2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Weighted Prioritization of Constraints in Optimization-Based Control

Axton Isaly, Sage C. Edwards, Zachary I. Bell, Warren E. Dixon

Abstract—**Optimization-based control laws are an effective method for developing controllers that simultaneously accomplish multiple tasks. Inspired by control barrier function methods, we focus on a framework where the control tasks are encoded as state-dependent inequality constraints on the control input and optimal control inputs satisfying the constraints are synthesized at each state. Some applications feature constraints that cannot be simultaneously satisfied at every state, which leads to infeasibilities in optimization-based control laws. Given a prioritized ranking of each control task, this paper develops an optimization-based control law that is always feasible while satisfying a mathematically-defined notion of the best possible combination of constraints based on an exponential weighting of the priority levels. The control law is implemented by solving two optimization problems in sequence, which for most control systems are a mixed integer linear program and a quadratic program.**

I. Introduction

For controlled dynamical systems, one interpretation of Lyapunov stability principles for controlled dynamical systems is that any given Lyapunov function defines a state-dependent inequality constraint on the control input [1]. Stability is guaranteed when a control law satisfies this constraint at each point in the state space. Using optimization to select inputs that satisfy constraints has seen growing prevalence with the recent interest in control barrier functions (CBF). The CBFs in this work, originating from the zeroing CBFs of [2], are a generalization of control Lyapunov functions that define constraints on the control input which, when satisfied, ensure the forward invariance or asymptotic stability of a set of states. The key novelty is that CBF problems typically focus on complex task specifications defined by multiple CBFs and additional state-dependent input constraints. Such task specifications lead to a feasibility problem where it must be determined whether inputs that simultaneously satisfy all of the constraints exist at each state [3].

Axton Isaly and Zachary I. Bell are with the Munitions Directorate, Air Force Research Laboratory, Eglin AFB, FL, USA. Email: axtonisaly1013@outlook.com, zachary.bell.10@us.af.mil.

Sage C. Edwards and Warren E. Dixon are with the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville FL 32611-6250, USA. Email: {sageedwards, wdixon}@ufl.edu

This research is supported in part by the Air Force Office of Scientific Research grant FA9550-19-1-0169, Air Force Research Laboratory Munitions Directorate Task Order FA8651-21-F-1025, Office of Naval Research grant N00014-21-1-2481, and an appointment to the NRC Research Associateship Program at the Air Force Research Laboratory Munitions Directorate, administered by the Fellowships Office of the National Academies of Sciences, Engineering, and Medicine. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsoring agency.

When multiple constraints are present in a task specification, conflicts can arise that lead to an infeasible problem at certain states. The presence of infeasibility is inherent to some control designs. For example, in adaptive cruise control [4], infeasibility arises when a controlled vehicle can't travel at the desired speed while avoiding a collision with another vehicle. This is an example of a situation where it is beneficial to include potentially infeasible constraints. To manage such control designs, a method is needed that prioritizes constraints based on some user-specified ranking. While the adaptive cruise control solution in [4] achieves prioritization between one high-priority constraint and one low-priority constraint, this paper studies general task specifications with an arbitrary number of priority levels and an arbitrary number of constraints in each level.

In the setting of prioritizing constraints, one approach that has received attention in [5] and [6] takes the view that high-priority constraints are infinitely more important than the lower-priority ones. If a high-priority constraint is infeasible, a solution is found that minimally violates the constraint. Hierarchical quadratic programs (QP) [5] implement this strategy by solving a cascade of QPs for each priority level. However, the lack of weighting between priorities can lead to situations where both high- and low-priority constraints are sacrificed, even though the lower-priority constraints could be satisfied.

This paper presents a prioritization scheme for optimization-based control that is considerate of low-priority constraints by using an exponential weighting between priority levels where each level is weighted double that of the previous one. A method for selecting control inputs that conform to our notion of best-prioritization is developed that uses a dual-step optimization where a mixed integer program is first solved to determine the best set of active constraints, and then the optimal control input is found using a second program where slack variables are added to the inactive constraints to guarantee feasibility of the optimization problem. For many practical applications, the resulting optimization problems are a mixed integer linear program (MILP) and a QP, respectively. Alternative prioritization methods are discussed with theoretical and computational comparisons between the approaches. A Monte-Carlo simulation is conducted for a target-tracking scenario with randomized trajectories. The alternative methods frequently produce inputs that do not satisfy the best prioritization of constraints (less than 60% of the simulation time), leading to between 3% and 19% less time spent tracking targets versus the developed method.

