# Efficient algorithms for collision avoidance at intersections

Alessandro Colombo
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA, 02139
acolombo@mit.edu

Domitilla Del Vecchio
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA, 02139
ddv@mit.edu

## ABSTRACT

We consider the problem of synthesising the least restrictive controller for collision avoidance of multiple vehicles at an intersection. The largest set of states for which there exists a control that avoids collisions is known as the maximal controlled invariant set. Exploiting results from the scheduling literature we prove that, for a general model of vehicle dynamics at an intersection, the problem of checking membership in the maximal controlled invariant set is NP-hard. We then describe an algorithm that solves this problem approximately and with provable error bounds. The approximate solution is used to design a supervisor for collision avoidance whose complexity scales polynomially with the number of vehicles. The supervisor is based on a hybrid algorithm that employs a dynamic model of the vehicles and periodically solves a scheduling problem.

## Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory, Scheduling* ; I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*

## General Terms

Algorithms, Theory

## Keywords

Intelligent transportation system, collision, complexity, hybrid, safety, scheduling

## 1. INTRODUCTION

An intelligent transportation system is a set of tightly interacting physical, computational, and communication resources that collaborate to enhance the performance of a transportation network. Of the possible performance metrics safety is, for obvious reasons, considered of primary importance [24]. A number of solutions have been proposed to improve vehicular safety, typically addressing the stability of the longitudinal or lateral dynamics of a vehicle, or reducing the risk of rear-end collisions [1–3]. A much harder problem, both from a technological and a computational point of view, is that of preventing collisions caused by drivers' mistakes when vehicles traverse an intersection. From a control perspective, this problem is ideally solved by determining the largest set of states for which there exists a control that avoids collisions, which is known as the *maximal controlled invariant set* [21]. The set of all controls that make this set invariant is the least restrictive among all control sets that enforce safety. A multi-objective control problem with safety as the primary objective and some other performance metric, for example fuel consumption, as secondary objective, can be solved hierarchically once this control set is determined, by optimizing within the control set. The problem of determining the maximal controlled invariant set for pairs of vehicles at an intersection is solved in [9, 10, 14, 15]. These results exploit the monotonicity of the vehicles' dynamics to derive a computationally efficient control law, but do not apply to more than two vehicles. An algorithm that addresses multi-vehicle collisions, based on abstraction, has been proposed in [5]. Here, the control problem is reduced to the control of a finite automaton, and the maximal controlled invariant set is approximated by a set of allowed states of the automaton. However, the size of the automaton scales exponentially with the size of the problem, thus limiting the applicability to a small number of vehicles. An algorithmic approach to safety enforcing based on time slot assignment, which can handle a larger number of vehicles, is found in [18]. It allows to design a safe control law, but it does not provide means to assess the performance of the result by any metric.

In the context of this paper, membership in the maximal controlled invariant set is determined by solving the following problem.

**Problem A (Safety, informal statement)** *Given the initial state of a set of $n$ agents moving along $n$ different paths crossing at an intersection, determine if there exists an input signal that leads all agents through the intersection avoiding collisions.*

Notice that we assume that agents move along a different paths, and that all paths intersect at a common point, as in Fig. 1. Extensions to our results that allow to handle more complex cases, like multilane intersections with multiple agents on each lane, are currently under study. The approach we follow consists in mapping Problem A onto a

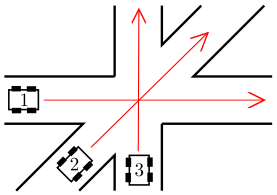


**Figure 1: Example of intersection topology.**

*scheduling problem* [22], where the intersection represents a machine, and the time spent by each agent in the intersection is the length of the job to be executed on the machine. We develop this analogy rigorously, proving that a formal restatement of the above problem is indeed equivalent, in a strong sense, to a modified version of a standard scheduling problem. Through this artifice, we can formally prove that Problem A, for a general class of vehicle dynamics, is NP-complete. This justifies the design of approximate algorithms to solve the problem. We revert again to the scheduling version of Problem A and we leverage results in the operations research literature to devise an approximate solution that has polynomial complexity and is within provable bounds from an optimal solution.

We use the exact solution of Problem A to design the least restrictive safety-enforcing controller, which solves the following problem.

**Problem B (Supervisor, informal statement)** *Given a set of agents as in Problem A, design a supervisor that, given a desired input, returns the desired input unless this will cause a collision at some future time, in which case it returns a safe input.*

The supervised system is hybrid: agents have continuous dynamics, while the allowed inputs are selected at discrete time instants through the solution of the scheduling version of Problem A. We then synthesise the above controller using the approximate solution of Problem A, obtaining an approximate solution of Problem B (Supervisor) within provable bounds of the exact one. The resulting controller enforces safety as long as all agents behave according to the model.

Our solutions exploit algorithms that are well known in the scheduling literature. The proof that these algorithms can be used to solve Problem A, and their use as solution of a control problem, constitute the main contributions of this paper, and are new results to the best of our knowledge.

The paper is organised as follows. In the next section, we introduce some standard concepts of computational complexity theory and operations research which are used in the paper. In Section 3, we formalize Problem A and we reformulate it in terms of a scheduling problem. Then, in Section 4, we prove that the two formulations are equivalent, and we prove complexity results. Section 5 proposes the exact and approximate solution of Problem A. Both solutions are used in Section 6 to provide a solution to Problem B (Supervisor). Finally, in Section 7 we apply the proposed algorithms on a simple model of vehicle dynamics.

## 2. MATHEMATICAL FOUNDATIONS

As mentioned in the Introduction, the results of this paper revolve around the translation of Problem A into a scheduling problem. In order to carry out the proofs, we introduce some standard machinery from the literature on scheduling and computational complexity theory.

Scheduling is the decision-making process of assigning to a number of jobs a *schedule*, that is, a set of execution times, to satisfy given requirements [22]. The standard formalism to describe a scheduling problem was introduced in [13]. It represents a problem by the string $\alpha|\beta|\gamma$, where the field $\alpha$ describes the machine environment (e.g. the number of machines), the field $\beta$ defines the jobs characteristics (such as unequal release dates or constraints on duration), and the field $\gamma$ defines the optimality criterion. This paper is concerned with two closely related scheduling problems: $1|\mathtt{r}_i|\mathtt{Lmax}$, and $1|\mathtt{r}_i, \mathtt{p}_i = 1|\mathtt{Lmax}$ introduced in the following definitions.

**Definition 2.1** ($1|\mathtt{r}_i|\mathtt{Lmax}$) *Given a set of $n$ jobs to be run on a single machine, with release times $\mathtt{r}_i \in \mathbb{R}_+$, deadlines $\mathtt{d}_i \in \mathbb{R}_+$, and durations $\mathtt{p}_i \in \mathbb{R}_+$, find a schedule $\mathtt{T} = (\mathtt{T}_1, ..., \mathtt{T}_n) \in \mathbb{R}_+^n$ such that, for all $i \in \{1, ..., n\}$,*

$$\mathtt{T}_i \geq \mathtt{r}_i,$$

*for all $i \neq j$,*

$$\mathtt{T}_i \geq \mathtt{T}_j \Rightarrow \mathtt{T}_i \geq \mathtt{T}_j + \mathtt{p}_j,$$

*and such that $\mathtt{Lmax} := \max_i(\mathtt{T}_i + \mathtt{p}_i - \mathtt{d}_i)$ is minimized.*

The second problem, $1|\mathtt{r}_i, \mathtt{p}_i = 1|\mathtt{Lmax}$, is identical to the one above, except that $\mathtt{p}_i = 1$ for all jobs.

The above problems are *optimization problems*. Computational complexity theory focuses instead on *decision problems*, problems that have a binary answer in $\{yes, no\}$ [6]. When a problem $P$ returns "*yes*" for an instance $I$ we say that $P$ *accepts* $I$, denoted $I \in P$. Any optimization problem can be cast into a decision problem by imposing a bound on the optimized quantity. Although the solution of a decision problem carries less information than that of the associated optimization problem, analysing the complexity of a decision algorithm is still relevant to the original optimization problem. Indeed, the computational complexity of a decision problem cannot be higher than that of the corresponding optimization problem, since an optimal solution is immediately mapped onto an answer to the decision problem.

We use the notation $DEC(\alpha|\beta|\gamma, \delta)$ to represent the decision problem "does $\alpha|\beta|\gamma$ have a solution with $\gamma \leq \delta$?" Thus, $DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0)$ is stated as follows.

**Definition 2.2** ($DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0)$) *Given a set of $n$ jobs to be run on a single machine, with release times $\mathtt{r}_i \in \mathbb{R}_+$, deadlines $\mathtt{d}_i \in \mathbb{R}_+$, and durations $\mathtt{p}_i \in \mathbb{R}_+$, determine if there exists a schedule $\mathtt{T} = (\mathtt{T}_1, ..., \mathtt{T}_n) \in \mathbb{R}_+^n$ such that, for all $i \in \{1, ..., n\}$,*

$$\mathtt{r}_i \leq \mathtt{T}_i \leq \mathtt{d}_i - \mathtt{p}_i,$$

*and for all $i \neq j$,*

$$\mathtt{T}_i \geq \mathtt{T}_j \Rightarrow \mathtt{T}_i \geq \mathtt{T}_j + \mathtt{p}_j.$$

The statement of $DEC(1|\mathtt{r}_i, \mathtt{p}_i = 1|\mathtt{Lmax}, 0)$ is obtained by adding the constraint $\mathtt{p}_i = 1$ for all $i$.

