NH-TTC: A gradient-based framework for generalized anticipatory collision avoidance

Bobby Davis*, Ioannis Karamouzas†, and Stephen J Guy*
*University of Minnesota †Clemson University

***Abstract*—We propose NH-TTC, a general method for fast, anticipatory collision avoidance for autonomous robots with arbitrary equations of motions. Our approach exploits implicit differentiation and subgradient descent to locally optimize the non-convex and non-smooth cost functions that arise from planning over the anticipated future positions of nearby obstacles. The result is a flexible framework capable of supporting high-quality, collision-free navigation with a wide variety of robot motion models in various challenging scenarios. We show results for different navigating tasks, with various numbers of agents (with and without reciprocity), on both physical differential drive robots, and simulated robots with different motion models and kinematic and dynamic constraints, including acceleration-controlled agents, differential-drive agents, and smooth car-like agents. The resulting paths are high quality and collision-free, while needing only a few milliseconds of computation as part of an integrated sense-plan-act navigation loop. For a video of further results and reference code, please see the corresponding webpage: http://motion.cs.umn.edu/r/NH-TTC/**

I. Introduction

Recent trends in robotics, machine learning, and computer graphics have significantly advanced the state-of-the-art in autonomous navigation of mobile robots and intelligent agents, often through improvements in high-quality, long-term planning. In many situations, an important task for the robot is to *immediately react* to its local surroundings while still making its best effort to follow its global, long-term plan. Whether it is an autonomous car driving on a highway [25], or a semi-autonomous smart-shelf navigating in an automated warehouse [21], a robot should be able to observe its surroundings, *anticipate* the expected behavior of nearby obstacles, and react accordingly, all within a tight sense-plan-act loop.

Broadly speaking, the problem of locally steering a robot in a dynamic environment has seen two types of approaches. On one hand, *exact* solutions have been proposed that faithfully capture the robot's motion model, but can be slow to compute (typically hundreds of milliseconds or much more on complex scenarios) [7, 26, 32]. On the other hand, approximate solutions exist that compute new controls very fast (often in well under a millisecond), but simplify or approximate the dynamics of the problem which can lead to overly conservative behavior [3, 6]. Fast and exact approaches primarily exist only for special cases such as when planning for a holonomic, velocity-controlled robot moving in an environment of fixed velocity obstacles [13] or when moving amidst other holonomic, velocity-controlled robots [51].

A similar problem to robots navigating among dynamic obstacles is faced by humans when navigating in crowded environments. Recent analysis of human motion has shown that humans anticipate collisions and react to each other's expected trajectories [40]. This analysis has lead to new pedestrian simulation methods, such as [28] and [31]. However, these methods are only directly applicable to holonomic, velocity-controlled robots.

Contributions. Inspired by these approaches derived from human data, we propose NH-TTC, short for non-holonomic time to collision, a generalized approach for real-time navigation of mobile robots in complex, dynamic environments. NH-TTC plans directly over the full control space of the robots using the exact robot dynamics. This allows us to naturally support planning over a wide variety of robot types with various kinematic or dynamic constraints. We leverage two gradient-based optimization techniques, implicit differentiation and subgradient descent, to penalize only true collisions and promote goal-oriented controls, even in the face of the discontinuities introduced by time-to-collision computation. To enable fast planning as part of an iterative, anytime framework, we consider single control trajectories allowing us to generate high-quality trajectories in under a millisecond.

II. Related Work

Our NH-TTC method falls under the class of *anticipatory collision avoidance* methods in which the robot predicts how its neighborhood evolves over time and react accordingly. Early anticipatory approaches include the dynamic window approach [15], velocity obstacles (VO) [13], and inevitable collision states [16, 41]. These techniques, especially VOs, served as inspiration for many decentralized, multi-agent navigation approaches, which often incorporate the notion of reciprocity, where agents explicitly share the responsibility for collision [50, 22]. The (reciprocal) VO concept has been further extended to include reciprocal collision avoidance between robots having more complicated dynamics [52, 44, 5], multi-robot teams walking in formation [30, 27], and formulations that account for uncertainty in the future trajectory of obstacles through explicit error modeling [17, 49] or by using deep-learning based approaches [33].

Anticipatory collision avoidance is also an important aspect to how humans navigate [28, 40], and models of anticipation have been key for socially-inspired navigation approaches that seek to understand, replicate, or draw inspiration from human collision-avoidance strategies [14, 11]. Recent work has shown advantages of enabling more long term anticipatory planning by using specialized, high level representations such using braids theory to plan passing coordination [36] or by warping paths to represent arcs over time [55].

A particularly popular approach for fast, anticipatory collision avoidance in multi-agent scenarios is the ORCA framework from van den Berg et al. [51]. ORCA conservatively approximates collision avoidance constraints on a robot's motion as half-planes in the space of velocities. The optimal collision-free velocity can then be quickly found by solving a convex optimization problem through linear programming. ORCA-based navigation is fast to compute, and consistently leads to collision-free navigation; however, it assumes the robot can directly choose any velocity (i.e., no kinematic motion constraints).

Several recent extensions to the velocity-based planning approach have focused on enabling safe navigation of kinematically or dynamically constrained robots, such as robots with differential-drive dynamics or robots with maximum acceleration caps. For example, the Generalized Velocity Obstacles (GVO) [53] uses random sampling in order to find kinematically feasible controls that get the robot closer to its goal. Sampling-based strategies have proven to be slow, leading to the rise of fast, specialized methods that propose geometric, ORCA-like optimization for specific motion models including approaches for steering differential-drives, car-like robots, and other non-holonomic agents [18, 48, 2, 1].

More closely related to our work are very recent methods which have been proposed to extend the ORCA-like approaches to very broad classes of robot motion models. Generalized Reciprocal Velocity Obstacles (GRVO) treats kinematically-constrained robots as if they were able to take arbitrary velocities and uses an LQR controller to steer to robot to reach the computed velocity [6]. Similarly, the Cooperative Collision Avoidance (CCA) approach replaces each robot with a virtual agent who has a dynamically enlarged extent [3], this extra space gives kinematically-constrained robots the breathing room to maneuver to their target velocities before any collisions. Our proposed navigation approach supports fast, realtime collision avoidance across the same wide range of motion models as GRVO or CCA, but removes the need to approximate robot motion models or collision computations. Reducing this approximation error is especially important for robots with highly constrained motion models, or those navigating in particularly dense scenarios or in close quarters.

III. Notation and Assumptions

We assume our environment contains a single robot navigating to a goal position $\mathbf{p}_g$ while avoiding a set of obstacles (multi-robots scenarios are discussed in Section VII). We assume both the robot itself and the various obstacles in its environment follow some known continuous time dynamics functions that define evolution of the state of the robot $\mathbf{x}(t, \mathbf{u})$ and the obstacles states $O(t) = \{\mathbf{o}_i(t), \ \forall i\}$ as follows:

$$\dot{\mathbf{x}}(t, \mathbf{u}) = f(\mathbf{x}(t, \mathbf{u}), \mathbf{u}) \tag{1}$$

$$\dot{\mathbf{o}}_i(t) = g_i(\mathbf{o}_i(t)), \tag{2}$$

where $\mathbf{u} \in \mathcal{U}$ is a valid control input, and f, g are (possibly non-linear) continuous-time equations of motion. The set $\mathcal{U}$ encodes constraints on the robots dynamics, such as maximum control limits. Likewise, we can define a collection of valid states $\mathcal{X}$ that can be used to constrain any aspect of the robot's state such that $\mathbf{x}(t, \mathbf{u}) \in \mathcal{X}$, $\forall t$. This is needed to specify state constraints that are not directly part of a robot's control (e.g., an acceleration-controlled robot with a maximum velocity).

To determine collisions between the robot and the obstacles, we model them both as disks. These disks are defined by projecting both the robot and obstacle states into a common Euclidean workspace, typically 2d or 3d, and then finding the minimal covering disk. As such, we define the robot's position as:

$$\bar{\mathbf{x}}(t, \mathbf{u}) = p(\mathbf{x}(t, \mathbf{u})), \tag{3}$$

where p maps from state space to the Euclidean workspace. The function p is chosen to place the center of the collision disk with a radius r_x so as to wrap the true shape of the robot as closely as possible. As an example, for a car-like robot, this will shift the center of the collision disk from the natural dynamics point at the center of the rear axle to the center of the car. Similarly, obstacle i's position is defined as:

$$\bar{\mathbf{o}}_i(t) = q_i(\mathbf{o}_i(t)) \tag{4}$$

and r_{o_i} be the center of the collision disk and its radius for obstacle i at time t, respectively, where the function q_i maps the obstacle's state space to the workspace.

IV. NH-TTC Problem Formulation

Our work considers the problem of a robot that must traverse among moving obstacles while navigating to a goal position. Here, we formulate the problem as one where the robot is following a tight sense-plan-act loop many times a second. As such, our approach is similar to classic "reactive" planning approaches as the robot is given only a few milliseconds to compute new controls each time step in response to its immediate sensor input. Unlike many of these approaches, we propose an anytime approach, meaning that it will quickly find an acceptable solution (typically, in under a millisecond) and iteratively refine the solution as time is available.

A. Optimization-based Formulation

Given the above setting, we can formally define the problem as follows. We are given the robot's current state, $\mathbf{x}(0)$, a set of obstacle states over time, $O(t) = \{\mathbf{o}_i(t), \ \forall i\}$, and the robot's goal position, $\mathbf{p}_{goal}$, which we assume is been computed by a high-level planning approach. The task for the robot is to find a collision-free trajectory, $\mathbf{x}(t) \ \forall t \geq 0$, that approaches the goal as fast as possible while obeying the constraints of the robot dynamics, $\dot{\mathbf{x}}(t, \mathbf{u}) = f(\mathbf{x}(t, \mathbf{u}), \mathbf{u})$, the control constraint set, $\mathcal{U}$, and accounting for the state constraint set $\mathcal{X}$.