Notation: For vectors $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $\|x\|$ denotes the Euclidean norm, $\|x\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\}$ denotes the infinity norm, and $(x, y) \triangleq \left[x^\top, y^\top\right]^\top$. The shorthand $[d] \triangleq \{1, 2, \ldots, d\}$ is used. For vector-valued function $B : \mathbb{R}^n \to \mathbb{R}^d$, we index each component as $B(x) = \left(B^{(1)}(x), B^{(2)}(x), \ldots, B^{(d)}(x)\right)$ and the inequality $B(x) \leq 0$ means that $B^{(i)}(x) \leq 0$ for all $i \in [d]$.

II. Preliminaries

This section provides a summary of the CBF methods from [3] to motivate the forthcoming development. A constrained differential inclusion with constraints on the state $x \in \mathbb{R}^n$ and input $u \in \mathbb{R}^m$ is modeled as

$$\dot{x} \in F(x, u) \qquad (x, u) \in C_u,$$

where $C_u \subset \mathbb{R}^n \times \mathbb{R}^m$ is the flow set and the flow map $F : \mathbb{R}^n \times \mathbb{R}^m \rightrightarrows \mathbb{R}^n$ is a set-valued mapping that associates every point $(x, u) \in \mathbb{R}^n \times \mathbb{R}^m$ to a set $F(x, u) \subset \mathbb{R}^n$. Differential inclusions can model uncertainty by allowing trajectories to move in a variety of directions for a given state and control input (x, u). The flow set can model physical limitations on the state, and represents arbitrary state-dependent constraints on the control input defined by the mapping $\Psi(x) \triangleq \{u \in \mathbb{R}^m : (x, u) \in C_u\}$. To define the input constraints analytically, we assume the existence of a function $\psi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{k_u}$ such that $\Psi(x) = \{u \in \mathbb{R}^m : \psi(x, u) \leq 0\}$.

CBFs are vector-valued functions $B : \mathbb{R}^n \to \mathbb{R}^d$ that encode control tasks by defining a safe set of states $\mathcal{S} \subset \Pi(C_u)$ as $\mathcal{S} = \{x \in \Pi(C_u) : B(x) \leq 0\}$, where $\Pi(C_u) \triangleq \{x \in \mathbb{R}^n : \exists u \in \mathbb{R}^m \; s.t. \; (x, u) \in C_u\}$ is the projection of the flow set onto the state space. Each component of the CBF defines a constraint on the control input. Assuming that B is continuously differentiable, consider the function $\Gamma : C_u \to \mathbb{R}^d$ with components $\Gamma_i(x, u) \triangleq \sup_{f \in F(x,u)} \langle \nabla B_i(x), f \rangle$, which specifies the worst-case growth of $B_i(x)$ for any possible direction of flow in $F(x, u)$. The set of safety-ensuring control inputs is defined as

$$K_c(x) \triangleq \{u \in \Psi(x) : \Gamma(x, u) \leq -\gamma(x)\}, \tag{1}$$

where $\gamma : \Pi(C_u) \to [-\infty, \infty]^d$ is a performance function that specifies the desired convergence properties (e.g., forward invariance or asymptotic stability). A CBF candidate is called a CBF when the set $K_c(x)$ is nonempty for all x in a neighborhood of the boundary of the safe set. Under the assumptions of [3, Thm. 2], selections of the mapping K_c (i.e., control laws $\kappa : \mathbb{R}^n \to \mathbb{R}^m$ with $\kappa(x) \in K_c(x)$) render the safe set $\mathcal{S}$ forward invariant. Forward invariance is associated with safety because it implies that trajectories of the closed-loop dynamics starting in $\mathcal{S}$ remain in $\mathcal{S}$ for all time.

The set $K_c(x)$ can be considered the feasible set in an optimization problem. Specifically, it is the feasible set for the following optimization-based control law

$$\begin{aligned} \kappa^*(x) \triangleq \; & \underset{u \in \mathbb{R}^m}{\arg\min} \; Q(x, u) \\ & s.t. \; \Gamma(x, u) \leq -\gamma(x), \\ & \psi(x, u) \leq 0, \end{aligned} \tag{2}$$

where $Q : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is a user-defined cost function often selected as $Q(x, u) = \|u - u_{nom}(x)\|^2$ for some nominal control law u_{nom}. It is assumed that the problem in (2) has a unique minimizer for each $x \in \mathbb{R}^n$, while many applications feature additional regularity that leads to the continuity of the minimizer as a function of the state [3, Thm. 3]. The constraints in Γ are affine in the control input if the dynamics are affine in the control input with the form $F(x, u) = F_d(x) + g_d(x) u$, where F_d is a set-valued mapping and g_d is single-valued (i.e., a function). However, the control law is only well-defined if the set $K_c(x)$ is nonempty at every state.

