A message-passing algorithm for multi-agent trajectory planning

Authored by:

Jonathan S. Yedidia
Jose Bento
Nate Derbinsky
Javier Alonso-Mora

Abstract

We describe a novel approach for computing collision-free emph{global} trajectories for p agents with specified initial and final configurations, based on an improved version of the alternating direction method of multipliers (ADMM) algorithm. Compared with existing methods, our approach is naturally parallelizable and allows for incorporating different cost functionals with only minor adjustments. We apply our method to classical challenging instances and observe that its computational requirements scale well with p for several cost functionals. We also show that a specialization of our algorithm can be used for {em local} motion planning by solving the problem of joint optimization in velocity space.

1 Paper Body

Robot navigation relies on at least three sub-tasks: localization, mapping, and motion planning. The latter can be described as an optimization problem: compute the lowest-cost path, or trajectory, between an initial and final configuration. This paper focuses on trajectory planning for multiple agents, an important problem in robotics [1, 2], computer animation, and crowd simulation [3]. Centralized planning for multiple agents is PSPACE hard [4, 5]. To contend with this complexity, traditional multi-agent planning prioritizes agents and computes their trajectories sequentially [6], leading to suboptimal solutions. By contrast, our method plans for all agents simultaneously. Trajectory planning is also simplified if agents are non-distinct and can be dynamically assigned to a set of goal positions [1]. We consider the harder problem where robots have a unique identity and their goal positions are statically prespecified. Both mixed-integer quadratic programming (MIQP) [7] and [more efficient, although local] sequential convex programming [8] approaches have been applied to the problem of computing collision-free trajectories for multiple

agents with pre-specified goal positions; however, due to the non-convexity of the problem, these approaches, especially the former, do not scale well with the number of agents. Alternatively, trajectories may be found by sampling in their joint configuration space [9]. This approach is probabilistic and, alone, only gives asymptotic guarantees. See Appendix A for further comments on discrete search methods. Due to the complexity of planning collision-free trajectories, real-time robot navigation is commonly decoupled into a global planner and a fast local planner that performs collision-avoidance. Many single-agent reactive collision-avoidance algorithms are based either on potential fields [10], which typically ignore the velocity of other agents, or ?velocity obstacles? [11], which provide improved performance in dynamic environments by formulating the optimization in velocity space instead of Cartesian space. Building on an extension of the velocity-obstacles approach, recent work on centralized collision avoidance [12] computes collision-free local motions for all agents whilst maximizing a joint utility using either a computationally expensive MIQP or an efficient, though local, QP. While not the main focus of this paper, we show that a specialization of our approach ?

This author would like to thank Emily Hupf and Noa Ghersin for their support while writing this paper.

1

to global-trajectory optimization also applies for local-trajectory optimization, and our numerical results demonstrate improvements in both efficiency and scaling performance. In this paper we formalize the global trajectory planning task as follows. Given p agents of different radii {ri }pi=1 with given desired initial and final positions, {xi (0)}pi=1 and {xi (T)}pi=1 , along with a cost functional over trajectories, compute collision-free trajectories for all agents that minimize the cost functional. That is, find a set of intermediate points {xi (t)}pi=1 , t 2 (0, T), that satisfies the ?hard? collision-free constraints that kxi (t) xj (t)k ¿ ri + rj , for all i, j and t, and that insofar as possible, minimizes the cost functional. The method we propose searches for a solution within the space of piece-wise linear trajectories, wherein the trajectory of an agent is completely specified by a set of positions at a fixed set of time instants {ts }?s=0 . We call these time instants break-points and they are the same for all agents, which greatly simplifies the mathematics of our method. All other intermediate points of the trajectories are computed by assuming that each agent moves with constant velocity in between break-points: if t1 and t2 ¿ t1 are consecutive break-points, then xi (t) = t2 1 t1 ((t2 t)xi (t1) + (t t1)xi (t2)) for t 2 [t1 , t2]. Along with the set of initial and final configurations, the number of interior break-points (? 1) is an input to our method, with a corresponding tradeoff: increasing ? yields trajectories that are more flexible and smooth, with possibly higher quality; but increasing ? enlarges the problem, leading to potentially increased computation. The main contributions of this paper are as follows: i) We formulate the global trajectory planning task as a decomposable optimization problem. We show how to solve the resulting sub-problems exactly and efficiently, despite their nonconvexity, and how to coordinate their solutions using message-passing. Our method, based on the ?three-weight? version of ADMM

