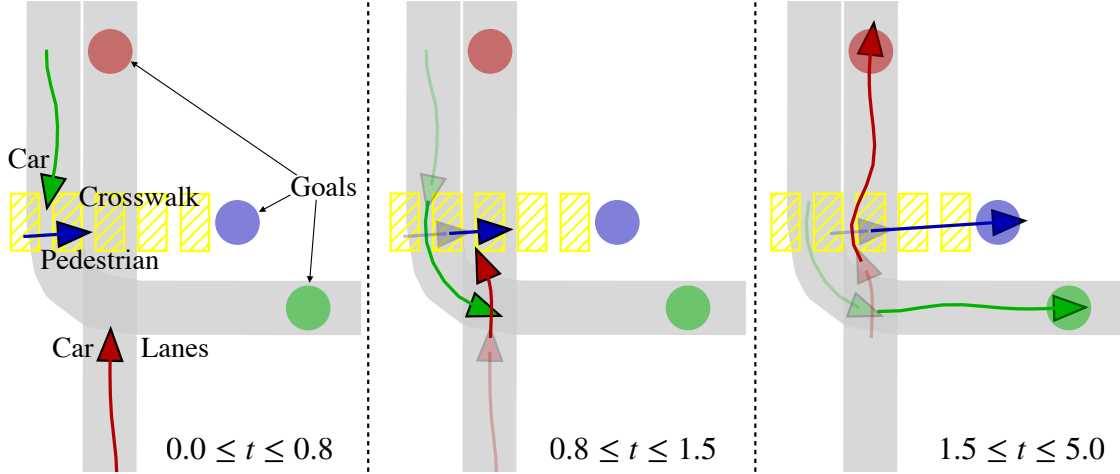
Fig. 4.2: Three-player intersection game. (Left) Green car seeks the lane center and then swerves slightly to avoid the pedestrian. (Center) Red car accelerates in front of the green car and slows slightly to allow the pedestrian to pass. (Right) Red car swerves left to give pedestrian a wide berth.

In these 500 random samples, only 6 did not converge and had to be resampled, and 5 converged to trajectories which were outliers from the clusters depicted in Figure. 4.1(A-C). These outliers are shown in Figure 4.1(D). We observe that, in these 5 cases, the players come within 0.5 m of one another.

Three-player intersection. Next, we consider a more complicated game intended to model traffic at an intersection. As shown in Fig. 4.2, we consider an intersection with two cars and one pedestrian, all of which must cross paths to reach desired goal locations. We use a time horizon of 5 s with discretization of 0.1 s, and Algorithm 2 terminates in under 0.25 s.

We model the pedestrian's dynamics as in (4.9) and each cars i's dynamics as follows:

$$\dot{p}_{x,i} = v_i \cos(\theta_i), \quad \dot{\theta}_i = v_i \tan(\phi_i)/L_i, \quad \dot{\phi}_i = \psi_i$$
$$\dot{p}_{y,i} = v_i \sin(\theta_i), \quad \dot{v}_i = a_i,$$

$$(4.14)$$

where the state variables are as before (4.9) except for front wheel angle ϕ_i. L_i is the inter-axle distance, and input $u_i := (\psi_i, a_i)$ is the front wheel angular rate and longitudinal acceleration, respectively. Together, the state of this game is 14-dimensional.

The running cost for each player i are specified as weighted sums of (4.11)–(4.13), and the following:

$$\text{lane center: } d_\ell(p_i)^2 \tag{4.15}$$
$$\text{lane boundary: } \mathbf{1}\{d_\ell(p_i) > d_{\text{lane}}\}(d_{\text{lane}} - d_\ell(p_i))^2 \tag{4.16}$$
$$\text{nominal speed: } (v_i - v_{\text{ref},i})^2 \tag{4.17}$$
$$\text{speed bounds: } \mathbf{1}\{v_i > \overline{v}_i\}(v_i - \overline{v}_i)^2$$
$$+ \mathbf{1}\{v_i < \underline{v}_i\}(\underline{v}_i - v_i)^2 \tag{4.18}$$

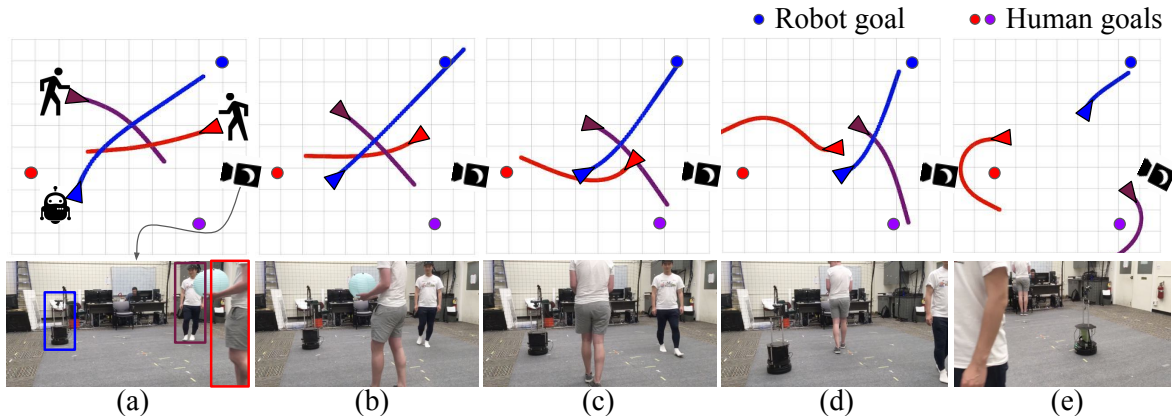**Fig. 4.3:** Time-lapse of a hardware demonstration of Algorithm 2. We model the inter-action of a ground robot (blue triangle) and two humans (purple and red triangles) using a differential game in which each agent wishes to reach a goal location while maintaining sufficient distance from other agents. Our algorithm solves receding horizon instantiations of this game in real-time, and successfully plans and executes interactive collision-avoiding maneuvers. Planned (and predicted) trajectories are shown in blue (robot), purple, and red (humans).

Here, d_{lane} denotes the lane half-width, and $d_\ell(p_i) := \min_{p_\ell \in \ell} \|p_\ell - p_i\|$ measures player i's distance to lane centerline ℓ. Speed v_i is penalized quadratically away from a fixed reference $v_{\text{ref},i}$ also outside limits $\underline{v}_i$ and $\overline{v}_i$.

Fig. 4.2 shows a time-lapse of the converged solution identified by Algorithm 2. These strategies exhibit non-trivial coordination among the players as they compete to reach their goals efficiently while sharing responsibility for collision-avoidance. Such competitive behavior would be difficult for any single agent to recover from a decoupled, optimal control formulation. Observe how, between $0 \le t \le 0.8\,\text{s}$ (left), the green car initially seeks the lane center to minimize its cost, but then turns slightly to avoid the pedestrian (blue). Between $0.8 \le t \le 1.5\,\text{s}$ (center), the red car turns right to pass in front of the green car, and then slows and begins to turn left to give the pedestrian time to cross. Finally (right), the red car turns left to give the pedestrian a wide berth.

