SPARTA: Fast global planning of collision-avoiding robot trajectories

Charles J. M. Mathy[1], **Felix Gonda**[1], **Dan Schmidt**[1], **Nate Derbinsky**[2], **Alexander A. Alemi**[1], **José Bento**[3], **Francesco M. Delle Fave**[1], **and Jonathan S. Yedidia**[1]

[1]Disney Research Boston, Cambridge, MA
{charles.mathy,felix.e.gonda}@gmail.com,
{dan.schmidt,alex.alemi,francesco.dellefave,yedidia}@disneyresearch.com
[2]Wentworth Institute of Technology, Boston, MA
derbinskyn@wit.edu
[3]Boston College, Chestnut Hill, MA
jose.bento@bc.edu

Abstract

We present an algorithm for obtaining collision-avoiding robot trajectories we call SPARTA (SPline-based ADMM for Robot Trajectory Avoidance). We break the problem of solving for collision-avoiding trajectories of robots into tractable subproblems, using the framework of a recently developed generalization of ADMM, the Three Weight Algorithm. The generated paths are smooth, include physical constraints of the robots, and the convergence speed is such that it becomes feasible for real-time applications. We physically implemented our algorithm using mobile robots and showcase the results.

The problem of finding smooth collision-avoiding paths for a set of moving objects arises in many different settings, such as robot trajectory planning, swarm robotics, and self-driving vehicles. There are two broad popular approaches to the problem. One involves searching for feasible paths by first performing a guided random search using e.g. Rapidly-Exploring Random Trees (RRTs) [1, 2], and later smoothing the paths via optimization around the feasible path. The other approach formulates the task as a constrained non-linear optimization problem directly, typically linearizing the constraints and then using some variant of Sequential Quadratic Programming (SQP) [3, 4, 5, 6, 7]. Such direct optimization-based approaches are fast; however, they can have difficulties when the starting guess for the algorithm involves large violations of the constraints. Robust methods are required to be both capable of dealing with initial conditions that are deep within the infeasible region, and able to refine a feasible solution to obtain smooth final results.

A promising approach, which satisfies the requirements just described, involves using a generalization of ADMM [8] called the Three Weight Algorithm (TWA) [9], which can be elegantly understood as a message passing algorithm. TWA considers the optimization problem where the function f that one is minimizing is the sum of hopefully simple functions which are denoted as factors :

$$\underset{z_1,\ldots,z_N}{\text{minimize}}\, f(z_1,\ldots,z_N) = \sum_\alpha f_\alpha(\{z_k\}_\alpha) \quad \text{subject to} \quad g_j(z) \le 0$$

where $\{z_k\}_\alpha$ are the variables that the factor f_α depends on, and g_j are the constraints. Ideally f_i depends on a small subset of variables. The constraints are treated as factors: add a function $f_\alpha = \infty \times I(-g_j(z))$ that is infinity if $g_j(z) > 0$, 0 otherwise ($I(x) = 0$ if $x > 0$, 1 otherwise).

Unlike approaches that incrementally move downhill, ADMM-based approaches combine solutions to subproblems with a guarantee that whenever the algorithm converges, all hard constraints are satisfied. ADMM will typically converge faster when the problem can be broken down into relatively large subproblems that can each be solved efficiently. TWA builds the analog of active set methods into ADMM: if a hard constraint is inactive, meaning that the messages it receives are in the feasibile set, it sends a zero-weight message, so as not to slow down convergence. This change takes ADMM's performance for circle packing from embarrassingly slow to state of the art [9]. For ADMM the lore is that it is slow for small problems, but eventually wins out on large dimensional problems because it takes advantage of distributed resources. In the problem at hand, TWA provides an algorithm that is fast for both small and large problems.

TWA for robot trajectory optimization has been shown [10, 11] to work well; however, there were limitations to the work, mainly that the paths were piecewise linear, so the acceleration was potentially infinite at every time step. Splines are standard in robot trajectory planning, as they provide a smooth representation and a relatively low-dimensional parameter space to optimize. The question we address in this work is whether we can build a TWA-based approach to robot optimization with splines as variables, which we call SPARTA (SPline-Based ADMM for Robot Trajectory Avoidance). Previous work relied on the fact that the subproblems that one has to solve in ADMM were exactly solvable. In this work we show this restriction can be lifted, because the subproblems are amenable to standard optimization algorithms. In this work we use Sequential Least Squares Quadratic Programming (SLSQP [12]) as a subroutine and find that it solves our subproblems very quickly. We compare our approach to SLSQP applied to the entire problem and find that SPARTA delivers on its promise of solving our optimization problem in a modular and robust way.