[13], is easily parallelized, does not require parameter tuning, and we present empirical evidence of good scalability with p. ii) Within our decomposable framework, we describe different sub-problems, called minimizers, each ensuring the trajectories satisfy a separate criterion. Our method is flexible and can consider different combinations of minimizers. A particularly crucial minimizer ensures there are no inter-agent collisions, but we also derive other minimizers that allow for finding trajectories with minimal total energy, avoiding static obstacles, or imposing dynamic constraints, such as maximum/minimum agent velocity. iii) We show that our method can specialize to perform local planning by solving the problem of joint optimization in velocity space [12]. Our work is among the few examples where the success of applying ADMM to find approximate solutions to a large non-convex problems can be judged with the naked eye, by the gracefulness of the trajectories found. This paper also reinforces the claim in [13] that small, yet important, modifications to ADMM can bring an order of magnitude increase in speed. We emphasize the importance of these modifications in our numerical experiments, where we compare the performance of our method using the three-weight algorithm (TWA) versus that of standard ADMM. The rest of the paper is organized as follows. Section 2 provides background on ADMM and the TWA. Section 3 formulates the global-trajectory-planning task as an optimization problem and describes the separate blocks necessary to solve it (the mathematical details of solving these subproblems are left to appendices). Section 4 evaluates the performance of our solution: its scalability with p, sensitivity to initial conditions, and the effect of different cost functionals. Section 5 explains how to implement a velocity-obstacle method using our method and compares its performance with prior work. Finally, Section 6 draws conclusions and suggests directions for future work.

2

Minimizers in the TWA

In this section we provide a short description of the TWA [13], and, in particular, the role of the minimizer building blocks that it needs to solve a particular optimization problem. Section B of the supplementary material includes a full description of the TWA. As a small illustrative example of how the TWA is used to solve optimization problems, suppose we want to solve minx2R3 f (x) = min{x1 ,x2 ,x3 } f1 (x1 , x3) + f2 (x1 , x2 , x3) + f3 (x3), where fi (.) 2 2

R[{+1}. The functions can represent soft costs, for example f3 (x3) = (x3 1)2 , or hard equality or inequality constraints, such as f1 (x1 , x3) = J(x1 ? x3), where we are using the notation J(.) = 0 if (.) is true or +1 if (.) is false. The TWA solves this optimization problem iteratively by passing messages on a bipartite graph, in the form of a Forney factor graph [14]: one minimizer-node per function fb , one equality-node per variable xj and an edge (b, j), connecting b and j, if fb depends on xj (see Figure 1-left). 1

g

=

1

2

g
=
2
3
g
=
3
? n1,1, ?1,1 ? n1,3, ?1,3
g1
? x1,1, ?1,1 ? x1,3, ?1,3
? n2,1, ?2,1 ? n2,2 , ?2,2 ? n2,3, ?2,3
g2
? x2,1, ?2,1 ? x2,2 , ??2,2 x2,3, ?2,3
? n3,3, ?3,3
g3
? x3,3, ?3,3