Receding horizon motion planning. Differential games are appropriate in a variety of applications including multi-agent modeling and coordinated planning. Here we present a proof-of-concept for their use in single-agent planning in a dynamic environment. In this setting, a single robot operates amongst multiple other agents whose true objectives are unknown. The robot models these objectives and formulates the interaction as a differential game. Then, crucially, the robot re-solves the differential game along a receding time horizon to account for possible deviations between the other agents' decisions and those which result from the game solution.

We implement Algorithm 2 in C++[3] within the Robot Operating System (ROS) frame-work, and evaluate it in a real-time hardware test onboard a TurtleBot 2 ground robot, in a motion capture room with two human participants. The TurtleBot wishes to cross the room

---

[3]Code available at: `github.com/HJReachability/ilqgames`

while maintaining $> 1\,\mathrm{m}$ clearance to the humans, and it models the humans likewise. We model the TurtleBot dynamics as (4.9) and humans likewise but with constant speed v_i, i.e.:

$$\dot{p}_{x,i} = v_i \cos(\theta_i)\,, \quad \dot{p}_{y,i} = v_i \sin(\theta_i)\,, \quad \dot{\theta}_i = \omega_i\,. \tag{4.19}$$

We use a similar cost structure as in the intersection example, and initialize Algorithm 2 with all agents' strategies identically zero (i.e., $P_i^0(\cdot), \alpha_i^0(\cdot) \equiv 0$). We re-solve the game in a 10 s receding horizon with time discretization of 0.1 s, and warm-start each successive receding horizon invocation with the previous solution. Replanning every 0.25 s, Algorithm 2 reliably converges in under 50 ms. We gather state information for all agents using a motion capture system. Fig. 4.3 shows a time-lapse of a typical interaction.

Initially, in frame (a) Algorithm 2 identifies a set of strategies which steer each agent to their respective goals while maintaining a large separation. Of course, the human participants do not actually follow these precise trajectories; hence later receding horizon invocations converge to slightly different strategies. In fact, between frames (c) and (d) the red participant makes an unanticipated sharp right-hand turn, which forces the (blue) robot to stay to the right of its previous plan and then turn left in order to maintain sufficient separation between itself and both humans. We note that our assumed cost structure models all agents as wishing to avoid collision. Thus, the resulting strategies may be less conservative than those that would arise from a non-game-theoretic motion planning approach.

Full-scale demonstration. We have also tested this game-theoretic receding horizon motion planner at scale in a Boeing test aircraft at a small commercial airport. The details of this demonstration are forthcoming. However, to summarize, the same underlying C++/ROS implementation discussed above was used to process received telemetry on the aircraft and a supposed taxiway intruder, and transmit real-time receding horizon game solutions to the actuators of the aircraft at 10 Hz.

Improvements due to feedback linearizable structure

In this Section, we shall consider the situation when game dynamics are feedback linearizable. That is, we presume that the game state $x \in \mathbb{R}^n$ evolves as

$$\dot{x} = f(x) + \sum_{i=1}^{N} b_i(x) u_i\,, \tag{4.20}$$

where $u_i \in \mathbb{R}^{m_i}$ is the control input of player i. In the examples below, x will be the concatenated states of multiple subsystems, but this is not strictly necessary. We assume that (4.20) is input-output feedback linearizable with no zero dynamics, i.e. there exist outputs $y = h(x)$ such that y and finitely many of its time derivatives evolve linearly as a function of some auxiliary inputs $z_i \in \mathbb{R}^{m_i}$, for some control law $u_i := u_i(x, z_i)$.

Summary of feedback linearization. This Section provides a brief review of feedback linearization, a geometric control technique popularly used across a wide range of robotic applications including manipulation, quadrotor flight, and autonomous driving.

Recall dynamics (4.20), and define the matrix $b(x) := [b_1(x), \dots, b_N(x)] \in \mathbb{R}^{n \times m}$ and vector $u^T = [u_1^T, \dots, u_N^T] \in \mathbb{R}^m$, with $m = \sum_i m_i$ the total control dimension. Thus, (4.20)

may be rewritten as

$$\dot{x} = f(x) + b(x)u, \; y = h(x) \tag{4.21}$$

where y is the output of the system, and the functions f, b, and h are sufficiently smooth.

Suppose that (4.21) has well-defined vector relative degree $(r_1, \ldots, r_m)$ [40, Definition 9.15] and is full-state feedback linearizable. Then, there exists a matrix $M(x)$ and vector $m(x)$ such that the time derivatives of the outputs y follow

$$[y_1^{(r_1)}, \ldots, y_m^{(r_m)}]^T = M(x)u + m(x). \tag{4.22}$$

Presuming the invertibility of the so-called "decoupling matrix" $M(x)$, we may design the following feedback linearizing control law as a function of both state x and an *auxiliary input*

$$u(x, z) = M^{-1}(x)(z - m(x)), \tag{4.23}$$

which renders the input-output dynamics linear in the new auxiliary inputs z:

$$[y_1^{(r_1)}, \ldots, y_m^{(r_m)}]^T = z. \tag{4.24}$$

Note that, as for u in (4.20) we shall consider $z^T = [z_1^T, \ldots, z_N^T]$ to be a concatenation of auxiliary inputs for each player, with $z_i \in \mathbb{R}^{m_i}$.

We have seen how a careful choice of feedback linearizing controller $u(x, z)$ renders the dynamics of the output y and its derivatives linear. Define the state of this linear system as $\chi := [y_1, \ldots, y_1^{(r_1-1)}, \ldots, y_m, \ldots, y_m^{(r_m-1)}]^T$. Just as there is a bijective map (4.23) between control u and auxiliary input z whenever $M(x)$ is invertible, there is also a bijection between state x and linear system state χ, $x = \lambda(\chi)$ [40] because (4.21) is full-state feedback linearizable. We shall use both bijective maps (and their derivatives) below to rewrite costs (4.2) in terms of the linearized dynamics (4.24).

Taking Advantage of Feedback Linearization Consider the following example of a (single player) 4D unicycle dynamical model:

$$\dot{x} = \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ w \\ a \end{bmatrix}, \; y = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{4.25}$$

representing the evolution of the positions p_x and p_y, the orientation θ, and speed v. The inputs w and a represent the angular rate and the acceleration. By taking time derivatives of the output y following the procedure above, we obtain the new set of states $\chi = [p_x, \dot{p}_x, p_y, \dot{p}_y]^T$ for the linearized system. Differentiation reveals that

$$\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \end{bmatrix} = \begin{bmatrix} -v\sin\theta & \cos\theta \\ v\cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} w \\ a \end{bmatrix}. \tag{4.26}$$

From this result, we compute the inverse decoupling matrix and drift term as

$$M^{-1}(x) = \begin{bmatrix} -\sin\theta/v & \cos\theta/v \\ \cos\theta & \sin\theta \end{bmatrix}, \; m(x) = 0. \tag{4.27}$$

