Safe Motion Planning under Partial Observability with an Optimal Deterministic Planner

Jeffrey Kane Johnson

Abstract—This paper derives a principled framework for efficiently and safely navigating partially observable multi-agent systems using an optimal deterministic planner. This is accomplished by decoupling the navigation problem into independent collision avoidance and guidance problems and by providing mechanisms for solving both efficiently. While the framework does forgo global optimality in order to compute solutions, we argue that such optimality is unattainable in practice due to intractability, so nothing is actually sacrificed. An example solution is demonstrated for a novel graph traversal problem using a deterministic, single-agent velocity profile planner in a partially observable, multi-agent setting.

Supplemental Material
Video demonstrations of the framework can be seen here: https://jeffreykanejohnson.com/publications/#pst.

I. INTRODUCTION

Robotics, fundamentally, is the problem of how machines engage in meaningful real-world interactions. These interactions require reasoning at varying levels of complexity, and as machines have become capable of more complex reasoning, robots have become capable of more complex interactions. This can be seen in industries such as mining, health care, and automated driving [1].

While such applications are impressive, there are still basic types of interaction problems that remain difficult. One such problem is how to navigate efficiently in an unstructured non-adversarial multi-agent system. This is the kind of problem pedestrians face when navigating crowded sidewalks or drivers face when navigating crowded roadways. While specific variants of this problem have received much attention from researchers, e.g. [2], [3], [4], general solutions that are also robust and efficient remain elusive. To help address this, we build on an idea with a long history in robotics motion planning: that under certain conditions multi-agent navigation can be achieved by decomposing the problem into separate collision avoidance and guidance sub-problems.

This paper builds on results from [5] to derive a framework that provides a rigorous methodology for performing the collision avoidance/guidance decomposition in dynamically constrained, non-adversarial multi-agent systems. The decomposition relies on the ability of the agents to identify when and how interaction effects in multi-agent systems can be factored out of the navigation problem. Once a problem has been factored, the decomposition allows robust and efficient solutions to be found. As a simple demonstration, a partially observable multi-agent path network traversal (PNT) problem is defined and solved under the framework using a deterministic, single-agent planner. The conclusion discusses future work applying this framework to larger, complex, and safety-critical systems.

II. RELATED WORK

In decentralized control theory person-by-person solution techniques [6] are those in which each agent assumes arbitrary control strategies for all other agents and optimizes its own actions according to the assumed strategies. Person-by-person strategies provide no guarantees of global optimality, but can be computationally efficient. A complementary technique, the designer’s approach [7], views the control problem from the perspective of a system designer who attempts to solve the decentralized problem in a centralized fashion by computing control laws for all agents in open-loop. By using an open-loop strategy, the designer’s approach attempts to find solutions that converge on some kind of global optimality without incurring the computational cost of accounting for future observations.

The designer’s approach is related to decision process problem models known variously as open-loop techniques [8], unobservable Markov Decision Processes (MDPs) [9], or non-observable MDPs (NOMDPs) [10]. These problem models are partially observable MDPs that have only a single ‘null’ observation. The null observation function reduces the problem complexity significantly from the general case, which is NEXPComplete [11]. However, lacking the perfect observation information of an MDP, the NOMDP is still NP-Complete [12]. A treatment of the types of partially observable systems most suitable to open-loop strategies is given in [13], where it is noted that
problems that admit global/local action space decomposition are particularly well-suited. From an empirical standpoint, [8] showed that open-loop techniques can be effective even in large-scale/high-dimensional robotics control problems.

All of this is to say that an established approach for dealing with multi-agent control problems is to modify the problem such that dimensions relating to agent interaction can be factored out. In fact, some sort of factorization must occur; the problem complexity is too high otherwise. In one form or another, factorization underlies many successful methods, such as the behavior fusion architecture [14], the subsumption architecture [15], the Dynamic Window Approach [16], CIRCA [17], ORCA [18], and the ROAD system [19]. However, these approaches often require very specific assumptions about the world, e.g., finite states, first-order dynamics, or that future trajectories can be accurately predicted. These baked-in requirements inherently limit the applicability of the approaches to specific types of problems and, as shown empirically in [20], unexpected collisions can result when such methods are applied to other problem types. Our approach addresses this limitation by only requiring that certain properties of the state space be computable without specifying how they should be computed.

III. DEFINITIONS & PROPERTIES

This section reviews properties and definitions used to derive the presented framework. See [5] for more details including derivations and proofs.

Definition 1. The agent state \( A(x) \) is a state representation containing both the dynamic state of an agent \( A \) and the volume of workspace it occupies at state \( x \).

