Cooperative Collision Avoidance via Proximal Message Passing

Hao Yi Ong[1] and J. Christian Gerdes[1]

***Abstract*— We propose a distributed controller to solve the Cooperative Collision Avoidance problem. We consider a network of vehicles, each with its own dynamic constraints and objective. The problem is to minimize the total network objective function subject to the vehicles' individual constraints and their shared collision avoidance constraints over a given time horizon. The proposed controller, a *proximal message passing* (PMP) algorithm, is iterative: At each iteration, every vehicle passes simple messages to its neighbors and then solves a convex program that minimizes its own objective function and a simple regularization term that only depends on the messages it received in the previous iteration. As a result, the method is completely decentralized and needs no global coordination other than synchronizing iterations. The problems that each vehicle solves can be done extremely efficiently and in parallel. We demonstrate the method on several examples using a model predictive control framework.**

I. Introduction

Trajectory planning, formation flying, and guidance and collision warning for large, complex networks have been major areas of research in recent years. Such networks include unmanned vehicle formations [1], satellite swarms [2], cooperative robots [3], and transportation networks [4]. From a control perspective, structural constraints on information flow constitute one of the major difficulties facing the use of large networks: each agent in the network must act based on limited information and even with global information, determining optimal inputs may be computationally prohibitive. Further, vehicles often have limited sensing, actuation, and computation capabilities, which require the guidance and control algorithms of the networks to be both simple and computationally efficient.

The above guidance and control problem can be captured as a general multivehicle Cooperative Collision Avoidance (CCA) problem, i.e., the problem of finding a trajectory of state-control pairs that satisfy each vehicle's linear dynamics and minimize each vehicle's convex stage-cost functions in a network with nonconvex collision avoidance constraints. In CCA, networked vehicles share obstacle and inter-vehicle collision avoidance constraints with their neighbors, which are encoded in their objective functions. The objective functions further encode operating costs such as fuel consumption and constraints on trajectory deviations and control inputs.

Finding optimal trajectories for CCA is in general a hard, nonconvex problem [5], [6], but can be convexified and solved to obtain approximate solutions. While generic convex optimization techniques and custom interior-point methods exist for these convexified problems, we are interested in distributed, real-time applications using model predictive control (MPC). MPC is a framework for controlling constrained dynamical systems by repeatedly solving a finite horizon optimal control problem. At each repetition, a convex program is generated from current sampled model states and solved to produce an input sequence. A subset of the input sequence is then executed as control input. This type of controller requires very high efficiency, scalability, and simplicity of the algorithm for hundreds to thousands of vehicles in a networked system.

Centralized mixed-integer quadratic programming and sequential convex programming (SCP) approaches have been applied to CCA by Mellinger et al. [7] and Augugliaro et al. [1], respectively, and work well for small vehicle networks. Morgan et al. [2] present an application-specific decentralized approach to an SCP formulation of CCA for small satellites formation flying. Bento et al. [3] present an efficient, parallelizable method, but rely on devising complicated solvers for different constraint and objective functions. In our paper, we propose a simple distributed message-passing algorithm that works for a general class of vehicle models with linear dynamics and scales well to large vehicle networks. The algorithm is derived using the alternating direction method of multipliers (ADMM), which had been explored for large-scale optimization problems by Eckstein and Fukushima [8], [9] and, more recently, Boyd et al. and Kraning et al. [10], [11].

The contributions of this paper are twofold. First, we derive a distributed algorithm by convexifying CCA and developing a message passing algorithm using ADMM to solve CCA to local optimality. Second, we develop and implement an MPC controller example based on our message passing algorithm that allows CCA to be solved in real-time on embedded systems. The rest of the paper is organized as follows: Section II provides the mathematical formulation of CCA, followed by Section III, which presents a method to convexify CCA. We derive the proximal message passing equations for CCA in Section IV, and thereafter discuss our simulations on several example MPC problems in Section V. Some concluding remarks are drawn and future works mentioned in Section VI.

II. Cooperative Collision Avoidance

We begin with an abstract definition of our network model and CCA and the compact notation we use to describe it.

[1]H. Y. Ong and J. C. Gerdes are with the Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 USA haoyi@stanford.edu; cgerdes@stanford.edu

A. Network Model

A network consists of finite, time-varying sets of vehicles $\mathcal{V}(t)$, communication links $\mathcal{E}(t)$, and nets $\mathcal{W}(t)$. In CCA, vehicles in $\mathcal{V}(t)$ are connected to nets through communication links. Subsets of vehicles in $\mathcal{V}(t)$ are linked to a net if these vehicles share collision avoidance constraints. Fig. 1 gives an example network of cars on a highway. Avoidance constraints are either inter-vehicle or vehicle-obstacle collision avoidance requirements, the latter of which are enforced between vehicles in $\mathcal{V}(t)$ and other obstacles. Some obstacles may be moving objects, such as other vehicles that do not operate on the standardized protocol as vehicles in $\mathcal{V}(t)$ do. Notice that a network is a bipartite graph with vehicles representing one set of vertices, nets representing the other set of vertices, and communication links forming the edges.

Each vehicle $v \in \mathcal{V}(t)$ is described by its own set of dynamics equations, and the interactions between itself and other vehicles and obstacles are captured by edges $e \in \mathcal{E}(t)$ between itself and neighboring nets. Each net $w \in \mathcal{W}(t)$ encodes inter-vehicle and vehicle-obstacle collision avoidance constraints shared by its neighboring vehicle nodes. We denote the set of vehicles connected to net w as $\mathcal{V}_w \subseteq \mathcal{V}(t)$. Each vehicle and net are also equipped to receive, compute, and transmit data for navigation purposes.