Figure 1: Left: bipartite graph, with one minimizer-node on the left for each function making up the overall objective function, and one equality-node on the right per variable in the problem. Right: The input and output variables for each minimizer block. Apart from the first-iteration message values, and two internal parameters1 that we specify in Section 4, the algorithm is fully specified by the behavior of the minimizers and the topology of the graph. What does a minimizer do? The minimizer-node g1 , for example, solves a small optimization problem over its local variables x1 and x3 . Without going into the full detail presented in [13] and the supplementary material, the estimates x1,1 and x1,3 are then combined with running sums of the differences between the minimizer estimates and the equality-node consensus estimates to obtain messages m1,1 and m1,3 on each neighboring edge that are sent to the neighboring equality-nodes along with corresponding certainty weights, ! ? 1,2 and ! ? 1,3 . All other minimizers act similarly. The equality-nodes receive these local messages and weights and produce consensus estimates for all variables by computing an average of the incoming messages, weighted by the incoming certainty weights ! ? . From these consensus estimates, correcting messages are computed and communicated back to the minimizers to help them reach consensus. A certainty weight for the correcting messages, ? , is also communicated back to the minimizers. For example, the minimizer g1 receives correcting messages n1,1 and n1,3 with corresponding certainty weights ? 1,1 and ? 1,3 (see Figure 1-right). When producing new local estimates, the bth minimizer node computes its local estimates {xj } by choosing a point that minimizes the sum of the local function fb and weighted squared distance from the incoming messages (ties are broken randomly): 2 3 X 1 {xb,j }j = gb {nb,j }j , { ? kb,j }j ? arg min 4fb ({xj }j) + ? b,j (xj nb,j)2 5 , (1) 2 j {xj }j

P where {}j and j run over all equality-nodes connected to b. In the TWA, the certainty weights {! ? b,j } that this minimizer outputs must be 0 (uncertain); 1 (certain); or ?0 , set to some fixed value. The logic for setting weights from minimizer-nodes depends on the problem; as we shall see, in trajectory

planning problems, we only use 0 or ?0 weights. If we choose that all minimizers always output weights equal to ?0 , the TWA reduces to standard ADMM; however, 0-weights allows equality-nodes to ignore inactive constraints, traversing the search space much faster. Finally, notice that all minimizers can operate simultaneously, and the same is true for the consensus calculation performed by each equality-node. The algorithm is thus easy to parallelize.

3

Global trajectory planning

We now turn to describing our decomposition of the global trajectory planning optimization problem in detail. We begin by defining the variables to be optimized in our optimization problem. In 1

These are the step-size and ?0 constants. See Section B in the supplementary material for more detail.

3

our formulation, we are not tracking the points of the trajectories by a continuous-time variable taking values in [0, T]. Rather, our variables are the positions {xi (s)}i2[p] , where the trajectories are indexed by i and break-points are indexed by a discrete variable s taking values between 1 and ? 1. Note that {xi (0)}i2[p] and {xi (?)}i2[p] are the initial and final configuration, sets of fixed values, not variables to optimize. 3.1

Formulation as unconstrained optimization without static obstacles

In terms of these variables, the non-collision constraints2 are k(?xi (s + 1) + (1 ?)xi (s)) (?xj (s + 1) + (1 for all i, j 2 [p], s 2 {0, ..., ? 1} and ? 2 [0, 1].

?)xj (s))k

ri + rj ,

(2)

The parameter ? is used to trace out the constant-velocity trajectories of agents i and j between break-points s + 1 and s. The parameter ? has no units, it is a normalized time rather than an absolute time. If t1 is the absolute time of the break-point with integer index s and t2 is the absolute time of the break-point with integer index s + 1 and t parametrizes the trajectories in absolute time then ? = (t t1)/(t2 t1). Note that in the above formulation, absolute time does not appear, and any solution is simply a set of paths that, when travelled by each agent at constant velocity between break-points, leads to no collisions. When converting this solution into trajectories parameterized by absolute time, the break-points do not need to be chosen uniformly spaced in absolute time. The constraints represented in (2) can be formally incorporated into an unconstrained optimization problem as follows. We search for a solution to the problem: min f cost ({xi (s)}i,s) +

{xi (s)}i,s

n X1 X

frcoll (xi (s), xi (s + 1), xj (s), xj (s + 1)), i ,rj

(3)

s=0 i¿j

where {xi (0)}p and {xi (?)}p are constants rather than optimization variables, and where the function f cost is a function that represents some cost to

be minimized (e.g. the integrated kinetic energy coll or the maximum velocity over all the agents) and the function fr,r 0 is defined as, coll 0 0 fr,r 0 (x, x, x , x) = J(k?(x
x0) + (1
?)(x
x0)k
r + r0 8? 2 [0, 1]).
(4)