III. Best Prioritization of Constraints

A. Motivation

Given a hierarchy of P control tasks, with level P being the highest priority, each level is associated with a CBF candidate $B_p : \mathbb{R}^n \to \mathbb{R}^{d_p}$. The CBF candidate for each level is vector-valued so that a single level could have multiple CBFs in it. The CBF candidate B_p defines a safe set $\mathcal{S}_p \triangleq \{x \in \mathbb{R}^n : B_p(x) \leq 0\}$ and a set of feasible controls $K_p(x) \triangleq \{u \in \Psi(x) : \Gamma_p(x, u) \leq \gamma_p(x)\}$. Ideally, the safe set defined by all of the CBFs $\mathcal{S}_{[P]} \triangleq \cap_{p=1}^{P} \mathcal{S}_p$ can be rendered forward invariant. However, following the discussion in Section II, this is only possible if the mapping $K_{[P]}(x) \triangleq \cap_{i=1}^{P} K_p(x)$ is nonempty nearby the boundary of $\mathcal{S}_{[P]}$. A test can be performed to determine whether $K_{[P]}(x)$ is nonempty using the tools in [3, Sec. V]. If the nonemptiness of $K_{[P]}(x)$ cannot be verified, compromises in the objectives are required.

Applications that motivate this paper are subject to additional input constraints defined by the set-valued mapping Ψ. Depending on the application, these input constraints may not have the highest priority. For example, it may be necessary to use excessive control effort to avoid collisions by performing an evasive maneuver. We thus generalize our development to consider arbitrary constraints defined by functions $\mathcal{C}_p : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{k_p}$, for $1 \leq p \leq P$, that define state-dependent constraints on the control input as $K_p(x) \triangleq \{u \in \mathbb{R}^m : \mathcal{C}_p(x, u) \leq 0\}$. Note that each priority level, with P being the highest, can have an arbitrary number (k_p) of constraints sharing the same priority.

B. Problem Statement

With the intent of developing a prioritization scheme that is considerate of low-priority levels, the best option for which set of constraints should be solved at a given state $x \in \mathbb{R}^n$ can be described in the following way. Consider the cost function $W(I) \triangleq \sum_{i \in I} 2^{i-1}$, where $I \subset \{1, \ldots, P\}$ is a set representing the inactive levels. An inactive level is a priority level whose constraints are not strictly enforced. The inactive constraints could be removed from the problem entirely, although we will instead make them soft constraints by relaxing them with slack variables. The cost W corresponds to the base 10 representation of a binary number with the i-th digit being set to 1 when level i is inactive (e.g., the number 0110 corresponds to the inactive index set $I = \{2, 3\}$). According to the cost W, making a high priority level inactive is weighted more than deactivating all of the levels below it combined. Thus, the best possible combination of inactive levels is the one that minimizes $W(I)$ subject to the active constraints being feasible. For example, with 3 levels, the best option is clearly for all levels to be active ($W(I) = 0$), the second best option is to relax the lowest priority level 1 ($W(I) = 1$), the third best is to relax level 2 with level 1 active $W(I) = 2$, and so on. For convenience, we define the active constraints for set I as $A(I) = \{1, \ldots, P\} \backslash I$. We then define the best possible set of inactive constraints at state x as

$$\mathcal{I}(x) \triangleq \underset{I \subset \{1,\ldots,P\}}{\arg\min} \sum_{i \in I} 2^{i-1} \quad s.t.\ \cap_{a \in A(I)} K_a(x) \neq \emptyset. \tag{3}$$

Then, the set of best-prioritized control inputs is defined as $\mathcal{P}(x) \triangleq \cap_{a \in A(\mathcal{I}(x))} K_a(x) = \{u \in \mathbb{R}^m : \mathcal{C}_a(x, u) \leq 0,\ \forall a \in A(\mathcal{I}(x))\}$. Note that $\mathcal{P}(x)$ is always nonempty because all of the constraints can be made inactive in (3). Thus, the controls problem motivated in Section III-A can be generalized in the following way.

Problem 1. Consider a prioritized set of constraint functions $\mathcal{C}_p : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{k_p}$, for $1 \leq p \leq P$, with the constraints in level P having the highest priority and where each level can have an arbitrary number (k_p) of constraints sharing the same priority. The constraints define state-dependent input constraints as $K_p(x) = \{u \in \mathbb{R}^m : \mathcal{C}_p(x, u) \leq 0\}$. Find a control law $\kappa : \mathbb{R}^n \to \mathbb{R}^m$ that is a selection of the mapping $\mathcal{P}$ of best-prioritized control inputs, meaning that $\kappa(x) \in \mathcal{P}(x)$ for all $x \in \mathbb{R}^n$.