We can formulate this task as a trajectory optimization problem as follows. Given an arbitrary trajectory $\mathcal{T} = \{\mathbf{x}(t), \ \forall t \geq 0\}$ we construct a cost function $C(\mathcal{T})$ which evaluates how well the trajectory does at providing efficient, collision-free motion

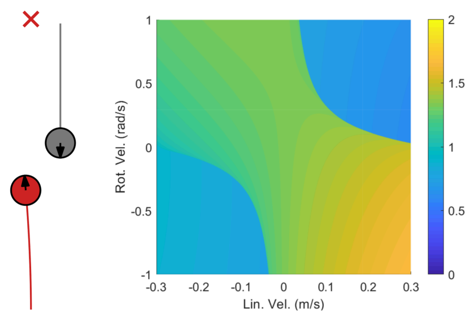


Fig. 1: **Gradient-Based Optimization in Control Space** (Left) A differential-drive robot, shown in red, has to reach the 'x' mark while avoiding a non-reactive obstacle shown in gray. (Right) The corresponding cost field of the robot is visualized here by taking samples from the robot's feasible controls. To better show the gradient, values are plotted in log scale with the color corresponding to $\log(C(\mathbf{u})+1)$. The optimal control (linear velocity = 0.3 m/s, rotational velocity = 0.03 rad/s) steers towards the goal, while gently avoiding any potential collisions.

towards the robot's current goal. In contrast to other trajectory optimization approaches, we treat collision constraints as an additional cost based on the time to collision with obstacles. As shown in [28], time to collision naturally balances soft avoidance of (temporally) distant collisions with hard avoidance of (urgent) nearby collisions. We can then represent our cost function as two separate terms:

$$C(\mathcal{T}) = C_{goal}(\mathcal{T}) + C_{col}(\mathcal{T}) \tag{5}$$

where $C_{goal}(\mathcal{T})$ evaluates how quickly the trajectory approaches its goal state (or goal position), and $C_{col}(\mathcal{T})$ assigns a penalty to trajectories which have a high risk of collision. Given sufficient computation time, our goal would be to find the complete trajectory which minimizes this cost function, e.g., via sampling as in [10, 9, 45], or using a POMDP-like formulation as in [42, 4]. However, these methods typically take several seconds or longer to converge, so they are inappropriate for the real-time setting considered here.

In order to allow the fast computation needed for use in a tight reactive planning loop, we consider only trajectories that are represented as a single, consistent, control $\mathbf{u}$ that is executed indefinitely. This critical assumption greatly accelerates the optimization process. As a result, we can reparameterize our cost function in terms of a single control $\mathbf{u}$. Formally, we seek to find the control $\mathbf{u}$ that minimizes:

$$C(\mathbf{u}) = C_{goal}(\mathbf{u}) + C_{col}(\mathbf{u}) \tag{6}$$

such that $\mathbf{u} \in \mathcal{U}$ and $\mathbf{x}(t, \mathbf{u}) \in \mathcal{X}\ \forall t \geq 0$ (see Figure 1). In practice, a robot would not take the optimal fixed-control trajectory indefinitely. Rather, it will re-run this optimization many times a second updating its planned trajectory as it approaches the goal and as local conditions change (as in a receding horizon planner [12, 35, 54]).

B. NH-TTC Cost Function

While many different cost functions could be used in our framework, we found that the sum of two simple cost functions works well in a wide variety of scenarios:

a) Goal Cost (C_{goal}): Because a fixed-control trajectory has no end point, we evaluate the position of the robot at some time t_{goal} into the future. The goal cost is then defined by how close the center of the robot (Equation 3) will be to its goal position at that time:

$$C_{goal}(\mathbf{u}) = \kappa_{goal} \left\| \bar{\mathbf{x}}(t_{goal}, \mathbf{u}) - \mathbf{p}_g \right\|, \tag{7}$$

where κ_{goal} is a scaling constant.

b) Collision Cost (C_{col}): Again, as our fixed-control trajectory has no endpoint, we evaluate collisions only for the next t_{horiz} seconds. Inspired by recent findings that suggest the urgency humans place on a collision follows an inverse power-law relationship with how imminent the collision is [28], we penalize trajectories with a term that is inversely proportional to the time until the nearest collision:

$$C_{col}(\mathbf{u}) = \max_{\mathbf{o}_i \in O} \frac{\kappa_{col}}{\tau(\mathbf{u}, \mathbf{o}_i)} \tag{8}$$

Here κ_{col} is a scaling constant and $\tau(\mathbf{u}, \mathbf{o}_i)$ computes the minimum time to collision between the robot's trajectory (as determined by the control, the current robot position, and the dynamics of the robot) and the expected future trajectory of obstacle $\mathbf{o}_i$. The result of this human-inspired collision cost is a natural balance between strongly avoiding urgent collisions and (when necessary) taking controls that will lead to collision in the far enough future, so that the robot will have a chance to re-plan well before the collision happens.

An important property of our cost function is that it rises to infinity as the time until the nearest collision approaches zero. This means any control which leads to immediate collisions has an effectively infinite penalty. As a result, in the limit as planning frequency approaches infinity, our approach is guaranteed to be collision-free so long as the optimizer has sufficient time to converge to a finite cost (assuming a collision-free control exists).

V. Control Optimization

While the above forms for the C_{goal} and C_{col} terms capture their respective costs in a straightforward fashion, the resulting cost function C is difficult to optimize as it is constrained, non-convex, and non-continuous. Specifically, time to collision has a sharp discontinuity between a glancing collision and no collision, jumping from a finite value to infinity, respectively (e.g., Figure 1). Due to these discontinuities, classical gradient descent will be ineffective (as will higher order, gradient-based optimization techniques). Smoothing this cost field (as was done in [29]) is not possible as we do not, in general, know a priori where the discontinuities will lie. We therefore use subgradient descent-based optimization [46, 8] which will allow us to find local minima even in the presence of a non-continuous, non-smooth cost function.

A. Subgradient Descent

To optimize Equation 6 using subgradient descent, as in standard gradient descent, the control $\mathbf{u}$ is iteratively updated based on the gradient of the cost with respect to $\mathbf{u}$:

$$\mathbf{u}_{k+1} = \mathbf{u}_k - \alpha\, \mathbf{s}_k, \tag{9}$$

where the search direction, $\mathbf{s}_k$, is based on the gradient $dC/d\mathbf{u}$. However, at points where C is not smooth, and therefore $dC/d\mathbf{u}$ is undefined, we choose either the left or right gradient. This is known as the subgradient, which we will call $\mathbf{g}_k$. While this gives us an optimization direction, it does not define the stepsize α, i.e., how far to update our control in that direction.

Numerous techniques have been proposed to choose an appropriate update size α. Experimentally, we found a Polyak update-based approach [43] to perform well in our domain. This method assumes the optimal possible control cost, c^*, is known in advance. Then, at each descent iteration k, given the current cost c_k:

$$\alpha = (c_k - c^*)/||\mathbf{s}_k||^2 \tag{10}$$

Given the complex, dynamic nature of our cost function, it is generally not possible to know the optimal value c^* in advance. As such, we compute an approximate optimal cost $\hat{c}^*$ by first taking the best cost seen in any of the iterations so far, c^+, and then interpolating it over iterations towards a conservative estimate of the lowest possible cost in this state, c_{lowest}:

$$\hat{c}^* = c^+ + \frac{10}{10+k}(c_{lowest} - c^+). \tag{11}$$

Here, we compute c_{lowest} by assuming the collision cost is zero and approximate a lower bound of the total cost using only the goal cost. The resulting update to α guarantees that the subgradient decent will converge on a true local minimum as k approaches infinity while still taking large updates when far away from the optimal control (similar strategies can be found in the subgradient optimization literature [47, 38]).

While it is possible to directly use the subgradient as the search direction (i.e., $\mathbf{s}_k = \mathbf{g}_k$), we find convergence to be improved by adding "momentum" to the search direction [39]. That is, our search direction at iteration k is based in part on the current subgradient $\mathbf{g}_k$ and in part on the previous search direction, $\mathbf{s_{k-1}}$. Experimentally, we found the following update rule to work well: $\mathbf{s}_k = \frac{1}{10}\mathbf{s}_{k-1} + \frac{9}{10}\mathbf{g}_k$.

Subgradient descent does not guarantee a cost decrease at each iteration, so the best control seen across the entire optimization is used as our final result. Additionally, we project the control computed at each iteration onto the control constraints, $\mathcal{U}$, to ensure the controls remain feasible. Furthermore, the trajectory should respect the state constraints, $\mathcal{X}$, and, if it does not, we project $\mathbf{u}$ to the nearest control that respects these constraints until at least the next planning cycle (see Section V-C). The resulting projected subgradient descent algorithm is shown in full in Algorithm 1. Here, we assume that the robot is given a time budget, $maxTime$, to compute its new control.

Algorithm 1: Subgradient-Based Control Optimization

```
Input  : $\mathbf{u}_0$, $\mathcal{U}$, $c_{lowest}$, $maxTime$
Output: $\mathbf{u}^*$
$k = 0$
$\mathbf{u}^* = \mathbf{u}_k = \mathbf{u}_0$
$c^+ = c_k = C(\mathbf{u}_0)$
$\mathbf{s}_{k-1} = \mathbf{0}_{n\times 1}$
while elapsed time < $maxTime$ do
    $\mathbf{g}_k = dC/d\mathbf{u}(\mathbf{u}_k)$
    $\mathbf{s}_k = \frac{1}{10}\mathbf{s}_{k-1} + \frac{9}{10}\mathbf{g}_k$
    $\hat{c}^*_k = c^+ + (c_{lowest} - c^+)10/(10+k)$
    $\alpha = (c_k - \hat{c}^*_k)/||\mathbf{s}_k||^2$
    $\mathbf{u}_{k+1} = \mathbf{u}_k - \alpha\,\mathbf{s}_k$
    Project $\mathbf{u}_{k+1}$ into $\mathcal{U}$ and $\mathcal{X}$ (see Section V-C)
    $c_{k+1} = C(\mathbf{u}_{k+1})$
    Update $c^+$ and $\mathbf{u}^*$
    $k = k + 1$
end
```

B. Subgradient Computation

Below, we first discuss how to compute the two terms of the cost function, C_{goal} and C_{col}, and then focus on the computation of the subgradient, $\mathbf{g}_k = dC/d\mathbf{u}(\mathbf{u}_k)$. Note that computing $\mathbf{g}_k$ is not trivial for many dynamics models as there may not be a closed form solution for the robot position, the obstacle position, or the time to collision.