Algorithm 3: Feedback Linearized Iterative LQ Games

Input: initial linearized system state $\chi(0)$ and control strategies $\{\gamma_i^0\}_{i \in [N]}$, time horizon T

Output: converged control strategies $\{\gamma_i^*\}_{i \in [N]}$ for the linearized system

1 for *iteration* $p = 1, 2, \ldots$ **do**

2 $\xi^p \equiv \{\hat{\chi}(t), \hat{z}_i(t)\}_{i \in [N], t \in [0,T]} \leftarrow$

3   getTrajectory$\left(\chi(0), \{\gamma_i^{p-1}\}\right)$;

4 $\{l_i(t), Q_i(t), R_{ij}(t)\} \leftarrow$ quadraticizeCost$\left(\xi^p\right)$;

5 $\{\tilde{\gamma}_i^p\} \leftarrow$ solveLQGame$\left(\{l_i(t), Q_i(t), R_{ij}(t)\}\right)$;

6 $\{\gamma_i^p\} \leftarrow$ stepToward$\left(\{\gamma_i^{p-1}, \tilde{\gamma}_i^p\}\right)$;

7   **if** *converged* **then**

8 $\lfloor$ **return** $\{\gamma_i^p\}$

Finally, we can also explicitly derive the state conversion map $\lambda(\chi)$

$$\lambda(\chi) = \begin{bmatrix} p_x \\ p_y \\ \sqrt{\dot{p}_x^2 + \dot{p}_y^2} \\ \tan^{-1}\left(\frac{\dot{p}_y}{\dot{p}_x}\right) \end{bmatrix}. \tag{4.28}$$

Now, consider a differential game with two players, each of whom independently follows dynamics (4.25). The inverse decoupling matrix $M^{-1}(x)$ and the Jacobian of the state conversion map λ for the full system will be block diagonal.

Transforming costs. So far, we have introduced feedback linearization and shown how to derive the mappings from auxiliary input z to control u and linearized system state χ to state x. To exploit the feedback linearizable structure of (4.21) when solving the game, we must rewrite running costs $g_i(t, x, u_1, \ldots, u_N)$ in terms of χ and z. Overloading notation, we shall denote the transformed running costs as

$$\begin{aligned} g_i(t; \chi; z_1; \ldots; z_N) &\equiv \\ g_i\Big(t; \lambda(\chi); u_1\big(\lambda(\chi), z_1\big); \ldots; u_N\big(\lambda(\chi), z_N\big)\Big), \end{aligned} \tag{4.29}$$

where $u_i(\lambda(\chi), z_i)$ is given in (4.23).

Algorithm 3 presents our main algorithm; a core step will be to compute first and second derivatives of each player's running cost with respect to the new state χ and inputs z_i. This may be done efficiently using the chain rule and exploiting known sparsity patterns for particular systems and costs. For completeness, however, we shall ignore sparsity and illustrate computing the first derivative of g_i with respect to the jth dimension of χ, denoted χ_j:

$$\frac{\partial g_i}{\partial \chi_j} = \sum_{p=1}^{n} \frac{\partial g_i}{\partial x_p} \frac{\partial x_p}{\partial \chi_j} + \sum_{n=1}^{N} \sum_{p=1}^{m_n} \frac{\partial g_i}{\partial u_{n,p}} \sum_{q=1}^{n} \frac{\partial u_{n,p}}{\partial x_q} \frac{\partial x_q}{\partial \chi_j} \tag{4.30}$$

where $u_{n,p}$ is the pth entry of the nth player's control input.

Second derivatives may be computed similarly, though again we stress that for specific dynamics and cost functions it is often much more efficient to exploit the *a priori* known sparsity of partial derivatives. Interestingly, we also observe that the terms arising from the second sum in (4.30), which account for the state-dependence of the feedback linearizing controllers (4.23), are often negligible in practice and may be dropped without significant impact on solution quality.

Core algorithm. Like the original iterative LQ game algorithm, we proceed from a set of initial strategies γ_i for each player—understood now to map from (t, χ) to z_i—and iteratively refine them by solving LQ approximations. Our main contribution, therefore, lies in the transformation of the game itself into the coordinates χ, z_i which correspond to feedback linearized dynamics. As we shall see in the results below, iterative LQ approximations are much more stable in the transformed coordinates and converge at least as quickly.

Algorithm 3 outlines the major steps in the resulting algorithm. We begin at the given initial condition $\chi(0)$ for the linearized system and strategies γ_i^0 for each player. Note that these strategies define control laws *for the linearized system*, i.e. $z_i(t) \equiv \gamma_i\big(t, \chi(t)\big)$.

At each iteration, we first (Algorithm 3, line 3) integrate the linearized dynamics (4.24) forward to obtain the current operating point $(\hat{\chi}(\cdot), \{\hat{z}_i(\cdot)\})$. Then (Algorithm 3, line 4), we compute a quadratic approximation to each player's running cost in terms of the variations $\delta\chi := \chi - \hat{\chi}$ and $\delta z_j := z_j - \hat{z}_j$

$$g_i(t; \chi; z_1; \ldots; z_N) - g_i(t; \hat{\chi}; \hat{z}_1; \ldots; \hat{z}_N) \approx \delta\chi^T l_i(t) +$$
$$\frac{1}{2}\delta\chi^T Q_i(t)\delta\chi + \frac{1}{2}\sum_{j=1}^{N} \delta z_j^T (R_{ij}(t)\delta z_j + 2r_{ij}(t)), \tag{4.31}$$

using the chain rule as in (4.30) to compute the terms l_i, Q_i and R_{ij} for each player.

Equipped with linear dynamics (4.24) and quadratic costs (4.31), the solution of the resulting general-sum LQ game is given by a set of coupled Riccati differential equations, which may be derived from the first order necessary conditions of optimality for each player [3, Chapter 6]. In practice (Algorithm 3, line 5), we numerically solve these equations in discrete-time using a time step of Δt. If a solution exists at the pth iteration, it is known to take the form

$$\tilde{\gamma}_i^p(t, \chi) \equiv \hat{z}_i(t) - P_i^p(t)\big(\chi(t) - \hat{\chi}(t)\big) - \alpha_i^p(t) \tag{4.32}$$

for matrix $P_i^p(t)$ and vector $\alpha_i^p(t)$ [3, Corollary 6.1].

We cannot simply use these strategies at the $(p+1)^{\text{th}}$ iteration or we risk diverging, however, without further assumptions on the curvature and convexity of running costs g_i. In fact, these costs are generally nonconvex when expressed in terms of χ and z_j (4.29), which necessitates some care in updating strategies. To address this issue (Algorithm 3, line 6), we follow a common practice in the ILQR and sequential quadratic programming literature (e.g., [48]) and introduce a step size parameter $\eta \in (0, 1]$:

$$\gamma_i^p(t, \chi) = \hat{z}_i(t) - P_i^p(t)\big(\chi(t) - \hat{\chi}(t)\big) - \eta\alpha_i^p(t). \tag{4.33}$$

Observe that, taking $\eta = 0$ and recalling that $\chi(0) = \hat{\chi}(0)$, we recover the previous open-loop control signal $\gamma_i^p(t, \chi) = \hat{z}_i, \forall t \in [0, T]$. Taking $\eta = 1$, we recover the LQ solution from this iteration (4.32). As is common in the literature, we perform a backtracking linesearch on η, starting with initial value η_0 and terminating when the trajectory that results from (4.33) satisfies a trust region constraint at level ϵ. In our experiments, we use an L_∞ constraint, i.e.

$$\|\chi(t) - \hat{\chi}(t)\|_\infty < \epsilon, \forall t, \tag{4.34}$$

and check that M^{-1} exists at each time.