The time-varying description of our network allows us to take into account changes in the state of each vehicle and thus the relevance of other vehicles and obstacles to its future trajectory (e.g., proximity of vehicles). For instance, a drone that has broken off from a flight formation no longer needs to consider the trajectories of its original fleet.

B. Problem Description

We formulate CCA as the deterministic, discrete-time, finite-horizon optimal control problem

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{v\in\mathcal{V}(t)}\sum_{\tau=1}^{T}\left(\phi_{v,\tau}(x_{v,\tau},u_{v,\tau})+\psi_{v,\tau}(x_{v,\tau},u_{v,\tau})\right) \\ & \displaystyle+\sum_{w\in\mathcal{W}(t)}\sum_{\tau=1}^{T}\mu_{w,\tau}(x_{\mathcal{V}_w,\tau}) \\ \text{subject to} & x_{v,0}=x_v^{\text{init}}, \\ & x_{v,\tau+1}=A_{v,\tau}x_{v,\tau}+B_{v,\tau}u_{v,\tau}+c_{v,\tau}, \\ & \forall v\in\mathcal{V}(t),\ \ \tau=0,\ldots,T-1, \end{array} \tag{1}$$

where $x_{v,\tau} \in \mathbf{R}^n$ and $u_{v,\tau} \in \mathbf{R}^m$ are the state and control variables, respectively, for each vehicle $v \in \mathcal{V}(t)$, and $x_{\mathcal{V}_w,\tau}$ are the concatenated state variables for the vehicles in $\mathcal{V}_w$. Note that we use τ to denote discrete time steps, where T is the given time horizon, and t to denote the actual time at which the problem is solved. The objective function is split into convex quadratic parts $\phi_{v,\tau}$, convex non-quadratic parts $\psi_{v,\tau}$, and nonconvex parts $\mu_{w,\tau}$ that encode the avoidance constraints. The convex quadratic terms $\phi_{v,\tau} : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ have the form

$$\phi_{v,\tau}(x,u) = \frac{1}{2}\begin{bmatrix} x_{v,\tau} \\ u_{v,\tau} \\ 1 \end{bmatrix}^T \begin{bmatrix} Q_{v,\tau} & S_{v,\tau} & q_{v,\tau} \\ S_{v,\tau}^T & R_{v,\tau} & r_{v,\tau} \\ q_{v,\tau}^T & r_{v,\tau}^T & 0 \end{bmatrix} \begin{bmatrix} x_{v,\tau} \\ u_{v,\tau} \\ 1 \end{bmatrix}$$

where

$$\begin{bmatrix} Q_{v,\tau} & S_{v,\tau} \\ S_{v,\tau}^T & R_{v,\tau} \end{bmatrix} \succeq 0$$

(i.e., symmetric positive definite). The convex non-quadratic terms $\psi_{v,\tau} : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$ are closed convex proper (CCP) (see [12]). These terms can encode convex constraints on the states and control inputs by indicator functions of CCP sets. For instance,

$$\psi_{v,\tau}(x_{v,\tau},u_{v,\tau}) = I_{\mathcal{C}_{v,\tau}}(x_{v,\tau},u_{v,\tau}) = \begin{cases} 0, & (x_{v,\tau},u_{v,\tau}) \in \mathcal{C}_{v,\tau} \\ \infty, & \text{otherwise} \end{cases}$$

enforces the convex state-control constraint $(x_{v,\tau},u_{v,\tau}) \in \mathcal{C}_{v,\tau}$ in (1). The nonconvex functions $\mu_{w,\tau} : \mathbf{R}^{n|\mathcal{V}_w|} \to \mathbf{R} \cup \{\infty\}$ are indicator functions on sets of the form

$$\left\| p_{i,\tau} - p_{j,\tau} \right\|_2 \geq D_{ij}, \tag{2}$$

where $p_{i,\tau}$ is the position of vehicle i, $p_{j,\tau}$ is that of another vehicle or obstacle j, and D_{ij} is the minimum separation required between i and j at time τ. The position variables of vehicles $v \in \mathcal{V}(t)$ are subsumed under the state variables $x_{v,\tau}$. Note that each $\mu_{w,\tau}$ is an indicator function on the intersection of a collection of sets of the form (2) corresponding to the net w. Each $\mu_{w,\tau}$ can also be a sum of indicator functions on the same sets.

For cases where a ball poorly estimates the forbidden region of a vehicle (e.g., a car is much longer on one side), multiple overlapping balls can be defined to cover it. Another way is to define minimum covering ellipsoids around the forbidden region, resulting in constraints of the form $\left\| p_{i,\tau} - p_{j,\tau} \right\|_E \geq D_{ij}$, where E parameterizes the ellipsoid. Other reasonable approximations include the ℓ_1-norm (box).

As formulated, CCA is a nonconvex optimization problem. To solve it efficiently, we convexify it using the concave-convex procedure to obtain a convex problem that can be solved approximately. This is shown in the next section.

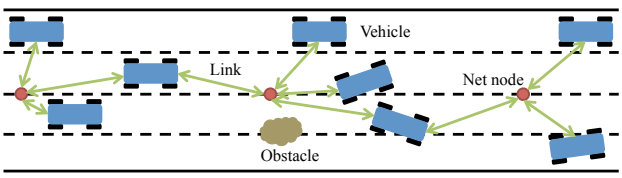

Fig. 1. A network of cars on a highway, with vehicles linked to fixed nets that represent communication relays/transponders that connect with cars within some threshold communication distance.