Remark 1. The weighting scheme in (3) ensures that the satisfaction of higher-priority constraints is never sacrificed to satisfy lower-priority constraints. One could consider a prioritization scheme where it is acceptable to satisfy numerous lower-priority levels at the expense of a higher-priority one, which could be accomplished, for example, by using a cost function $\sum_{i \in I} w^{i-1}$ for some $w \in [1, 2]$. In particular, choosing $w = 1$ weights each priority tier equally, so that the problem in (3) always satisfies the maximum possible number of constraints.

C. Control Law Guaranteeing Best Prioritization

A naive approach to making a selection of $\mathcal{P}$ is to simply attempt to solve numerous optimization problems in the order of their suitability (as defined by the cost function in (3)) until one is found that is feasible. However, this approach requires attempting to solve up to 2^P optimization problems. An alternative optimization scheme can be formulated to solve the same problem. It involves solving two optimization problems. The first is a mixed integer program that solves

$$\begin{aligned}(\mathcal{U}^*(x), d^*(x)) \triangleq & \underset{u \in \mathbb{R}^m,\, d \in \mathbb{R}^P}{\arg\min} \sum_{p=1}^{P} 2^{p-1} d_p \\ & s.t.\ \mathcal{C}_p(x, u) \leq D d_p, \\ & \quad d_p \in \{0, 1\},\ \forall p \in [P]\end{aligned} \tag{4}$$

where D is a large positive constant. The optimization problem in (4) relaxes the constraints in level i when $d_i = 1$. Analogous to the set $\mathcal{I}(x)$ in (3), the solution $d^*(x)$ therefore represents the best possible inactive constraints. Next, we take the binary variables $d^*(x)$ from (4) and solve for the control input using the program

$$\begin{aligned}(\kappa^*(x), \delta^*(x)) \triangleq & \underset{u \in \mathbb{R}^m,\, \delta \in \mathbb{R}^P}{\arg\min} Q(x, u) + k_\delta \|\delta\|^2 \\ & s.t.\ \mathcal{C}_p(x, u) \leq d_p^*(x) \delta_p,\ \forall p \in [P],\end{aligned} \tag{5}$$

where k_δ is a penalization for the slack variables and we have assumed that the minimizer of (5) is unique. In (5), the slack variables δ_i are applied only to the constraints in levels that were made inactive by the first program. The slack variables ensure that the inactive constraints are always feasible, while the cost function minimizes the slack variables for minimal violation of the inactive constraints. Such an application of slack variables is common in the CBF literature [2], [7].

Remark 2. For CBF problems with constraints that are affine in u, (4) is a mixed integer linear program (MILP), and provided the cost is quadratic with the form $Q(x, u) = u^T H(x) u + u^T c(x)$, (5) is a QP. The minimizer $d^*(x)$ in (4) is always unique because the cost function takes a unique value for each combination of binary variables. When (5) is a QP and $H(x)$ is positive definite, the minimizer $(\kappa^*(x), \delta^*(x))$ is unique [8]. The developed programs could be applied in a more general setting where the problems are nonlinear programs, in which case conditions for (5) to have a unique minimizer are well-characterized [9].

The following result shows that the controller κ^* resulting from the multi-step optimization described above is a selection of the mapping $\mathcal{P}$ of best-prioritized control inputs, and the best active constraint set is defined by $d^*(x)$. The only requirement is that the constant D in (4) be large enough to ensure feasibility, as indicated in the following assumption.

Assumption 1. The constant D in (4) is large enough so that the following set is nonempty:

$$\mathcal{P}_D(x) = \left\{ \begin{array}{rl} u \in \mathbb{R}^m : & \mathcal{C}_a(x, u) \leq 0,\ \forall a \in A(\mathcal{I}(x)) \\ & \mathcal{C}_i(x, u) \leq D,\ \forall i \in \mathcal{I}(x) \end{array} \right\}.$$

Theorem 1. *Given $x \in \mathbb{R}^n$, let Assumption 1 hold. Then $\mathcal{I}(x) = \{i \in [P] : d_i^*(x) = 1\}$ and the solution κ^* of (5) is such that $\kappa^*(x) \in \mathcal{P}(x)$. In particular, the feasible set for (5) is equivalent to $\mathcal{P}(x)$.*