Effect of feedback linearization. In comparison to the non-feedback linearized case of Section 4.3, the linearized dynamics (4.24) are independent of trajectory (and hence also of iteration). That is, in such cases [14], each iteration begins by constructing a Jacobian linearization of dynamics (4.20); this is superfluous in our case. As a consequence, large changes in auxiliary input z between iterations—which lead to large changes in state trajectory—are trivially consistent with the feedback linearized dynamics (4.24). By contrast, a large change in control u may take the nonlinear dynamics (4.20) far away from the previous Jacobian linearization, which causes the algorithm from [14] to be fairly sensitive to step size η and trust region size ϵ. We study this sensitivity more carefully in the results below.

Finally, it is important to note that while many systems of interest (e.g., manipulators, cars, and quadrotors) are feedback linearizable, this is not true of all systems. Additionally, there are two drawbacks of our algorithm that deserve mention. First, we must take care to avoid regions in which M^{-1} does not exist. We accomplish this by designing costs that penalize proximity to singularities. While this can potentially limit the range of behaviors, many motion problems naturally incorporate these costs. Second, the transformed costs $g_i(t; \chi; \dots)$ may have much more varied, extreme curvature than the original costs $g_i(t, x, \dots)$. In some cases, this can make Algorithm 3 sensitive to linesearch parameters η_0 and ϵ, even offsetting the benefits mentioned above. We defer further discussion and empirical study to the results below.

Results. To showcase the benefits of our feedback linearization-based approach, we study the empirical sensitivity of solutions to the initial step size η_0 and trust region size ϵ hyperparameters. We shall consider a three-player intersection example and compare the strategies identified by Algorithm 3 with those identified on the original dynamics, using the algorithm from [14]. Here, two cars, modeled with bicycle dynamics

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cos\theta \\ v \sin\theta \\ (v/L)\tan\phi \end{bmatrix}, \quad \begin{bmatrix} \dot{v} \\ \dot{\phi} \\ \dot{a} \end{bmatrix} = \begin{bmatrix} a \\ \omega \\ \kappa \end{bmatrix} \tag{4.35}$$

(with inter-axle distance L and inputs ω controlling front wheel rate $\dot{\phi}$ and κ controlling jerk), and a pedestrian modeled with dynamics (4.25) navigate an intersection. Like (4.25), bicycle dynamics (4.35) are feedback linearizable in the outputs (p_x, p_y). We place quadratic penalties on each player's distance from the appropriate lane center and from a fixed goal
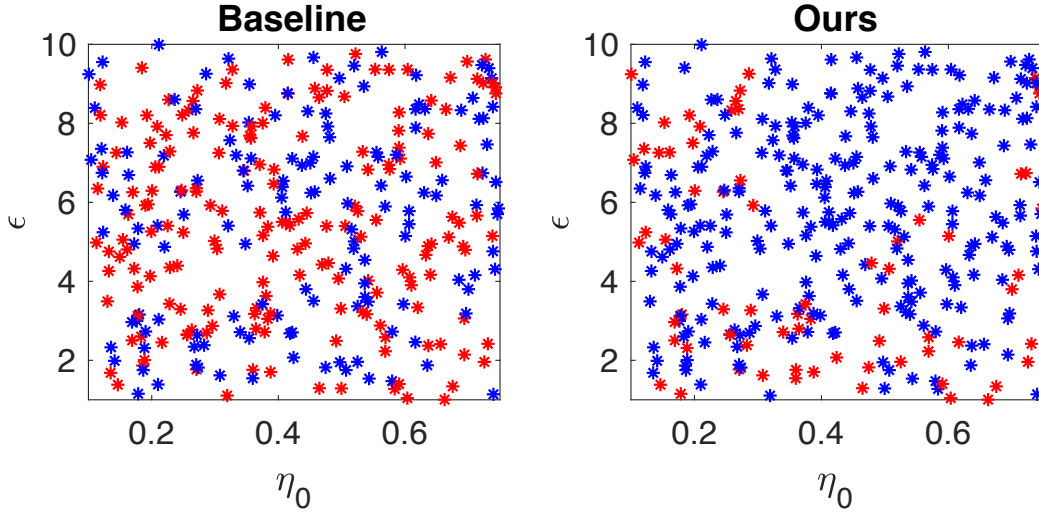
Fig. 4.4: Distribution of pairs (η_0, ϵ) colored by quality metric q. Pairs with low q are colored blue, and high q pairs are colored red.

location, as well as on the difference between speed v and a fixed nominal speed $\bar{v}$. Players are also penalized quadratically within a fixed distance of one another [4].

In order to assess the quality of a trajectory $\xi = (\chi, z_1, \ldots, z_N)$ generated by a particular algorithm, we define the *similarity metric* to the desired trajectory to be:

$$q(\xi, \tilde{\xi}) := \max_{t \in [0,T]} \|\chi(t) - \tilde{\chi}(t)\|_{2,(p_x,p_y)}. \tag{4.36}$$

Here, we take $\tilde{\xi} := (\tilde{\chi}, \tilde{z}_1, \ldots, \tilde{z}_N)$ to be the equilibrium trajectory which that algorithm ideally converges to. The norm measures Euclidean distance only in the (p_x, p_y) dimensions. Trajectories that diverge or converge to unreasonable solutions yield high values for q, while trajectories that closely match $\tilde{\xi}$ incur low values.

We fix the initial conditions and cost weights identically for both algorithms. Thus, any trajectory ξ identified by the solver will solely be a function of the initial step size η_0 and trust region size ϵ. Therefore, we will overload the penalty metric notation as $q(\eta_0; \epsilon)$. Given this metric we study the quality of solutions over the ranges $\eta_0 \in [0.1, 0.75]$ and $\epsilon \in [1.0, 10.0]$, and test 324 uniformly sampled (η_0, ϵ) pairs.

Fig. 4.4 displays the sampled pairs over the space of η_0 and ϵ. For clarity, we set a *success threshold* q^* and color "successful" pairs with $q(\eta_0; \epsilon) \leq q^*$ blue, and "unsuccessful" pairs red. Fig. 4.5 shows histograms of solution quality q for each algorithm, with a horizontal line denoting threshold q^*. We observe that solving the game using feedback linearization converges much more reliably than solving it for the original nonlinear system. Moreover, for converged trajectories with low q-value, the average computation time was 0.3982 ± 0.3122 s (mean $\pm$ standard deviation) for our method and 0.8744 ± 0.9582 s for the baseline.