1) Cost Computation: To compute the goal cost, C_{goal} (Equation 7), we need to compute the position at t_{goal}. As we may not have a closed form solution for $\mathbf{x}(t_{goal}, \mathbf{u})$, we approximate it using fourth order Runge-Kutta integration (RK4). To improve accuracy, we iteratively run multiple steps of RK4, such that each step is, at most, some small time horizon, dt_{max}. This parameter allows for tuning the accuracy of our position estimation, at the expense of computation time. Once $\mathbf{x}(t_{goal}, \mathbf{u})$ has been computed, we pass it through the position mapping function p to obtain the Euclidean position, $\bar{\mathbf{x}}(t_{goal}, \mathbf{u})$, which, along with the goal position, fully defines the goal cost.

To compute the collision cost, C_{col} (Equation 8), we need to compute the most imminent time to collision over all the obstacles. Assuming both objects can be approximated by disks, the time to collision between an agent $\mathbf{x}$ and object $\mathbf{o}$ can be defined as the time at which the two disks touch, i.e.:

$$||\bar{\mathbf{x}}(\tau(\mathbf{u},\mathbf{o}),\mathbf{u})) - \bar{\mathbf{o}}(\tau(\mathbf{u},\mathbf{o}))||^2 - (r_x + r_o)^2 = 0, \tag{12}$$

where $\bar{\mathbf{x}}$ (Equation 3) and $\bar{\mathbf{o}}$ (Equation 4) return the agent and obstacle collision centers respectively. However, for many systems, solving this equation for τ is not feasible, as there may not be a closed form solution for $\bar{\mathbf{x}}$ and/or $\bar{\mathbf{o}}$, or the resulting equation is too complex. Instead, we utilize a similar approach to that used to compute the goal cost. We forward propagate the state of the robot and the state of each obstacle using RK4, and perform linear continuous collision checks between the resulting states to estimate the first moment of

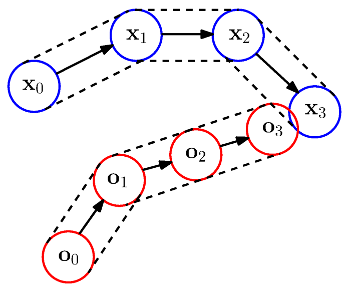


Fig. 2: **Time to Collision Computation**: Robot and obstacle states are forward propagated using RK4 integration, and then linear continuous collision checks are done between each discrete state.

collision that may have occurred during the integration steps (see Figure 2).

Knowing how to compute the (propagated) cost C, we next show how to compute the gradient of first the goal cost and then the collisions cost with respect to the controls $\mathbf{u}$.

2) Goal Cost Gradient: As C_{goal} (Equation 7) indirectly relies on $\mathbf{u}$, we must apply the chain rule to compute the gradient:

$$\frac{dC_{goal}(\mathbf{u})}{d\mathbf{u}} = \frac{dC_{goal}(\mathbf{u})}{d\bar{\mathbf{x}}(t_{goal},\mathbf{u})}\frac{d\bar{\mathbf{x}}(t_{goal},\mathbf{u})}{d\mathbf{x}(t_{goal},\mathbf{u})}\frac{d\mathbf{x}(t_{goal},\mathbf{u})}{d\mathbf{u}} \tag{13}$$

The first term can be directly computed as:

$$\frac{dC_{goal}(\mathbf{u})}{d\bar{\mathbf{x}}(t_{goal},\mathbf{u})} = \frac{\kappa_{goal}}{2\left\|\bar{\mathbf{x}}(t_{goal},\mathbf{u}) - \mathbf{p}_g\right\|} \tag{14}$$

The second term, $d\bar{\mathbf{x}}(t_{goal},\mathbf{u})/d\mathbf{x}(t_{goal},\mathbf{u})$, can be computed directly given the projection function p (this term will be 1 if the robot's control point is the center of the collision disk.

All that remains is to compute the third term, $d\mathbf{x}(t_{goal},\mathbf{u})/d\mathbf{u}$. Given a discrete time dynamics function, this gradient can be computed iteratively as:

$$\frac{d\mathbf{x}(t+dt,\mathbf{u})}{d\mathbf{u}} = \frac{\partial\mathbf{x}(t+dt,\mathbf{u})}{\partial\mathbf{u}} + \frac{\partial\mathbf{x}(t+dt,\mathbf{u})}{\partial\mathbf{x}(t,\mathbf{u})}\frac{d\mathbf{x}(t,\mathbf{u})}{d\mathbf{u}} \tag{15}$$

which follows directly from the multivariate chain rule. We start from $d\mathbf{x}(0,\mathbf{u})/d\mathbf{u} = 0$, as the current position is independent of the upcoming control, and apply Equation 15 iteratively until the gradient at some user-defined time, T, is obtained. However, we may not have a closed form solution for $\mathbf{x}(t,\mathbf{u})$, and therefore no closed form solution for the discrete time dynamics. We can approximate this gradient using a series of trapezoidal integration steps on the (known) continuous dynamics as follows:

$$\begin{aligned}\mathbf{x}^+ &= \mathbf{x}(t,\mathbf{u}) + dt\cdot\dot{\mathbf{x}}(\mathbf{x}(t,\mathbf{u}),\mathbf{u})\\ \mathbf{x}(t+dt,\mathbf{u}) &\approx \mathbf{x}(t,\mathbf{u}) + \frac{dt}{2}(\dot{\mathbf{x}}(\mathbf{x}(t,\mathbf{u}),\mathbf{u}) + \dot{\mathbf{x}}(\mathbf{x}^+,\mathbf{u}))\end{aligned} \tag{16}$$

Using these approximate dynamics, we can iteratively compute the partial derivatives required by Equation 15 as:

$$\frac{\partial\mathbf{x}(t+dt,\mathbf{u})}{\partial\mathbf{u}} \approx \frac{dt}{2}\left(\frac{\partial\dot{\mathbf{x}}(\mathbf{x}(t,\mathbf{u}),\mathbf{u})}{\partial\mathbf{u}} + \frac{\partial\dot{\mathbf{x}}(\mathbf{x}^+,\mathbf{u})}{\partial\mathbf{u}} + \frac{\partial\dot{\mathbf{x}}(\mathbf{x}^+,\mathbf{u})}{\partial\mathbf{x}^+}\frac{\partial\mathbf{x}^+}{\partial\mathbf{u}}\right) \tag{17}$$

Algorithm 2: Position Control Gradient

```
Input  : $\mathbf{u}$, $\mathbf{x}(0,\mathbf{u})$, $T$, $dt_{max}$
Output: $\mathbf{x}(T,\mathbf{u})$, $d\mathbf{x}(T,\mathbf{u})/d\mathbf{u}$
$t = 0$
$\frac{d\mathbf{x}(t,\mathbf{u})}{d\mathbf{u}} = 0$
while $t < T$ do
    $dt = \min(dt_{max}, T - t)$
    $\mathbf{x}(t+dt,\mathbf{u}) = \mathrm{RK4}(\mathbf{x}(t,\mathbf{u}),\mathbf{u},dt)$
    $\frac{d\mathbf{x}(t+dt,\mathbf{u})}{d\mathbf{u}} =$ Equation 15
    $t = t + dt$
end
```

$$\frac{\partial\mathbf{x}(t+dt,\mathbf{u})}{\partial\mathbf{x}(t,\mathbf{u})} \approx I + \frac{dt}{2}\Big(\frac{\partial\dot{\mathbf{x}}(\mathbf{x}(t,\mathbf{u}),\mathbf{u})}{\partial\mathbf{x}(t,\mathbf{u})} + \frac{\partial\dot{\mathbf{x}}(\mathbf{x}^+,\mathbf{u})}{\partial\mathbf{x}^+}\frac{\partial\mathbf{x}^+}{\partial\mathbf{x}(t,\mathbf{u})}\Big) \tag{18}$$

where the half step partial derivatives are computed as:

$$\frac{\partial\mathbf{x}^+}{\partial\mathbf{u}} = dt\frac{\partial\dot{\mathbf{x}}(\mathbf{x}(t,\mathbf{u}),\mathbf{u})}{\partial\mathbf{u}} \tag{19}$$

$$\frac{\partial\mathbf{x}^+}{\partial\mathbf{x}(t,\mathbf{u})} = \mathbf{I} + dt\frac{\partial\dot{\mathbf{x}}(\mathbf{x}(t,\mathbf{u}))}{\partial\mathbf{x}(t,\mathbf{u})} \tag{20}$$

The full iterative process for computing the $d\mathbf{x}(T,\mathbf{u})/d\mathbf{u}$ at time T is shown in Algorithm 2 (here, we set $T = t_{goal}$). Finally, we combine the resulting gradient as in Equation 13 to compute the total gradient for the goal cost term.

3) Collision Cost Gradient: Similar to the goal cost term, the collision cost term, C_{col} (Equation 8), relies indirectly on $\mathbf{u}$, so we must again compute its gradient via the chain rule:

$$\frac{dC_{col}(\mathbf{u})}{d\mathbf{u}} = \frac{dC_{col}(\mathbf{u})}{d\tau(\mathbf{u},\mathbf{o}^*)}\frac{d\tau(\mathbf{u},\mathbf{o}^*)}{d\mathbf{u}} \tag{21}$$

where $\mathbf{o}^*$ is the obstacle with the closest time-to-collision.