The concepts of *reducibility* and *equivalence* [6, 19] are used when comparing the complexity of different problems.

**Definition 2.3** *A problem $P1$ is reducible to a problem $P2$ if for every instance $I$ of $P1$ an instance $I'$ of $P2$ can be constructed in polynomial-bounded time, such that $I \in P1 \Leftrightarrow I' \in P2$. In this case, we write $P1 \propto P2$.*

Two problems $P1$ and $P2$ are equivalent, denoted $P1 \simeq P2$, if $P1 \propto P2$ and $P2 \propto P1$.

Problem $DEC(1|r_i|Lmax, 0)$ has been shown to be $NP$-complete by reduction of Knapsack [19]. It can be solved by enumerative algorithms that systematically test all the possible permutations of jobs. Unlike $DEC(1|r_i|Lmax, 0)$, $DEC(1|r_i, p_i = 1|Lmax, 0)$ has an exact $O(n^3)$-time solution, reported in [12] and implemented in Algorithm 1. The solution of the decision problem only requires the binary answer, but Algorithm 1 returns a schedule $T$ along with the binary answer. The information on the schedule is used in the following sections to design an approximate solution to Problem A. The algorithm starts by computing a set of *forbidden regions*, i.e., intervals $F_i \subset \mathbb{R}$ during which no job can be started. Then, it schedules jobs by increasing deadline order, ensuring no job is started during a forbidden region.

**Algorithm 1** Solution of $DEC(1|r_i, p_i = 1|Lmax, 0)$

**procedure** PolynomialTime($\mathbf{r}, \mathbf{d}$)
  **for all** $i \in \{1, \dots, n\}$ **do** $F_i \leftarrow \emptyset$, $c_i \leftarrow \emptyset$
  **end for**
  sort jobs in increasing $r_i$ ($r_1 \le r_2 \le r_3 \le ...$)
  **for** $i = n$ downto 1 **do** ▷ Part I: forbidden regions
    **for all** $j \in \{1, \dots, n\}$ such that $d_j \ge d_i$ **do**
      **if** $c_j = \emptyset$ **then** $c_j \leftarrow d_j - 1$
      **else** $c_j \leftarrow c_j - 1$
      **end if**
      **while** $c_j \in F_k$ for some $F_k$ **do** $c_j \leftarrow \inf(F_k)$
      **end while**
    **end for**
    **if** $i = 1$ or $r_{i-1} < r_i$ **then** $c \leftarrow \min_i(c_i)$
      **if** $c < r_i$ **then return** $\{\emptyset, no\}$
      **end if**
      **if** $c \in [r_i, r_i + 1]$ **then** $F_i \leftarrow [c - 1, r_i]$
      **end if**
    **end if**
  **end for**
  $t \leftarrow 0$
  **for** $i = 1$ to $n$ **do** ▷ Part II: schedule
    $r_{min} \leftarrow \min\{r_j :$ job $j$ has not been scheduled$\}$
    $t \leftarrow \max\{t, r_{min}\}$
    **while** $t \in F_j$ for some $j$ **do** $t \leftarrow \sup(F_j)$
    **end while**
    $j \leftarrow \{i :$ job $i$ has least deadline among those ready at $t\}$
    $T_j \leftarrow t$
    $t \leftarrow t + 1$
  **end for**
  **return** $\{T, yes\}$
**end procedure**

# 3. FORMALIZATION OF PROBLEM A

Consider the system

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \quad \mathbf{y} = h(\mathbf{x}) \tag{1}$$

given by the parallel composition of the dynamics of $n$ (possibly different) agents

$$\dot{x}_i = f_i(x_i, u_i), \quad y_i = h_i(x_i) \tag{2}$$

with $x_i \in X_i \subseteq \mathbb{R}^r$, $u_i \in U_i \subset \mathbb{R}^s$, $y_i \in Y_i \subseteq \mathbb{R}$, and $h_i(x_i)$ continuous in $x_i$. The functional space of the inputs signals $u_i(t)$, $t \in [0, \infty)$, is $\mathcal{U}_i$. The aggregate states, inputs and outputs of all agents are denoted $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{y}$, respectively. The corresponding sets are $\mathcal{U}$, $X$, $U$, and $Y$, respectively. Assume that (1) has unique solutions. This is ensured, for example, if (1) is locally Lipschitz [17], while uniqueness conditions for piecewise smooth systems are reported in [11]. Assume that systems (2) are output-monotone [4] and that the positivity cone of the output $y_i$ is $\mathbb{R}_+$. This means that there exist two convex cones $K_x^i \subset R^r$ and $K_u^i \subset R^s$, such that $x_i(0) - x_i'(0) \in K_x^i$ and $u_i(t) - u_i'(t) \in K_u^i$ for all $t \ge 0$ imply $y_i(t) - y_i'(t) \in \mathbb{R}_+$ for all $t \ge 0$. Assume also that the set $U_i$ is compact and is a partially ordered set [8] with respect to its positivity cone, with a unique maximum, called $u_{i,max}$, and minimum, called $u_{i,min}$. Finally, assume that $\dot{y}_i$ is bounded to a strictly positive interval $[\dot{y}_{i,min}, \dot{y}_{i,max}]$ for all $i$. When the state of agent $i$ at a specified time $t_0$ is $x_i(t_0)$, a trajectory starting at $x_i(t_0)$ or the corresponding output signal at time $t \ge t_0$, with input $u_i$ in $[t_0, t]$, are denoted $x_i(t, u_i, x_i(t_0))$ and $y_i(t, u_i, x_i(t_0))$, respectively. For the sake of brevity, when $t_0 = 0$ we use the notation $x_i(t, u_i)$ and $y_i(t, u_i)$. When the particular input is not important, we write simply $x_i(t)$ and $y_i(t)$, and when time is not important we simply use the variable name without arguments.

We assign to each system $i$ an open interval $(a_i, b_i)$, and we say that two agents $i$ and $j$ *collide* if the outputs of the corresponding systems verify the condition $y_i(t) \in (a_i, b_i)$ and $y_j(t) \in (a_j, b_j)$ at the same time instant $t$. We say that a set of agents $\{i, j, ..\}$ collides if any pair of agents in the set collides. The subset of $Y$ of collision points is called the *bad set*, denoted $B_+$:

$$B_+ := \{\mathbf{y} \in Y : y_i \in (a_i, b_i) \text{ and } y_j \in (a_j, b_j), \text{ for some } i \ne j\}.$$

We start by formalizing Problem A. Notice that it is a decision problem, in the sense introduced in Section 2. Then, we propose an alternative formulation in the form of another decision problem, that can be proved to be equivalent.

**Problem A (Safety, formal statement)** *Given initial conditions $\mathbf{x}(0)$, determine if there exists an input signal $\mathbf{u}(t)$ that guarantees that $\mathbf{y}(t, \mathbf{u}) \notin B_+$ for all $t \ge 0$.*

An instance of Problem A is described by the initial conditions $\mathbf{x}(0)$, and by the set

$$\Theta := \{f, h, X, U, \mathcal{U}, Y, a_1, \dots, a_n, b_1, \dots, b_n\}. \tag{3}$$

The instance $\{\mathbf{x}(0), \Theta\} \in$ Problem A if and only if $\mathbf{x}(0)$ belongs to the maximal controlled invariant set of system (1), with parameters $\Theta$. In order to reformulate Problem A as a scheduling problem, we introduce three quantities: $R_i$, $D_i$, and $P_i$. These play a similar role to the release times, deadlines, and job durations of Section 2. Specifically, for each agent $i$, if $y_i(0) \le a_i$ define

$$R_i := \inf_{u_i \in \mathcal{U}_i} \{t : y_i(t, u_i) = a_i\}, \; D_i := \sup_{u_i \in \mathcal{U}_i} \{t : y_i(t, u_i) = a_i\},$$

and set $R_i = D_i = 0$ if $y_i(0) > a_i$. Since (2) is output-monotone we have that, if $y_i(0) \le a_i$,

$$R_i = \{t : y_i(t, u_{i,max}) = a_i\}, \quad D_i = \{t : y_i(t, u_{i,min}) = a_i\}. \tag{4}$$

These two quantities are, respectively, the minimum and maximum time at which the output of system $i$ can reach $a_i$. Notice that $R_i$ and $D_i$ are always well defined, since (1) has unique solutions and $\dot{y}_i \ge \dot{y}_{i,min} > 0$. For each agent $i$ such that $y_i(0) \le a_i$, given a real number $T_i$, define

$$P_i(T_i) := \inf_{u_i \in \mathcal{U}_i} \{t \ge 0 : y_i(t, u_i) = b_i\} \tag{5}$$

with constraint

$$y_i(t, u_i) \le a_i \, \forall \, t < T_i. \tag{6}$$

If the constraint cannot be satisfied, set $P_i(T_i) := \infty$. If $y_i(0) \in (a_i, b_i)$ define $P_i(T_i) := \{t : y_i(t, u_{i,max}) = b_i\}$, and if $y_i(0) \ge b_i$ define $P_i(T_i) := 0$. $P_i(T_i)$ is the earliest time that $i$ can reach $b_i$, if it does not pass $a_i$ before $T_i$.

Recall that quantities $R_i$, $D_i$, and $P_i(T_i)$ defined above depend on the initial condition $x_i(0)$ through $y_i(t)$. Problem A is reformulated as follows.