Unfortunately, these results do not generalize to all games. As discussed above, in some cases the cost landscape gets much more complicated when expressed in linearized system

---

[4]For details concerning weighing of different cost terms we refer the reader to our github repository at https://github.com/HJReachability/ilqgames
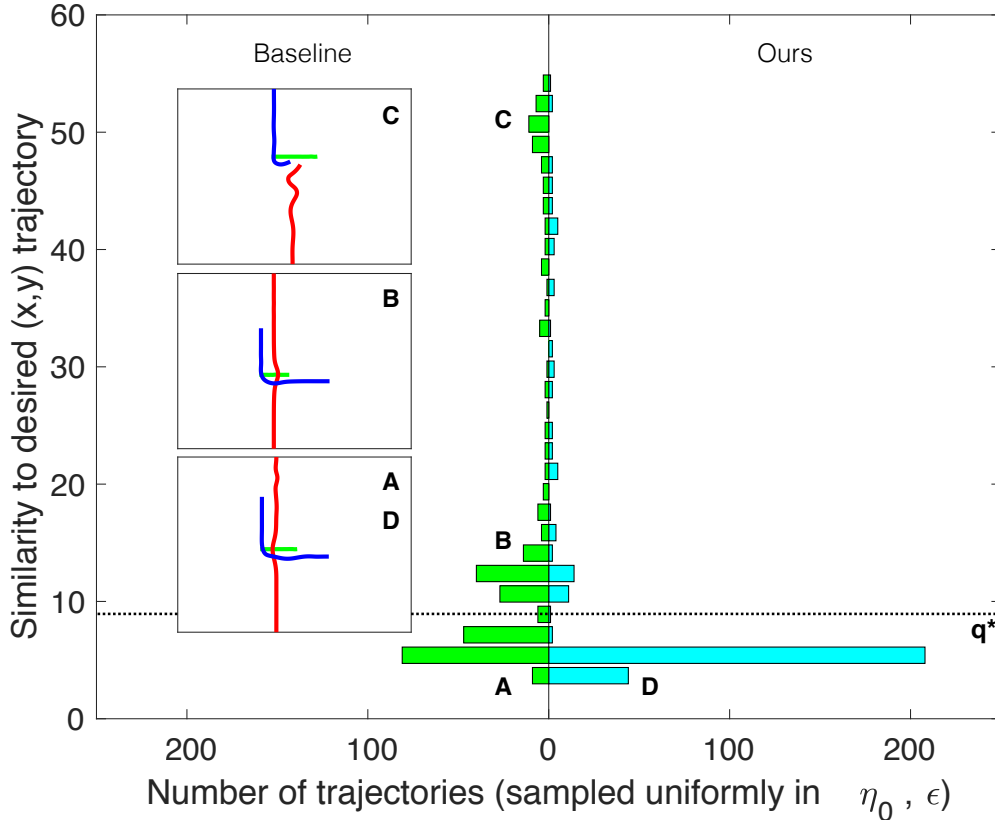
Fig. 4.5: Comparison of the proposed algorithm with the state of the art [14] for a three player intersection game. Histograms (left, baseline; right, ours) show that our method is much more numerically stable and converges more frequently. Insets labelled {A, B, C, D} show a typical trajectory for the associated bin. The dotted horizontal line shows threshold q^*, used to distinguish samples from Fig. 4.4.

coordinates χ, z_i. For example, a simple quadratic penalty on a single player's speed difference from nominal $\bar{v}$ in (4.25) is nonconvex and non-smooth near the origin when expressed as a function of linearized system state χ:

$$(v - \bar{v})^2 \iff \left(\bar{v} - \sqrt{\dot{p}_x^2 + \dot{p}_y^2}\right)^2. \tag{4.37}$$

Consequences vary; the effect is negligible in the intersection example from Fig. 4.5, but it is more significant in the roundabout example below, where cars must slow down before turning into the roundabout.

Fortunately, in practical settings of interest it is typically straightforward to design smooth, semantically equivalent costs explicitly as functions of the linearized system coordinates χ. For example, we can replace the nominal speed cost of (4.37) with a time-varying quadratic penalty in that player's position (p_x, p_y):

$$(v - \bar{v})^2 \implies \left(p_x(t) - \bar{p}_x(t)\right)^2 + \left(p_y(t) - \bar{p}_y(t)\right)^2, \tag{4.38}$$

where $\left(\bar{p}_x(\cdot), \bar{p}_y(\cdot)\right)$ defines the point on the lane center a distance $\bar{v}t$ from the initial condition.
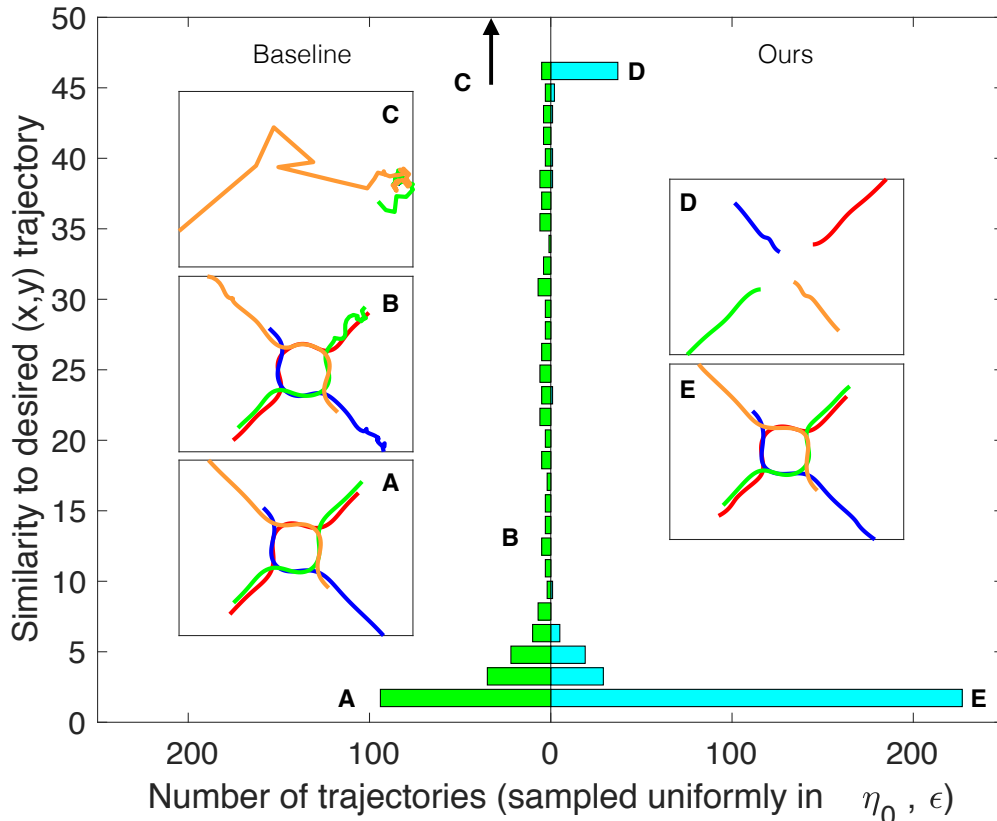
Fig. 4.6: Relative performance obtained for a roundabout from using the structure of feedback linearization.