However, unlike in the goal cost gradient, the time to collision, τ, cannot generally be written explicitly as a function of the controls $\mathbf{u}$, which prevents us from computing $d\tau(\mathbf{u},\mathbf{o}^*)/d\mathbf{u}$. To address this issue, we propose the use of *implicit differentiation*. This allows us to have an analytic expression of the gradient implicitly written as a function of $\tau(\mathbf{u},\mathbf{o}^*)$. The implicit relationship between τ and $\mathbf{u}$ is shown in Equation 12. Using this relationship, we can find $d\tau(\mathbf{u},\mathbf{o}^*)/d\mathbf{u}$ by taking the derivative of Equation 12 with respect to $\mathbf{u}_j$ (the jth element of the control), and solving for $d\tau(\mathbf{u},\mathbf{o}^*)/d\mathbf{u}_j$:

$$\frac{d\tau(\mathbf{u},\mathbf{o}^*)}{d\mathbf{u}_j} = -\frac{(\bar{\mathbf{x}} - \bar{\mathbf{o}}^*)^T(\frac{d\bar{\mathbf{x}}}{d\mathbf{u}_j})}{(\bar{\mathbf{x}} - \bar{\mathbf{o}}^*)^T(\frac{d\bar{\mathbf{x}}}{d\tau} - \frac{d\bar{\mathbf{o}}^*}{d\tau})} \tag{22}$$

where $\bar{\mathbf{x}} = \bar{\mathbf{x}}(\tau(\mathbf{u},\mathbf{o}^*),\mathbf{u}))$ and $\bar{\mathbf{o}}^* = \bar{\mathbf{o}}^*(\tau(\mathbf{u},\mathbf{o}^*))$.

Note that $d\bar{\mathbf{x}}/d\tau$ and $d\bar{\mathbf{o}}^*/d\tau$ are the known continuous time dynamics of the robot and the obstacle, and $d\bar{\mathbf{x}}/d\mathbf{u}_j$ can be computed via the chain rule as:

$$\frac{d\bar{\mathbf{x}}}{d\mathbf{u}_j} = \frac{d\bar{\mathbf{x}}}{d\mathbf{x}}\frac{d\mathbf{x}}{d\mathbf{u}_j} \tag{23}$$

where $d\bar{\mathbf{x}}/d\mathbf{x}$ is dependent on the dynamics model, and $d\mathbf{x}/d\mathbf{u}_j$ can be computed with Algorithm 2, setting $T = \tau(\mathbf{u}, \mathbf{o}_i)$. If τ is infinite (i.e. there is no collision), this derivative is 0.

C. State Constraints

While collisions are handled through the cost term C_{col}, other state constraints (such as maximum velocities) are handled as true constraints. For these constraints, a small modification needs to be applied to NH-TTC. Since we are generating trajectories with a single control, it is not always possible to guarantee that such constraints will be satisfied for the entire trajectory. For example, applying a non-zero acceleration will eventually violate any velocity magnitude constraint. To address this issue, we only allow controls that will not violate the state constraints within the next timestep. For example, in each iteration of subgradient descent, we project the control of an acceleration controlled robot as follows:

$$\mathbf{u}_{proj} = \begin{cases} \mathbf{a} & \text{if } ||\mathbf{v} + \mathbf{a}\,dt|| \leq v_{max} \\ \dfrac{\mathbf{v}^* - \mathbf{v}}{dt}, & otherwise \end{cases} \tag{24}$$

where $\mathbf{v}^*$ is $\mathbf{v} + \mathbf{a}\,dt$ projected onto v_{max}. This ensures the control $\mathbf{u}_{proj}$ will not violate the state constraints within the next timestep.

In addition to enforcing the state constraints for the next timestep, we must additionally account for these constraints in predictions of our future states, such as in the collision search. We do this by modifying the continuous dynamics to forbid exceeding the state constraints. In the above example, we would modify the continuous dynamics of an acceleration controlled robot as follows:

$$\dot{\mathbf{v}} = \begin{cases} \dfrac{\mathbf{a}}{100}, & \text{if } ||\mathbf{v}|| > v_{max} \text{ and } \mathbf{a}^T\mathbf{v} > 0 \\ \mathbf{a}, & otherwise \end{cases} \tag{25}$$

While it may be natural to zero out the acceleration if the constraint is violated, this could result in the gradient of the velocity with respect to acceleration going to zero as well, and no optimization would occur. Instead, we limit the acceleration to a very small value to avoid the vanishing gradient problem. Note that our above approach is only applicable when the state constraints can be expressed as a convex constraint on the control.

D. Implementation Details

In our experiments, the cost parameters, κ_{goal} and κ_{col}, are both set to 1. For the time-to-collision search, t_{horiz} is set to 5 s and dt_{max} is set to 0.1 s. The goal distance time, t_{goal}, is set to 1 s. Trajectories are planned using 10 ms of planning time, and controls are updated at 10 Hz. All results are generated on a single thread on an Intel Xeon 3.0GHz CPU (for physical robots, each robot planned in its own thread). We implemented our subgradient-based optimization framework in C++ using Eigen [20] to efficiently handle matrix and vector operations.

The exact terms of the cost gradient computation (Equations 13-22) will vary based on the dynamics of the robot under consideration. Given the equations of motion, these derivatives can be computed analytically. While our framework supports many robot dynamics models, our results primarily consider four motion dynamics: (V) a robot that can directly control its x- and y-velocities, (A) an acceleration-controlled robot, (DD) a robot that that has differential drive kinematics, and (Car) a robot which has car-like kinematics. This collection includes span a range of 1st and 2nd order dynamics, with or without kinematic constraints, and includes both holonomic and non-holonomic systems. Importantly, including 2nd order dynamics models allows our method to explicitly enforce the smoothness of the planned trajectories.

For all of these motion models, we impose a constraint on the maximum linear velocity of the robot. Additionally, the acceleration model caps the maximum acceleration, the differential-drive and car-like models both cap the maximum angular velocity, and the car-like model has a maximum steering angle. In all of our experiments, unless otherwise specified, we use the following constraints: 0.3 m/s for linear velocity, 1.0 rad/s for angular velocity, 1.0 m/s^2 for linear acceleration, π rad/s^2 for angular acceleration, and $\pi/4$ rad for steering angle.

	Optimization Time		
Dynamics	1 ms	5 ms	10 ms
V	99.7% (5.9)	99.9% (2.0)	99.9% (3.2)
A	99.7% (5.3)	99.9% (2.6)	99.9% (2.6)
DD	99.5% (7.3)	99.5% (6.7)	99.6% (6.1)
Car	99.0% (9.7)	99.5% (6.8)	99.6% (6.4)

TABLE I: **Percent of Collision-Free Frames**: The average percent and variance (reported in percentage points) of collision-free frames is shown for the scenario in Figure 3(a-c) for various dynamics models. Experiments were run for 1000 frames, and averaged over 1000 runs per dynamics model.

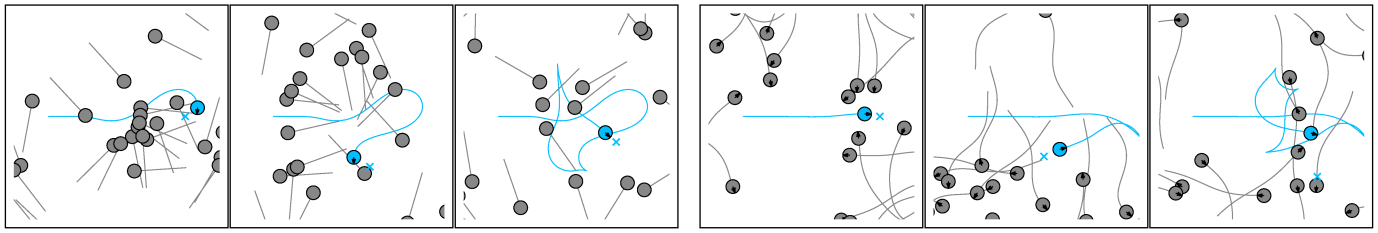
VI. Single-Agent Collision Avoidance

A key application of anticipatory collision avoidance is to allow a robot to avoid nearby dynamic obstacles as it moves to its goal. Our method supports high-quality, collision-free navigation for various dynamics models on many such scenarios.

A. Random Agents Scenario

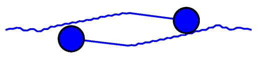
To test the collision avoidance performance in a challenging scenario, we tasked a simulated robot to navigate to randomly generated goals while using NH-TTC to avoid 40 simultaneously moving, non-reactive linear velocity obstacles in a densely packed scenario (Figure 3(a-c)). This scenario is very challenging as obstacles occasionally "box-in" the robot, and create scenarios in which collisions are unavoidable. Note, this extreme scenario is the only scenario we tested for which there are any collisions with our method.

Table I shows how our method performs with different agent dynamics models. In general, increasing the complexity of the dynamics model increases the average number of colliding frames. Overall though, even in such a challenging setting

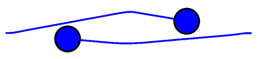


(a) $t = 8.5\,\mathrm{s}$ (b) $t = 13.2\,\mathrm{s}$ (c) $t = 26.0\,\mathrm{s}$ (d) $t = 6.0\,\mathrm{s}$ (e) $t = 14.1\,\mathrm{s}$ (f) $t = 28.0\,\mathrm{s}$

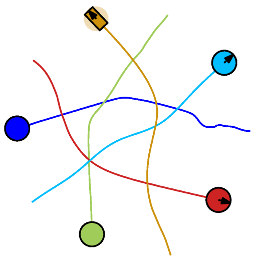
Fig. 3: **Smooth Differential Drive Robot Among Random Obstacles**: The cyan robot moves to a random goal position, indicated by the colored x, while avoiding a number of grey dynamic obstacles. The recent trajectories traced by the robot and the obstacles are displayed with solid lines. (a-c) The obstacles follow known linear paths. (d-f) The obstacles follow unknown sinusoidal paths, where the robot is only given the current linear and angular velocities of each obstacle.