**Problem 1 (Scheduling)** *Given initial conditions* $\mathbf{x}(0)$*, determine if there exists a schedule* $\mathbf{T} = (T_1, \dots, T_n) \in \mathbb{R}^n$ *such that, for all* $i$*,*

$$R_i \le T_i \le D_i, \tag{7}$$

*and for all* $i \ne j$*,*

$$T_i \ge T_j \Rightarrow T_i \ge P_j(T_j). \tag{8}$$

As for Problem A, an instance of Problem 1 is described by the set $\{\mathbf{x}(0), \Theta\}$.

Notice that the quantity $D_i$ in (7) plays a role that is formally similar to the quantity $\mathtt{d}_i - \mathtt{p}_i$ in the definition of $DEC(\mathtt{1}|\mathtt{r}_i|\mathtt{L_{max}})$, that is, $D_i$ is a deadline minus a job duration. This choice, as opposed to letting $D_i$ be a deadline, allows to simplify the proofs in the following section.

**Example 3.1** Consider system (1) with $f_i(x_i, u_i) = u_i$, $h_i(x_i) = x_i$, $x_i \in \mathbb{R}$, $u_i \in [1, 2]$, and $i \in \{1, 2, 3\}$. Let $(x_1(0), a_1, b_1) = (0, 2, 4)$, $(x_2(0), a_2, b_2) = (2, 4, 6)$, $(x_3(0), a_3, b_3) = (4, 6, 8)$. According to (4) we have $(R_1, D_1) = (1, 2)$, $(R_2, D_2) = (2, 4)$, $(R_3, D_3) = (3, 6)$, and according to (5), $P_i(T_i) = T_i + 1$, since $b_i - a_i = 2$ and $u_{i,max} = 2$ for all $i$. Therefore, any schedule $\mathbf{T}$ with $T_1 \in [1, 2]$, $T_2 \in [T_1 + 1, 4]$, $T_3 \in [T_2 + 1, 6]$, is feasible for Problem 1. Similarly, the schedule $\mathbf{T}$ with $T_1 \in [1, 2]$, $T_2 = 4$, $T_3 = 3$ is feasible.

# 4. PROBLEMS A AND 1 ARE EQUIVALENT AND NP-HARD

**Theorem 4.1** *Problem A* $\simeq$ *Problem 1.*

Proof. We must show that Problem A is reducible to Problem 1 and *vice versa*. Notice that for both Problems an instance is fully described by the set $\{\mathbf{x}(0), \Theta\}$, thus the mapping between instances of the two problems is the identity (which has obviously polynomially bounded running time). Proving equivalence thus amounts to proving that for a given $\{\mathbf{x}(0), \Theta\}$,

$$\{\mathbf{x}(0), \Theta\} \in \text{ Problem A} \Leftrightarrow \{\mathbf{x}(0), \Theta\} \in \text{ Problem 1.}$$

($\{\mathbf{x}(0), \Theta\} \in$ Problem A $\Rightarrow \{\mathbf{x}(0), \Theta\} \in$ Problem 1): Assume that $\tilde{\mathbf{y}}(t, \tilde{\mathbf{u}})$ satisfies the constraints of Problem A. The time instants at which $\tilde{\mathbf{y}}(t, \tilde{\mathbf{u}})$ crosses each of the planes $y_i = a_i$ define a vector $\mathbf{T}$ (notice that synchronous crossings are forbidden by $\tilde{\mathbf{y}} \notin B_+$), and we can set $T_i = 0$ if $y_i(0) > a_i$. This $\mathbf{T}$ satisfies (7) given the definition of $R_i$ and $D_i$. Moreover, the time instants at which $\tilde{\mathbf{y}}(t, \tilde{\mathbf{u}})$ crosses the planes $y_i = b_i$ defines a vector $\tilde{\mathbf{P}} = (\tilde{P}_1, \dots, \tilde{P}_n)$, and for all $T_i \ge T_j$, $T_i \ge \tilde{P}_j$, otherwise $\tilde{\mathbf{y}}(t, \tilde{\mathbf{u}}) \cap B_+ \ne \emptyset$. Since $P_i(T_i)$ is the minimum time at which agent $i$ can exit the intersection, provided it enters no earlier than $T_i$, we have that $P_i(T_i) \le \tilde{P}_i$. Therefore $\mathbf{T}$ also satisfies (8).
($\{\mathbf{x}(0), \Theta\} \in$ Problem A $\Leftarrow \{\mathbf{x}(0), \Theta\} \in$ Problem 1): Assume that Problem 1 accepts the instance $\{\mathbf{x}(0), \Theta\}$, i.e., there exists a schedule $\mathbf{T}$ that satisfies Problem 1 given $\mathbf{x}(0)$. Assume that $y_i(0) \le a_i$ for all $i$ and, without loss of generality, that $T_n \ge T_i$ for all $i \in \{1, \dots, n-1\}$. To satisfy condition (8), the schedule must be such that $P_i(T_i)$ is finite for all $i \in \{1, \dots, n-1\}$. Thus by definition of $P_i(T_i)$, there exists an input such that $y_i(t) \le a_i$ for all $t < T_i$, and $y_i(t) = b_i$ when $t = P_i(T_i)$. To satisfy (7) and (8), with this input each $y_i$ enters the interval $(a_i, b_i)$ no earlier than $T_i$, leaves the interval at $P_i(T_i)$, and the intervals $(T_i, P_i(T_i))$ do not intersect. Thus agents $1, \dots, n-1$ do not collide. Then, setting $u_n = u_{n,min}$, we know that agent $n$ reaches $a_n$ at time $t_n = D_n$. By (4), $t_n \ge T_n$, and by (8) $T_n \ge P_i(T_i)$ for all $i \in \{1, \dots, n-1\}$. Thus when $y_n \in (a_i, b_i)$, $y_i \ge n_i$ for all $i \in \{1, \dots, n-1\}$, hence agents $1, \dots, n$ do not collide.

If for some systems $i$ we have $y_i(0) > a_i$, then $R_i = 0$ and $D_i = 0$. By (7) this implies $T_i = 0$. For agents with $y_i(0) > a_i$, by definition of $P_i$ we have that $P_i(0) = 0$ if $y_i(0) \ge b_i$, and $P_i(0) > 0$ otherwise. Problem 1 accepts $\{\mathbf{x}(0), \Theta\}$ only if there exists at most one agent such that $a_i < y_i(0) < b_i$, otherwise we would have $T_i = T_j = 0$ and $P_i(T_i), P_j(T_j) > 0$ which contradicts (8). Assume, without loss of generality, that agents $1, \dots, m-1$ have $y_i(0) \ge b_i$, and agent $m$ has $y_m(0) \in (a_m, b_m)$. Agents $1, \dots, m-1$ do not collide regardless of the input. Agent $m$ has input such that $y_m(P_m(T_m)) = b_i$, and agents $m+1, \dots, n$ reach $a_i$ at $t \ge P_m(T_m)$ with a collision-free input, by the reasoning above. □

Now let $\mathcal{PC}(U_i)$ be the set of piecewise constant functions $\mathbb{R} \to U_i$. To prove $NP$-hardness of Problems A and 1 we consider the particular set of instances obtained by fixing

$$\begin{gathered} f_i(x_i, u_i) = u_i, \quad h_i(x_i) = x_i, \quad X_i = \mathbb{R}, \quad Y_i = \mathbb{R} \\ U_i = [u_{i,min}, u_{i,max}] \subset \mathbb{R}, \quad \mathcal{U}_i = \mathcal{PC}(U_i), \end{gathered} \tag{9}$$

and show that $DEC(\mathtt{1}|\mathtt{r}_i|\mathtt{Lmax}, 0)$, which is NP-complete (see [19]), can be reduced to Problem 1.

**Theorem 4.2** *Consider* $\mathtt{1}|\mathtt{r}_i|\mathtt{Lmax}$ *with* $n > 1$ *jobs, and Problem 1 with* $n$ *agents. Then,*

$$DEC(\mathtt{1}|\mathtt{r}_i|\mathtt{Lmax}, 0) \propto \text{ Problem 1.}$$

Proof. Consider the set of instances of Problem 1 that satisfy (9). One such instance with $n$ agents is described by $n$ sets of numbers $(x_i(0), a_i, b_i, u_{i,min}, u_{i,max})$, while an instance of $DEC(\mathtt{1}|\mathtt{r}_i|\mathtt{Lmax}, 0)$ with $n$ jobs is described by

$n$ sets of numbers $(\mathtt{r}_i, \mathtt{d}_i, \mathtt{p}_i)$. Let us call $\mathbf{x}(0)$, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{u}_{min}$, $\mathbf{u}_{max}$, $\mathbf{r}$, $\mathbf{d}$, $\mathbf{p}$ the $n$-dimensional vector of the corresponding quantities. To prove the theorem, we must find a mapping $g : (\mathbf{r}, \mathbf{d}, \mathbf{p}) \mapsto (\mathbf{x}(0), \mathbf{a}, \mathbf{b}, \mathbf{u}_{min}, \mathbf{u}_{max})$, and prove that

$$(\mathbf{r}, \mathbf{d}, \mathbf{p}) \in DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0) \Leftrightarrow g(\mathbf{r}, \mathbf{d}, \mathbf{p}) \in \text{Problem 1},$$

which is equivalent to

$$\begin{gathered}(\mathbf{r}, \mathbf{d}, \mathbf{p}) \in DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0) \Rightarrow g(\mathbf{r}, \mathbf{d}, \mathbf{p}) \in \text{Problem 1} \\ \text{and} \\ (\mathbf{r}, \mathbf{d}, \mathbf{p}) \notin DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0) \Rightarrow g(\mathbf{r}, \mathbf{d}, \mathbf{p}) \notin \text{Problem 1}.\end{gathered} \tag{10}$$

We begin by assuming, without loss of generality, that $\mathtt{r}_i, \mathtt{d}_i > 0$ for all $i$. This can always be ensured by adding a constant to $\mathbf{r}$ and $\mathbf{d}$. To construct the mapping $g$, partition the set of instances $(\mathbf{r}, \mathbf{d}, \mathbf{p})$ in two groups: (i) instances such that, for some $i$, $\mathtt{d}_i - \mathtt{p}_i < \mathtt{r}_i$; (ii) all other instances.