We demonstrate the effectiveness of this substitution in two examples—merging into a roundabout, and overtaking a lead vehicle—in which the original cost (4.37) led to instability in Algorithm 3. In both cases, we also use simple quadratic penalties for z_i (rather than transforming $\|u_i\|^2$ into linearized coordinates), albeit with different weightings. Results for the roundabout merging and overtaking examples are shown in Figures 4.6 and 4.7, respectively. From the 324 samples in each (drawn from expanded ranges $\eta_0 \in [0.1, 1.0], \epsilon \in [1, 50]$), we see that Algorithm 3 converged more frequently than the method of [14]. Moreover, when successful, the average computational time in the roundabout example was $0.2797 \pm 0.1274\,\mathrm{s}$ for our method and $0.4244 \pm 0.5259\,\mathrm{s}$ for the baseline. Runtimes for the overtaking example were $0.5112 \pm 0.3228\,\mathrm{s}$ (ours) and $0.4417 \pm 0.4142\,\mathrm{s}$ (baseline). Observe how runtimes for our approach cluster more tightly around the mean, indicating a more reliable convergence rate.

Discussion

We have presented a novel algorithm for finding local solutions in multi-player general-sum differential games. Our approach is closely related to the iterative linear-quadratic regulator (ILQR) [26], and offers a straightforward way for optimal control practitioners to directly account for multi-agent interactions via differential games. We performed a Monte Carlo study which demonstrated the reliability of our algorithm and its ability to identify
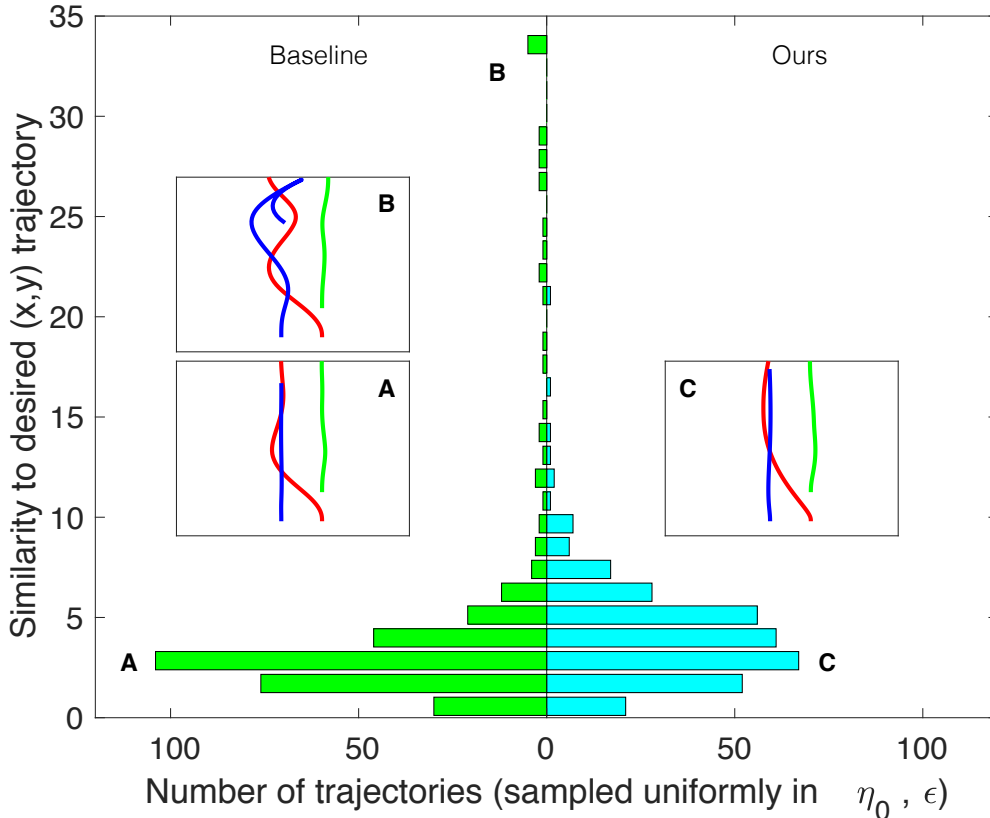
Fig. 4.7: Comparison for a three vehicle high speed overtaking maneuver.

complex interactive strategies for multiple agents. These solutions display the competitive behavior associated with local Nash equilibria, although there are subtle differences. We also showcased our method in a three-player 14-dimensional traffic example, and tested it in real-time operation in a hardware robot navigation scenario, following a receding time horizon and in a full-scale demonstration on an airplane. Additionally, we have presented a further adaptation of our general approach to situations in which the underlying differential game has feedback linearizable dynamics. Here, we showed that these games could sometimes be solved more quickly and reliably by using this additional structure.

There are several other approaches to identifying local solutions in differential games, such as iterative best response [52]. We have shown the computational efficiency of our approach. However, quantitatively comparing the solutions identified by different algorithms is challenging due to differences in equilibrium concept, information structure (feedback vs. open loop), and implementation details. Furthermore, in arbitrary general-sum games, different players may prefer different equilibria. Studying the qualitative differences in these equilibria is an important direction of future research.

Although our experiments show that our algorithm converges reliably, we have no *a priori* theoretical guarantee of convergence from arbitrary initializations. Future work will seek a theoretical explanation of this empirical property. Improving performance and addressing convergence are important directions of future research. We also intend to investigate inequality-constrained differential games; here, we believe that interior point methods may

present a promising direction. Finally, it will be critical to develop a theory for online estimation of other players' objectives, and for understanding the sensitivity of local solutions to misspecified objectives and sub-optimal play.

Chapter 5

Conclusion

5.1 What's missing

This dissertation has presented several *practical* algorithms for building reliable autonomous systems. Chapter 2 presented a method for obtaining adversarial disturbance-rejection in real-time motion planning problems. Chapter 3 investigated multi-agent situations, in which the uncertainty came from the motion of other agents rather than from internal system dynamics, and presented a probabilistic online method for adapting a measure of confidence one should have in a predictive utility model. Chapter 4 brought the two preceding chapters together in some sense. Here, the focus was on coupling prediction to planning and this resulted in a differential game, and the Chapter presented a novel solution strategy that was amenable to real-time operation.

Moving forward, of course there are many interesting directions for future research and Section 5.2 will discuss a few of the most immediate. However, the remainder of this Section will attempt to enumerate a more complete (though certainly only partial) list of problems. For simplicity, however, each problem shall not attempt a full description of any related work and will only state the underlying problem as it pertains to what has been described previously.