C. Notation

In subsequent sections, x_v, $x_{\mathcal{V}_w}$, and x denote the concatenated states (i.e., trajectories) for vehicle v, vehicles in net w, and all vehicles in $\mathcal{V}(t)$. Likewise, u_v and u denote the concatenated controls for vehicle v and all vehicles in $\mathcal{V}(t)$. We define

$$\begin{aligned} \mathcal{D}_v = \{(x_v,u_v) \mid\ & x_{v,0} = x_v^{\text{init}}, \\ & x_{v,\tau+1} = A_{v,\tau}x_{v,\tau} + B_{v,\tau}u_{v,\tau} + c_{v,\tau}, \\ & \tau = 0,\ldots,T-1\}, \end{aligned}$$

which is the set of state-control pairs that satisfy the dynamics of (1), and we use $I_{\mathcal{D}_v}$ to denote the indicator function of the set $\mathcal{D}_v$, which is CCP. We further define

$$\phi_v(x_v,u_v)=\sum_{\tau=1}^{T}\phi_{v,\tau}(x_{v,\tau},u_{v,\tau}),$$
$$\psi_v(x_v,u_v)=\sum_{\tau=1}^{T}\psi_{v,\tau}(x_{v,\tau},u_{v,\tau}),$$
$$\mu_w(x_{\mathcal{V}_w})=\sum_{\tau=1}^{T}\mu_{w,\tau}(x_{\mathcal{V}_w,\tau}),$$

and compactly write each vehicle's objective function as

$$f_v(x_v,u_v)=\phi_v(x_v,u_v)+\psi_v(x_v,u_v)+I_{\mathcal{D}_v}(x_v,u_v).$$

With this notation, CCA can be compactly expressed as

$$\text{minimize}\quad \sum_{v\in\mathcal{V}(t)} f_v(x_v,u_v)+\sum_{w\in\mathcal{W}(t)}\mu_w(x_{\mathcal{V}_w}).$$

III. Convex-Concave Procedure

As formulated in (1), CCA is nonconvex due to the avoidance constraints (2). To address this, we express the avoidance constraints as "difference of convex functions" in the form $f-g\le 0$, with f and g convex, and locally solve CCA by the convex-concave procedure (CCCP). Our resulting algorithm is iterative: At each iteration, it uses an estimate of the solution from the previous iteration to approximate the nonconvex constraints and solves the convexified problem. The initial guess is randomly made or warm started.

A. Convexification

Following CCCP, we replace the concave part of the original constraint (2) with the affine approximation

$$\left\|p_{i,\tau}^k-p_{j,\tau}^k\right\|_2+g^T\left(\left(p_{i,\tau}-p_{j,\tau}\right)-\left(p_{i,\tau}^k-p_{j,\tau}^k\right)\right)\ge D_{ij},$$

where

$$g=\left(p_{i,\tau}^k-p_{j,\tau}^k\right)\Big/\left\|p_{i,\tau}^k-p_{j,\tau}^k\right\|_2 \tag{3}$$

is the subgradient of the approximation at iteration k of the CCCP procedure. With the defined subgradient, the affine approximation can be simplified further, giving

$$g^T\left(p_{i,\tau}-p_{j,\tau}\right)\ge D_{ij}. \tag{4}$$

Since g is a subgradient of (2) at $\left(p_{i,\tau}^k,p_{j,\tau}^k\right)$,

$$\left\|p_{i,\tau}-p_{j,\tau}\right\|_2\ge g^T\left(p_{i,\tau}-p_{j,\tau}\right)\ge D_{ij}.$$

In other words, the convexified constraint is more restrictive than the original one, and a single iteration of CCCP would result in a solution feasible to (1). Conceptually, the ball that defines the forbidden region is approximated by a halfspace tangent to the ball and perpendicular to the line segment connecting the nominal positions of i and j. For an ellipsoidal or an ℓ_1-norm box forbidden region proposed in II-B, g would simply be replaced with a subgradient to the ellipsoid or box.

B. Iterative Algorithm

With the convexified constraints, we iteratively solve the approximate convex problem

$$\text{minimize}\quad \sum_{v\in\mathcal{V}(t)} f_v(x_v,u_v)+\sum_{w\in\mathcal{W}(t)} g_w^k(x_{\mathcal{V}_w}), \tag{5}$$

where $g_w^k:\mathbf{R}^{n|\mathcal{V}_w|T}\to\mathbf{R}\cup\{\infty\}$ are indicator functions on convex sets in the form of (4) for each net. These convex sets are approximated by estimates of the solution $x_{\mathcal{V}_w}^k$ at the k^{th} iteration of CCCP.

C. Convergence and Stopping Criteria

Because CCCP approximates our nonconvex constraints with conservative halfspace bounds, we could exclude feasible solutions. At worst, the convexification may turn a feasible problem infeasible. To alleviate this problem, we can replace hard constraints with penalty terms for constraint violation. This way, even if a single iteration's solution were infeasible, we get the best infeasible solution, and future iterations can still converge to feasible solutions under new constraints. In general, however, CCCP has been shown to converge to locally optimal solutions [13], [14]. In practice, CCCP has been used extensively in solving many nonconvex programs that appear in machine learning and observed to converge very quickly [15].

For our stopping criterion, we use the convergence of objective values between iterations of (5) to

$$\epsilon_{\text{CCCP}}^{abs}\sqrt{(n+m)T}+\epsilon_{\text{CCCP}}^{rel}\left|f_{\text{obj}}^k\right|. \tag{6}$$

IV. Proximal Message Passing

We develop proximal message passing, a decentralized optimization method that efficiently solves CCA across every vehicle in the network *within* each CCCP iteration. We derive the message passing equations by reformulating (5) using the alternating direction method of multipliers (ADMM) and then simplifying the resulting equations. ADMM is a *decomposition-coordination* technique in which the solutions to small local subproblems are coordinated to find a solution to a large global problem. We leverage ADMM and the decomposable structure of CCA to arrive at our message passing equations. The reader is referred to Boyd et al.'s discussion of distributed optimization in [10] for a thorough review of ADMM.