For all instances in group (i), set the image of $g$ to an arbitrary non-accepted instance of Problem 1, e.g. set $x_i(0) \in (a_i, b_i)$, $x_j(0) \in (a_j, b_j)$ for some $i \neq j$. This is possible since $n > 1$. Given that for all instances in group (i) $(\mathbf{r}, \mathbf{d}, \mathbf{p}) \notin DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0)$, and that, by construction, $f(\mathbf{r}, \mathbf{d}, \mathbf{p}) \notin$ Problem 1, mapping $g$ restricted to instances of group (i) satisfies (10).

For the instances in group (ii), set $(\mathtt{r}_i, \mathtt{d}_i, \mathtt{p}_i) \mapsto (x_i(0), a_i, b_i, u_{i,min}, u_{i,max})$ with

$$\begin{gathered}x_i(0) = 0, \quad a_i = \mathtt{r}_i, \quad b_i = (\mathtt{p}_i + \mathtt{r}_i), \\ u_{i,min} = \mathtt{r}_i/(\mathtt{d}_i - \mathtt{p}_i), \quad u_{i,max} = 1.\end{gathered} \tag{11}$$

The two above sets of equations transform every instance of $DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0)$ in group (ii) into an instance of Problem 1. Writing (4) and (5) explicitly for a system that satisfies (9), we have

$$R_i = \frac{a_i - x_i(0)}{u_{i,max}}, \quad D_i = \frac{a_i - x_i(0)}{u_{i,min}}, \quad P_i(T_i) = \frac{b_i - a_i}{u_{i,max}} + T_i.$$

Rewriting the right-hand sides of these using (11) we obtain $R_i = \mathtt{r}_i$, $D_i = \mathtt{d}_i - \mathtt{p}_i$, $P_i(T_i) = \mathtt{p}_i + T_i$. The above equations formally transform the conditions on $(\mathtt{r}_i, \mathtt{d}_i, \mathtt{p}_i)$ of $DEC(1|\mathtt{r}_j|\mathtt{Lmax}, 0)$ into the conditions on $(R_i, D_i, P_i)$ in (7) and (8) of Problem 1, therefore $g$ restricted to instances of group (ii) satisfies (10). This concludes the proof. □

From the above theorem, and from the equivalence of Problems 1 and A, we obtain

**Lemma 4.3** *Problems 1 and A are NP-hard*

PROOF. $DEC(1|\mathtt{r}_i|\mathtt{Lmax}, 0)$ is reducible to Problem 1, and since the first is NP-complete, the second is NP-hard. Also, by Theorem 4.1, Problem A is NP-hard. □

# 5. SOLUTION OF PROBLEM A

Consider the set $\mathcal{P}$ of all permutations of the index vector $(1, \ldots, n)$. Let $\pi$ be a member of $\mathcal{P}$, and $\pi_i$ be the $i$-th index of the permutation. Once the parameters of system (1) are specified, Problem 1 can be solved for an arbitrary set of initial conditions $\mathbf{x}(0)$ by Algorithm 2. Since Problems A and 1 are equivalent, the same algorithm solves Problem A. Algorithm 2 explores all the possible permutations in $\mathcal{P}$, and since the cardinality of the search space grows factorially in the number of agents $n$, so does the running time of the algorithm. As we have seen, Problem 1 is NP-hard, so

**Algorithm 2** Solution of Problem 1

```
1: procedure ExactSolution(x(0))
2:     for all i do
3:         given x_i(0) calculate R_i, D_i
4:     end for
5:     for all π ∈ P do
6:         T_{π_1} ← R_{π_1}
7:         for i = 2 → n do
8:             T_{π_i} ← max(P_{π_{i-1}}(T_{π_{i-1}}), R_{π_i})
9:         end for
10:        if T_i ≤ D_i for all i ∈ {1, ..., n} then
11:            return {T, yes}
12:        end if
13:    end for
14:    return {∅, no}
15: end procedure
```

unsurprisingly there is no known way to find a solution to all instances in polynomial time with respect to $n$. Moreover, since our ultimate goal is to use an algorithm that solves Problem A to construct the control law required by Problem B (Supervisor), and this must run in real time, even an algorithm with good average running time, but exponential worst-case time, would not be good enough. Using results from the scheduling literature, however, we can design algorithms that provide an approximate solution to Problem 1 in polynomial time. The approach that we propose consists in adding an additional constraint to Problem 1, so that it becomes solvable in polynomial time. Then we gauge the effect of this additional constraint on the solutions. We start by proving the following

**Lemma 5.1** *Consider system (2) and $R_i$, $D_i$ defined in (4), and assume that $\mathcal{U}_i$ is path connected, that solutions of (1) depend continuously on the input, and that $h_i$ is continuous. If $y_i(0) < a_i$ then, for any $T_i \in [R_i, D_i]$ there exists a $u_i \in \mathcal{U}_i$ such that $y_i(T_i, u_i) = a_i$.*

PROOF. By assumption solutions of (1) depend continuously on the input, therefore by continuity of $h_i$, $y_i(t, u_i)$ depends continuously on $u_i$. Since $y_i(0) < a_i$, $\dot{y}_i \geq \dot{y}_{i,min} > 0$, and (1) has unique solutions, $\{t : y_i(t, u_i) = a_i\}$ defines a single-valued, continuous map $M : \mathcal{U}_i \to [R_i, D_i]$. Finally, since $\mathcal{U}_i$ is path connected, there is a continuous path in $\mathcal{U}_i$ connecting the inputs corresponding to $R_i$ and $D_i$. The image of a continuous path under a continuous map is a continuous path, connecting points $R_i$ and $D_i$, and therefore covers the whole interval $[R_i, D_i]$, that is, $M$ is surjective. □

Now define the quantity

$$\delta_{max} := \max_{i \in \{1, \ldots, n\}} \sup_{x_i(0) \in X_i : y_i(0) = a_i} \{t : y_i(t, u_{i,max}) = b_i\}. \tag{12}$$

This is the minimum worst-case time necessary for $y_i$ to go from $a_i$ to $b_i$. From here on we assume the hypotheses of Lemma 5.1. This implies that there exists an input $u_i \in \mathcal{U}_i$ such that $y_i(T_i, u_i) = a_i$, and using (12) that $y_i(T_i + \delta_{max}, u_i) \geq b_i$, so that by the definition of $P_i(T_i)$, $P_i(T_i) \leq T_i + \delta_{max}$. We modify Problem 1 as follows.

**Problem 2 (Approximation)** *Given initial conditions $\mathbf{x}(0)$, determine if there exists a schedule $\mathbf{T} = (T_1, \ldots, T_n) \in \mathbb{R}^n$*

*such that, for all $i$,*

$$R_i \le T_i \le D_i,$$

*and for all $i \neq j$, if $T_j = 0$ then*

$$T_i \ge T_j \Rightarrow T_i \ge P_j(T_j),$$

*if $T_j > 0$ then*

$$T_i \ge T_j \Rightarrow T_i \ge T_j + \delta_{max}.$$

Any schedule that satisfies Problem 2 also satisfies Problem 1, since $T_j + \delta_{max} \ge P_j(T_j)$. By solving Problem 1 however, we allocate the resource (the intersection) for more time that is strictly needed by each agent. We are thus trading maximum traffic flow with computational speed.