Unknown objectives in differential games

A key assumption in Chapter 4 was that we knew the costs for all players *a priori*. Yet, for many applications of practical interest such as traffic, this is not necessarily the case. We need, therefore, a method for estimating each player's cost in real-time, given that they are acting according to a solution of the game, e.g., a Nash equilibrium. Although this seems extremely difficult and ill-posed, we do have some hope in some cases. Namely, in the traffic problems considered in the examples in Chapter 4 and which are broadly descriptive of the problems faced by autonomous car companies, there may be very good priors. For example, we can reasonably guess that a reasonable driver wishes to do stay in his or her lane, make forward progress, not collide, etc., and these wishes are easily encoded in a set of weighted costs of the sort in Chapter 4. Therefore, the problem may reduce to learning a set of weights on a small set of known functions, which is likely much easier to do.

Partial observability in game dynamics

So far, we have also presumed that strategies have *full state feedback.* This is particularly important as a distinction from the open loop information pattern which is common in model predictive control and for which there is also a closed-form LQ game solution [3, Chapter 6]. However, full state observability is a strong assumption that may not hold in some cases, such as long-occluded regions at intersections which may or may not contain one or more cars. Unfortunately, it is theoretically incredibly challenging to handle partial state observability in differential games; yet, it is certainly a topic of practical interest.

Randomness in differential games

We have presented coordinated prediction and planning as though they equate with a differential game; however, to fully account for uncertainty in human drivers' behavior it is almost certain that we need to consider *stochastic* differential games. These have certainly been studied, and the interested reader is directed to [3], however, a significant amount of work is required to model the uncertain dynamics of certain, e.g, traffic games, and devise a real-time solution strategy.

Team effects

In many situations which may be understood as differential games, natural solutions which arise in practice involve multiple players collaborating. It is not clear if these coalitions are well-modeled by a greedy Nash equilibrium, or if it is very important to account for the coalition's coordinated decision-making explicitly. A deeper investigation is certainly warranted.

Equilibrium accuracy

This issue is really two separate problems: first, it is important to understand what exactly the iterative LQ method of Chapter 4 finds at equilibrium and how it relates to a Nash equilibrium; second, it is not at all clear how closely the notion of a Nash equilibrium models the behavior of real people. The first problem is readily apparent from the brief discussion of Section 10. The second, however, is both empirical and theoretical. It will be important both to measure how closely peoples' behavior in differential games qualitatively matches greedy Nash behavior (vs. other notions of equilibrium such as Stackelberg or Conjectural Variations) and how closely it quantitatively matches (e.g., how close to people come to equilibrium behavior even though they are not necessarily computing it precisely).

Equilibrium alignment

In all kinds of games, including differential games, there are generally multiple Nash equilibria (not including the presence of local equilibria). For example, [30, 31] not only show the presence of such equilibria, but also show that policy gradient algorithms do not generally converge to such equilibria. In the traffic games of Chapter 4, these equilibria correspond

to qualitatively different sets of strategies, e.g., at an intersection. It will be extremely important to devise a real-time method for an autonomous system to *align* itself to the choice of equilibrium one or more other (human) agents are choosing. Moreover, it is not necessarily clear that other agents will play an equilibrium strategy; indeed, any theoretical assumption in that regard must be validated empirically.

Accounting for a receding horizon in differential games

In practice, almost all practical motion planning applications solve the underlying problem in a *receding time horizon*, of the type described in Chapter 4. The community has devoted significant attention to recursive feasibility issues in model predictive control, which also follows a receding horizon; still, it will be important to construct costs for differential games that afford some notion or robustness in a receding time horizon.

Non-adversarial disturbances

Many systems of practical interest afford relatively simple, often physical, models; yet, these models do not capture the precise state evolution of the real system due to other effects. In these cases, the mismatch of the model is not truly adversarial, and it is not necessarily best to study it as such. In Chapter 2, we presumed that this model mismatch could be captured by presuming the presence of a bounded adversarial disturbance injected by "nature." This may be a reasonable assumption for the case of a quadrotor undergoing variable gusts of wind, but it is probably not the best assumption in all cases, e.g., a Baxter robot with internal joint springs and modeled using classic open-loop manipulator chain dynamics. It will be important to address these non-adversarial model mismatch issues in the future, and thereby facilitate real-time operation of complex systems using simple but known and incorrect models.

5.2   What's next

In this Section, I shall briefly describe progress on two of these problems; the remainder of these many issues I happily leave for future work.

Progress on equilibrium alignment

In the recent paper [35], we present a novel method for inferring the equilibrium other players are operating at in a differential game. We use the method of Chapter 4 to identify equilibria of a game, and a particle filter with random equilibria (generated by seeding each player with a random strategy and finding the corresponding equilibrium identified by the solver). The particle filter uses a straightforward noise model to process observations of each player's state and re-weights each particle according to how likely it is to have generated the observed data. The ego vehicle then executes its strategy for the most likely equilibrium at each time step.

Going forward, additional work on this problem will be very important. For example, future work should also address situations in which the other agents do not all agree on a

single equilibrium of the game. Additionally, future work ought not to assume that all agents operate at precisely the equilibria (or necessarily that they agree on the *type* of equilibrium), not to mention the difficult issue of whether other agents will even operate at a (possibly) local equilibrium at all.

Learning to correct non-adversarial disturbances

The recent papers [53, 54] develop a new technique for using model-based control techniques such as feedback linearization in contexts with unknown model mismatch. Here, we develop a method which uses a model-free policy gradient reinforcement learning algorithm for modifying a nominal controller both offline and online. Our procedure has theoretical guarantees of stability and convergence only in the restrictive case in which the learned modification to an existing controller is *linearly parameterized*; in practice, however, we find that it can work well in hardware with learned neural network controllers.

In the future, this general approach for treating model mismatch is extremely promising. That is, a paradigm which will emerge in the learning and control literature is that analytical models can provide "mostly correct" control schemes and the remaining mismatch (e.g., in the behavior of the *a priori* designed controller) may be "fixed" with learning techniques. These model-based control schemes are often more straightforward to design and analyze than learned counterparts; yet by learning the remainder we can hope to correct the mismatch that is inevitably present.

Bibliography

[1]   Chris L Baker, Joshua B Tenenbaum, and Rebecca R Saxe. "Goal inference as inverse planning". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 29. 2007.

[2]   Somil Bansal et al. "Hamilton-Jacobi reachability: A brief overview and recent advances". In: *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE. 2017, pp. 2242–2253.

[3]   Tamer Başar and Geert Jan Olsder. *Dynamic noncooperative game theory*. SIAM, 1999.

[4]   Felix Berkenkamp et al. "Safe model-based reinforcement learning with stability guarantees". In: *Advances in Neural Information Processing Systems*. 2017, pp. 908–918.

[5]   Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[6]   Giuseppe C Calafiore and Laurent El Ghaoui. *Optimization models*. Cambridge university press, 2014.

[7]   Mo Chen et al. "Decomposition of reachable sets and tubes for a class of nonlinear systems". In: *IEEE Transactions on Automatic Control* (2018).

[8]   L. C. Evans and P. E. Souganidis. "Differential games and representation formulas for solutions of Hamilton-Jacobi-Isaacs equations". In: *Indiana University mathematics journal* 33.5 (1984), pp. 773–797.

[9]   Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided cost learning: Deep inverse optimal control via policy optimization". In: *International Conference on Machine Learning*. 2016, pp. 49–58.