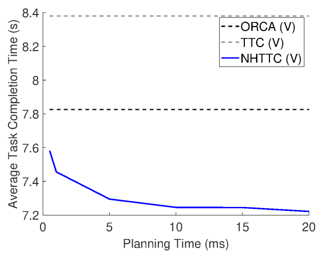
(a) No Reciprocity



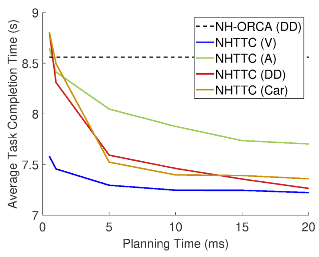
(b) With Reciprocity



(c) 5-Agent Circle



(d) Performance Comparison



(e) Convergence Comparison

Fig. 4: **(a-b) Reciprocity**: Two velocity-controlled agents swap positions. In (a), reciprocity is disabled and each agent takes full responsibility to resolve the collision. In (b), reciprocity is enabled. This shows the smoothing effect reciprocity has on the resulting trajectories. **(c) 5-Agent Circle**: Each of the agents attempts to move to its antipodal position on a circle. The dark blue agent is velocity controlled, the green agent is acceleration controlled, the red agent is differential drive controlled, the light blue agent is smooth differential drive controlled, and the yellow agent is a simple car. **(d-e) Performance Comparisons**: Task completion time for homogeneous agents in the 5-Agent Circle who all share the same dynamics model. NH-TTC is able to outperform ORCA, TTC, and NH-ORCA on average, even with more restrictive motion models. Results averaged over 100 runs.

that is unlikely to happen in real life, collisions are very rare, typically occurring in less than 0.5% of frames. For nearly all of the dynamics models, increasing the optimization time both reduces the average number of colliding frames and the variance in the number of colliding frames. Though most of the cost improvement occurs within the first 5 ms.

We also tested the performance of our method in a variant of the above scenario, where now the future trajectories of the obstacles are unknown, and the robot makes predictions based only on the current actions of each obstacle. Figure 3(d-f) shows results with a smooth differential drive robot avoiding differential drive obstacle following randomized sinusoidal paths. Again, the robot is able to smoothly avoid all obstacles while progressing through multiple goals.

B. *Additional Scenarios and Results*

Additional scenarios are shown in the supplemental video. These results include additional dynamics models such as an acceleration-controlled version of both the differential drive dynamics and the car-like dynamics. Similarly, we also show results from changing various control constraints.

VII. Multi-Agent NH-TTC

Directly optimizing the cost function in Equation 6 is not the correct behavior to take when the obstacles are also actively avoiding collisions with the robot (e.g., the obstacles are other robots). In these scenarios, avoiding the full collision at every time step can result in oscillatory behavior. This is due the fact that each agent tries to resolve the collision by itself without accounting for the fact that the other agent, by symmetry, is facing the exact same condition. This will lead to a pattern of robots alternatively over- and then under-correcting for collisions, which results in jerky motion (see Figure 4a).

We can address this issue through allowing *reciprocity* between the two agents, where they share the effort of averting mutual collisions [50]. Inspired by the approach taken by ORCA (only avoiding half the collision), we only update the control to halfway between the new optimal control and the previous control. This modification results in smoother paths, while still fully avoiding collisions (see Figure 4b). Even if the obstacle is not actual reacting to avoid the robot, the robot still converges to a collision-free control after a few time steps.

A. *Heterogeneous Circle*

To highlight how our approach can handle interactions between heterogeneous agents, i.e. agents that can have different motion models and state spaces, we consider a scenario with five agents, each with a different robot model. In this scenario, each of the five agents are simultaneously planning in a decentralized manner as they attempt to move to antipodal points on a circle. Figure 4c shows the paths taken by each agent. As can also be observed in the supplementary video, NH-TTC generates controls that lead to collision-free and smooth paths. Note that the jitter in the initial portion of the velocity-controlled agent comes from the implicit coordination between the agents. After the first few seconds, once the agents have come to an implicit consensus on the paths to take, the paths are smooth for the rest of execution.

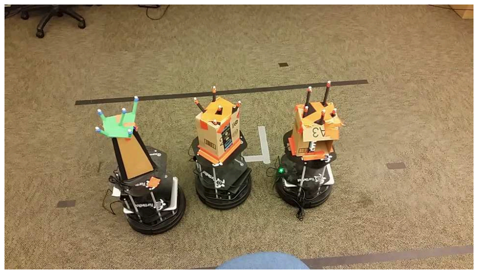
B. *Comparison to Other Techniques*

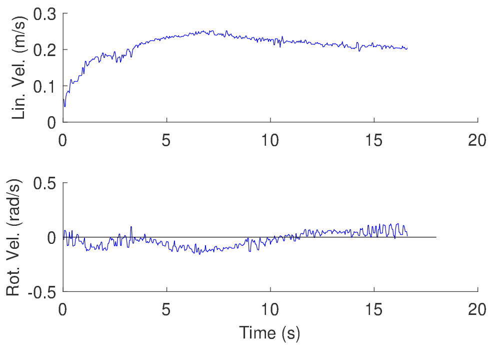
To empirically analyze the efficiency of the paths generated by NH-TTC, we compare its performance in a 5-agent circle scenario to TTC [28], ORCA [51], and NH-ORCA [1] as implemented by Hennes et al. [23]. This scenario is set up similar to that in Figure 4c, but with homogeneous models. Both ORCA and TTC are typically deterministic, but we observed ORCA agents frequently deadlocking in nearly symmetric scenarios. To alleviate this shortcoming, we add a small amount of noise to the goal velocity at each timestep of ORCA. This noise amount, $\pm 0.1 m/s$, was chosen as the smallest amount of noise that resulted in a 100% success rate. Even with very small optimization times (less than 1 ms), NH-TTC is able to outperform ORCA and TTC in the velocity scenario (see Figure 4d). NH-TTC is also able to perform well compared to NH-ORCA in a differential drive scenario.

We also compare NH-TTC perfomance across different motion models. Note that standard TTC and ORCA only support velocity controlled models, while NH-ORCA only supports differential drive models. Other techniques (such as [3] and [48]) have extended ORCA to non-velocity motion models. However, all of these techniques rely on increasing the collision radius of the robot to account for changing the motion model. This radius increase will shrink the feasible control space, frequently resulting in longer paths. In comparison, NH-TTC is able to achieve similar performance as in the velocity controlled-case accross other first order models (DD and Car, see Figure 4e). Even with a second order, acceleration-controlled model (A), NH-TTC is able to plan efficient paths that outperform the holonomic models mentioned above. Note that even for very small (sub-millisecond) optimization times, our planned trajectories are still collision free.

C. *Execution on Physical Robots*

To test the applicability of our method to real robots, we implemented our framework on three Turtlebot2 robots (a differential drive system). We used an OptiTrack system for position and orientation localization, and used the internal odometry of the robots to get the linear and angular velocities. Each robot communicated its current pose and velocity to the other robots, and asynchronously planned over the latest received state of the world.


(a) Following



(b) Robot Controls

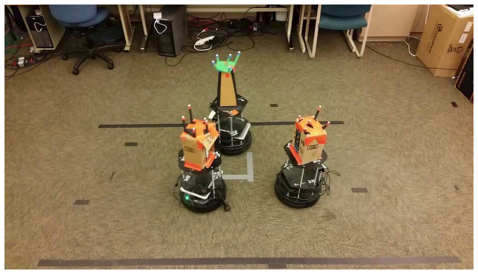
Fig. 5: **Physical Robots - Following Scenario**: One robot attempts to slip between two unresponsive robots (a simulated version is shown in Figure 8). In (b), the controls executed by the controlled robot are shown, as measured by wheel encoders and gyroscope. The robot is able to quickly accelerate and slide between the two agents as soon as an appropriate gap opens up.

We tested the applicability of our approach to physical robots on two scenarios:

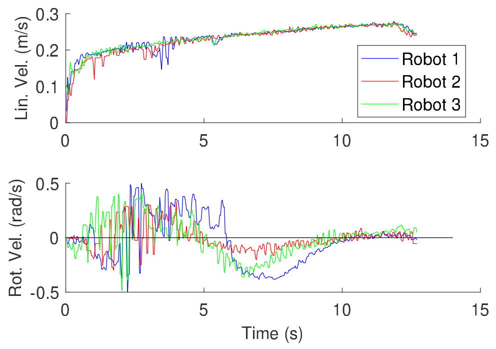
- *"Car" Following*: Two non-reactive robots moving at a constant velocity of 0.2 and 0.15 m/s, respectively, with a single controlled robot having a maximum velocity of 0.3 m/s. As in the simulated version of this case, we check the goal distance at multiple temporal points in the future to reduce oscillatory behavior.
- *3-Robot Circle*: Three robots simultaneously try to move to antipodal points on a circle.

In all of our physical robot experiments, we left the maximum linear velocity at 0.3 m/s, but reduced the maximum rotational velocity to be 0.5 rad/s in order to match the actual limits of the robots used in the experiments. Results from these experiments can be seen in the companion video (Extension 1).

To further study the quality of the generated trajectories, we plotted the controls taken by the robot over time, as measured by its wheel encoders and gyroscope, in Figures 5 and 6. While there is some inherent noise in the measured controls, the robots were able to smoothly achieve their goals without colliding, taking admissible controls that stayed within the



(a) 3-Robot Circle



(b) Robots' Controls

Fig. 6: **Physical Robots - 3-Robot Circle**: Three robots avoid each other while moving to antipodal positions on a circle. The controls executed by each robot, as measured by wheel encoders and gyroscope, are shown in (b), illustrating the ability of the differential drives to smoothly adapt their linear and angular velocities to resolve impending collisions.

given limits in all scenarios tested.

In the car-following scenario, while in motion, the average linear acceleration of the controlled robot was $0.008\,\mathrm{m/s^2}$ and its average rotational acceleration was $0.002\,\mathrm{rad/s^2}$. In the first 8 s, the robot gradually adapts its linear and angular speed to smoothly slip between the two obstacles. Then, in the next 7 s, it starts decelerating in order to align itself with the speed and orientation of the leader obstacle, after which it maintains an almost zero linear and angular acceleration.