Definition 2. For a state \( x \) and path \( P \), the stopping path (SP) \( (SP) \) is the minimal set of agent states that \( A \) must occupy while reaching a reference velocity \( 1 \) from \( x \) along \( P \).

Definition 3. For a given agent state let \( P \) be the set of all feasible paths. The stopping region (SR) \( (SR) \) is the disjoint union of all SPs for \( P \in P \).

Definition 4. The actions of two agents are said to require coordination when the safety of either agent’s action sequence is not independent of the other’s.

Definition 5. A contingency plan \( (CP) \) is a control sequence that an agent can execute that guarantees no collisions.

Definition 6. Behavioral interactions are factorizable if control can maintain contingencies independently of interaction effects and without coordination.

Property 1. SP Disjointness is the property that a system is guaranteed to be factorizable if all agents have one or more SPs that are disjoint from the SRs of all other agents.

IV. PROBLEM DEFINITION

This section defines the specific multi-agent navigation problem that our framework will address. The problem is defined as a variant of the reciprocal \( n \)-body collision avoidance problem given in [18].

Problem 1. Let \( \mathcal{A} \) be a set of agents navigating a shared space and assume that collision is never initially inevitable. Assume each agent can observe the instantaneous dynamic state of the environment. Assume each agent has knowledge of the physical dynamic properties of the environment and of other agents, but that each agent actuates according to a unique decision process. Each agent may assume with certainty that other agents will prefer both to avoid collision and to avoid causing collision, but that otherwise the decision processes of other agents are not fully observable except through coordination. When \(|\mathcal{A}| > 2\), how can an agent choose a control with the guarantee that it satisfies the first priority below while attempting to satisfy the second?:

1) Ensure that it is possible to remain collision free.
2) Make progress toward a desired goal state.

In this paper, the solution to Problem 1 exploits Property 1 to decompose the problem into independent sub-problems for each of the objectives. As shown in [5], this decomposition is safe as long as agent interactions do not require coordination for collision avoidance. When coordination is not required, each agent can apply a person-by-person control strategy, which allows collision avoidance and guidance to be solved independently. The policies assumed for other agents must be self-preserving, but can otherwise be defined however is expedient and without regard for non-collision interactions. In fact, Property 1 also allows for agents to interact selectively with each other. Such a system could then use a decentralized joint-planning framework as in [21]. Such systems are outside the scope of this paper, but are an area of promising future work and may be highly relevant for automated vehicles with V2V capabilities.

A. Interaction Factorization

A basic result of multi-agent systems is that if all agents at all times possess at least one contingency plan, collision can always be avoided. Thus, if control can maintain contingencies independently of interaction effects, then the interactions can effectively be factored out of the control problem. This notion of interaction factorization rests on two assumptions: that the system consists of non-adversarial entities, and that agents in the system have reasonable understandings of the system’s dynamics.

The intuition behind Property 1 is that when contingency plans are SPs, they are independent of interactions and computable strictly from system dynamics. As such, the system as a whole is capable of remaining collision free through the independent maintenance of SP Disjointness by the agents. This process is what is referred to in this work as interaction factorization. The following sections will derive a navigation framework built around this factorization.

V. THE FRAMEWORK

Our framework consists of a guidance controller, a local collision avoidance controller, and a fusion module. The
guidance controller is part of the “assumed strategies” of the person-by-person control approach, so its structure is arbitrary and unimportant to the framework. We therefore only define the local controller and fusion module, leaving definition of the guidance controller to framework users.

A. Solution Strategy

In a pure optimization-based approach, an agent would solve the sub-goals of Problem $\Phi$ jointly with a stochastic optimal controller. But, as noted, such formulations are not generally tractable. However, an agent can solve Problem $\Phi$ by formulating the sub-goals as two independent tasks and fusing them in a way that maintains Property 1.

1) A guidance controller computes a control term $u^d$ that directs an agent to some goal

2) A local collision avoidance controller maintains a safe control set $U_{safe}$ that satisfies Property 1

The fusion module takes $u^d$ as input and computes an output control $u$ as follows:

1) If $U_{safe} \neq \emptyset$, take $\arg \min_{u \in U_{safe}} \mu(u,u^d)$

2) Else, compute $u$ to restore Property 1

A similar solution methodology is given in [22]. This decomposition is what allows optimal deterministic planners to be used to solve partially observable problems: The collision avoidance sub-problem is a strictly deterministic contingency planning problem, so a deterministic planner can solve it. As with any non-globally optimal strategy, deadlocks may occur. Resolution strategies vary by application, but future work will examine some general strategies.