Proof: The solution with $d_i(x) = 1$ if and only if $i \in \mathcal{I}(x)$ minimizes the cost function in (4) by definition of $\mathcal{I}(x)$. Under Assumption 1, a feasible input $u \in \mathbb{R}^m$ exists for this choice of binary variables so that (4) correctly identifies the best inactive constraints as $\mathcal{I}(x) = \{i \in [P] : d_i^*(x) = 1\}$. Equivalently, $a \in A(\mathcal{I}(x)) = \{1, \ldots, P\} \backslash \mathcal{I}(x)$ if and only if $d_a^*(x) = 0$. Thus, the solution $\kappa^*(x)$ to (5) is a selection of $\mathcal{P}(x)$ because $\mathcal{C}_a(x, \kappa^*(x)) \leq 0$ whenever $d_a^*(x) = 0$. Since the slack variables can relax the inactive constraints to an arbitrary extent, equivalence between the feasible set of (5) and $\mathcal{P}(x)$ follows. ■

IV. Alternative Methods

The feasible set for the prioritization method in Section III is the largest possible in that it is exactly equal to $\mathcal{P}(x)$. A possible limitation of the developed approach is that the control input is generally discontinuous because the set $\mathcal{P}(x)$ may change abruptly when the set of active constraints changes. In this section, we discuss other methods from the literature and some novel methods that will be used for comparison in the next section. The first two methods produce continuous control laws under some continuity and convexity conditions imposed on the constraints and cost function (see [3, Thm. 3]). However, none of the methods are guaranteed to yield selections of $\mathcal{P}$, thereby not providing the best prioritization according to the exponential weighting scheme developed in this paper.

The following prioritization scheme is discussed in [10]. It consists of giving slack variables priorities relative to each other through constraints. In the following, we drop the optimal slack variables from the left-hand side of the minimization problem for brevity, and assume that the minimizer $\kappa^*(x)$ for the decision variables $u \in \mathbb{R}^m$ is unique.

$$\begin{aligned} \kappa^*(x) \triangleq \underset{u \in \mathbb{R}^m,\, \delta \in \mathbb{R}^P}{\arg\min}\ & Q(x,u) + k_\delta \|\delta\|^2 \\ s.t.\ & \mathcal{C}_p(x,u) \leq \delta_p,\ \forall p \in [P], \\ & \delta_p \leq \delta_{p-1},\ \forall p \in \{2, \ldots, P\}. \end{aligned} \tag{6}$$

The outcome of (6) is that the constraints in higher-priority levels are only relaxed if the ones in lower-priority levels have been relaxed. However, a disadvantage is that certain situations necessitate that higher-priority constraints are relaxed, in which case the lower-priority constraints will be forced to be relaxed. To see this numerically, consider a case where the constraints in levels 4 and 5 are incompatible so that any input satisfying the higher-priority level 5 constraint requires that $\delta_4 = 200$ at a minimum. Necessarily, $\delta_1 \geq \delta_2 \geq \delta_3 \geq 200$ so that the feasible set for (6) contains control inputs that do not satisfy the constraints in levels 1-3, even though the best prioritization in $\mathcal{P}(x)$ could satisfy levels 1-3 and 5 simultaneously.

A more heuristic approach could be considered where the slack variables are prioritized in the cost function of the optimization:

$$\begin{aligned} \kappa^*(x) \triangleq \underset{u \in \mathbb{R}^m,\, \delta \in \mathbb{R}^P}{\arg\min}\ & Q(x,u) + k_\delta \sum_{i=1}^{P} w_c^{i-1} \delta_i^2 \\ s.t.\ & \mathcal{C}_p(x,u) \leq \delta_p,\ \forall p \in [P]. \end{aligned} \tag{7}$$

where $w_c \geq 1$ is a weighting factor for the slack variables. Such a weighting scheme is heuristic for two reasons. First, there is a blending between the control cost Q and the slack variable cost, thus neither objective will be minimized in general. Even if only the slack cost were included in the cost function, it is not possible to guarantee a proper prioritization using continuous slack variables due to the scaling between variables. The method in Section III is inspired by a lexicographic solution [11] to problem (7), where constraining the slack variables to be binary removes the scaling issues in the first step of the solution.