In this section, x and x represent the position of an arbitrary agent of radius r at two consecutive break-points and x0 and x0 the position of a second arbitrary agent of radius r0 at the same breakpoints. In the expression above J(.) takes the value 0 whenever its argument, a clause, is true and coll takes the value +1 otherwise. Intuitively, we pay an infinite cost in fr,r 0 whenever there is a collision, and we pay zero otherwise. In (3) we can set f cost (.), to enforce a preference for trajectories satisfying specific properties. For example, we might prefer trajectories for which the total kinetic energy spent by the set of agents is small. In this case, defining fCcost (x, x) = Ckx xk2 , we have, p n 1
f cost ({xi (s)}i,s) =
1 X X cost f (xi (s), xi (s + 1)). pn i=1 s=0 Ci,s
(5)

where the coefficients {Ci,s } can account for agents with different masses, different absolute-time intervals between-break points or different preferences regarding which agents we want to be less active and which agents are allowed to move faster. More simply, we might want to exclude trajectories in which agents move faster than a certain amount, but without distinguishing among all remaining trajectories. For this case we can write, fCcost (x, x) = J(kx
xk ? C).
(6)

In this case, associating each break-point to a time instant, the coefficients {Ci,s } in expression (5) would represent different limits on the velocity of different agents between different sections of the trajectory. If we want to force all agents to have a minimum velocity we can simply reverse the inequality in (6). 2 We replaced the strict inequality in the condition for non-collision by a simple inequality ? ? to avoid technicalities in formulating the optimization problem. Since the agents are round, this allows for a single point of contact between two agents and does not reduce practical relevance.

4

3.2

Formulation as unconstrained optimization with static obstacles

In many scenarios agents should also avoid collisions with static obstacles. Given two points in space, xL and xR , we can forbid all agents from crossing the line segment from xL to xR by adding Pp Pn 1 the following term to the function (3): i=1 s=0 fxwall (xi (s), xi (s + 1)). We recall that ri is L ,xR ,ri the radius of agent i and fxwall (x, x) = J(k(?x + (1 L ,xR ,r

Notice that f
coll

?)x)
can be expressed using f
(xR + (1
wall
coll 0 0 fr,r 0 (x, x, x , x)
)xL)k
r for all ?,
. In particular,
wall 0 = f0,0,r+r 0 (x
x, x0
2 [0, 1]).
x).
(7) (8)

We use this fact later to express the minimizer associated with agent-agent collisions using the minimizer associated with agent-obstacle collisions. When agents move in the plane, i.e. xi (s) 2 R2 for all i 2 [p] and s+1 2 [? +1], being able to avoid collisions with a general static line segment allows to automatically avoid collisions with multiple static obstacles of arbitrary polygonal shape. Our numerical experiments only consider agents in the plane and so, in this paper, we only describe the minimizer block for wall collision for a 2D world. In higher dimensions, different obstacle primitives need to be considered. 3.3

Message-passing formulation

To solve (3) using the TWA, we need to specify the topology of the bipartite graph associated with the unconstrained formulation (3) and the operation performed by every minimizer, i.e. the ! ?weight update logic and x-variable update equations. We postpone describing the choice of initial values and internal parameters until Section 4. We first describe the bipartite graph. To be concrete, let us assume that the cost functional has the form of (5). The unconstrained formulation (3) then tells us that the global objective function is the sum of ?p(p + 1)/2 terms: ?p(p 1)/2 functions f coll and ?p functions fCcost . These functions involve a total of (? + 1)p variables out of which only (? 1)p are free (since the initial and final configurations are fixed). Correspondingly, the bipartite graph along which messages are passed has ?p(p + 1)/2 minimizer-nodes that connect to the (? + 1)p equality-nodes. In particular, the equalitynode associated with the break-point variable xi (s), ? ¿ s ¿ 0, is connected to 2(p 1) different cost g coll minimizer-nodes and two different gC minimizer-nodes. If s = 0 or s = ? the equality-node cost only connects to half as many g coll nodes and gC nodes. We now describe the different minimizers. Every minimizer basically is a special case of (1). 3.3.1