Under this framework, the fusion module is responsible for priority blending and for maintaining Property 1. The local controller computes interaction factorizations, which requires the ability to compute SRs. While SR computation can be highly domain dependent, certain aspects of the computation are universally required. The following sections describe nominal routines that can be tailored to individual problem instances.

The local controller is defined in Algorithm 1. Line 3 provides a check against problem assumptions: if the disjointness condition is ever violated, then some assumption, either of the environment or interactions of other agents, has been broken and collision may become imminent. In this case, a mitigation strategy should be employed because collision avoidance can no longer be guaranteed.

B. Computing Stopping Regions

Let $\text{ComputeSR}$ be a function that computes SRs. Assume there exist functions to compute feasible paths for an agent, and to compute the SP given each of the feasible paths. The resulting swept volumes are combined into a composite set, the SR, using a disjoint union operation. Note that $\text{ComputeSR}$ is where the framework can use an optimal deterministic planner.

In principle, the set of feasible paths will be infinite in size, and each path may also be infinite in length. Thus, most practical applications will require some kind of discreteness and finiteness approximations to ensure computability. Luckily, as the next section will show, these types of approximations need not break Property 1.

C. Computing SP Disjointness

To compute SP Disjointness for the whole system, a solving algorithm needs to collect all indices of SPs, compute those that intersect the other agent’s SRs, and return the complement of that set. This can be accomplished with a relative complement operation. The $\text{DisjointSPs}$ function, defined in Algorithm 2, performs this operation.

Algorithm 1 For an agent state $A(x)$ and goal-directed control $u^d$, compute $u^*$ that allows the agent to remain collision free with respect to obstacles $O$ over horizon $T$ while minimizing $\mu(u, u^d)$.

1: procedure LOCALCONTROL($A(x)$, $u^d$, $O$, $T$)
2: $\Phi \leftarrow \text{SPDisjointControls}(A(x), O, T)$
3: if $\Phi$ is $\emptyset$ then
4: Compute mitigation control $u^*$
5: else
6: $U \leftarrow \text{InitialControls}(\Phi)$
7: $u^* \leftarrow \arg \min_{u \in U} \mu(u, u^d)$
8: end if
9: return $u^*$
10: end procedure

Algorithm 2 Find the SPs $e$ in SR that are disjoint from SR$^*$.

1: procedure DISJOINTSPs($SR, SR^*$)
2: $I \leftarrow$ index set from SR
3: $Q \leftarrow SR \cap SR^*$
4: $I_Q \leftarrow$ index set from $Q$
5: return $I \setminus I_Q$
6: end procedure

It is important to note that the relative complement operation with respect to the SRs of other agents is required for correctness. Lemma 1 shows that it is not sufficient to simply check for the existence of a mutually disjoint subset of SPs between agents.

Fig. 2: Illustration of the counterexample used in the proof of Lemma 1. Agents $A_1$ and $A_2$ have mutually disjoint SPs available, but no SPs are disjoint from the other’s SR.
Fig. 3: This figure illustrates the conservative SP Disjointness computation described by Lemma 2. A conservative SR boundary δSR it shown for a disc agent, along with stopping path subset that is made up of SP1 and SP2. Conservative stopping paths SP1* (dashed boundary) and SP2* (dotted boundary) are defined such that SP1* ∪ SP2* covers δSR.

**Lemma 1.** The existence of mutually disjoint SPs does not imply that the system is factorizable.

**Proof.** Proof proceeds by counterexample. Consider two agents A1 and A2, where A1 has SP1 and SP2 in its SR, and A2 has SP3 and SP4 in its SR. If SP1 only intersects SP3 and SP2 only intersects SP4, then the stopping paths SP1 and SP4, and SP2 and SP3 are mutually disjoint between the agents, but neither agent’s stopping paths are disjoint of the other’s SR. Thus, no SP is guaranteed collision free without coordination, so by Definition 6 the system is not factorizable. See Fig. 2.

To compute the full interaction factorization, implementations can exploit the fact that Property 1 does not require exactness. Lemma 2 shows that the property can be guaranteed so long as the SR boundary, δSR, can be conservatively approximated with respect to one or more of its SPs.

**Lemma 2.** For a stopping region SR1, let δSR1 be a conservative approximation to the boundary of SR1. Let SR1* ⊂ SR1 be a subset of SPs. For each SP ∈ SR1* let SP be SP with its boundary expanded such that the volume enclosed by δSR1 is covered by all SP* (as in Fig. 3). Let SR1* be the stopping region composed of all SP*. For any other SR2, if SP disjointness holds between SR1* and SR2, then it must hold between SR1 and SR2.