A. ADMM and the Proximal Message Passing Algorithm

We first cast the k^{th} CCCP iteration of CCA in consensus form:

$$\begin{aligned}&\text{minimize}\quad \sum_{v\in\mathcal{V}(t)} f_v(x_v,u_v)+\sum_{w\in\mathcal{W}(t)} g_w^k(\tilde{x}_{\mathcal{V}_w})\\&\text{subject to}\quad x=\tilde{x}.\end{aligned} \tag{7}$$

Following [10]'s notation, we form the augmented Lagrangian

$$\begin{aligned}L_\rho(x,u,\tilde{x},y)=&\sum_{v\in\mathcal{V}(t)} f_v(x_v,u_v)+\sum_{w\in\mathcal{W}(t)} g_w^k(\tilde{x}_{\mathcal{V}_w})\\&+(\rho/2)\left(\|x-\tilde{x}+y\|_2^2\right),\end{aligned} \tag{8}$$

with the ADMM parameter ρ and scaled dual variable $y = y_x/\rho : \mathbf{R}^{n(T)|\mathcal{V}(t)|}$, where $y_x : \mathbf{R}^{n(T)|\mathcal{V}(t)|}$ is the dual variable associated with the state consistency constraints. Notice that we can split the quadratic regularizer in (8) across either vehicles or nets:

$$\begin{aligned}\|x-\tilde{x}+y\|_2^2 &= \sum_{v\in\mathcal{V}(t)} \|x_v-\tilde{x}_v+y_v\|_2^2 \\ &= \sum_{w\in\mathcal{W}(t)} \|x_{\mathcal{V}_w}-\tilde{x}_{\mathcal{V}_w}+y_{\mathcal{V}_w}\|_2^2,\end{aligned}$$

The resulting ADMM algorithm is thus given by the iterations

$$\begin{aligned}\left(x_v^{\kappa+1},u_v^{\kappa+1}\right) &:= \underset{(x_v,u_v)}{\operatorname{argmin}}\Big(f_v\left(x_v,u_v\right)+(\rho/2) \\ &\qquad \|x_v-\tilde{x}_v^{\kappa}+y_v^{\kappa}\|_2^2\Big),\ \ v\in\mathcal{V}(t), \\ \tilde{x}_{\mathcal{V}_w}^{\kappa+1} &:= \underset{\tilde{x}_{\mathcal{V}_w}}{\operatorname{argmin}}\Big(g_w^k\left(\tilde{x}_{\mathcal{V}_w}\right)+(\rho/2) \\ &\qquad \left\|x_{\mathcal{V}_w}^{\kappa}-\tilde{x}_{\mathcal{V}_w}+y_{\mathcal{V}_w}^{\kappa}\right\|_2^2\Big),\ \ w\in\mathcal{W}(t), \\ y_{\mathcal{V}_w}^{\kappa+1} &:= y_{\mathcal{V}_w}^{\kappa}+\left(x_{\mathcal{V}_w}^{\kappa}-\tilde{x}_{\mathcal{V}_w}^{\kappa}\right),\ \ w\in\mathcal{W}(t),\end{aligned}$$

where the first step is carried out in parallel by all vehicles, and then the second and third steps are carried out in parallel by all nets. Note that we use κ to denote the ADMM iteration index, as opposed to the CCCP iteration index k.

Since g_w^k is an indicator function for each net w and there is no coupling between variables at each time step, the second step reduces to T parallel projections onto polyhedra (i.e., (4)). We can thus rewrite the second step as

$$\tilde{x}_{\mathcal{V}_w,\tau}^{\kappa+1} := \prod_{\mathcal{C}_{w,\tau}^k}\left(x_{\mathcal{V}_w,\tau}^{\kappa}+y_{\mathcal{V}_w,\tau}^{\kappa}\right),\ \ w\in\mathcal{W}(t),\ \ \tau=1,\ldots,T,$$

where $\mathcal{C}_{w,\tau}^k$ is the convex set that encodes the constraints of net w at the τ^{th} time step, on the k^{th} CCCP iteration.

Assuming that the objective functions are CCP, we can simplify the above to yield *proximal message passing*:

1) *Prox controller updates.*

$$\left(x_v^{\kappa+1},u_v^{\kappa+1}\right) := \mathbf{prox}_{f_v,\rho}\left(\tilde{x}_v^{\kappa}-y_v^{\kappa}\right),\ \ v\in\mathcal{V}(t).$$

2) *Avoidance projection updates.*

$$\tilde{x}_{\mathcal{V}_w,\tau}^{\kappa+1} := \prod_{\mathcal{C}_{w,\tau}^k}\left(x_{\mathcal{V}_w,\tau}^{\kappa}+y_{\mathcal{V}_w,\tau}^{\kappa}\right),\ \ w\in\mathcal{W}(t),\ \ \tau=1,\ldots,T.$$

3) *Scaled price updates.*

$$y_{\mathcal{V}_w}^{\kappa+1} := y_{\mathcal{V}_w}^{\kappa}+\left(x_{\mathcal{V}_w}^{\kappa}-\tilde{x}_{\mathcal{V}_w}^{\kappa}\right),\ \ w\in\mathcal{W}(t),$$

where the prox function for a function h is defined as

$$\mathbf{prox}_{h,\rho}(\theta) = \underset{\xi}{\operatorname{argmin}}\left(h(\xi)+(\rho/2)\|\theta-\xi\|_2^2\right),$$

and is guaranteed to exist when h is CCP.

Our message passing algorithm thus alternates between evaluating the prox functions (in parallel) on each vehicle and computing the avoidance projection and price updates on each net. As long as each vehicle has access to the avoidance projection and price update variables for the nets it shares with its neighbors, this algorithm is completely decentralized. Then, our algorithm consists of each vehicle planning its own trajectory and broadcasting its plans to its neighbors.