As mentioned in Section I, the hierarchical prioritization scheme of [5] does not aim to satisfy the best configuration of constraints defined by $\mathcal{P}(x)$, but rather aims to relax the high-priority constraints by a minimal amount. However, we include the hierarchical approach to obtain performance comparisons in the next section. In general, the hierarchical programming problem can be represented recursively by a cascade of P optimization problems, which is solved in the order of $p = (P, P-1, \ldots, 1)$,

$$\begin{aligned} \delta_p^*(x) \triangleq \underset{u \in \mathbb{R}^m,\ \delta \in \mathbb{R}}{\arg\min}\ & \delta^2 \\ s.t.\ & \mathcal{C}_p(x,u) \leq \delta \\ & \mathcal{C}_q(x,u) \leq \delta_q^*(x),\ \forall q \in \{p+1, \ldots, P\}. \end{aligned} \tag{8}$$

Analogous to (5), it is beneficial to determine the optimal control input according to some user-defined cost function by solving a final program $\kappa^*(x) \triangleq \arg\min_{u \in \mathcal{P}_\delta(x)} Q(x,u)$, where $\mathcal{P}_\delta(x) \triangleq \{u \in \mathbb{R}^m : \mathcal{C}_p(x,u) \leq \delta_p^*(x),\ \forall p \in [P]\}$. The minimizer κ^* may not be continuous even in the special case of quadratic programming, as discussed in [5, Sec. 4]. It should be noted that the methods in (6) and (7) generally do not produce the optimal choice of slack variables as defined by (8) (i.e., the solution $\kappa^*(x)$ is not necessarily an element of $\mathcal{P}_\delta(x)$).

V. Simulation Study

This section provides numerical comparisons between the best-prioritization method in Section III and the alternative methods in Section IV. The study uses an example dynamical system that models a single controlled agent equipped with a camera with a fixed viewing box that must track a number of moving agents. The moving agents are each assigned a priority level. Because the motivation of this simulation is to demonstrate differences between prioritization schemes, the priority levels of the agents are randomly assigned. A more sophisticated solution to the problem could assign priorities dynamically based on clustering of the agents.

Table I
SIMULATION RESULTS

	Method 1	Method 2	Method 3	Method 4
Solve Time (s)	4.49	2.67	2.76	17.19
Tracking Time (Agent 9)	5.71	3.96	4.41	5.85
Tracking Time (Agent 1)	4.18	3.80	3.81	3.76
Weighted Tracking Time*	2680	1950	2130	2610
Input Difference**	5.05	0.46	0.57	0.60
Time in $\mathcal{P}(x)$ (%)	100	36.0	32.2	58.5
Distance from $\mathcal{P}(x)$	0	20.4	19.1	16.4
Distance from $\mathcal{P}_\delta(x)$	20.5	18.3	4.97	0

* Computed as $\sum_{p=1}^{P-1} 2^{p-1} t_p$, where t_p is the time in seconds spent tracking agent p.

** Computed as the average value of $dt \cdot \sum_{k=1}^{t_f \cdot dt} \|\kappa^*(x_k) - \kappa^*(x_{k-1})\|$ over each run.

A. Model

The leader agent has state $x_L \in \mathbb{R}^2$ with fully actuated dynamics $\dot{x}_L = u$. There are P other agents, each having the dynamics $\dot{x}_p = v_p(t)$, where $v_p : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$ is the agent's unknown velocity. The flow set for this example is $(z, u) \in C_u \triangleq \mathbb{R}^{2(N+1)} \times \mathbb{R}_{\geq 0} \times \mathbb{R}^2$, where $z \triangleq (x_L, x_1, \ldots, x_P, t)$ is a concatenated state vector. The leader agent is equipped with a camera that has a square viewing box of side length $2\bar{d}$, with $\bar{d} = 5$. Thus, the ideal control objective is to design a trajectory for the leader agent that keeps the leader agent within a distance of $\bar{d}$ from all of the remaining agents. This objective is encoded by defining a vector-valued CBF $B_p : \Pi(C_u) \to \mathbb{R}^4$ for each agent $p \in [P]$ as $B_p(z) \triangleq \left((x_p - x_L) - \bar{d}, -(x_p - x_L) - \bar{d}\right)$. An asymptotic tracking constraint is imposed on each agent of the form

$$\bar{\Gamma}_p(z, u) \triangleq \begin{bmatrix} \bar{v}_p - u \\ \bar{v}_p + u \end{bmatrix} \leq -K_b B_p(z), \tag{9}$$