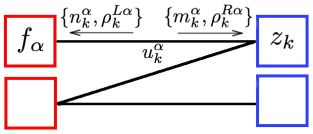

f_α
$\{n_k^\alpha, \rho_k^{L\alpha}\}$
$\{m_k^\alpha, \rho_k^{R\alpha}\}$
u_k^α
z_k

Figure 1: Pictorial representation of the general TWA.

TWA builds a distributed approach to this problem. Borrowing a pictorial representation from the message-passing literature is very convenient (see fig. 1). We put the factors f_α on the left and the variables z_k on the right. A line is drawn connecting each factor to each variable it depends on. For each factor, we create local copies x_k^α of the variables it depends on, associated u_k^α variables, weight variables to the left (right) $\rho_k^{L\alpha}$ ($\rho_k^{R\alpha}$), and messages to the left (right) $n_k^\alpha(m_k^\alpha)$. The algorithm is described in [9] and summarized in appendix A. At each iteration, each factor receives its copy of the variables with associated weights. It solves a subproblem independently of the other factors, which involves minimizing its f_α plus a quadratic penalty for differing from the messages it received:

$$x_k^\alpha \leftarrow \arg\min_{x_k^\alpha} \left[f_\alpha(\{x_k^\alpha\}) + \sum_k \frac{\rho_k^{L\alpha}}{2} (x_k^\alpha - n_k^\alpha)^2 \right],$$

then sends out its solution with weights to the right. Once the factors are done, the weighted messages on the right are averaged; the average is sent to the left with an appropriate weight.

In regular ADMM all the weights would be equal to some constant ρ. In TWA, however, one can change the weights. If a factor enforcing a hard constraint, which we call a hard factor, is receiving messages in the feasible set with respect to its constraints, it sends out zero weight messages to the right. This stops an inactive constraint from slowing down the algorithm. The weight to the left is simply $\rho_k^{L\alpha} = \max_\alpha \rho_k^{R\alpha}$: the variable is as informative as the most informative message associated with it[1]. The factors enforcing a soft objective send out constant weight messages, with weight given by a fixed value ρ. Since we have such factors, for SPARTA $\rho_k^{L\alpha} = \rho$ always.

The problem we tackle is as follows: consider N spherical robots in a D-dimensional space, each with its own radius R_a, moving during the time interval $[0, T]$. The position of robot a at time t is $\vec{r}^{\,a}(t)$, and its velocity is $\dot{\vec{r}}^{\,a}(t)$. We must specify a desired path for each robot, $r^{Da}(t)$.

We seek to minimize the quadratic departure from the velocity of the desired path:

$$f(\vec{r}_1, \vec{r}_2, \ldots) = \gamma \sum_{a=1}^{N} \int_0^T (\dot{\vec{r}}^{\,a}(t) - \dot{\vec{r}}^{\,Da}(t))^2 dt \tag{1}$$

[1]In practice, for a zero weight we send a small weight, e.g. 10^{-4}. TWA also allows for the sending of infinite weight messages, which we don't use in this work.