B. Convergence and Stopping Criteria

The primal and dual residuals for (7) are

$$r^{\kappa} = x^{\kappa}-\tilde{x}^{\kappa},\quad s^{\kappa} = \rho\left(\tilde{x}^{\kappa}-\tilde{x}^{\kappa-1}\right).$$

A suitable stopping criterion is

$$\|r^{\kappa}\|_2 \le \epsilon^{\text{pri}},\quad \|s^{\kappa}\|_2 \le \epsilon^{\text{dual}},$$

where $\epsilon^{\text{pri}} > 0$ and $\epsilon^{\text{dual}} > 0$ are tolerances for primal and dual feasibility, respectively. These tolerances can be fixed values or assigned to scale with the problem size and typical variable values [11]:

$$\begin{aligned}\epsilon^{\text{pri}} &= \epsilon^{\text{abs}}\sqrt{(n+m)\,T}+\epsilon^{\text{rel}}\max\left\{\|x^{\kappa}\|_2,\|\tilde{x}^{\kappa}\|_2\right\}, \\ \epsilon^{\text{dual}} &= \epsilon^{\text{abs}}\sqrt{(n+m)\,T}+\epsilon^{\text{rel}}\|y^{\kappa}\|_2,\end{aligned} \tag{9}$$

where $\epsilon^{\text{abs}} > 0$ and $\epsilon^{\text{rel}} > 0$ are absolute and relative tolerances, respectively.

Provided that a feasible solution exists and that all objective functions are CCP, we have (see [10] for proof):

1) *Residual convergence.* $r^{\kappa} \to 0$ as $\kappa \to \infty$, i.e., the iterates approach consensus.
2) *Objective convergence.* $f\left(x^{\kappa},u^{\kappa}\right)+g\left(\tilde{x}^{\kappa}\right) \to f_0^{\star}$ as $\kappa \to \infty$, i.e., the objective function of the iterates approaches the optimal value $f_0^{\star}$.
3) *Dual variable convergence.* $y^{\kappa} \to y^{\star}$ as $\kappa \to \infty$, i.e., the dual variables approach the dual optimal point $y^{\star}$.

Note that convergence is to the optimal solution of (7), the convexified problem in each CCCP iteration, as opposed to the globally optimal solution of (1). In practice, ADMM-based approaches converge to modest accuracy (three significant figures) within tens of iterations, which is sufficient for control purposes. Our algorithm can also be augmented with other methods to produce high accuracy solutions [16].

C. Choice of ρ

The value of the ADMM parameter ρ can greatly affect the convergence rate of the message passing algorithm. To the best of our knowledge, there are no known methods for choosing ρ optimally *a priori*, except in special cases [17], [18]. Good methods for picking ρ for offline and online applications are discussed in [10]. We ran trials with a range of ρ values on small networks and picked the one that yielded the best convergence rate to use for larger networks.

D. Implementation of updates

1) Proximal functions: Each vehicle is responsible for implementing its proximal function. In general, evaluating the proximal function requires solving a linear-convex optimal control problem, for which many efficient methods are available. Examples include operator splitting methods [19], [20] and custom interior-point methods [21], [22].

2) Projections and pricing: Projecting onto a polyhedron is in general equivalent to solving a quadratic program and thus cannot be solved analytically. We can apply the same methods used to evaluate proximal functions.

As the number of affine inequalities of the form (4) increases quadratically with the number of vehicles and obstacles, the number of overlapping forbidden halfspaces may also increase. In this case, pruning redundant inequalities may increase computational efficiency. In our numerical examples in the following section, we achieved good results even without additional pruning. Additionally, if $p \neq x$ (i.e., there are other state variables than position), each vehicle v only needs to broadcast p_v. Similarly, the only price variables needed are those associated with the *position* consistency constraints. This decreases the amount of data needed for broadcasting and thus the chance of corruption and the amount of communication bandwidth required, as well as the complexity of the prox and projection operations.

In practice, the solution would be affected by how quickly and reliably packets can be delivered. To account for this, solutions from previous iterations could be used in case of delays, but may cause infeasibility. The penalty formulation discussed in III-C may alleviate this, but a better approach may be to robustify our algorithm to cope with asynchronous, unreliable communications in a way similar to [23].

V. Numerical Examples

We illustrate the speed and scaling of our algorithm with a range of examples. These examples are intentionally kept simple for illustration purposes, but can easily be extended with more refined networks and vehicle models. We present our vehicle model, our problem cast as CCA using an MPC framework, and our network generation method. We then discuss the implementation and simulation results.

A. Vehicle Model

We consider a kinematic bicycle model of the car, derived using its longitudinal symmetry and assuming no tire slip angle. With reference to Fig. 2, the model is described by

$$\dot{p_x} = v\cos(\theta), \quad \dot{p_y} = v\sin(\theta), \quad \dot{\theta} = \frac{v}{L}\tan(\delta), \quad (10)$$

where (p_x, p_y) is the center of the rear axle, θ is the vehicle heading with respect to the global horizontal axis (positive counter-clockwise), v is the (constant) forward speed in the vehicle frame, δ is the steer angle (positive counter-clockwise), and L is the vehicle wheelbase. This results in a nonlinear set of equations with continuous vehicle state $x = (p_x, p_y, \theta)$ and input δ.