where $\bar{v}_p \geq \|v_p(t)\|$ for all $t \geq 0$ so that $\bar{\Gamma}_p$ is an upper bound of the unknown function $\dot{B}_p(z, u) = (v_p(t) - u, -v_p(t) + u)$. The constants were $K_b = 13$ and $\bar{v}_p = 4$ for every agent p. Whenever the control input is such that $\dot{B}_p \leq \bar{\Gamma}_p \leq -K_b B_p$ along a trajectory of the closed-loop dynamics, asymptotic stability of the set $\mathcal{S}_p \triangleq \{z \in \Pi(C_u) : \|x_p - x_L\|_\infty \leq \bar{d}\}$ is guaranteed (see [2], [3]). Note that asymptotic stability implies forward invariance, meaning that if an agent is being tracked it will continue to be tracked as long as the constraint in (9) is enforced. It can be verified analytically that for a single agent $p \in [P]$, there exist control inputs at every state and for any trajectory v_p that satisfy the four constraints in (9). Thus, the highest-priority agent can always be tracked. However, for an arbitrary number of agents and most trajectories, the leader will not be able to simultaneously track all of the agents. Nonetheless, it is useful to enforce the constraints for all agents so that the leader is aware of the lower-priority agents and tracks them when this objective does not conflict with tracking the highest-priority one.

B. Methods & Results

A Monte-Carlo simulation was conducted with 500 runs to study the difference between five optimization methods, where Method 1 is the best prioritization approach in Section III, Method 2 prioritizes slack variables according to (6), Method 3 weighs slack variables only in the cost function according to (7), and Method 4 is the hierarchical approach in (8). In each of the 500 runs, the leader agent was initialized at the origin, and a set of initial conditions and continuous velocity trajectories v_p were randomly generated for the remaining agents. The same initial conditions and trajectories were used to test all five methods. The initial condition for each agent is selected to ensure each agent is initially visible. A trajectory for the leader agent was computed for each method where the control input was determined at each timestep by the specified control law.

For all methods, a min-norm control law was used by setting the cost function as $Q(x, u) = \|u\|^2$. Some method-specific parameters were $k_\delta = 10^3$ for Methods 1 and 2, and $k_\delta = 10$ and $w_c = 3$ for Method 3. Given this cost function and the control-affine constraints in (9), the optimization problem in (4) is a MILP and the remaining programs are QPs. The simulation was implemented in MATLAB (2019b), while the optimization problems were solved using Gurobi (v10.0) [12]. The simulations were implemented on a workstation running Windows 10 with a 3.2 GHz Intel Core i5-4570 processor and 8 GB of RAM. Each test featured $P = 10$ agents aside from the leader, and was run for $t_f = 10s$ with a timestep of $dt = 0.01s$. As explained in Section V-A, tracking the highest-priority constraints corresponding to Agent 10 are guaranteed to be feasible in this example. Consequently, slack variables were never applied to the constraints corresponding to that agent. Simulation results are tabulated in Table I, and explained in the following paragraph. All statistics are averaged over every run.

The solve time statistic in Table I represents the time spent solving optimization problems (i.e., calling methods from the Gurobi core library) for each control law as a summation over the entire simulation. Because each run included $t_f/dt = 1000$ iterations, the results indicate an average solve time per iteration of approximately 5ms for Method 1, 3ms for Methods 2 and 3, and 17ms for Method 4. The tracking time in Table I represents the total time over the 10-second simulation that was spent tracking the second

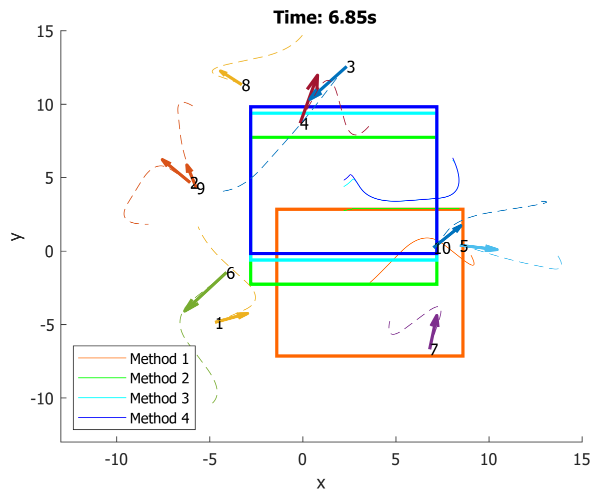

Figure 1. This image depicts a time instant where Method 1 tracked the optimal configuration of agents while the other methods failed. Each method tracked the same agent trajectory. The numbered vectors represent the position (tail of vector) and velocity of an agent, with Agent 10 being the highest priority. The solid lines indicate the leader trajectory (center of box) for the remaining time in the 10s simulation, and the dashed lines indicate the trajectories for the remaining agents. Method 1 was able to keep high-priority Agents 5 and 7 in its viewing box, while the other methods were pulled towards other agents that either could not be tracked or had lower priority.