Agent-agent collision minimizer

We start with the minimizer associated with the functions f coll , that we denoted by g coll . This minimizer receives as parameters the radius, r and r0 , of the two agents whose collision it is avoiding. The minimizer takes as input a set of incoming n-messages, {n, n, n0 , n0 }, and associated ? -weights, { ? , ? , ? 0 , ? 0 }, and outputs a set of updated x-variables according to expression (9). Messages n and n come from the two equality-nodes associated with the

positions of one of the agents at two consecutive break-points and n0 and n0 from the corresponding equality-nodes for the other agent. g coll (n, n, n0 , n0 , ? , ? , ? 0 , ? 0 , r, r0) = arg

min

{x,x,x0 ,x0 }

coll 0 0 fr,r 0 (x, x, x , x)

? ?0 0 ? ?0 0 kx nk2 + kx nk2 + kx n0 k2 + kx n0 k2 . (9) 2 2 2 2 The update logic for the weights ! ? for this minimizer is simple. If the trajectory from n to n for an agent of radius r does not collide with the trajectory from n0 to n0 for an agent of radius r0 then set all the outgoing weights ! ? to zero. Otherwise set them all to ?0 . The outgoing zero weights indicate to the receiving equality-nodes in the bipartite graph that the collision constraint for this pair of agents is inactive and that the values it receives from this minimizer-node should be ignored when computing the consensus values of the receiving equality-nodes. +

The solution to (9) is found using the agent-obstacle collision minimizer that we describe next. 5

3.3.2

Agent-obstacle collision minimizer

The minimizer for f wall is denoted by g wall . It is parameterized by the obstacle position {xL , xR } as well as the radius of the agent that needs to avoid the obstacle. It receives two n-messages, {n, n}, and corresponding weights { ? , ? }, from the equality-nodes associated with two consecutive positions of an agent that needs to avoid the obstacle. Its output, the x-variables, are defined as g wall (n, n, r, xL , xR , ? , ?) = arg min fxwall (x, x) + L ,xR ,r {x,x}

? kx 2

nk2 +

? kx 2

nk2 .

(10)

When agents move in the plane (2D), this minimizer can be solved by reformulating the optimization in (10) as a mechanical problem involving a system of springs that we can solve exactly and efficiently. This reduction is explained in the supplementary material in Section D and the solution to the mechanical problem is explained in Section I. The update logic for the ! ? -weights is similar to that of the g coll minimizer. If an agent of radius

r going from n and n does not collide with the line segment from xL to xR then set all outgoing weights to zero because the constraint is inactive; otherwise set all the outgoing weights to ?0 . Notice that, from (8), it follows that the agent-agent minimizer g coll can be expressed using g wall . More concretely, as proved in the supplementary material, Section C, ? ? g coll (n, n, n0 , n0 , ? , ? , ? 0 , ? 0 , r, r0) = M2 g wall M1 .{n, n, n0 , n0 , ? , ? , ? 0 , ? 0 , r, r0 } , for a constant rectangular matrix M1 and a matrix M2 that depend on {n, n, n0 , n0 , ? , ? , ? 0 , ? 0 }. 3.3.3

Minimum energy and maximum (minimum) velocity minimizer

When f cost can be decomposed as in (5), the minimizer associated with the functions f cost is denoted by g cost and receives as input two n-messages, {n, n}, and corresponding weights, { ? , ? }. The messages come from two equality-nodes associated with two consecutive positions of an agent. The minimizer is also parameterized by a cost factor c. It outputs a set of updated x-messages defined as g cost (n, n, ? , ? , c) = arg min fccost (x, x) + {x,x}

? kx 2

nk2 +

? kx 2

nk2 .

(11)

The update logic for the ! ? -weights of the minimum energy minimizer is very simply: always set all outgoing weights ! ? to ?0 . The update logic for the ! ? -weights of the maximum velocity minimizer is the following. If kn nk ? c set all outgoing weights to zero. Otherwise, set them to ?0 . The update logic for the minimum velocity minimizer is similar. If kn nk c, set all the ! ? -weights to zero. Otherwise set them to ?0 . The solution to the minimum energy, maximum velocity and minimum velocity minimizer is written in the supplementary material in Sections E, F, and G respectively.