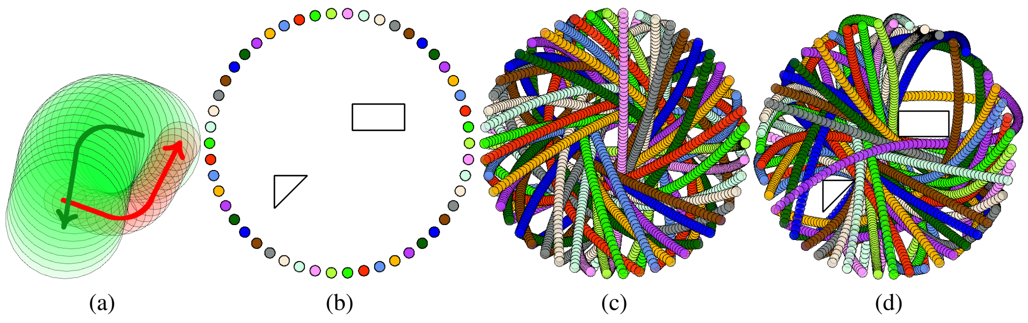


Figure 2: (a) Exact solution of the optimization problem for two robots of different radii with fixed initial and final positions and no physical constraints, which gives intuition regarding the solution for more robots. The robots first move in a straight line until they graze each other, then they rotate around each other, and finally go in a straight line again. The solution is therefore continuous with discontinuous second derivative at the two time points. See appendix B for a precise description of the solution. (b) Initial condition for all tests in this work. N robots start on a circle of radius R_s ($N = 50$ in this case), with radii $r = 2\pi/(4N)$. We set $v_{max} = 10R_s/\text{s}$, $a_{max} = 10R_s/\text{s}^2$, $\dot{\theta}_{max} = 10\,\text{rad/s}$. Each path is made up of 3 splines, and they must end at the antipodal point on the circle. (c) Solution found by SPARTA for 50 robots with no line obstacles in 1.1 seconds, using 4 cores. We are showing 50 sampled points from each continuous path. (d) Solution found by SPARTA with the 7 line obstacles shown in (b), in 4.4 seconds using 4 cores. (c) and (d) illustrate that SPARTA finds smooth and tight solutions.

where $\vec{r}_a$ is the entire path of robot a from $t = 0$ to $t = T$. We require the robots not to collide during the motion, the first derivatives of their paths to be continuous, their velocity to stay below v_{max}, their acceleration to stay below a_{max}, and their angular velocity to stay below $\dot{\theta}_{max}$. All results in this work are in two dimensions $D = 2$; however, the algorithm works for any dimension[2], and in particular we have tested it in three dimensions. γ is a constant prefactor, which technically does not affect the true global minimum[3]. We fix the initial position of the robots, and can optionally choose to additionally set the final position. Note that SPARTA can handle other objectives, especially since as we describe below we can use SLSQP as a subroutine, which can handle nonlinear objectives.

We add a desired path for the case where the final point is not set, as otherwise the robots would simply stand still. When fixing initial and final positions we set $\vec{r}^{Da}(t) = 0$ which reproduces the objective function used in [10, 11][4].

We now represent the paths as a set of second order splines and require the first derivative to be continuous. We have n_t time points t_i where $i = 1, \ldots, n_t$, and N robots labelled by $a = 1, \ldots, N$. Each robot's path is made up of $n_t - 1$ splines $\vec{P}_i^{\vec{Y}a}(t)$ for $t \in [t_i, t_{i+1}]$, with $i = 1, \ldots, n_t - 1$.

$$\vec{P}_i^{\vec{Y}a}(t) = \vec{Y}_i^{Pa}(t_{i+1} - t)\delta t_i^{-1} + \vec{Y}_i^{Ca}(t - t_i)\delta t_i^{-1} + \vec{Y}_{i+1}^{Pa}(t - t_{i+1})(t - t_i)\delta t_i^{-2}$$

where $\delta t_i = t_{i+1} - t_i$, and $\vec{Y} = \vec{z}$ if the spline represents the spline on the right, meaning the actual value output by SPARTA, or $\vec{Y} = \vec{x}_\alpha$ if the spline is the local copy of the variable associated with a factor on the left. We omit the index α which labels the factor because the relevant factor is clear from context. We define spline variables $\vec{Y}_i^{Sa} = \{\vec{Y}_i^{Pa}, \vec{Y}_i^{Ca}, \vec{Y}_{i+1}^{Pa}\}$ which are a vector of all the variables that define the spline of robot a at time slice i. The time points t_i can be set arbitrarily; in this work we choose equally spaced time points $t_i = (i-1) * T/(n_t - 1)$.

Each factor α depends on a set of spline variables $\{\vec{z}_i^{Sa}\}$. It carries its own set of Lagrange multiplier variables $\{\vec{u}_{\alpha i}^{Sa}\}$, message variables $\{\vec{u}_{\alpha i}^{Sa}\}$ and output variables $\{\vec{x}_{\alpha i}^{Sa}\} = \{\vec{x}_{\alpha i}^{Pa}, \vec{x}_{\alpha i}^{Ca}, \vec{x}_{\alpha i+1}^{Pa}\}$. We

[2]Angular velocity is not defined in $D > 2$ so we restrict this factor to $D = 2$.

[3]γ has to be set appropriately for SPARTA to converge: one should find a value that works for $N = 2$ robots, and scale it as $1/N$.