In the 3-robot circle scenario, averaged across the times when the three differential drive robots were moving, the average linear acceleration was $0.008\,\mathrm{m/s^2}$ and the average rotational acceleration was $0.011\,\mathrm{rad/s^2}$. Here, the robots are able to quickly resolve the collisions within the first 5 s, and then smoothly adapt their orientations and accelerate to reach their goals. It is worth noting that the quick rotational velocity changes that the robots exhibit during the first 4 s is due to the symmetric nature of the scenario; the robots are on track to arrive in the center of the environment nearly at the same time, and attempt to break the symmetry by trying to implicitly agree on whether to perform clockwise or counter clockwise avoidance maneuvers.

VIII. Conclusion

In this paper we have proposed NH-TTC, a new generalizable framework for anticipatory collision avoidance. Our method is able to optimize directly in the control space of the robot in an anytime fashion, allowing collision-free trajectories to be computed over very short planning times for a wide variety of robot motion models. To do so, we cast local navigation as a control optimization problem and employ an anticipatory cost function that focuses on the expected future values of robot controls. As such a function is non-convex and suffers from discontinuities, we minimize it using subgradient descent, and use implicit differentiation to capture the dynamics of future collisions for arbitrary motion models.

Limitations: While our approach performs well in many scenarios, there are some navigation tasks it does not address well. Since we optimize over single control trajectories, we are unable to operate on any unstable systems (such as a humanoid robot) where a single control cannot be taken over long horizons. In addition, greedily optimizing goal distance at each planning cycle limits the maneuvers we can generate. However, this difficulty could be alleviated through a more complex goal distance function that evaluates the resulting trajectory quality holistically, rather than measuring the distance to the goal at a single point in time.

Future Work: We are excited to test the application of NH-TTC to other mobile robot types, especially those having 3D dynamics such as quadrotors. In the future, we would also like to extend NH-TTC to account for motion and sensing uncertainty in the future trajectories of obstacles. Prior work on uncertainty-aware local navigation [24, 14] can provide some interesting ideas in this direction. Furthermore, we would like to relax some of the assumptions that our framework makes, such as that the robot and obstacles will maintain a constant control input over a finite time horizon. Integrating our method with obstacle prediction techniques [37, 19] could result in better trajectories in crowded settings. Finally, as we currently fit a single collision disk around each robot which can underestimate the true time-to-collision value, we would like to better approximate the robots using, e.g., the medial axis transform [34].

Acknowledgments

This work was supported in part by the National Science Foundation under Grants IIS-1748541 and CNS-1544887.

References

[1] Javier Alonso-Mora, Andreas Breitenmoser, Paul Beardsley, and Roland Siegwart. Reciprocal collision avoidance for multiple car-like robots. In *IEEE International Conference on Robotics and Automation*, pages 360–366, 2012.

[2] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Distributed Autonomous Robotic Systems*, pages 203–216. Springer, 2013.

[3] Javier Alonso-Mora, Paul Beardsley, and Roland Siegwart. Cooperative collision avoidance for nonholonomic robots. *IEEE Transactions on Robotics*, 34(2):404–420, 2018.

[4] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*, 35(14):1760–1778, 2016.

[5] Daman Bareiss and Jur van den Berg. Reciprocal collision avoidance for robots with linear dynamics using LQR-obstacles. In *IEEE International Conference on Robotics and Automation*, pages 3847–3853, 2013.

[6] Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

[7] Kostas E Bekris and Lydia E Kavraki. Greedy but safe replanning under kinodynamic constraints. In *IEEE International Conference on Robotics and Automation*, pages 704–710, 2007.

[8] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[9] J. Denny, M. Morales, S. Rodriguez, and N. M. Amato. Adapting rrt growth for heterogeneous environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1772–1778, 2013.

[10] Andrew Dobson, Kiril Solovey, Rahul Shome, Dan Halperin, and Kostas E Bekris. Scalable asymptotically-optimal multi-robot motion planning. In *2017 international symposium on multi-robot and multi-agent systems (MRS)*, pages 120–127. IEEE, 2017.

[11] Teófilo Dutra, Ricardo Marques, Joaquim B. Cavalcante-Neto, Creto Augusto Vidal, and Julien Pettré. Gradient-based steering for vision-based crowd simulation algorithms. *Computer Graphics Forum*, 36(2), 2017.

[12] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE Transactions on Control Systems Technology*, 15(3):566–580, 2007.

[13] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17:760–772, 1998.

[14] Zahra Forootaninia, Ioannis Karamouzas, and Rahul Narain. Uncertainty models for TTC-based collision avoidance. In *Robotics: Science and Systems*, 2017.

[15] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[16] Thierry Fraichard and Hajime Asama. Inevitable collision states - A step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[17] Chiara Fulgenzi, Anne Spalanzani, and Christian Laugier. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *IEEE International Conference on Robotics and Automation*, pages 1610–1616, 2007.

[18] Andrew Giese, Daniel Latypov, and Nancy M Amato. Reciprocally-rotating velocity obstacles. In *IEEE International Conference on Robotics and Automation*, 2014.

[19] Julio Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria L Gini. Moving in a crowd: Safe and efficient navigation among heterogeneous agents. In *IJCAI*, pages 294–300, 2016.

[20] Gaël Guennebaud, Benoît Jacob, and Others. Eigen v3. http://eigen.tuxfamily.org, 2010.

[21] Eric Guizzo. Three engineers, hundreds of robots, one warehouse. *IEEE spectrum*, 45(7):26–34, 2008.

[22] Stephen J Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187, 2009.

[23] Daniel Hennes, Daniel Claes, Wim Meeussen, and Karl Tuyls. Multi-robot collision avoidance with localization uncertainty. In *AAMAS*, pages 147–154, 2012.

[24] Daniel Hennes, Daniel Claes, Wim Meeussen, and Karl Tuyls. Multi-robot collision avoidance with localization uncertainty. In *International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 147–154. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[25] Thomas M Howard, Colin J Green, Alonzo Kelly, and Dave Ferguson. State space sampling of feasible motions for high-performance mobile robot navigation in complex environments. *Journal of Field Robotics*, 25(6-7):325–345, 2008.

[26] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[27] Ioannis Karamouzas and Stephen J. Guy. Prioritized group navigation with formation velocity obstacles. In *IEEE International Conference on Robotics and Automation*, pages 5983–5989, 2015.

[28] Ioannis Karamouzas, Brian Skinner, and Stephen J. Guy. Universal power law governing pedestrian interactions. *Physical Review Letters*, 113:238701, 2014.

[29] Ioannis Karamouzas, Nick Sohre, Rahul Narain, and Stephen J. Guy. Implicit crowds: Optimization integrator for robust crowd simulation. *ACM Trans. Graph.*, 36(4): 136:1–136:13, July 2017. ISSN 0730-0301.

[30] Andrew Kimmel, Andrew Dobson, and Kostas Bekris. Maintaining team coherence under the velocity obstacle framework. In *International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 247–256, 2012.

[31] Ross A Knepper and Daniela Rus. Pedestrian-inspired sampling-based multi-robot collision avoidance. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 94–100. IEEE, 2012.

[32] Yanbo Li, Zakary Littlefield, and Kostas E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[33] Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *Robotics and Automation Letters*, 2(2):656–663, 2017.

[34] Yuexin Ma, Dinesh Manocha, and Wenping Wang. Efficient reciprocal collision avoidance between heterogeneous agents using ctmat. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1044–1052. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[35] Jacob Mattingley, Yang Wang, and Stephen Boyd. Receding horizon control. *IEEE Control Systems*, 31(3): 52–65, 2011.

[36] Christoforos I Mavrogiannis and Ross A Knepper. Multi-agent path topology in support of socially competent navigation planning. *The International Journal of Robotics Research*, 2018.

[37] Christoforos I Mavrogiannis and Ross A Knepper. Multi-agent trajectory prediction and generation with topological invariants enforced by hamiltonian dynamics. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, 2018.

[38] Angelia Nedic and Dimitri P Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12(1):109–138, 2001.

[39] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer, 2003.

[40] Anne-Hélène Olivier, Antoine Marin, Armel Crétual, and Julien Pettré. Minimal predicted distance: A common metric for collision avoidance during pairwise interactions between walkers. *Gait Posture*, 36(3):399–404, 2012.

[41] Stéphane Petti and Thierry Fraichard. Safe motion planning in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2215, 2005.

[42] Robert Platt, Russ Tedrake, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems*, 2010.

[43] Boris T Polyak. Introduction to optimization. translations series in mathematics and engineering. *Optimization Software*, 1987.

[44] Martin Rufli, Javier Alonso-Mora, and Roland Siegwart. Reciprocal collision avoidance with motion continuity constraints. *IEEE Transactions on Robotics*, 29(4):899–912, 2013.

[45] Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 2574–2581. IEEE, 2015.

[46] Naum Zuselevich Shor. Minimization methods for non-differentiable functions. In *Springer Series in Computational Mathematics*. Springer, 1985.

[47] NZ Shor. *Nondifferentiable Optimization and Polynomial Problems*, volume 24. Springer Science & Business Media, 1998.

[48] Jamie Snape, Jur van den Berg, Stephen J Guy, and Dinesh Manocha. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4584–4589, 2010.

[49] Jamie Snape, Jur van den Berg, Stephen J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, Aug 2011.

[50] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[51] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research: The 14th International Symposium ISRR*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer, 2011.

[52] Jur van den Berg, Jamie Snape, Stephen J Guy, and Dinesh Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *IEEE International Conference on Robotics and Automation*, pages 3475–3482, 2011.

[53] David Wilkie, Jur van den Berg, and Dinesh Manocha. Generalized Velocity Obstacles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5573–5578, 2009.

[54] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic MPC for model-based reinforcement learning. In *IEEE International Conference on Robotics and Automation*, 2017.

[55] David Wolinski and Ming C Lin. Generalized warpdriver: Unified collision avoidance for multi-robot systems in arbitrarily complex environments. In *Robotics: Science and Systems*, 2018.