4

Numerical results

We now report on the performance of our algorithm (see Appendix J for an important comment on the anytime properties of our algorithm). Note that the lack of open-source scalable algorithms for global trajectory planning in the literature makes it difficult to benchmark our performance against other methods. Also, in a paper it is difficult to appreciate the gracefulness of the discovered trajectory optimizations, so we include a video in the supplementary material that shows final optimized trajectories as well as intermediate results as the algorithm progresses for a variety of additional scenarios, including those with obstacles. All the tests described here are for agents in a twodimensional plane. All tests but the last were performed using six cores of a 3.4GHz i7 CPU. The different tests did not require any special tuning of parameters. In particular, the step-size in [13] (their ? variable) is always 0.1. In order to quickly equilibrate the system to a reasonable set of variables and to wash out the importance of initial conditions, the default weight ?0 was set equal to a small value (?p ? 10 5) for the first 20 iterations and then set to 1 for all further iterations. 6

The first test considers scenario CONF1: p (even) agents of radius r, equally spaced around on a circle of radius R, are each required to exchange position with the corresponding antipodal agent, r = (5/4)R sin(?/2(p 4)). This is a classical difficult test scenario because the straight line motion of all agents to their goal would result in them all colliding in the center of the circle. We compare the convergence time of the TWA with a similar version using standard ADMM to perform the optimizations. In this test, the algorithm?s initial value for each variable in the problem was set to the corresponding initial position of each agent. The objective is to minimize the total kinetic energy (C in the

energy minimizer is set to 1). Figure 2-left shows that the TWA scales better with p than classic ADMM and typically gives an order of magnitude speed-up. Please see Appendix K for a further comment on the scaling of the convergence time of ADMM and TWA with p. Convergence time (sec) ?

Number of occurrences
?
?=4 ?
?
1500
?=8 ?
? ?
0
?
0
? ?
20
? ? ? ?
?
? ?
?
? ? ? ?
? ?
?
40
3
4
5
6
7
2500
60
? = ?6
25
20
15
10
5
? = ?4
80
100
p = 100 2000
1500
?
1000
? ? 500
?

0 100
200
Number of agents, p
300
400
500
600
700
Physical cores (? 12)
?
500
2
Convergence time (sec)
?=6
2000
1000
1
?
?=8
0
0
p = 80
?
?
?
p ?= 60 ? ? p?? ? ? 40 ? ? ?
?
?
? 5
Objective value of trajectories
?
? ?
? ?
?
?
?
?
?
?
? ? ?
10
? ? ?
? ?
? ? ?
15
? ? ?

? ?
? ? ?
? ?
? ? ?
20
Number of cores

Figure 2: Left: Convergence time using standard ADMM (dashed lines) and using TWA (solid lines). Middle: Distribution of total energy and time for convergence with random initial conditions (p = 20 and ? = 5). Right: Convergence time using a different number of cores (? = 5). The second test for CONF1 analyzes the sensitivity of the convergence time and objective value when the variables? value at the first iteration are chosen uniformly at random in the smallest spacetime box that includes the initial and final configuration of the robots. Figure 2-middle shows that, although there is some spread on the convergence time, our algorithm seems to reliably converge to relatively similar-cost local minima (other experiments show that the objective value of these minima is around 5 times smaller than that found when the algorithm is run using only the collision avoidance minimizers without a kinetic energy cost term). As would be expected, the precise trajectories found vary widely between different random runs. Still for CONF1, and fixed initial conditions, we parallelize our method using several cores of a 2.66GHz i7 processor and a very primitive scheduling/synchronization scheme. Although this scheme does not fully exploit parallelization, Figure 2-right does show a speed-up as the number of cores increases and the larger p is, the greater the speed-up. We stall when we reach the twelve physical cores available and start using virtual cores.