**Proof.** Assume SR1, SR1*, and SR2 as defined in the lemma. Let Q = SR1 ∩ SR2 and Q’ = SR1* ∩ SR2. Because SR1* is constructed conservatively, then it must be that Q ⊆ Q’, and, by definition, SP1 ⊆ SP1*. It follows directly that if there exists SP1* ∈ SR1* such that SP1* ≠ Q’, then there must exist SP1 ∈ SR1 such that SP1 ≠ Q. Thus, if Property 1 is satisfied for Q’, it must also be for Q.

Discreteness and finiteness assumptions among the SPs are therefore safe to make provided both conservative SPs and a conservative approximation to δSR can be computed. This result is a powerful tool for constructing efficient implementations.

**D. Preserving SP Disjointness over Time**

Most practical systems will compute controls with respect to intervals of time rather than points in time. This means that in order to maintain Property 1 over time, the framework also needs to compute disjointness with respect to time horizons. This horizon-based disjointness problem is formulated below, followed by a nominal algorithmic solution.

**Problem 2.** Under the constraints and conditions of Problem 1 assume an agent $A$ has a control set $U$. Given the system state at time $t$, what is the subset of controls $U^* \subseteq U$ for $A$ that preserves Property 1 up to a time horizon $T$?

Problem 2 is not defined with respect to agent decision processes, so a solution must account for any decision agents may make within $T$. To do this, we note that the magnitude of the state change due to decision processes is bounded by the magnitude of the state change possible due to dynamics. Thus, in theory, computing disjointness with respect to the union of SRs over all states that may be occupied within $T$ suffices to ensure Property 1. In practice, however, this is generally infeasible because it can require computing the union of an infinite number of swept volumes. As an alternative, it is usually possible to efficiently approximate just the boundary of this infinite union, especially when $T$ is small, and ensure disjointness from it. It is important to note that computing this boundary does not require accurate predictions of future interactions; it only requires bounded estimates of physical dynamics. Algorithm 3 presents an approximation algorithm for a single object with an obvious extension to multiple objects.

**Algorithm 3** For agent state $A(x)$ and object $O$, find control sequences $\Phi$ for $A$ that maintain Property 1 over horizon $T$. Let $\hat{\delta}(\cdot)$ be an SR boundary approximation function.

1: procedure SPDISJOINTCONTROLS($A(x), O, T$)
2: $\Phi \leftarrow Control\Sequences(A(x), T)$
3: for $\phi \in \Phi$ do
4: for $u \in \phi$ do
5: $A'(x) \leftarrow AgentStateAt(A(x), u_i)$
6: $SR \leftarrow ComputeSR(A'(x))$
7: if $\forall SP \in SR, SP \cap \hat{\delta}(O)$ is not $\emptyset$ then
8: $\text{EraseFrom}(\phi, \Phi)$
9: break
10: end if
11: end for
12: end for
13: return $\Phi$
14: end procedure

**VI. Implementation & Evaluation**

This section introduces the path network traversal problem used to evaluate the presented framework. The evaluation
Problem 3. For a reciprocal below as a variant of Problem 1.

A. Completion time variance and failure rate

To define CTV, let $\Gamma$ be some problem scenario that is parameterized by a vector $\theta$, and let $f(\Gamma, \theta)$ be a simulation function that computes the time it takes an agent to reach a goal state in scenario $\Gamma$ under parameter vector $\theta$. Let a sequence $\Theta_1, \ldots, \Theta_n$ be i.i.d. random parameter vectors and let $\Omega^* = \{ \theta_1, \ldots, \theta_n \}$ be a sample of size $n$ drawn from $\Theta_1, \ldots, \Theta_n$. For $n > 1$, CTV is equal to the unbiased sample variance estimate of $f(\Gamma; \theta)$ applied across $\Omega^*$.

B. The path network traversal problem

The PNT problem is a partially observable multi-agent navigation problem. As noted in [1], optimal solutions to these problems are computationally intractable. It is defined below as a variant of Problem [1]

**Problem 3.** For a reciprocal $n$-body collision avoidance problem, let $A$ be a set of agents navigating a graph $G = (V, E)$. Assume all agents are initialized at some $v \in V$ and that collision is never initially inevitable. Assume $[\hat{s}_{\min}, \hat{s}_{\max}]$ and $[\hat{s}_{\min}, \hat{s}_{\max}]$ are bounds on speed and acceleration. Assume an agent at $\hat{s}_{\max}$ can traverse at most $e_{\max}$ edges and can stop within at most $e_{\min}$ edges. The problem for a given agent is to navigate from a start vertex to a goal vertex without collision within a time horizon $T$.