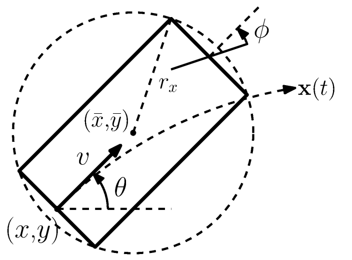


Fig. 7: **Smooth Car**: Definitions of the states of the smooth car (x, y, θ, v, ϕ), in addition to the collision disk $(\bar{x}, \bar{y}, r_x)$. Given zero controls, the car will follow the trajectory labeled $\mathbf{x}(t)$.

Appendix

We present four appendices to our paper. Appendix A provides a detailed case study overviewing how the NH-TTC framework can be applied to a the complex case of a vehicle with smooth car-like dynamics with rear-wheel (off-center) dynamics. Appendix B discusses how to apply NH-TTC to scenarios with dynamic goals. Appendix C gives a detailed comparison to ORCA [51] in a constrained scenario. Appendix D analyzes the performance of the optimization approach underlying our implementation.

A. Case Study: Smooth Car

We define a smooth car by a 5d state space in which each state is represented by 2d position, orientation, linear velocity, and steering angle as $\mathbf{x} = (x, y, \theta, v, \phi)$. Figure 7 shows the relationships between these states. To make the velocity and the steering angle of the car vary continuously over time, we define a 2d control $\mathbf{u} = (a, \psi)$ that represents the car's linear acceleration and the rate of change of the steering angle. In addition to constraints on a and ψ, we also impose state constraints on v and ϕ. As described in section V-C, we introduce the following functions to help enforce the state constraints:

$$k_a(v, a) = \begin{cases} 1 & \text{if } |v| > v_{max} \text{ and } a \cdot v > 0 \\ \frac{1}{100} & \textit{otherwise} \end{cases}$$
$$k_\psi(\phi, \psi) = \begin{cases} 1 & \text{if } |\phi| > \phi_{max} \text{ and } \psi \cdot \phi > 0 \\ \frac{1}{100} & \textit{otherwise} \end{cases} \tag{26}$$

Using these functions, we can define the continuous time dynamics of the system as:

$$\begin{aligned} \dot{x} &= v\cos(\theta) \qquad \dot{y} = v\sin(\theta) \qquad \dot{\theta} = v\tan(\phi)/L \\ \dot{v} &= k_a(v, a)\ a \quad \dot{\phi} = k_\psi(\phi, \psi)\ \psi \end{aligned} \tag{27}$$

where L is the length of the car. Note that $\dot{v}$ and $\dot{\phi}$ are modified to maintain the velocity and steering angle constraints during forward propagation.

From these dynamics, we can compute the partial derivatives with respect to both the state and the controls (only non-zero derivatives are shown). The partial derivatives with respect to the state are:

$$\begin{aligned} \frac{\partial \dot{x}}{\partial \theta} &= -v\sin(\theta) & \frac{\partial \dot{x}}{\partial v} &= \cos(\theta) \\ \frac{\partial \dot{y}}{\partial \theta} &= v\cos(\theta) & \frac{\partial \dot{y}}{\partial v} &= \sin(\theta) \\ \frac{\partial \dot{\theta}}{\partial v} &= \tan(\phi)/L & \frac{\partial \dot{\theta}}{\partial \phi} &= v/(L\cos^2(\phi)) \end{aligned} \tag{28}$$

and the partial derivatives with respect to the controls are:

$$\frac{\partial \dot{v}}{\partial a} = k_a(v, a) \quad \frac{\partial \dot{\phi}}{\partial \psi} = k_\psi(\phi, \psi) \tag{29}$$

Using the continuous time dynamics and these partial derivatives, we can compute discrete time dynamics via trapezoid integration, and find the partial derivatives necessary for Algorithm 2. The discrete time dynamics are approximated as:

$$\begin{aligned} x_{t+dt} &\approx x_t + \frac{dt}{2}(v_t\cos(\theta_t) + v_{t+dt}\cos(\theta_{t+dt})) \\ y_{t+dt} &\approx x_t + \frac{dt}{2}(v_t\sin(\theta_t) + v_{t+dt}\sin(\theta_{t+dt})) \\ \theta_{t+dt} &\approx \theta_t + \frac{dt}{2L}(v_t\tan(\phi_t) + v_{t+dt}\tan(\phi_{t+dt})) \\ v_{t+dt} &\approx v_t + dt\ a\ k_a(v_t, a) \\ \phi_{t+dt} &\approx \phi_t + dt\ \psi\ k_\psi(\phi_t, \psi) \end{aligned} \tag{30}$$

Using this dynamics function, we can analytically compute the form of the partial derivatives with respect to the state at time t and the controls, as required by Equation 15 (again only showing non-zero elements). First, the partials of the x-component of the state with respect to the controls are:

$$\begin{aligned} \frac{\partial x_{t+dt}}{\partial x_t} &= 1 \\ \frac{\partial x_{t+dt}}{\partial \theta_t} &= -\frac{dt}{2}(v_t\sin(\theta_t) + v_{t+1}\sin(\theta_{t+1})) \\ \frac{\partial x_{t+dt}}{\partial v_t} &= \frac{dt}{2}(\cos(\theta_t) + \cos(\theta_{t+1}) - v_{t+1}\sin(\theta_{t+1})\frac{\partial \theta_{t+1}}{\partial v_t}) \\ \frac{\partial x_{t+dt}}{\partial \phi_t} &= -\frac{dt}{2}v_{t+1}\sin(\theta_{t+1})\frac{\partial \theta_{t+1}}{\partial \phi_t} \\ \frac{\partial x_{t+dt}}{\partial a} &= \frac{dt}{2}(dt\cos(\theta_{t+dt}) - v_{t+1}\sin(\theta_{t+1})\frac{\partial \theta_{t+1}}{\partial a}) \\ \frac{\partial x_{t+dt}}{\partial \psi} &= -\frac{dt}{2}v_{t+1}\sin(\theta_{t+dt})\frac{\partial \theta_{t+1}}{\partial \psi} \end{aligned} \tag{31}$$

Similarly, the partials of the y-component of the state are:

$$\begin{aligned} \frac{\partial y_{t+dt}}{\partial x_t} &= 1 \\ \frac{\partial y_{t+dt}}{\partial \theta_t} &= \frac{dt}{2}(v_t\cos(\theta_t) + v_{t+1}\cos(\theta_{t+1})) \\ \frac{\partial y_{t+dt}}{\partial v_t} &= \frac{dt}{2}(\sin(\theta_t) + \sin(\theta_{t+1}) + v_{t+1}\cos(\theta_{t+1})\frac{\partial \theta_{t+1}}{\partial v_t}) \\ \frac{\partial y_{t+dt}}{\partial \phi_t} &= \frac{dt}{2}v_{t+1}\cos(\theta_{t+1})\frac{\partial \theta_{t+1}}{\partial \phi_t} \\ \frac{\partial y_{t+dt}}{\partial a} &= \frac{dt}{2}(dt\sin(\theta_{t+dt}) + v_{t+1}\cos(\theta_{t+1})\frac{\partial \theta_{t+1}}{\partial a}) \\ \frac{\partial y_{t+dt}}{\partial \psi} &= \frac{dt}{2}v_{t+1}\cos(\theta_{t+dt})\frac{\partial \theta_{t+1}}{\partial \psi} \end{aligned} \tag{32}$$

The partials of the orientation θ_{t+1} are:

$$\begin{aligned}
\frac{\partial \theta_{t+1}}{\partial \theta_t} &= 1 \\
\frac{\partial \theta_{t+1}}{\partial v_t} &= \frac{dt}{2L}(\tan(\phi_t) + \tan(\phi_{t+1})) \\
\frac{\partial \theta_{t+1}}{\partial \phi_t} &= \frac{dt}{2L}(\frac{v_t}{\cos^2(\phi_t)} + \frac{v_{t+1}}{\cos^2(\phi_{t+1})}) \\
\frac{\partial \theta_{t+1}}{\partial a} &= \frac{dt^2\, k_a(v_t, a)}{2L}\tan(\phi_{t+1}) \\
\frac{\partial \theta_{t+1}}{\partial \psi} &= \frac{dt^2\, v_{t+1}\, k_\psi(\phi_t, \psi)}{2L\cos^2(\phi_{t+1})}
\end{aligned} \tag{33}$$

Finally the partials with respect to the velocity v and steering angle ϕ are:

$$\begin{aligned}
\frac{\partial v_{t+1}}{\partial v_t} &= 1 \\
\frac{\partial v_{t+1}}{\partial a} &= dt\, k_a(v_t, a)
\end{aligned} \tag{34}$$

and:

$$\begin{aligned}
\frac{\partial \phi_{t+1}}{\partial \phi_t} &= 1 \\
\frac{\partial \phi_{t+1}}{\partial \psi} &= dt\, k_\psi(\phi_t, \psi)
\end{aligned} \tag{35}$$

We also need to define the function, p, mapping the state, $\mathbf{x}$, to the Euclidean workspace, and its derivatives. Because we are modeling the car from the center of the rear axle, we can minimize the encompassing area of the collision avoidance circle by shifting the collision center to lie on the center of the car rather than on the real axle (Figure 7):

$$\begin{aligned}
\bar{x} &= x + \frac{L}{2}\cos(\theta) \\
\bar{y} &= x + \frac{L}{2}\sin(\theta)
\end{aligned} \tag{36}$$

The non-zero derivatives of the collision disk center are:

$$\begin{aligned}
\frac{\partial \bar{x}}{\partial x} &= 1 & \frac{\partial \bar{y}}{\partial y} &= 1 \\
\frac{\partial \bar{x}}{\partial \theta} &= -\frac{L}{2}\sin(\theta) & \frac{\partial \bar{y}}{\partial \theta} &= \frac{L}{2}\cos(\theta)
\end{aligned} \tag{37}$$

The above derivatives, along with those of the continuous dynamics (Equation 27), and those of the trapezoidal integration (Equations 31-35), fully define the goal cost gradient (Equation 13).