To cast our problem as a linear-convex control problem, we Taylor expand (10) about some nominal trajectory $\bar{x}$:

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -v\sin(\bar{\theta}) \\ 0 & 0 & v\cos(\bar{\theta}) \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{v}{L} \end{bmatrix} \delta + \begin{bmatrix} v\cos(\bar{\theta}) + v\bar{\theta}\sin(\bar{\theta}) \\ v\sin(\bar{\theta}) - v\bar{\theta}\cos(\bar{\theta}) \\ 0 \end{bmatrix}.$$

Discretizing the above using a zero-order hold, we get

$$x_{\tau+1} = A_\tau x_\tau + B_\tau \delta_\tau + c_\tau,$$

where A_τ, B_τ, and c_τ are constant dynamics coefficients numerically evaluated at the time period τ.

In the examples, each vehicle tracks a nominal trajectory while maintaining a smooth steering input. The objective function for each vehicle can be written as

$$\sum_{\tau=1}^{T} \left\| p_\tau^{\text{veh}} - \bar{p}_\tau^{\text{veh}} \right\|_2^2 + \lambda \sum_{\tau=0}^{T-2} (\delta_{\tau+1} - 2\delta_\tau + \delta_{\tau-1})^2,$$

with parameter λ for the cost on the mean-square curvature of steer and $p^{\text{veh}} = (p_x, p_y)$. The vehicle's steer inputs are constrained by

$$|\delta_\tau| \leq \delta^{\max}, \qquad |\delta_\tau - \delta_{\tau-1}| \leq \delta^{\text{slew}}. \quad (11)$$

Vehicles are further limited by collision avoidance constraints with other vehicles and obstacles of the form (2). The forbidden regions covering each vehicle and obstacle are modeled as balls of radii R^{veh} and R^{obs} about p^{veh} and p^{obs}, respectively. Convexifying these avoidance constraints, we obtain the affine inequalities

$$g_{vv}^T \left(p_{i,\tau}^{\text{veh}} - p_{j,\tau}^{\text{veh}} \right) \geq 2R^{\text{veh}}, \;\; g_{vo}^T \left(p_{i,\tau}^{\text{veh}} - p_{j,\tau}^{\text{obs}} \right) \geq R^{\text{veh}} + R^{\text{obs}} \quad (12)$$

between objects i and j, where g_{XX} are the subgradients computed from the affine approximations of the forbidden regions at a prior iteration of CCCP as shown in (3).

B. MPC Formulation of CCA

We consider an MPC algorithm that solves CCA for T discrete time steps and repeatedly executes the first input of the control sequence. The MPC optimization problem is

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in \mathcal{V}(t)} \left(\lambda \sum_{\tau=0}^{T-2} (\delta_{v,\tau+1} - 2\delta_{v,\tau} + \delta_{v,\tau-1})^2 + \sum_{\tau=1}^{T} \left\| p_\tau^{\text{veh}} - \bar{p}_\tau^{\text{veh}} \right\|_2^2 \right) + \sum_{w \in \mathcal{W}(t)} g_w^k (x_{\mathcal{V}_w}) \\ \text{subject to} \quad & x_{v,0} = x_v^{\text{init}}, \\ & x_{v,\tau+1} = A_{v,\tau} x_{v,\tau} + B_{v,\tau} \delta_{v,\tau} + c_{v,\tau}, \\ & \forall v \in \mathcal{V}(t), \;\; \tau = 0, \ldots, T-1, \end{aligned}$$

where g_w^k is the indicator function for net w on polyhedra of the form (12) at the k^{th} CCCP iteration.

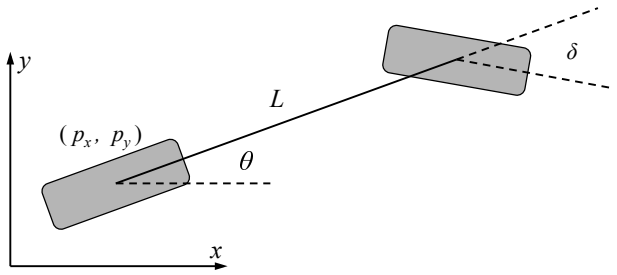

Fig. 2. The kinematic bicycle car model in a global reference frame.

C. Network Topology and Simulation

All networks consisted of a time-invariant set of vehicles $\mathcal{V}$ and time-varying sets of links $\mathcal{E}(t)$ and nets $\mathcal{W}(t)$. Cars and static obstacles were distributed at a ratio of 5:1 randomly on a square of length s^{bdry}, and cars were redistributed if they violated (12). Car speeds were chosen randomly from a uniform distribution: $v \sim \mathcal{U}\left(v^{\min}, v^{\max}\right)$. We dynamically generated nominal trajectories from steer inputs sampled from the normal distribution: $\bar{\delta}_\tau \sim \mathcal{N}\left(\bar{\delta}_{\tau-1}, \delta^{\text{slew}}/3\right)$; trajectories were regenerated if they violated the boundary or the inputs violated (11). Links were dynamically formed (at each MPC time step) by clustering cars and obstacles within a threshold and connecting the car nodes to a net node in each cluster. Thresholds were set at the maximum distance each car can travel in the MPC time horizon of 4 s. Each simulation ran at discrete time periods of $\Delta\tau = 0.1$ s for a total simulation time of T^{sim}. We used the following problem parameters.

$$\begin{array}{lll} \delta^{\max} = 30 \text{ deg} & \delta^{\text{slew}}/\Delta\tau = 20 \text{ deg/s} & \\ v^{\min} = 20 \text{ mph} & v^{\max} = 30 \text{ mph} & L = 3 \text{ m} \\ R^{\text{veh}} = 3 \text{ m} & R^{\text{obs}} = 2 \text{ m} & \lambda = 1 \end{array}$$