[4]We can equivalently set $\vec{r}^{Da}(t)$ to be a straight line at constant velocity $\vec{v}^{\,a}$ from start point $\vec{S}$ to end point $\vec{E}$: $\dot{\vec{r}}^{Da}(t) = (\vec{E} - \vec{S})/T$. In this case $\int_0^T \dot{\vec{r}}^a(t) \cdot \dot{\vec{r}}^{Da}(t)dt = (\vec{E} - \vec{S})^2/T$, therefore $\int_0^T (\dot{\vec{r}}^a(t) - \dot{\vec{r}}^{Da}(t))^2 dt = \int_0^T \dot{\vec{r}}^a(t)^2 - (\vec{E} - \vec{S})^2/T$ and we can equivalently use $\gamma \sum_a \int_0^T \dot{\vec{r}}^a(t)^2 dt$ as the objective.

will omit the label α because context dictates which factor we are talking about; we merely add it here to emphasize that each factor carries its own local copies of these variables. The variables $\{\vec{z}_i^{Sa}\}$ can be read off as the final solution to the problem.

We now discuss the factors used to solve the problem at hand. For piecewise linear segments the subproblems of each factor were exactly solvable [10, 11]. For SPARTA we did not derive exact solutions but found that using SLSQP for the factors we could not solve exactly worked well. For further mathematical details concerning the factors, see appendix C.

Collision avoidance. For each pair of robots a and b with radii R_a and R_b, and time interval $[t_i, t_{i+1}]$, we require that the splines of these robots in that time interval remain at a minimum distance $R_{ab} = R_a + R_b$ for all $t \in [t_i, t_{i+1}]$. The relevant factor must minimize $\frac{\rho}{2}(\vec{x}_i^{Sa} - \vec{n}_i^{Sa})^2 + \frac{\rho}{2}(\vec{x}_i^{Sb} - \vec{n}_i^{Sb})^2$ over $\vec{x}_i^{Sa}, \vec{x}_i^{Sb}$ subject to $||\vec{P}_i^{\vec{x}a}(t) - \vec{P}_i^{\vec{x}b}(t)|| \geq R_{ab}$ for all $t \in [t_i, t_{i+1}]$. To formulate this semi-infinite program as a problem SLSQP can solve, call $t^*(\vec{x}_i^{Sa}, \vec{x}_i^{Sb})$ the time in $[t_i, t_{i+1}]$ where $||\vec{P}_i^{\vec{x}a}(t) - \vec{P}_i^{\vec{x}b}(t)||$ is minimized, which depends on $\vec{x}_i^{Sa}, \vec{x}_i^{Sb}$. The program is then equivalent to

$$\min_{\vec{x}_i^{Sa}, \vec{x}_i^{Sb}} \frac{\rho}{2}(\vec{x}_i^{Sa} - \vec{n}_i^{Sa})^2 + \frac{\rho}{2}(\vec{x}_i^{Sb} - \vec{n}_i^{Sb})^2 \text{ s.t. } ||\vec{P}_i^{\vec{x}a}(t^*(\vec{x}_i^{Sa}, \vec{x}_i^{Sb})) - \vec{P}_i^{\vec{x}b}(t^*(\vec{x}_i^{Sa}, \vec{x}_i^{Sb}))|| \geq R_{ab}$$

which can be fed to SLSQP[5]. If the splines corresponding to the incoming messages are the solution, i.e. they are non-colliding, then $\rho^R = 0$, otherwise $\rho^R = \rho$. SLSQP is started at $\vec{x}^{Sa} = \vec{n}^{Sa}$, $\vec{x}^{Sb} = \vec{n}^{Sb}$. For 50 robots these factors take up about 30% of the computation time.

Soft objective. The soft objective factors exist for each robot a and time interval $[t_i, t_{i+1}]$. They must minimize $\frac{\rho}{2}(\vec{x}_i^{Sa} - \vec{n}_i^{Sa})^2 + \gamma \int_{t_i}^{t_{i+1}} (\dot{\vec{P}}_i^{\vec{x}a}(t) - \dot{\vec{r}}^{Da}(t))^2 dt$ over $\vec{x}_i^{Sa} = \{\vec{x}_i^{Pa}, \vec{x}_i^{Ca}, \vec{x}_{i+1}^{Pa}\}$.This is a quadratic form whose minimum can be found by solving a linear set of equations, which is done very efficiently with standard methods. If the spline is at $t = 1$ ($t = N - 1$) and the initial position (final position) is set, then that variable doesn't exist in the problem; in this factor it is set to its correct value and there is no associated penalty cost.