Convergence time, one epoch (sec)

Finally, Figure 3-left compares the convergence time to optimize the total energy with the time to simply find a feasible (i.e. collision-free) solution. The agents initial and final configuration is randomly chosen in the plane (CONF2). Error bars indicate ? one standard deviation. Minimizing the kinetic energy is orders of magnitude computationally more expensive than finding a feasible solution, as is clear from the different magnitude of the left and right scale of Figure 3-left. ?

?=8
12
?=8 10
1500
?=6 ?
?=4
8
1200 ?
6
? ?
?
4 ?
2 0

?
0
?
20
? ?
? ? ?
40
?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
60
? ? ? ?
? ? ?
600
?
?=6 ?
900
? 300
?=4
Minimum energy
Feasible
1800
Convergence time (sec)
Convergence time (sec)
Convergence time (sec)
30
?
0 80
3.0
2.5
Pink: MIQP
2.0
Light blue: TWA
1.5
1.0
0.5
0.0 8
100
Number of agents, p
10* 12
14* 16
18* 20
24* 24
30* 32
40* 40

50* 52

Number of agents, p

Figure 3: Left: Convergence time when minimizing energy (blue scale/dashed lines) and to simply find a feasible solution (red scale/solid lines). Right: (For Section 5). Convergence-time distribution for each epoch using our method (blue bars) and using the MIQP of [12] (red bars and star-values).

7

5

Local trajectory planning based on velocity obstacles

In this section we show how the joint optimization presented in [12], which is based on the concept of velocity obstacles [11] (VO), can be also solved via the message-passing TWA. In VO, given the current position {xi (0)}i2[p] and radius {ri } of all agents, a new velocity command is computed jointly for all agents minimizing the distance to their preferred velocity {viref }i2[p] . This new velocity command must guarantee that the trajectories of all agents remain collision-free for at least a time horizon ? . New collision-free velocities are computed every ?? seconds, ? ¡ 1, until all agents reach their final configuration. Following [12], and assuming an obstacle-free environment and first order dynamics, the collision-free velocities are given by, X minimize Ci kvi viref k2 s.t. k(xi (0) + vi t) (xj (0) + vj t)k ri + rj 8 i 2 [p], t 2 [0, ?]. {vi }i2[p]

i2[p]

Since the velocities {vi }i2[p] are related linearly to the final position of each object after ? seconds, {xi (?)}i2[p] , a simple change of variables allows us to reformulate the above problem as, X 2 minimize Ci0 kxi xref i k {xi }i2[p]

i2[p]

s.t. k(1 ?)(xi (0) xj (0)) + ?(xi xj)k ri + rj 8 j ¿ i 2 [p], ? 2 [0, 1] (12) 2 ref ref where = Ci /? , xi = xi (0) + vi ? and we have dropped the ? in xi (?). The above problem, extended to account for collisions with the static line segments {xRk , xLk }k , can be formulated in an unconstrained form using the functions f cost , f coll and f wall . Namely, X X XX ref min fCcost frcoll (xi (0), xi , xj (0), xj) + fxwall (xi (0), xi). (13) 0 (xi , xi) + i ,rj R k ,xL k ,ri Ci0

{xi }i

i

i2[p]

i¿j

i2[p] k

{xref i }i

Note that {xi (0)}i and are constants, not variables being optimized. Given this formulation, the TWA can be used to solve the optimization. All corresponding minimizers are special cases of minimizers derived in the previous section for global trajectory planning (see Section H in the supplementary material for details). Figure 3-right shows the distribution of the time to solve (12) for CONF1. We compare the mixed integer quadratic programming (MIQP) approach from [12] with ours. Our method finds a local minima of exactly (13), while [12] finds a global minima of an approximation to (13). Specifically, [12]

requires approximating the search domain by hyperplanes and an additional branch-and-bound algorithm while ours does not. Both approaches use a mechanism for breaking the symmetry from CONF1 and avoid deadlocks: theirs uses a preferential rotation direction for agents, while we use agents with slightly different C coefficients in their energy minimizers (Cith agent = 1 + 0.001i). Both simulations were done on a single 2.66GHz core. The results show the order of magnitude is similar, but, because our implementation is done in Java while [12] uses Matlab-mex interface of CPLEX 11, the results are not exactly comparable.