[10]   Jaime F. Fisac et al. "Probabilistically Safe Robot Planning with Confidence-Based Human Predictions". In: *Robotics: Science and Systems*. 2018.

[11]   David Fridovich-Keil, Jaime F Fisac, and Claire J Tomlin. "Safely probabilistically complete real-time planning and exploration in unknown environments". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7470–7476.

[12]   David Fridovich-Keil, Vicenç Rubies-Royo, and Claire J Tomlin. "An Iterative Quadratic Method for General-Sum Differential Games with Feedback Linearizable Dynamics". In: *arXiv preprint arXiv:1910.00681* (2019).

[13] David Fridovich-Keil et al. "Confidence-aware motion prediction for real-time collision avoidance". In: *The International Journal of Robotics Research* 39.2-3 (2019), pp. 250–265.

[14] David Fridovich-Keil et al. "Efficient Iterative Linear-Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games". In: *arXiv preprint arXiv:1909.04694* (2019).

[15] David Fridovich-Keil et al. "Planning, Fast and Slow: A Framework for Adaptive Real-Time Safe Trajectory Planning." In: *IEEE Conference on Robotics and Automation* (2018).

[16] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 3067–3074.

[17] Michael Green and David JN Limebeer. *Linear robust control*. Courier Corporation, 2012.

[18] Sylvia L. Herbert et al. "FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning". In: *IEEE Conference on Decision and Control*. 2017.

[19] Rufus Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1999.

[20] Boris Ivanovic and Marco Pavone. "The Trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 2375–2384.

[21] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[22] Sertac Karaman and Emilio Frazzoli. "Sampling-based optimal motion planning for non-holonomic dynamical systems". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 5041–5047.

[23] Alexander B Kurzhanski and Pravin Varaiya. "Ellipsoidal techniques for reachability analysis: internal approximation". In: *Systems & control letters* 41.3 (2000), pp. 201–211.

[24] Alexander B Kurzhanski and Pravin Varaiya. "On ellipsoidal techniques for reachability analysis. part ii: Internal approximations box-valued constraints". In: *Optimization methods and software* 17.2 (2002), pp. 207–237.

[25] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[26] Weiwei Li and Emanuel Todorov. "Iterative linear quadratic regulator design for nonlinear biological movement systems." In: *ICINCO*. 2004, pp. 222–229.

[27] Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2011.

[28] R. Duncan Luce. *Individual choice behavior: A theoretical analysis*. Wiley, 1959.

[29] Eric V Mazumdar, Michael I Jordan, and S Shankar Sastry. "On finding local nash equilibria (and only local nash equilibria) in zero-sum games". In: *arXiv preprint arXiv:1901.00838* (2019).

[30] Eric Mazumdar and Lillian J Ratliff. "On the convergence of gradient-based learning in continuous games". In: *arXiv preprint arXiv:1804.05464* (2018).

[31] Eric Mazumdar et al. "Policy-Gradient Algorithms Have No Guarantees of Convergence in Linear Quadratic Games". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 2020, pp. 860–868.

[32] Ian Mitchell. *A Toolbox of Level Set Methods*. `http://people.cs.ubc.ca/~mitchell/ToolboxLS/index.html`. 2009.

[33] Ian M. Mitchell, A. M. Bayen, and C. J. Tomlin. "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games". In: *IEEE Transactions on Automatic Control* 50.7 (2005), pp. 947–957. DOI: `10.1109/TAC.2005.851439`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463302`.

[34] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games". In: *IEEE Transactions on automatic control* 50.7 (2005), pp. 947–957.

[35] Lasse Peters et al. "Inference-Based Strategy Alignment for General-Sum Differential Games". In: *arXiv preprint arXiv:2002.04354* (2020).

[36] Morgan Quigley et al. "ROS: an Open-Source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.

[37] Lillian J Ratliff, Samuel A Burden, and S Shankar Sastry. "On the characterization of local Nash equilibria in continuous games". In: *Transactions on Automatic Control* 61.8 (2016), pp. 2301–2307.

[38] Vicenç Rubies-Royo et al. "A Classification-based Approach for Approximate Reachability". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7697–7704.

[39] Dorsa Sadigh et al. "Planning for Autonomous Cars that Leverage Effects on Human Actions". In: *Robotics: Science and Systems (RSS)* (2016).

[40] Shankar Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer, 1999.

[41] Edward Schmerling et al. "Multimodal Probabilistic Model-Based Planning for Human-Robot Interaction". In: *arXiv preprint arXiv:1710.09483* (2017).

[42] Alan Wilbor Starr and Yu-Chi Ho. "Nonzero-sum differential games". In: *Journal of optimization theory and applications* 3.3 (1969), pp. 184–206.

[43] AW Starr and Yu-Chi Ho. "Further properties of nonzero-sum differential games". In: *Journal of Optimization Theory and Applications* 3.4 (1969), pp. 207–219.

[44] Ioan A Şucan and Lydia E Kavraki. "Kinodynamic motion planning by interior-exterior cell exploration". In: *Algorithmic Foundation of Robotics VIII*. Springer, 2009, pp. 449–464.

[45]   Ioan A Şucan, Mark Moll, and Lydia E Kavraki. "The open motion planning library". In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82.

[46]   Zachary N Sunberg and Mykel J Kochenderfer. "Online algorithms for POMDPs with continuous state, action, and observation spaces". In: *Twenty-Eighth International Conference on Automated Planning and Scheduling*. 2018.

[47]   Akio Tanikawa, Hiro Mukai, and Min Xu. "Local Convergence of the Sequential Quadratic Method for Differential Games". In: *Transactions of the Institute of Systems, Control and Information Engineers* 25.12 (2012), pp. 349–357.

[48]   Yuval Tassa, Nicolas Mansard, and Emo Todorov. "Control-limited differential dynamic programming". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1168–1175.

[49]   Peter Trautman and Andreas Krause. "Unfreezing the robot: Navigation in dense, interacting crowds". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 797–803.

[50]   John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press Princeton, NJ, 1945.

[51]   Mingyu Wang et al. "Game Theoretic Planning for Self-Driving Cars in Competitive Scenarios". In: *Robotics: Science & Systems*. 2019.

[52]   Zijian Wang, Riccardo Spica, and Mac Schwager. "Game Theoretic Motion Planning for Multi-robot Racing". In: *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 225–238.

[53]   Tyler Westenbroek et al. "Feedback Linearization for Unknown Systems via Reinforcement Learning". In: *arXiv preprint arXiv:1910.13272* (2019).

[54]   Tyler Westenbroek et al. "Technical Report: Adaptive Control for Linearizable Systems Using On-Policy Reinforcement Learning". In: *arXiv preprint arXiv:2004.02766* (2020).

[55]   Stephen J Wright. *Applying new optimization algorithms to more predictive control*. Tech. rep. Argonne National Lab., IL (United States), 1996.

[56]   Kemin Zhou and John Comstock Doyle. *Essentials of robust control*. Vol. 104. Prentice Hall Upper Saddle River, NJ, 1998.

[57]   Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *AAAI*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.