Continuous first derivatives. These factors exist for each robot and time point between splines. They minimize $\frac{\rho}{2}(\vec{x}_i^{Sa} - \vec{n}_i^{Sa})^2 + \frac{\rho}{2}(\vec{x}_{i+1}^{Sa} - \vec{n}_{i+1}^{Sa})^2$ subject to $\dot{\vec{P}}_i^{\vec{x}a}(t_{i+1}) = \dot{\vec{P}}_{i+1}^{\vec{x}a}(t_{i+1})$. Each dimension decouples. This can be solved exactly: it can be formulated as a minimization of $\frac{\rho}{2}\sum(\vec{v}_i - \vec{n}_i)^2$ subject to $\sum_i \mu_i \vec{v}_i = 0$ with appropriately defined $\{\mu_1, \ldots, \mu_5\}$ where $\vec{v} = \{\vec{x}_i^{Pa}, \vec{x}_i^{Ca}, \vec{x}_{i+1}^{Pa}, \vec{x}_{i+1}^{Ca}, \vec{x}_{i+2}^{Pa}\}$ (see appendix C). Using a Lagrange multiplier we find $\vec{v}_i = \vec{n}_i - \frac{\sum_j \mu_j \vec{n}_j}{\sum_k \vec{\mu}_k^2}\vec{\mu}_i$. The output weights are $\rho^R = \rho$.

Line obstacles. Consider a line segment going from $\vec{v}_1$ to $\vec{v}_2$. We must minimize $\frac{\rho}{2}(\vec{x}_i^{Sa} - \vec{n}_i^{Sa})^2$ subject to $\forall t \in [t_i, t_{i+1}], \forall \alpha \in [0, 1] : ||\vec{P}_i^a(t) - (\alpha\vec{v}_2 + (1-\alpha)\vec{v}_1)|| \geq R_a$ for robot a at time slice i. We find the point of closest overlap between the spline and line segment and turn this semi-infinite program into a program with one constraint, that of having the point of closest approach between the spline and line segment be at least a distance R_a apart, which we solve with SLSQP.

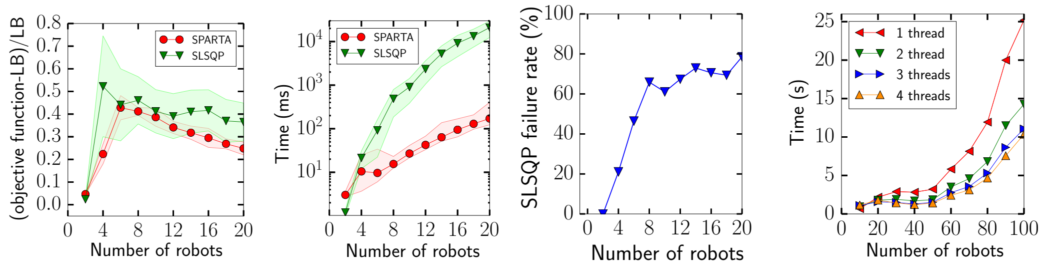
Maximum velocity. The velocity depends linearly on time in a spline. Therefore, to enforce a maximum $\dot{\vec{P}}_i^{\vec{x}a}(t)$ we need only enforce it at the endpoints t_i. For t_i, we must minimize $\frac{\rho}{2}(\vec{x}_i^{Sa} - \vec{n}_i^{Sa})^2$ subject to $||\dot{\vec{P}}_i^{\vec{x}a}(t_i)||^2 \leq v_{max}^2$. If the messages satisfy the constraint, we send out $\vec{x}_i^{Sa} = \vec{n}_i^{Sa}$ and $\rho^R = 0$; otherwise, the constraint becomes active: $||\dot{\vec{P}}_i^{\vec{x}a}(t_i)||^2 = v_{max}^2$ and $\rho^R = \rho$. A quadratic objective with quadratic constraint can be solved exactly.