To solve Problem 3 the following problem-specific versions of the nominal routines are needed. For brevity, we omit the precise details of the functions and only summarize how they are computed.

1) **ComputeSR:** As noted in [V-B] it is this function that can utilize a deterministic planner. To solve the PNT problem, we will use the optimal speed profile planner from [23]. **ComputeSR** needs the set of feasible paths and the minimum swept volumes along those paths; together these pieces of information are the SPs, which define the SR.

- **Feasible Paths:** $G$ may have infinite feasible paths, but only SPs are needed. Since the maximum length of these paths is fully determined by $\hat{s}_{\max}$, $\hat{s}_{\max}$, and $A(x)$, this function need only find all paths of such length. The length is computed by the speed profile planner, and paths are found by simple graph search over a number of edges determined by $e_{\max}$.

- **Minimum Swept Path:** The agent models are discs and the control sequences $\phi_i$ are always $\hat{s}_{\min}$, so the swept paths are 2D capsules [24] that are easily computed given the stopping speed profile computed by the planner, the path, $\hat{s}_{\min}$, $A(x)$, and $\phi_i$.

2) $\hat{\delta}(\cdot)$: This is the SR boundary approximation function. Since $T$ is short, we simplify the operation by discarding the time dimension of SR computation. Both the number and maximum length of paths $A$ can traverse within $T$ are determined by the agent’s dynamics and the speed profile planner. Thus, the true set of SRs is contained by a union of capsules that are each the union of the swept volumes along each path with the SR computed from the end of the path.

3) **SPDisjointControls:** Once the SR approximations are computed, they are mapped in polynomial time [23] to obstacles in the planning space used by our deterministic planner, and the planner computes collision-free control sets.

4) **LocalControl:** This method can now be implemented directly as specified in Algorithm [1].

C. Results & Analysis

This section provides results of computing CTV for a single agent in a PNT problem defined on a graph $G$, which is a $6 \times 6$ grid with 12 agents that attempt to traverse from one side to the other in a straight line. All agents are unit discs with reference points at their centers (Fig. [1]). All information about $G$ and dynamic constraints is stored in $\Gamma$. Mean parameter vector $\mu_0$ is initialized with agent positions along the outside edges of $G$ oriented toward the opposite side with initial speeds and accelerations zero. Agent guidance controllers are random walks over $[0, \hat{s}_{\max}]$. Each agent’s assumed strategy for all others is simple constant velocity. The sample count $n$ for all trials is 100. For all agents $H = 10$, $[\hat{s}_{\min}, \hat{s}_{\max}] = [0, 10]$, $[\hat{s}_{\min}, \hat{s}_{\max}] = [-4, 4]$.

Two types of trials were conducted: one in which only the initial conditions were randomized, and another in which online speed estimates were additionally noisy. Results for the first are summarized in Table II and the second in Table III.

<table>
<thead>
<tr>
<th>Objects</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\Delta}$ (ms)</td>
<td>27.85</td>
<td>42.23</td>
<td>69.66</td>
<td>106.2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
In Table I, the CTV trends down toward ~0.6 then levels off. We hypothesize that this is because $\mu$ positions the objects in a regular array along the outer edges of $G$, which can lead to traffic jams. However, as $\sigma_{\text{position}}$ increases, the objects become more evenly distributed, which eases out traffic flow, reducing expected completion times. This behavior indicates that the expected completion time is directly affected by traffic density and that the algorithm is capable of negotiating dense traffic at the expense of completion time.

For the trials in Table II there is less of a clear trend in CTV, which can be expected because of the persistent state uncertainty in these trials. Interestingly, average completion time and CTV do not differ greatly from those in Table I. The increased uncertainty caused agents to engage in more conservative turn-taking behavior, which allowed traffic to clear without jamming. Note that these behavioral interactions were emergent, which indicates that the proposed framework has inherent robustness properties.

Table III shows the growth of running time per simulation run as a function of object count. The simulation used state observation uncertainty $\sigma_s = 0.2$. The data indicate that, despite the partial observability, complexity growth is polynomial in the number of agents. This is to be expected, as the framework itself is polynomial in the complexity of the deterministic planner, which is itself polynomial.

VII. CONCLUSIONS & FUTURE WORK

This paper derived a framework that enables efficient navigation in partially observable multi-agent domains. The framework builds on previous results in a general way to enable application to a wide variety of problems and scenarios while maintaining guarantees on tractability and collision avoidance. A sample problem was presented and problem-specific routines were derived under the framework to enable application to a wide variety of problems and navigation in partially observable multi-agent domains. The framework builds on previous results in a general way to enable application to automotive active safety systems.

REFERENCES