By (4) $R_i = 0 \Leftrightarrow D_i = 0$, and agents with $R_i = D_i = 0$ do not contribute to the combinatorial complexity of Problem 1 as $T_i$ has a unique possible value. By normalizing the data of Problem 2 to make $\delta_{max} = 1$, and then setting $R_i = \mathtt{r}_i$, $D_i = \mathtt{d}_i - 1$, $T_i = \mathtt{T}_i$, Problem 2 for agents with $R_i, D_i > 0$ becomes formally equivalent to $DEC(\mathtt{1}|\mathtt{r}_i, \mathtt{p}_i = \mathtt{1}|\mathtt{Lmax}, 0)$, which is solved in polynomial time by Algorithm 1. Algorithm 3 solves Problem 2 treating separately agents with $y_i(0) \ge a_i$, for which $R_i = D_i = 0$, and agents with $y_i(0) < a_i$. In the pseudocode of ApproximateSolution, without loss of generality, we assume that $y_i(0) \ge a_i$ for $i = 1, \dots, m$, and $y_i(0) < a_i$ for $i = m+1, \dots, n$. Since Algorithm 3 provides

**Algorithm 3** Solution of Problem 2

```
1: procedure ApproximateSolution(x(0))
2:     for all i ∈ {1,...,n} do given x_i(0) calculate R_i, D_i
3:     end for
4:     if y_i(0) ∈ [a_i, b_i) for two different i ∈ {1,...,m} then
5:         return {∅, no}
6:     end if
7:     for all i ∈ {1,...,m} do T_i ← 0
8:     end for
9:     R_bound ← max{P_1(0),...,P_m(0)}
10:    for all i ∈ {m+1,...,n} do R_i ← max(R_i, R_bound)
11:    end for
12:    set δ_max as in (12)
13:    r = (R_{m+1}/δ_max, ..., R_n/δ_max)
14:    d = (D_{m+1}/δ_max + 1, ..., D_n/δ_max + 1)
15:    {T_{m+1}, ..., T_n, answer}=PolynomialTime(r, d)
16:    for i = m+1 → n do T_i ← T_i δ_max
17:    end for
18:    return {T, answer}
19: end procedure
```

an approximate solution to Problem 1, we need a measure of the quality of this solution. We perform this by providing an upper bound to the quantity $\sup_{\mathbf{u}\in\mathcal{U}} \inf_{t\ge 0, b\in B_+} \|\mathbf{y}(t,\mathbf{u}) - b\|_\infty$, calculated over all $\mathbf{x}(0)$ for which Algorithm 3 returns "*no*". This is the maximum over all possible inputs $\mathbf{u} \in \mathcal{U}$ of the distance of $\mathbf{y}(t,\mathbf{u})$ from the bad set $B_+$, and, as such, it gives a measure of how much Algorithm 3 "overestimates" $B_+$. To provide the upper bound we first introduce the following result.

**Lemma 5.2** *For a given $\mathbf{x}(0)$, take an arbitrary $\mathbf{u} \in \mathcal{U}$, and define a schedule $\mathbf{T}$ as $T_i = \{t : y_i(t, u_i) = a_i\}$ for all $i$ such that $y_i(0) < a_i$, and $T_i = 0$ for all other $i$. Assume that, for some $i$ and $j$, $y_i(0) < a_i$, $y_j(0) \le a_j$, $T_i \ge T_j$, and $T_i - T_j \le \delta_{max}$, that is, two jobs are scheduled within $\delta_{max}$ of each other. Then*

$$\inf_{t\ge 0, b\in B_+} \|\mathbf{y}(t,\mathbf{u}) - b\|_\infty \le \max_{i\in\{1,\dots,n\}} \dot{y}_{i,max} \left(\delta_{max} - \frac{b_i - a_i}{\dot{y}_{i,max}}\right),$$

Proof. Since $\dot{y}_j \le \dot{y}_{j,max}$, $y_j$ remains in the interval $(a_j, b_j)$ for a time interval greater or equal to $(b_j - a_j)/\dot{y}_{j,max}$, therefore $y_j(T_i, u_j) - b_j \le \dot{y}_{j,max}\left(\delta_{max} - \frac{b_j - a_j}{\dot{y}_{j,max}}\right)$, while $y_i(T_i, u_i) = a_i$. The set of points in $Y$ with $y_i = a_i$, $y_j = b_j$ is on the boundary of $B_+$, therefore

$$\inf_{b\in B_+} \|\mathbf{y}(T_i,\mathbf{u}) - b\|_\infty \le y_j(T_i, u_j) - b_j \le \max_{i\in\{1,\dots,n\}} \dot{y}_{i,max}\left(\delta_{max} - \frac{b_i - a_i}{\dot{y}_{i,max}}\right).$$

□

**Theorem 5.3** *If for a given $\mathbf{x}(0)$* ApproximateSolution *returns "no", then*

$$\sup_{\mathbf{u}\in\mathcal{U}} \inf_{t\ge 0, b\in B_+} \|\mathbf{y}(t,\mathbf{u}) - b\|_\infty \le \max_{i\in\{1,\dots,n\}} \dot{y}_{i,max}\left(\delta_{max} - \frac{b_i - a_i}{\dot{y}_{i,max}}\right). \quad (13)$$

Proof. ApproximateSolution returns "*no*" if $y_i(0) \in [a_i, b_i)$ for two different $i$, or if PolynomialTime at line 15 returns "*no*". In the first case the left hand side of (13) is equal to 0, since $\mathbf{y}(0)$ is on the boundary of $B_+$, and (13) is verified. In the second case, if PolynomialTime returns "*no*" then, for any schedule $\mathbf{T}$ with $T_i \in [R_i, D_i]$ for all $i$, there exist $i$ and $j$ with $y_i(0) < a_i$, $y_j(0) < a_j$, $T_j \le T_i$, such that $T_i - T_j < \delta_{max}$. This is a consequence of the fact that PolynomialTime solves $DEC(\mathtt{1}|\mathtt{r}_i, \mathtt{p}_i = \mathtt{1}|\mathtt{Lmax}, 0)$ exactly. By the reasoning above, for any $\mathbf{u} \in \mathcal{U}$, the schedule $\mathbf{T}$ defined by $T_i = \{t : y_i(t, u_i) = a_i\}$ if $y_i(0) < a_i$, $T_i = 0$ otherwise, has $T_i \in [R_i, D_i]$ for all $i$, and satisfies the hypotheses of Lemma 5.2. This completes the proof. □

According to the above theorem, if ApproximateSolution cannot find a feasible schedule for an initial condition $\mathbf{x}(0)$, then all outputs $\mathbf{y}(t,\mathbf{u})$ with $\mathbf{u} \in \mathcal{U}$ intersect the *extended bad set*

$$\hat{B}_+ := \left\{\mathbf{y} : \inf_{b\in B_+} \|\mathbf{y} - b\|_\infty \le \max_{i\in\{1,\dots,n\}} \dot{y}_{i,max}\left(\delta_{max} - \frac{b_i - a_i}{\dot{y}_{i,max}}\right)\right\}. \quad (14)$$

# 6. SOLUTION OF PROBLEM B

Problem B (Supervisor) requires to design a supervisor that, given the current state of the system and a desired input, returns the desired input if this does not cause a collision at some future time, or a safe input otherwise. Thus, the input returned by the supervisor must keep the state of system (1) within the maximal controlled invariant set. As we have seen in the previous sections, membership in this set is determined by solving Problem A or its scheduling version, Problem 1. In Section 5 we have provided an exact and an approximate algorithm to solve these problems. We can exploit these algorithms in the solution of Problem B (Supervisor) by designing the supervised system as a hybrid system. This is obtained from matching the continuous dynamics of (1) with a discrete-time control map. At the $k$-th iteration, the control map takes as arguments the current

state $\mathbf{x}(k\tau)$ and desired (constant) value $\mathbf{v}_k \in U$ of the input for $t \in [k\tau, (k+1)\tau]$, and returns an input signal $\mathbf{u}_{out}$, with value in $U$, defined for $t \in [k\tau, (k+1)\tau]$. The choice of the returned input signal is based on the solution of Problem 1. The sought supervisor is the map $s : (\mathbf{x}(k\tau), \mathbf{v}_k) \mapsto \mathbf{u}_{out}$. To give a precise meaning to the statement of Problem B (Supervisor), we must formally define the conditions under which a desired input may cause a collision. Given $\mathbf{v}_k$, consider the two signals $\bar{\mathbf{u}}_k$ and $\bar{\mathbf{u}}_k^\infty$ defined as follows: the first is defined on the interval $[k\tau, (k+1)\tau]$ and identically equal to $\mathbf{v}_k$; the second is an element of $\mathcal{U}$ defined on $[k\tau, \infty)$, and such that $\bar{\mathbf{u}}_k^\infty(t) = \bar{\mathbf{u}}_k(t)$ when $t \in [k\tau, (k+1)\tau]$. Additionally, given $\mathbf{x}(k\tau)$, call $\mathbf{u}_{k,safe}^\infty(t) \in \mathcal{U}$ a control signal such that $\mathbf{y}(t, \mathbf{u}_{k,safe}^\infty, \mathbf{x}(k\tau)) \notin B_+$ for all $t \geq k\tau$ (if such control exists), and call $\mathbf{u}_{k,safe}$ the restriction of $\mathbf{u}_{k,safe}^\infty$ to the interval $[k\tau, (k+1)\tau]$. If $\mathbf{u}_{k,safe}^\infty$ does not exists, let $\mathbf{u}_{k,safe}^\infty, \mathbf{u}_{k,safe} = \emptyset$. Problem B (Supervisor) is formally stated as follows

**Problem B (Supervisor, formal statement)** *Design the supervisor $s(\mathbf{x}(k\tau), \mathbf{v}_k)$ for system (1) such that*

$$s(\mathbf{x}(k\tau), \mathbf{v}_k) = \begin{cases} \bar{\mathbf{u}}_k & \text{if } \exists\, \bar{\mathbf{u}}_k^\infty(t) \in \mathcal{U} : \\ & \quad \mathbf{y}(t, \bar{\mathbf{u}}_k^\infty, \mathbf{x}(k\tau)) \notin B_+ \forall\, t \geq k\tau \\ \mathbf{u}_{k,safe} & \text{otherwise,} \end{cases}$$

*and so that it is non-blocking: if $\mathbf{u}_{out} = s(\mathbf{x}(k\tau), \mathbf{u}_k) \neq \emptyset$, then for any $\mathbf{v}_{k+1}$, $k \geq 0$, $s\left(\mathbf{x}((k+1)\tau, \mathbf{u}_{out}, \mathbf{x}(k\tau)), \mathbf{v}_{k+1}\right) \neq \emptyset$.*