Maximum acceleration. a_{max} is simple to enforce: since $\ddot{\vec{P}}_i^{\vec{x}a}(t) = 2\vec{x}_i^{Ca}/\delta t_i^2$, we are simply imposing that $||\vec{x}_i^{Ca}|| \leq a_{max}\frac{\delta t_i^2}{2}$. We must minimize $\frac{\rho}{2}(\vec{x}_i^{Ca} - \vec{n}_i^{Ca})^2$ subject to $||\vec{x}_i^{Ca}|| < \lambda$, where $\lambda = a_{max}\frac{\delta t_i^2}{2}$. If the incoming message is feasible, i.e. $||\vec{n}_i^{Ca}|| \leq a_{max}\frac{\delta t_i^2}{2}$, then $\vec{x}_i^{Ca} = \vec{n}_i^{Ca}$ and $\rho^R = 0$ for all variables. Otherwise, $\vec{x}_i^{Ca} = \vec{n}_i^{Ca}\lambda/||\vec{n}_i^{Ca}||$ and $\rho^R = \rho$ for all variables.

Maximum angular velocity. We solve it with SLSQP. The angular velocity $\dot{\theta}$ is only defined in two dimensions, although generalizations exist in higher dimensions. For this factor we restrict ourselves

[5]We use the implementation in the NLopt library [13].

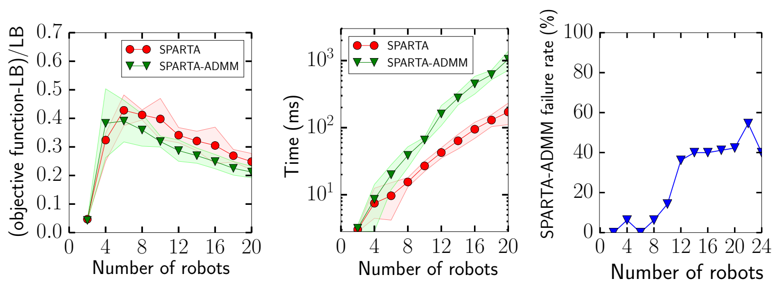
to $D = 2$. Given a path $(x(t), y(t))$, the angular velocity is equal to $\dot{\theta}(t) = \frac{\dot{x}\ddot{y}-\ddot{x}\dot{y}}{\dot{x}^2+\dot{y}^2}$. In this case the x and y coordinates are $(\vec{x}_i^{Pa})_1, (\vec{x}_i^{Pa})_2$ respectively. Once again we formulate this problem as minimizing $\frac{\rho}{2}(\vec{x}_i^{Sa} - \vec{n}_i^{Sa})^2$ subject to the maximum angular velocity factor being less than or equal to $\dot{\theta}_{max}$. Therefore we need to find the $t^*(\vec{x}_i^{Pa})$ in $[t_i, t_{i+1}]$ that maximizes $||\dot{\theta}(t)||$. The numerator $(\dot{\vec{P}}_i^{\vec{x}a}(t))_1(\ddot{\vec{P}}_i^{\vec{x}a}(t))_2 - (\ddot{\vec{P}}_i^{\vec{x}a}(t))_1(\dot{\vec{P}}_i^{\vec{x}a}(t))_2$ is independent of time in this case, and the denominator is $(\dot{\vec{P}}_i^{\vec{x}a}(t))^2$; therefore, we need only minimize $(\dot{\vec{P}}_i^{\vec{x}a}(t))^2$, which is a quadratic function of t, in $[t_i, t_{i+1}]$.



(a) (b) (c) (d)

Figure 3: Comparison of SPARTA to SLSQP applied to the whole problem, with the same conditions as in fig. 2(c). We run each configuration 30 times initializing the paths to straight lines from start to endpoint, adding a random number in $[-0.05, 0.05]$ to each variable. We set the step size to $\beta = 0.3$. In (a) and (b) we plot the mean (line with markers), and the lines one standard deviation away. (a) Objective function, compared to a lower bound LB to the objective function obtained by exactly solving the problem where only robots starting on antipodal points interact (see appendix B). (b) Convergence times. We run the algorithms until all variables do not change by more than 10^{-3}. (c) Failure rate of SLSQP, meaning the fraction of times it does not converge, as a function of the number of robots (SPARTA always converged in these experiments). (d) Convergence time as a function of the number of threads, for a representative run for each N.