D. Implementation

All computation was carried out on a system with a quad-core Intel i7 processor, with clock speed 2.7 GHz and 8 GB of RAM, running Mac OS X. The CCA solver and simulator for all examples were implemented on a single thread in MATLAB and C, which interacted with MATLAB through a MEX-interface. C code for the convex optimization was generated using CVXGEN [24]. When running our message passing algorithm, we report the average time required to reach an accuracy corresponding to ϵ^{abs} and ϵ^{rel} in (9) all set to 10^{-3}. We also report the average time required to reach an accuracy of $\epsilon^{abs}_{\text{CCCP}}$ and $\epsilon^{rel}_{\text{CCCP}}$ in (6) set to 10^{-3} for any CCCP iteration. With these tolerances, the objective value obtained at termination was never more than 1% suboptimal for simulations with $|\mathcal{V}| \leq 10$, as judged by the objective value obtained by running a centralized solver for CCCP iterations with CVX [25], [26], a parser-solver that uses Gurobi [27] (larger problems took CVX too long to run practically). We used $\rho = 1.2$ for the message-passing algorithm.

We have not yet created a fully peer-to-peer, bulk synchronous parallel implementation of our message passing algorithm, but have carefully tracked solve times in our serial implementation to facilitate a first order analysis of such a system. In a peer-to-peer implementation, the prox controller updates occur in parallel across all vehicles followed by avoidance projections and scaled price updates occurring in parallel across all nets. The computation time per iteration is thus the maximum time over all vehicles to evaluate the prox function of their objective, added to the maximum time over all nets to project their trajectories back to feasibility and update their prices. Since the price updates only require a small number of vector operations that can be performed extremely quickly, the determining factor in solve time is in evaluating the prox functions and projections. In our examples, the maximum time taken to evaluate any prox function and projection are 12.6 ms and 17.5 ms, respectively, while average solve times were an order of magnitude smaller.

E. Results

Fig. 3 presents average timing results for solving CCA for a family of examples, using our serial implementation, with networks of size $|\mathcal{V}| = 5,\ 10,\ 25,\ 50,\ 100,\ 250$, and 500. For each network size, we generated and solved 5 network instances to compute average solve times and confidence intervals around those averages. These times do not include time to parse and transform the problem. The solve times per second of simulation were modeled with a log-normal distribution. Fig. 4 shows the trajectories of vehicles in a single instance of a network with $|\mathcal{V}| = 10$. Here, we observe that the vehicles avoid collision by changing direction, driving in the same direction at a safe distance from each other, or crossing over the trajectory of another car after allowing it to pass. Table I summarizes representative results for small, medium, and large networks ($|\mathcal{V}| = 5,\ 50$, and 500), showing that the solve times are constant with network size. The table also reports the maximum and average number of cars connected to a net, $|\mathcal{V}_w|^{\max}$ and $|\mathcal{V}_w|^{\text{avg}}$.

For network instances with $|\mathcal{V}| = 500$ vehicles, the problem has over 160,000 variables, and it took each MPC iteration took an average of 35.0 ms to solve in parallel using proximal message passing. By fitting a line to the proximal message passing runtimes, we find that our implementation empirically scales as $O\left(|\mathcal{V}|^{1.145}\right)$; i.e., solve time is essentially linear in network size. For a peer-to-peer implementation, we expect the runtime of our algorithm to be essentially constant and, in particular, independent of network size. To solve a problem with $|\mathcal{V}| = 500$ with approximately 4 CCCP iterations and 6 proximal message-passing iterations in each CCCP iteration then takes only 35 ms (per MPC iteration every 100 ms). And since the data in each MPC problem at each time period is similar to the data

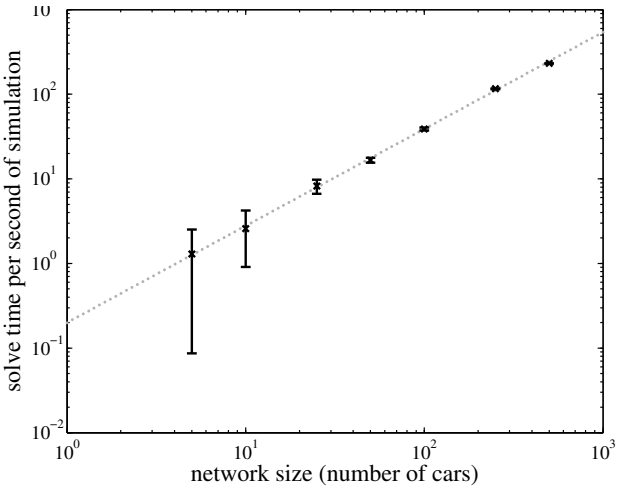

Fig. 3. Average execution times for a family of networks on a single thread. Error bars show 95% confidence bounds. The dotted line shows the least-squares fit to the data, resulting in a scaling exponent of 1.145.

TABLE I

CCA SIMULATION RESULTS FOR THE MPC EXAMPLES

	Small	Medium	Large
Number of cars in network	5	50	500
Total variables	1640	16400	164000
Simulation time T^{sim} (s)	20	20	10
Square side length s^{bdry} (km)	0.32	1.01	3.20
Serial solve time (s)	26	334	2318
Number of nets	1.42	13.1	129.5
Number of net links $\lvert\mathcal{V}_w\rvert^{\text{avg}}$	1.89	2.47	2.49
Max. number of net links $\lvert\mathcal{V}_w\rvert^{\text{max}}$	5	10	15
MPC solve time (ms)	21.6	34.5	35.0
Max. MPC solve time (ms)	44.2	48.7	49.6
CCCP iterations	3.38	3.09	3.45
CCCP solve time (ms)	5.75	9.28	9.33
PMP iterations	2.76	4.46	5.69
PMP solve time (ms)	2.11	2.31	2.21

All data presented are mean values unless otherwise stated.