Given a system of the form (1) and the state $\mathbf{x}(k\tau)$ at some time $k\tau$, the procedure ExactSolution in Algorithm 2 returns a binary value (*yes*/*no*), and a schedule $\mathbf{T}$. We can use this information to design the supervisor in Problem B. To this end, introduce the operator $\sigma(x_i(0), T_i)$, associated to the function $P_i(T_i)$. For all agents with $y_i(0) \leq a_i$, let

$$\sigma(x_i(0), T_i) := \arg \inf_{u_i \in \mathcal{U}} \{t \geq 0 : y_i(t, u_i) = b_i\}, \tag{15}$$

with constraint

$$y_i(t, u_i) \leq a_i \,\forall\, t < T_i, \tag{16}$$

This is the input $u_i$ that brings $y(t, u_i, x_i)$ at $b_i$ at $t = P_i(T_i)$ (see (5)). If the constraint cannot be satisfied, set $\sigma(x_i(0), T_i) := \emptyset$. If $y_i(0) \in (a_i, b_i)$ define $\sigma(x_i(0), T_i) := u_{i,max}$, and if $y_i(0) \geq b_i$ define $\sigma(x_i(0), T_i) := 0$. If the input is not unique, let $\sigma$ return one among the possible solutions. Call $\sigma(\mathbf{x}(0), \mathbf{T})$ the vector $(\sigma(x_1(0), T_1), \ldots, \sigma(x_n(0), T_n))$. Assume that, at $t = 0$, we have ExactSolution$(\mathbf{x}(0)) = \{\mathbf{T}_0, yes\}$, and define $\mathbf{u}_{0,safe}^\infty = \sigma(\mathbf{x}_0, \mathbf{T}_0)$ and $\mathbf{u}_{0,safe}$ as the restriction of $\mathbf{u}_{0,safe}^\infty$ to the time interval $[0, \tau]$. At each iteration $k = 0, 1, 2, \ldots$, the supervisor map $s(\mathbf{x}(k\tau), \mathbf{v}_k)$ is defined by Algorithm 4, using the current state $\mathbf{x}(k\tau)$, the desired input $\mathbf{v}_k$, and the value $\mathbf{u}_{k,safe}$ calculated at the previous iteration. The algorithm returns $\bar{\mathbf{u}}_k$ (the desired input), if the state reached with this input is within the maximal controlled invariant set; otherwise it returns $\mathbf{u}_{k,safe}$. To prove that Algorithm 4 correctly solves Problem B, we use the two following lemmas as intermediate results.

**Lemma 6.1** *If* ExactSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$*, then* $\sigma(\mathbf{x}(k\tau), \mathbf{T}) \neq \emptyset$.

Proof. The existence of an input $\sigma(\mathbf{x}(k\tau), \mathbf{T})$ corresponding to the schedule $\mathbf{T}$ is proved as in the proof of Theorem 4.1. □

**Algorithm 4** Implementation of the supervisor map

```
1: procedure s(x(kτ), v_k)
2:     ū_k(t) ← v_k ∀ t ∈ [kτ, (k+1)τ]
3:     {T, answer} ←
4:       ExactSolution(x((k+1)τ, ū_k, x(kτ)))
5:     if answer = yes then
6:         u^∞_{k+1,safe} ← σ(x((k+1)τ, ū_k, x(kτ)), T)
7:         u_{k+1,safe} ← u^∞_{k+1,safe} restricted to [kτ, (k+1)τ]
8:         return ū_k
9:     else
10:        {T, answer} ←
11:          ExactSolution(x((k+1)τ, u_{k,safe}, x(kτ)))
12:        u^∞_{k+1,safe} ← σ(x((k+1)τ, u_{k,safe}, x(kτ)), T)
13:        u_{k+1,safe} ← u^∞_{k+1,safe} restricted to [kτ, (k+1)τ]
14:        return u_{k,safe}
15:    end if
16: end procedure
```

**Lemma 6.2** *If* ExactSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$*, defining* $\mathbf{u} := \sigma(\mathbf{x}(k\tau), \mathbf{T})$*,* ExactSolution$(\mathbf{x}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau)))$ *returns "yes".*

Proof. The input $\mathbf{u}$ is well defined by Lemma 6.1. Call $\mathbf{y}(t, \mathbf{u}, \mathbf{x}(k\tau))$ the corresponding output trajectory, defined from time $(k\tau)$, and let $\tilde{\mathbf{u}}$ be $\mathbf{u}$ restricted to the interval $[(k+1)\tau, \infty)$. Clearly if $\mathbf{y}(t, \mathbf{u}, \mathbf{x}(k\tau)) \cap B_+ = \emptyset$, then $\mathbf{y}(t, \tilde{\mathbf{u}}, \mathbf{x}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau)) \cap B_+ = \emptyset$, therefore $\{\mathbf{x}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau)), \Theta\} \in$ Problem A, with $\Theta$ defined by (3). Since Problem A $\simeq$ Problem 1, $\{\mathbf{x}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau)), \Theta\} \in$ Problem 1. □

**Theorem 6.3** *Assume that $s(\mathbf{x}(0), \mathbf{v}_0) \neq \emptyset$. Then, the supervisor $s(\mathbf{x}(k\tau), \mathbf{v}_k)$ defined by Algorithm 4 solves Problem B.*

Proof. To be a solution of Problem B, $s(\mathbf{x}(k\tau), \mathbf{v}_k)$ (i) must return $\bar{\mathbf{u}}_k$ unless all possible $\bar{\mathbf{u}}_k^\infty$ would eventually cause a collision and (ii) it must be nonblocking.

To prove (i), note that Algorithm 4 returns $\bar{\mathbf{u}}_k$ unless ExactSolution at line 4 returns "*no*". If 4 returns "*no*", then $\{\mathbf{x}((k+1)\tau, \bar{\mathbf{u}}, \mathbf{x}(k\tau)), \Theta\} \notin$ Problem 1, and by equivalence of Problems A and 1, $\{\mathbf{x}((k+1)\tau, \bar{\mathbf{u}}, \mathbf{x}(k\tau)), \Theta\} \notin$ Problem A. Thus by definition of Problem A and of $\bar{\mathbf{u}}_k^\infty$, for all $\bar{\mathbf{u}}_k^\infty$, $\mathbf{y}(t, \bar{\mathbf{u}}_k^\infty, \mathbf{x}(k\tau)) \cap B_+ \neq \emptyset$.

To prove (ii) we can proceed by induction: by assumption $s(\mathbf{x}(0), \mathbf{v}_0) \neq \emptyset$, and we must show that if $\mathbf{u}_{out} = s(\mathbf{x}(k\tau), \mathbf{v}_k) \neq \emptyset$, then $s(\mathbf{x}((k+1)\tau, \mathbf{u}_{out}, \mathbf{x}(k\tau)), \mathbf{v}_{k+1}) \neq \emptyset$. First notice that $s(\mathbf{x}((k+1)\tau, \mathbf{u}_{out}, \mathbf{x}(k\tau)), \mathbf{v}_{k+1}) \neq \emptyset$ as long as $\mathbf{u}_{k+1,safe} \neq \emptyset$, so all we have to do is to show that $\mathbf{u}_{k+1,safe} \neq \emptyset$. State $\mathbf{x}((k+1)\tau, \mathbf{u}_{out}, \mathbf{x}(k\tau))$ is reached either with an input $\mathbf{u}_{out} = \bar{\mathbf{u}}$ (lines 5-8 in Algorithm 4) or $\mathbf{u}_{out} = \mathbf{u}_{k,safe}$ (lines 9-14). In the first case, by Lemma 6.1 line 4 of Algorithm 4 ensures that $\sigma$ at line 6 is nonempty. In the second case, by Lemma 6.2 the procedure ExactSolution at line 11 must return $\{\mathbf{T}, yes\}$, and by Lemma 6.1 this implies that $\sigma$ at line 12 is nonempty. □

Notice that, if the step $\tau$ is increased, the "restrictiveness" of Algorithm 4 is unaffected. Indeed, for any value of $\tau$ the algorithm returns the desired input if and only if this does not cause collisions. The size of $\tau$ is thus only a matter of engineering and design convenience.

Algorithm 4 is based on the procedure ExactSolution, whose running time scales factorially with the number of

agents. Therefore, it can be applied only to relatively small problems. To achieve a control law that scales polynomially with the number of controlled agents, we proceed as follows. Define $\sigma_{approx}$ as $\sigma$ in (15) with constraint (16) replaced by $y_i(T_i, u_i) = a_i$. Notice that Lemma 5.1 ensures that an input satisfying this constraint exists if $y_i(0) \leq a_i$ and $T_i \in [R_i, D_i]$. Then, at lines 4 and 11 of Algorithm 4 substitute ExactSolution with ApproximateSolution defined by Algorithm 3, and at lines 6 and 12 substitute $\sigma$ with $\sigma_{approx}$.

Through the substitution, the supervisor retains the nonblocking property defined by Problem B, but it allows only a subset of all collision-free trajectories. More precisely we have the following result.