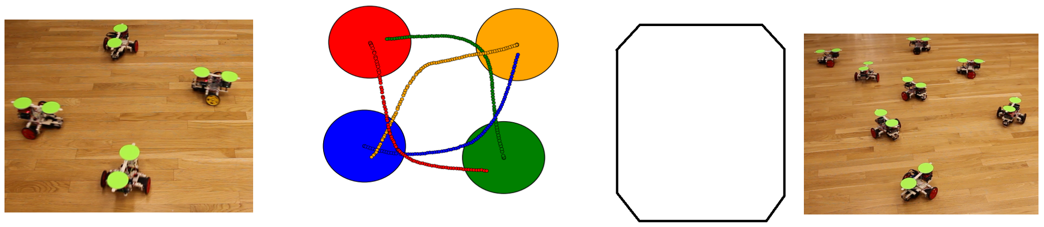
We compare SPARTA to SLSQP in fig. 3, for N robots starting in a circle and having to move to the antipodal point. SPARTA finds lower objective solutions more quickly. As the number of robots N increases, SLSQP increasingly does not find a solution. In most cases, when SLSQP gets stuck the paths have points where some pairs of robots get very close to each other, and it is not able to resolve this. SPARTA always converged for these problems. SPARTA is also trivial to parallelize.



(a) (b) (c)

Figure 4: Comparison of SPARTA to SPARTA-ADMM where all weights are equal to each other. Tests and plot labels are the same as in fig. 3. (a) Objective functions. (b) Convergence times. (c) Failure rate of SPARTA-ADMM.

To see the effects of having non-uniform weights, we compare SPARTA to SPARTA-ADMM where all weights are equal in fig. 4. We find that SPARTA-ADMM finds slightly lower objective function, however it is slower and suffers the same problem as SLSQP of failing to converge with a probability that grows with the number of robots.



(a) (b) (c) (d)

Figure 5: Implementation of SPARTA on mobile robots. The robots are about 5 inches in size, and move at about 15 inches per second. A camera tracks the car positions. Users define a desired velocity and angular velocity, which gives a predicted path given to SPARTA as an initial point and a set of desired trajectories $r^{Da}(t)$. SPARTA then outputs paths satisfying no-collision and physical constraints, which are fed to a controller of the robot motors. (a) Snapshot of 4 mobile robots in the middle of executing a robot exchange. (b) Measured paths of the robots during robot exchange. (c) Virtual bounding area for 8 mobile robots. We add a little line at the edges to prevent SPARTA from producing paths ending in the corner. (d) Snapshot of 8 robots all given a full throttle. Here SPARTA is run without fixed final points, but with $\vec{r}^{Da}(t)$ equal to a linear path starting at the current position and moving straight with the maximum velocity. We ran the robots for 10 minutes and they never collided or left the bounding region.

To test the real-time capacity of SPARTA we built simple two-wheeled mobile robots, with Arduinos [14] controlling servo motors for the wheels and connected to xbees [15] for wireless control, fig. 5(a) and (d). A camera overlooks the robots and tracks the two green circles on each car using clustering, from which one derives position and orientation. Given a position and desired path, we built a simple PID controller that sends controls to the wheels of the robot to follow the path. This controller can follow the path to within about two inches, an error margin we incorporate in the effective robot radius in SPARTA. We can run the system in two modes: setting a desired final trajectory, or responding to user inputs in real-time. The first case is shown in fig. 5(b), where we execute the robot exchange described previously, for 4 robots. At a frequency of 10 Hz we set the measured positions of the robots as the initial positions and the targets as desired final positions. We run SPARTA for 10 ms and feed its output to the PID controller. The cars indeed execute collision avoiding trajectories. Tracking takes about 5 ms and sending controls takes another 5 ms, which means we have about 70 ms of computation time left per iteration.

In the second use case, we have users (possibly AIs) for each robot that send controls to the system, such as desired velocity and angular velocity. Given the measured position and user controls, a predicted path $\vec{r}^{Da}(t)$ is calculated. For example, if the user wants to go straight with velocity v, $\vec{r}^{Da}(t)$ is a straight line from the current robot position at velocity v with the current robot orientation. SPARTA then outputs the collision-avoiding physically realisable paths which are close to $\vec{r}^{Da}(t)$ thanks to the objective function. The paths are given to the PID controller. As an example of this use case, we take 8 robots with "AI" for all robots pressing full throttle: we set $\vec{r}^{Da}(t)$ to be a straight line initially at the measured robot positions and with velocity $v_{max} = 15$ inches/sec. We put a bounding area in SPARTA, see fig. 5(c).Without SPARTA the cars would simply go straight and collide. With SPARTA in the loop, the cars never collide, and execute smooth collision-avoiding trajectories. Tracking takes longer (10 ms) for 8 cars,and PID control also takes about 10 ms, so we can run SPARTA for 20 ms without problems.