highest-priority Agent 9 and the lowest-priority Agent 1. Agent 9 is tabulated since Agent 10 was always tracked for the entire 10s duration. The weighted tracking time in Table I scores the agent tracking performance of each method by weighting the higher-priority agents more heavily. In terms of average tracking time over every experiment, Method 1 had a higher tracking time than Methods 2 and 3 on an agent-by-agent basis ranging from a 3% to 19% improvement. Method 4 had a 1.4% higher tracking time for Agent 9, while Method 1 outperformed Method 4 for all other agents by 3% to 5%. As discussed in Section IV, Method 4 is designed to provide minimal relaxation of the second priority level at the expense of the lower levels. The input difference indicates the smoothness of the control input by measuring the change in the input at each timestep. The time in $\mathcal{P}(x)$ in Table I represents the percentage of simulation time that the control input was a selection of the best feasible set $\mathcal{P}(x)$, which is a measure of how well each method prioritized the constraints. The distances from $\mathcal{P}(x)$ and $\mathcal{P}_\delta(x)$ were computed by solving a QP to project the control input from each method onto the respective sets.

C. Discussion

The simulation results show that the best-prioritization method led to improved tracking and significantly better prioritization of lower-priority constraints. Method 1 is the only method with a guarantee of producing the optimal prioritization of constraints. Methods 2-4 selected control inputs from the best-prioritized set $\mathcal{P}(x)$ less than 40% of the time, while Method 4 selected from $\mathcal{P}(x)$ 58.5% of the time. Figure 1 shows a typical example when Method 1 led to better tracking than the other methods. Method 1 was more responsive to changes in the set $\mathcal{P}(x)$, which shifts abruptly when the active constraint set changes. The responsiveness of Method 1 is reflected in large input differences in Table I, indicating a discontinuous control input. Future work will investigate the possibility of obtaining continuous control laws that are selections of the best-prioritized set $\mathcal{P}(x)$. Such an approach would involve finding a subset of $\mathcal{P}(x)$ with better continuity properties (in the sense of set-valued mappings), if such continuous subsets exist.

The computation time for Method 1 was approximately twice that of Methods 2 and 3. This is not surprising because two optimization problems are solved in Method 1. Similarly, the hierarchical programming of Method 4 was significantly slower than the other methods because of the need to solve 11 QPs. The authors of [5] develop a more computationally efficient solution method for hierarchical QPs. Method 1 appears to scale well compared to Method 4. During a 10-simulation run where the number of agents was increased to $P = 100$, the average solve time for Method 1 was 12.6s, 8.1s for Method 2, 2.4s for Method 3, and 163.0s for Method 4.

References

[1] R. A. Freeman and P. V. Kokotovic, *Robust Nonlinear Control Design: State-Space and Lyapunov Techniques*. Boston, MA: Birkhäuser, 1996.

[2] X. Xu, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Robustness of control barrier functions for safety critical control," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 54–61, 2015.

[3] A. Isaly, M. Ghanbarpour, R. G. Sanfelice, and W. E. Dixon, "On the feasibility and continuity of feedback controllers defined by multiple control barrier functions," in *Proc. Am. Control Conf.*, Jun. 2022.

[4] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[5] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[6] N. Somani, M. Rickert, A. Gaschler, C. Cai, A. Perzylo, and A. Knoll, "Task level robot programming using prioritized non-linear inequality constraints," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 430–437.

[7] J. Usevitch, K. Garg, and D. Panagou, "Strong invariance using control barrier functions: A Clarke tangent cone approach," in *Proc. IEEE Conf. Decis. Control*. IEEE, 2020, pp. 2044–2049.

[8] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[9] A. V. Fiacco and Y. Ishizuka, "Sensitivity and stability analysis for nonlinear programming," *Annals of Operations Research*, vol. 27, no. 1, pp. 215–235, 1990.

[10] G. Notomista, S. Mayya, Y. Emam, C. Kroninger, A. Bohannon, S. Hutchinson, and M. Egerstedt, "A resilient and energy-aware task allocation framework for heterogeneous multirobot systems," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 159–179, 2021.

[11] M. J. Rentmeesters, W. K. Tsai, and K.-J. Lin, "A theory of lexicographic multi-criteria optimization," in *Proceedings of ICECCS'96: 2nd IEEE International Conference on Engineering of Complex Computer Systems (held jointly with 6th CSESAW and 4th IEEE RTAW)*. IEEE, 1996, pp. 76–79.

[12] I. Gurobi Optimization, "Gurobi optimizer reference manual (version 10.0)."