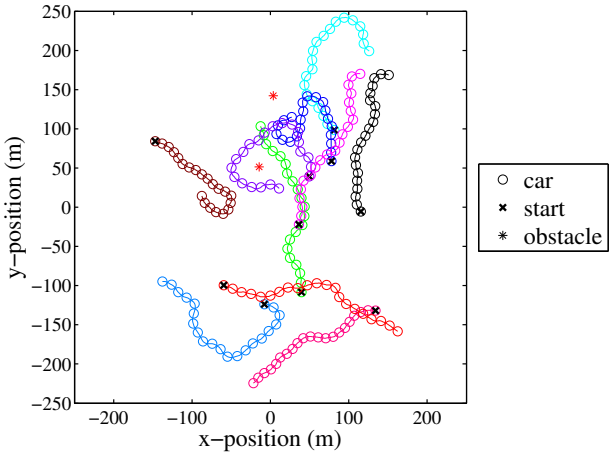

Fig. 4. Car trajectories and obstacles; circles plotted at 1 s intervals.

from the previously solved problem, we can initialize the solver with the solution from the previous problem to obtain a speed-up over the existing cold start implementation.

VI. CONCLUSION AND FUTURE WORK

In this brief, we have demonstrated a message passing algorithm that can solve CCA rapidly. Simple MPC problems can be solved in tens of milliseconds on average, and the algorithm scales well to arbitrarily large networks. Future work will include simulations with more accurate car models (e.g., linear dynamical bicycle models with tires that account for sideslip) and nets that encode constraints and objectives beyond those for collision avoidance. We will also explore methods to robustify our algorithms against network delays.

REFERENCES

[1] F. Augugliaro *et al.*, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Oct. 2012.

[2] D. Morgan *et al.*, "Model predictive control of swarms of spacecraft using sequential convex programming," in *AIAA/AAS Astrodynamics Specialist Conf.*, Aug. 2012.

[3] J. Bento *et al.*, "A message-passing algorithm for multi-agent trajectory planning," in *Advances in Neural Information Processing Systems*, 2013, pp. 521–529.

[4] K. Treleaven *et al.*, "Asymptotically optimal algorithms for pickup and delivery problems with application to large-scale transportation systems," *IEEE Transactions on Automatic Control*, 2012.

[5] J. Reif, "Complexity of the mover's problem and generalizations," in *IEEE Annu. Symp. on Foundations of Computer Science*, Oct. 1979, pp. 421–427.

[6] J. Hopcroft *et al.*, "On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the warehouseman's problem," *The Int. Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.

[7] D. Mellinger *et al.*, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *IEEE Conf. on Robotics and Automation*, May 2012, pp. 477–483.

[8] J. Eckstein, "Parallel alternating direction multiplier decomposition of convex programs," *Journal of Optimization Theory and Applications*, vol. 80, no. 1, pp. 39–62, 1994.

[9] J. Eckstein and M. Fukushima, "Some reformulations and applications of the alternating direction method of multipliers," in *Large Scale Optimization: State of the Art*, 1993, pp. 119–138.

[10] S. Boyd *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 1–122, 2010.

[11] M. Kraning *et al.*, "Dynamic network energy management via proximal message passing," *Foundations and Trends in Optimization*, vol. 1, no. 2, pp. 70–122, 2013.

[12] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ: Princeton Univ. Press.

[13] A. Yuille and A. Rangarajan, "The concave-convex procedure," *Neural Computation*, vol. 15, no. 4, pp. 915–936, 2003.

[14] B. Sriperumbudur and G. Lanckriet, "A proof of convergence of the concave-convex procedure using zangwill's theory," *Neural Computation*, vol. 24, no. 6, pp. 1391–1407, 2012.

[15] A. Yuille, "CCCP algorithms to minimize the bethe and kikuchi free energies: Convergent alternatives to belief propagation," *Neural Computation*, vol. 14, no. 7, pp. 1691–1722, 2002.

[16] J. Eckstein and M. C. Ferris, "Operator-splitting methods for monotone affine variational inequalities, with a parallel application to optimal control," *INFORMS Journal on Computing*, vol. 10, pp. 218–235, 1998.

[17] A. U. Raghunathan and S. D. Cairano, "Optimal step-size selection in alternating direction method of multipliers for convex quadratic programs and model predictive control," in *21st Int. Symp. on Mathematical Theory of Networks and Systems*, Jul. 2014.

[18] E. Ghadimi *et al.*, "On the optimal step-size selection for the alternating direction method of multipliers," *Estimation and Control of Networked Systems*, vol. 3, no. 1, pp. 139–144, 2012.

[19] B. O'Donoghue *et al.*, "A splitting method for optimal control," *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 6, pp. 2432–2442, 2013.

[20] G. Stathopoulos *et al.*, "A hierarchical time-splitting approach for solving finite-time optimal control problems," in *European Control Conf.*, 2013.

[21] A. Domahidi *et al.*, "Efficient interior point methods for multistage problems arising in receding horizon control," in *IEEE Conf. on Decision and Control*, Dec. 2012, pp. 668–674.

[22] J. Mattingley *et al.*, "Receding horizon control: Automatic generation of high-speed solvers," in *IEEE Control Syst. Mag.*, vol. 31, no. 3, Jun. 2011, pp. 52–65.

[23] F. Zanella *et al.*, "Asynchronous Newton-Raphson consensus for distributed convex optimization," in *Estimation and Control of Networked Systems*, 2012, pp. 133–138.

[24] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, no. 1, pp. 1–27, 2012.

[25] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," 2006. [Online]. Available: https://cvxr.com/cvx/

[26] ——, "Graph implementations for nonsmooth convex programs," in *Recent advances in learning and control*, 2008, pp. 95–110.

[27] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2014. [Online]. Available: http://www.gurobi.com