6

Conclusion and future work

We have presented a novel algorithm for global and local planning of the trajectory of multiple distinct agents, a problem known to be hard. The solution is based on solving a non-convex optimization problem using TWA, a modified ADMM. Its similarity to ADMM brings scalability and easy parallelization. However, using TWA improves performance considerably. Our implementation of the algorithm in Java on a regular desktop computer, using a basic scheduler/synchronization over its few cores, already scales to hundreds of agents and achieves real-time performance for local planning. The algorithm can flexibly account for obstacles and different cost functionals. For agents in the plane, we derived explicit expressions that account for static obstacles, moving obstacles, and dynamic constraints on the velocity and energy. Future work should consider other restrictions on the smoothness of the trajectory (e.g. acceleration constraints) and provide fast solvers to our minimizers for agents in 3D. The message-passing nature of our algorithm hints that it might be possible to adapt our algorithm to do planning in a decentralized fashion. For example, minimizers like g coll could be solved by message exchange between pairs of agents within a maximum communication radius. It is an open problem to build a practical communication-synchronization scheme for such an approach.

8

2 References

[1] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Roland Siegwart, and Paul Beardsley. Image and animation display with multiple mobile robots. 31(6):753?773, 2012. [2] Peter R. Wurman, Raffaello D?Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI Magazine, 29(1):9?19, 2008. [3] Stephen J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 177?187, 2009. [4] John H. Reif. Complexity of the mover?s problem and generalizations. In IEEE Annual Symposium on Foundations of Computer Science, pages 421?427, 1979. [5] John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the ?warehouseman?s prob-

lem?. The International Journal of Robotics Research, 3(4):76?88, 1984. [6] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. Robotics and Autonomous Systems, 41(2?3):89?99, 2002. [7] Daniel Mellinger, Alex Kushleyev, and Vijay Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In IEEE International Conference on Robotics and Automation, pages 477?483, 2012. [8] Federico Augugliaro, Angela P. Schoellig, and Raffaello D?Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1917?1922, 2012. [9] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. The International Journal of Robotics Research, 20(5):378?400, 2001. [10] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. The International Journal of Robotics Research, 5(1):90?98, 1986. [11] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. The International Journal of Robotics Research, 17(7):760?772, 1998. [12] Javier Alonso-Mora, Martin Rufli, Roland Siegwart, and Paul Beardsley. Collision avoidance for multiple agents with joint utility maximization. In IEEE International Conference on Robotics and Automation, 2013. [13] Nate Derbinsky, Jos?e Bento, Veit Elser, and Jonathan S. Yedidia. An improved three-weight messagepassing algorithm. arXiv:1305.1961 [cs.AI], 2013. [14] G. David Forney Jr. Codes on graphs: Normal realizations. IEEE Transactions on Information Theory, 47(2):520?548, 2001. [15] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. arXiv preprint arXiv:1005.0416, 2010. [16] R. Glowinski and A. Marrocco. Sur l?approximation, par e? l?ements finis d?ordre un, et la r?esolution, par p?enalisization-dualit?e, d?une class de probl‘ems de Dirichlet non lin?eare. Revue Franc?aise d?Automatique, Informatique, et Recherche Op?erationelle, 9(2):41?76, 1975. [17] Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. Computers & Mathematics with Applications, 2(1):17?40, 1976. [18] Hugh Everett III. Generalized lagrange multiplier method for solving problems of optimum allocation of resources. Operations Research, 11(3):399?417, 1963. [19] Magnus R. Hestenes. Multiplier and gradient methods. Journal of Optimization Theory and Applications, 4(5):303?320, 1969. [20] Magnus R. Hestenes. Multiplier and gradient methods. In L.A. Zadeh et al., editor, Computing Methods in Optimization Problems 2. Academic Press, New York, 1969. [21] M.J.D. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, Optimization. Academic Press, London, 1969. [22] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning, 3(1):1?122, 2011.

9