**Theorem 6.4** *Consider the extended bad set $\hat{B}_+$ defined in (14). Call $\hat{s}(\mathbf{x}(k\tau), \mathbf{v}_k)$ the supervisor defined in Problem B substituting $\hat{B}_+$ to $B_+$, and call $s_{approx}(\mathbf{x}(k\tau), \mathbf{v}_k)$ the supervisor defined by Algorithm 4 modified as detailed above. Then $s_{approx}(\mathbf{x}(k\tau), \mathbf{v}_k)$ is no more restrictive than $\hat{s}(\mathbf{x}(k\tau), \mathbf{v}_k)$, that is, if $s_{approx}(\mathbf{x}(k\tau), \mathbf{v}_k) = \mathbf{u}_{k,safe}$ then $\hat{s}(\mathbf{x}(k\tau), \mathbf{v}_k) = \mathbf{u}_{k,safe}$. Moreover if $s_{approx}(\mathbf{x}(0), \mathbf{v}_0) \neq \emptyset$ then the supervisor is non-blocking in the sense defined in Problem B.*

To prove this result we use two intermediate lemmas.

**Lemma 6.5** *If* ApproximateSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$, *then* $\sigma_{approx}(\mathbf{x}(k\tau), \mathbf{T}) \neq \emptyset$.

Proof. By the definition of Problems 1 and 2, ApproximateSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$ implies that ExactSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$, the result follows from Lemma 6.1. □

**Lemma 6.6** *If* ApproximateSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$, *defining* $\mathbf{u} := \sigma_{approx}(\mathbf{x}(k\tau), \mathbf{T})$, ApproximateSolution$(\mathbf{x}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau))$ *returns "yes".*

Proof. Assume, without loss of generality, that $\mathbf{y}(k\tau)$ is such that $y_i(k\tau) \geq a_i$ for $i \in \{1, \ldots, m\}$, while $y_i(k\tau) < a_i$ for all other agents. Also, assume that $\mathbf{y}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau))$ is such that $y_i((k+1)\tau, u_i, x_i(k\tau)) \geq a_i$ for $i \in \{1, \ldots, p\}$ with $p \geq m$, while $y_i((k+1)\tau, u_i, x_i(k\tau)) < a_i$ for all other agents. ApproximateSolution$(\mathbf{x}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau))$ returns a positive answer provided that (i) no more than one of the agents $1, \ldots, p$ are in $[a_i, b_i)$ and (ii) PolynomialTime at line 15 finds a feasible schedule. (i) is ensured by condition ApproximateSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$ and by having $\mathbf{u} := \sigma_{approx}(\mathbf{x}(k\tau), \mathbf{T})$. To have (ii) PolynomialTime must find a feasible schedule. According to Definition 2.2, this is a schedule $\mathtt{T}$ such that $\mathtt{T}_i \in [\mathtt{r}_i, \mathtt{d}_i]$, and such that $|\mathtt{T}_i - \mathtt{T}_j| \geq 1$ for all $i \neq j$. Since PolynomialTime solves $DEC(\mathtt{1}|\mathtt{r}_i, \mathtt{p}_i = \mathtt{1}|\mathtt{Lmax}, 0)$ exactly, it always returns a positive answer if a feasible schedule exists. Since ApproximateSolution$(\mathbf{x}(k\tau)) = \{\mathbf{T}, yes\}$, elements of $\mathbf{T}$ corresponding to agents $m+1, \ldots, n$ satisfy $|T_i - T_j| \geq \delta_{max}$ for $i \neq j$. By the definition of $\sigma_{approx}$, $y_i(T_i + k\tau, u_i, x_i(k\tau)) = a_i$. Consider $\mathbf{x}((k+1)\tau) := \mathbf{x}((k+1)\tau, \mathbf{u}, \mathbf{x}(k\tau))$, and the input $\tilde{\mathbf{u}}$ equal to $\mathbf{u}$ restricted to the interval $[(k+1)\tau, \infty)$. For all $i \in \{p, \ldots, n\}$, call $T_i' = \{t : y_i(t, \tilde{u}_i, x_i((k+1)\tau)\} = a_i$, and call $R_i'$ and $D_i'$ the quantities defined in (4) with respect to initial condition $x_i((k+1)\tau)$. For all such $i$, $T_i' = T_i - \tau$, therefore all $T_i'$ are at least at distance $\delta_{max}$ from each other. Moreover, $T_i' \in [R_i', D_i']$ by construction. The schedule $\mathtt{T} = (T_p'/\delta_{max}, \ldots, T_n'/\delta_{max})$ is thus a feasible schedule for $DEC(\mathtt{1}|\mathtt{r}_i, \mathtt{p}_i = \mathtt{1}|\mathtt{Lmax}, 0)$. □

Proof of theorem 6.4. By Theorem 5.3 and by (14), the procedure ApproximateSolution returns "*yes*" if there is an input that keeps the state outside of $\hat{B}_+$, thus $s_{approx}(\mathbf{x}(k\tau), \mathbf{v}_k)$ is no more restrictive than $\hat{s}(\mathbf{x}(k\tau), \mathbf{v}_k)$. To prove nonblockingness, one can proceed as in the proof of nonblockingness of Theorem 6.3, substituting Lemmas 6.5 and 6.6 and procedure ApproximateSolution to Lemmas 6.1 and 6.2 and procedure ExactSolution. □

# 7. EXAMPLES

To compute the quantities $P_i(T_i)$ and $\sigma$ or $\sigma_{approx}$ required by Algorithm 4, we must solve an optimization problem on the set of inputs $\mathcal{U}$. In general, such a problem may be solved numerically [7, 16]. For illustration purposes, here we discuss the case in which $f_i(x, u)$ in (2) is a double integrator with saturation on the input function:

$$f_i(x_i, u_i) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x_i + \begin{pmatrix} 0 \\ C \end{pmatrix} u, \quad h(x_i) = (1, 0) \cdot x_i, \tag{17}$$

with $x_i \in \mathbb{R}^2$ and $u_i \in \mathbb{R}$, where

$$C := \begin{cases} 1 \text{ if } (0,1) \cdot x_i \in [\dot{y}_{i,min}, \dot{y}_{i,max}] \text{ or} \\ (0,1) \cdot x_i = \dot{y}_{i,max} \text{ and } u_i < 0 \text{ or} \\ \quad (0,1) \cdot x_i = \dot{y}_{i,min} \text{ and } u_i > 0 \\ 0 \text{ otherwise.} \end{cases}$$

The quantity $y_i = (1, 0) \cdot x_i$ is the position of agent $i$ along its path, while $\dot{y}_i = (0, 1) \cdot x_i$ is the velocity. This is a simple model of longitudinal vehicle dynamics when friction is negligible, and it satisfies the assumptions of Section 3 and of Lemma 5.1. Notice that the presented algorithm can handle more general models, including linear and nonlinear friction terms, such as those discussed in [25].

In the case of equations (17), the optimization problem on $\mathcal{U}$ is solved analytically using standard variational calculus [16, 20]. When no feasible input allows to reach $a_i$ at time $T_i$ with $\dot{y}_i(T_i) = \dot{y}_{i,max}$, extremal solutions can be proved to have the signal $u_i$ composed of three segments with values $u_{i,min}, 0, u_{i,max}$ (in this order), or two segments with values $u_{i,min}, u_{i,max}$ (in this order), or a single segment with value $u_{i,min}$ or $u_{i,max}$. In the case that $\dot{y}_{i,max}$ can be attained before reaching $a_i$, there exists a continuum of optimal solutions. Since $\sigma$ and $\sigma_{approx}$ are required to return a unique solution, in these cases we fix their image to the unique optimal solution obtained with input sequence $u_{i,min}, 0, u_{i,max}, 0$.

Using the optimal inputs defined above to construct $\sigma$ and $\sigma_{approx}$, we have implemented the supervisor described by Algorithm 4, for a set of identical agents with dynamics described by (17), with $\dot{y}_{i,min} = 5Km/h$ $(1.39m/s)$, $\dot{y}_{i,max} = 50Km/h$ $(13.9m/s)$, $u_{i,min} = -2m/s^2$, $u_{i,max} = 1m/s^2$, and an interval $(a_i, b_i)$ that is $10m$ wide for all agents. The supervisor runs at discrete time steps of length $\tau = 1/10s$. In all cases, we have assigned to each agent a fixed "desired speed", that the driver tries to maintain, by accelerating or braking if necessary, unless forced to a different input by the supervisor. For a 3-agent system, Figure 2 represents a "slice" in the space $Y$, for fixed velocities $\dot{\mathbf{y}}(0) = (9, 11, 13)$ $m/s$, of the complement of the maximal controlled invariant

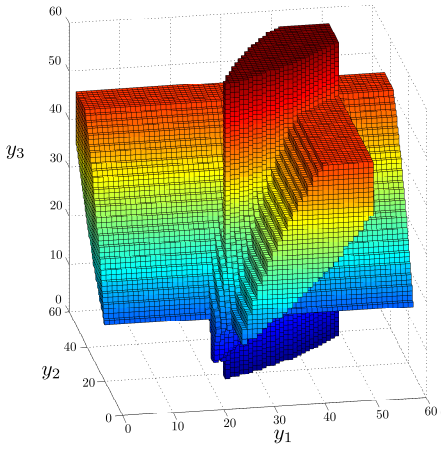


**Figure 2: Complement of the maximal controlled invariant set, in the space $Y$ for fixed velocities. Axes are in meters. For all agents, $(a_i, b_i) = (40, 50)$.**