To compute collisions, we also define the radius of the collision disk, assuming a 2-to-1 length to width ratio for the car:

$$r_x = \frac{L\sqrt{5}}{4} \tag{38}$$

Equation 38 together with the continuous dynamics (Equation 27) and the offset collision circle center (Equation 36), allows us to compute the linear continuous collision checks between RK4 integration steps in order to estimate the time to collision with any obstacles in the scene. After computing the time to collision, we can compute the collision cost, using Equation 8, and the collision cost gradient, by combining

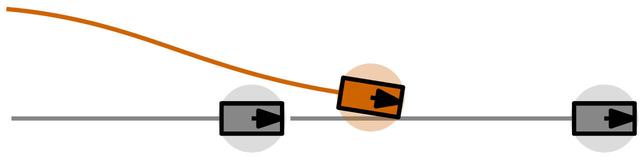


Fig. 8: **Car Following**: A smooth car-like robot, shown in orange, navigates to follow the lead car on the right.

Equations 22, 31-35, and 37. Combining the collision cost gradient with the goal cost gradient gives the full gradient, which can then be used in Algorithm 1 for the control update.

B. Dynamic Goals

The goal cost function defined in Equation 7 focuses on a single static goal. However, in many cases robots can have dynamic goals (e.g., chasing a moving target). In such cases, the robot needs to understand the moving nature of its goal to successfully reach it. Our framework can be easily adapted to support such dynamic goals. Simply greedily minimizing the distance to a dynamic goal can result in oscillations around the goal as it moves. To remedy this oscillatory behavior, we can compute the goal cost, C_{goal}, at multiple temporal points, and then average those costs:

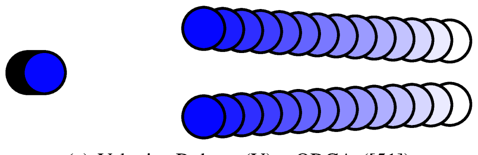
$$C_{dyn\_goal} = \frac{1}{n}\sum_{i=1}^{n} C_{goal}(\mathbf{u}, \mathbf{p}_{goal}^{t_i}) \tag{39}$$

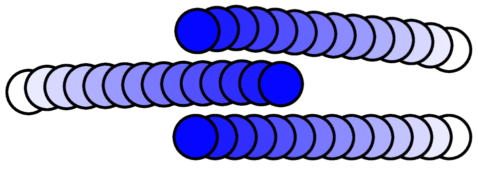
This requires the robot not only to reach the goal, but then to stay on top of it as it keeps moving. In effect, it requires the robot's velocity to synchronize with that of the goal. We show this type of dynamic goal behavior in the following scenario:

Car-following: In this scenario, a smooth car-like robot is attempting to follow a passive car moving at a constant speed, while a third slower moving car has gotten between them. The robot tries to maintain a small following distance behind the lead obstacle, but the trailing obstacle starts too close for the robot to move in between. As such, the car needs to wait for a gap to open between the obstacles before sliding in. See Figure 8 for an overview.

C. Comparison to ORCA: 2 vs 1 Oncoming

In this scenario, two agents standing on one side of an environment have to move toward a third agent that is standing on the opposite side. As shown in Figure 9a, using the ORCA framework to plan for holonomic, velocity-controlled agents results in the lone agent staying put until the other two agents have moved around it. This is due to the fact that ORCA conservatively approximates the set of forbidden velocities with half-planes throwing away too many feasible velocities that the agents could have taken. In contrast, by using our subgradient-based optimization framework, all three agents are able to quickly resolve the collisions and reach to their goals in a timely manner as depicted in Figure 9b. In addition, as compared to ORCA and many of its extensions, such as [2] and [6], our approach plans directly in the agent's control space allowing us to find smooth and collision-free


(a) Velocity Robots (V) - ORCA ([51])


(b) Velocity Robots (V) - NH-TTC

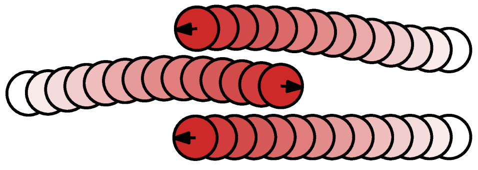

(c) Differential Drive Robots (DD) - NH-TTC

Fig. 9: **2 vs 1 Oncoming**: Two simulated robots move from right to left while a third moves from left to right. All figures show the robots after 4 seconds of simulation. The traces of the robots are shown as colored disks which are light at their initial positions and dark at their current positions. (a) Using the ORCA framework, the standalone robot is reluctant to move forward until the other two robots have walked around it. (b) In contrast, using NH-TTC, all three robots are able to safely reach to their goals in a timely manner. (c) Similar behavior is obtained when NH-TTC plans for differential-drive agents.

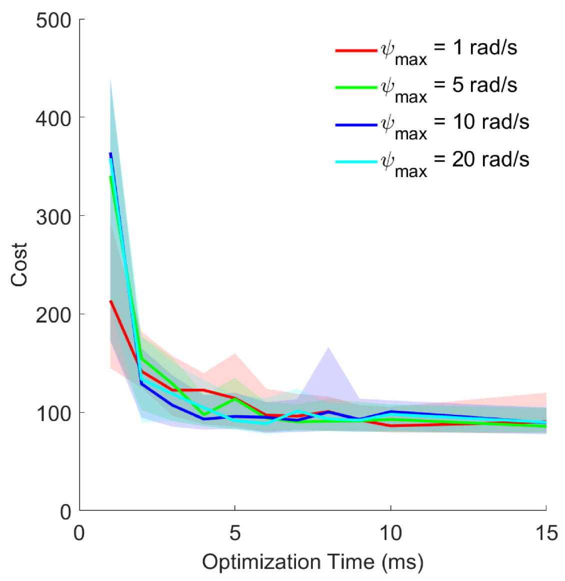


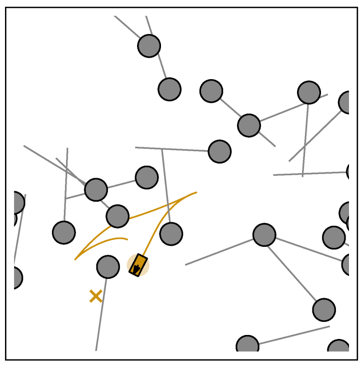
Fig. 10: **Trajectory Cost Over Optimization Time**: Performance analysis of NH-TTC for various control bounds in the car-following scenario shown in Figure 8. The plot depicts the sum of the costs of the taken controls as a function of the available planning time for a variety of maximum steering angle velocities, ψ. Solid lines denote averages over 1,000 runs, with each run lasting 300 frames, and shaded regions denote 90% confidence intervals. For a large variety of bounds, our approach is able to quickly find locally optimal solutions exhibiting low variance.

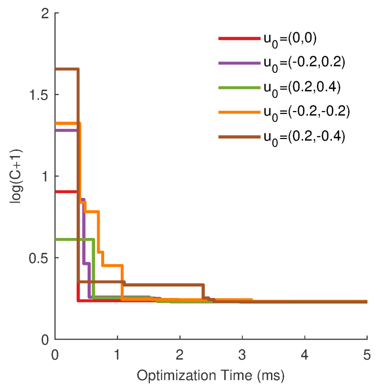
paths for different motion models without being required to cast controls into an intermediate velocity space. As an example, see Figure 9c for trajectories obtained by NH-TTC for differential-drive robots.

D. Optimization Performance Analysis

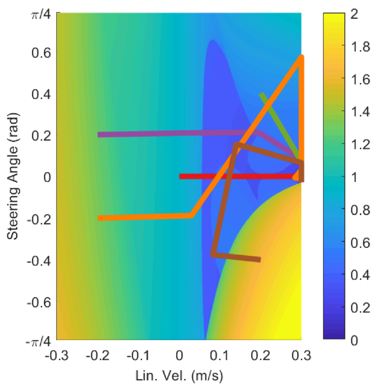
We analyze the performance of our NH-TTC approach in the car-following scenario (see Section VIII-B) by varying the time that the robot has at its disposal to plan for a new control, as well as the maximum steering angle velocity that the car can attain. Figure 10 reports the total cost of the controlled-robot averaged over 1,000 runs for various planning times, ranging from 1 ms to 15 ms, and four distinct control bounds. Overall, as can be seen in the figure, our subgradient descent implementation requires only a small optimization time to start finding low cost trajectories. Using, for example, a very tight constraint of 1 rad/s on the steering angle, NH-TTC is able to find near-optimal solutions within 5 ms of planning per time step. As we increase the control bounds from 1 rad/s to 20 rad/s, the quality of the trajectories returned by NH-TTC remains nearly unchanged while the cost obtained still exhibits low variance across different runs. This highlights the ability of our approach to efficiently search the control space regardless of its size. In contrast, a sampling-based control approach would require more and more time to find good trajectories as the range of the robot's valid steering angle increases, making it impractical for real-time planning settings.

To further show the robustness of our subgradient-based optimization, we test its sensitivity to the initial control, $\mathbf{u}_0$, given as input to Algorithm 1. In particular, we chose a snapshot from the random scenario shown in Figure 11a, where a simple car-like robot is interacting closely with a large number of dynamic obstacles. We ran NH-TTC using five different initial controls while allowing 5 ms of planning time for the car. Across all five runs, NH-TTC completes between 164 and 185 subgradient descent iterations within the 5 ms of the given planning time. Figures 11b-c show the evolution over time of the best cost seen so far and the corresponding control for each of the runs, where the first descent iteration is delayed by 0.3 ms to pre-compute the obstacle trajectories. As highlighted by the cost field in Figure 11c, the car has to solve a complex optimization problem, with many local minima. However, while the initial costs across all of the runs has a large spread, NH-TTC is able to quickly converge to similar near-optimal solutions in only 3 ms.


(a) Scenario



(b) Control Cost Over Time



(c) Cost Field

Fig. 11: **Effect of Initial Control Guess**: (a) A simple car navigating to the goal (the yellow X), avoiding the random velocity agents. (b) The log cost of the best control found so far for a variety of initializations over 5 ms of planning. (c) The cost field corresponding to the simple car's controls. To better show the gradient, values are plotted in log scale with the color corresponding to $\log(C(\mathbf{u}) + 1)$. The evolution of each optimization run is shown in its corresponding color.