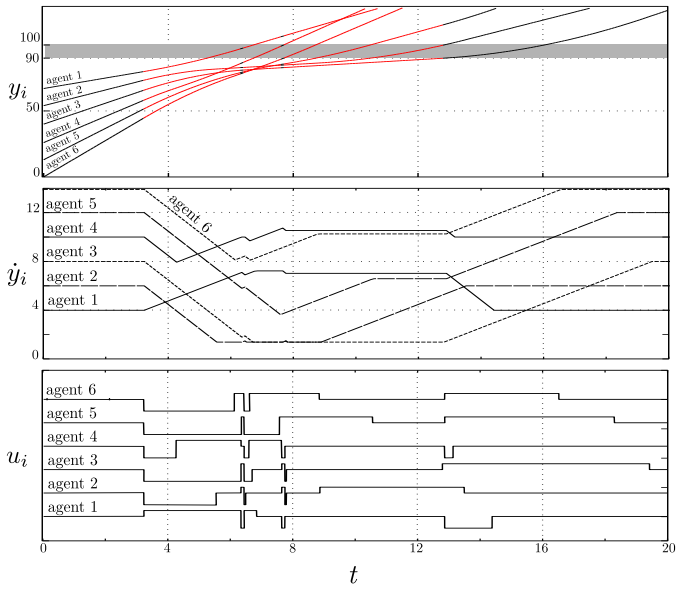


**Figure 3: Positions ($y_i$), velocities ($\dot{y}_i$), and input ($u_i$) of 6 agents controlled by the supervisor in Algorithm 4, using ExactSolution and $\sigma$. The interval $(a_i, b_i) = (90, 100)$ is represented in gray and equal for all agents. Position curves are in black when the supervisor accepts the agents' desired inputs, in red when the supervisor overrides the desired input.**

set. Computing this set for systems with large state space is known to be a hard problem, and research has been focusing on calculating approximations [23]. The procedure ExactSolution in Algorithm 2, exploiting the system's structure, provides a practical technique to determine if a state is inside this set for few agents (0.01 seconds for 6 agents on a modern laptop). Using the procedure ApproximateSolution in Algorithm 3, the same test can be executed approximately, with error bound given by Theorem 5.3, on much larger problems. Figures 3 and 4 depict the trajectories of six agents controlled by the supervisor, using ExactSolution and $\sigma$, or ApproximateSolution and $\sigma_{approx}$, respectively. The interval $(a_i, b_i) = (90, 100)$ is equal for all agents. In these simulations, the initial positions and velocities were selected so that, in the absence of supervisor, all agents would enter the interval $(a_i, b_i)$ (in gray) at the same time. For the sake of simplicity, we let Algorithm 4 return an input (desired input, or override) for all agents, before and after the intersection. Agents past the intersection could apply an arbitrary input without affecting safety, but including this option would make the algorithm longer without significantly improving the result.

Notice that, while in Figure 3 agents occupy the gray band in contiguous time intervals, in Figure 4 there is some idle time between the instant when an agent leaves the gray band and when following agent enters it. The maximum distance of the trajectory in Figure 4 from $B_+$ is bounded as proved in Theorem 5.3. For the parameters specified above, the bound is equal to $35.77m$. To prove that this bound is tight, we have repeated the simulation for 15 agents with the same parameters as above. The result is shown in Figure 5, where the trajectory reaches the bound exactly.

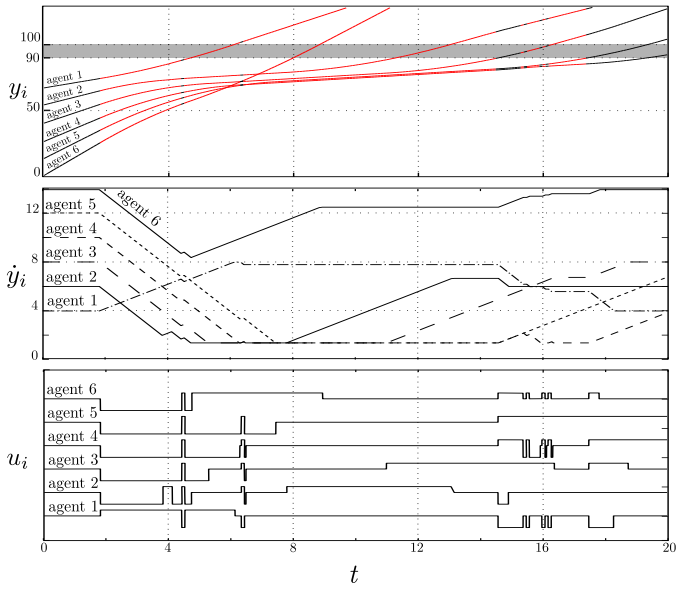


**Figure 4: The same numerical experiment of Figure 3 is executed using the supervisor in Algorithm 4, using ApproximateSolution and $\sigma_{approx}$.**

## 8. CONCLUSIONS

We have considered the problem of determining membership in the maximal controlled invariant set for a general class of systems describing vehicle dynamics at an intersection. Using results from the scheduling literature and computational complexity theory we have proved that the

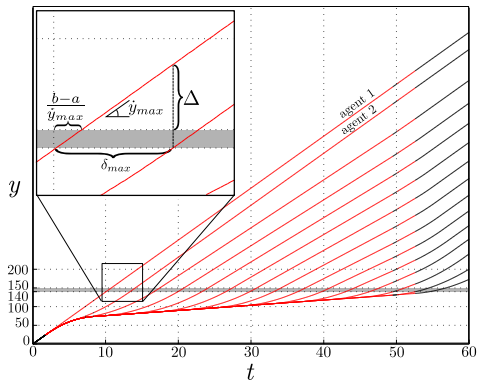


**Figure 5: 15 agents supervised using Algorithm 4, with procedure ApproximateSolution. The bound given in Theorem 5.3, here denoted $\Delta$, is reached exactly by the trajectory.**

exact solution of this problem is NP-complete. We have proposed an approximate solution whose running time scales polynomially with the number of agents. Based on these results, we have designed the least restrictive supervisor (solving Problem B (Supervisor) exactly), whose running time scales exponentially with the number of controlled agents. The supervised system has a hybrid structure, where the continuous dynamics of the agents are controlled based on the result of a scheduling problem solved at regular time intervals. By modifying this supervisor, we have obtained an approximate solution with polynomial running time, and we have provided a tight bound on the approximation. The supervisor acts as a filter between a desired input, here assumed to be generated by the driver, and the physical system. This structure is easily coupled with other controllers, acting between the driver input and the supervisor, to pursue secondary performance objectives within the set of safe control actions allowed by the supervisor.

The results presented here assume that each agent moves along a different path and computes the exact or approximate supervisor through a centralised algorithm. We are currently working on relaxing both constraints, allowing multiple agents to move on the same path or along merging paths, and implementing the solution as a distributed algorithm, to further improve scalability.

## 9. REFERENCES


[1] Car 2 Car Communication Consortium. http://www.car-to-car.org.

[2] Cooperative Intersection Collision Avoidance Systems (CICAS). http://www.its.dot.gov/cicas.

[3] Vehicle Infrastructure Integration Consortium (VIIC). http://www.vehicle-infrastructure.org.

[4] D. Angeli and E. D. Sontag. Monotone control systems. *IEEE Trans. Autom. Control*, 48:1684–1698, 2003.

[5] A. Colombo and D. Del Vecchio. Supervisory control of differentially flat systems based on abstraction. In *50th IEEE Conference on Decision and Control*, 2011.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[7] B. Dacorogna. *Direct Methods in the Calculus of Variations*. Springer, 2008.

[8] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[9] V. Desaraju, H. C. Ro, M. Yang, E. Tay, S. Roth, and D. Del Vecchio. Partial order techniques for vehicle collision avoidance: Application to an autonomous roundabout test-bed. In *ICRA '09. IEEE International Conference on Robotics and Automation*, 2009.

[10] J. Duperret, M. Hafner, and D. Del Vecchio. Formal design of a provably safe robotic roundabout system. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[11] A. F. Filippov. *Differential Equations with Discontinuous Righthand Sides*. Kluwer Academic Publishers, Dordrecht, 1988.

[12] M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM J. Comput.*, 6:416–426, 1981.

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 4:287–326, 1979.

[14] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio. Automated vehicle-to-vehicle collision avoidance at intersections. In *Proc. of ITS World Congress*, 2011.

[15] M. R. Hafner and D. Del Vecchio. Computational tools for the safety control of a class of piecewise continuous systems with imperfect information on a partial order. *SIAM J. Contr. Opt.*, To Appear.

[16] E. Bryson Jr. and Y. Ho. *Applied optimal control*. Ginn and Company, 1969.

[17] H. K. Khalil. *Nonlinear systems*. Prentice-Hall, 2002.

[18] H. Kowshik, D. Caveney, and P. R. Kumar. Provable systemwide safety in intelligent intersections. *IEEE Trans. Veh. Technol.*, 60:804–818, 2011.

[19] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of discrete mathematics*, 1:343–362, 1977.

[20] D. G. Luenberger. *Optimization by vector space methods*. Wiley, 1969.

[21] J. Lygeros, C. Tomlin, and S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35:349–370, 1999.

[22] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.

[23] C. J. Tomlin, I. Mitchell, A. M. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proc. IEEE*, 91:986–1001, 2003.

[24] U.S. Department of Transportation. ITS Strategic Research Plan 2010-2014. http://www.its.dot.gov/strategic_plan2010_2014/.

[25] R. Verma, D. Del Vecchio, and H. K. Fathy. Development of a scaled vehicle with longitudinal dynamics of an HMMWV for an ITS testbed. *IEEE/ASME Transactions on Mechatronics*, 13:1–12, 2008.