In conclusion, we have shown that TWA can be used with splines to quickly compute smooth collision-avoiding robot trajectories. Our preliminary results on mobile robots are promising. The algorithm could potentially be used in 3 dimensions on drones. Since we were able to use SLSQP as a subroutine, this frees us from needing exactly solvable factors, and one could try a similar approach on robots or obstacles with more complicated shapes. To go to more robots (e.g. $N = 1000$), one could try combining SPARTA with a version of TWA with dynamic pruning of factors as was done to pack millions of circles [16]: one would put the robots in an R-tree and each iteration only enable collision-avoiding factors for robots close enough to potentially collide.

References

[1] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[2] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[3] Philip E Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM journal on optimization*, 12(4):979–1006, 2002.

[4] Javier Alonso-Mora, Andreas Breitenmoser, Paul Beardsley, and Roland Siegwart. Reciprocal collision avoidance for multiple car-like robots. In *IEEE International Conference on Robotics and Automation*, pages 360–366, 2012.

[5] 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.

[6] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[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] 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] Nate Derbinsky, José Bento, Veit Elser, and Jonathan S. Yedidia. An improved three-weight message-passing algorithm. *arXiv:1305.1961 [cs.AI]*, 2013.

[10] José Bento, Nate Derbinsky, Javier Alonso-Mora, and Jonathan S. Yedidia. A message-passing algorithm for multi-agent trajectory planning. In *Advances in neural information processing systems*, pages 521–529, 2013.

[11] José Bento, Nate Derbinsky, Charles Mathy, and Jonathan S. Yedidia. Proximal operators for multi-agent path planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[12] Dieter Kraft et al. *A software package for sequential quadratic programming*. DFVLR Obersfaffeuhofen, Germany, 1988.

[13] Steven G. Johnson. The nlopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt.

[14] https://www.arduino.cc/.

[15] http://www.digi.com/lp/xbee.

[16] Nate Derbinsky, José Bento, and Jonathan S. Yedidia. Scalable methods to integrate task knowledge with the three-weight algorithm for hybrid cognitive processing via optimization. *Biologically Inspired Cognitive Architectures*, 8:107–117, 2014.

Appendix for "SPARTA: fast global planning of collision-avoiding robot trajectories"

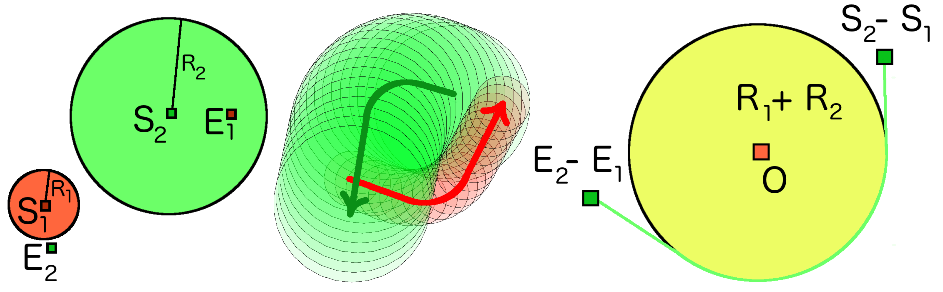
A Summary of TWA algorithm

To initialize: set all x_k^α to their initial guesses, set all n_k^α, m_k^α, u_k^α to zero, and all weights $\rho_k^{R\alpha}, \rho_k^{L\alpha}$ to 0.

Loop until convergence:

1. Update the values for the variables on the left by finding the proximal optimum for each factor:
$$x_k^\alpha \leftarrow \arg\min_{x_k^\alpha} \left[f_\alpha(\{x_k^\alpha\}) + \sum_k \frac{\rho_k^{L\alpha}}{2} (x_k^\alpha - n_k^\alpha)^2 \right]$$
2. Compute new values for the right-going weights $\rho_k^{R\alpha}$ depending on the logic of the minimizers.
3. Compute the new right-going messages: $m_k^\alpha \leftarrow x_k^\alpha + u_k^\alpha$
4. Compute the new weighted average values of the variables on the right: $z_k \leftarrow \frac{\sum_\alpha \rho_k^{R\alpha} m_k^\alpha}{\sum_\alpha \rho_k^{R\alpha}}$
5. Compute the left-going weights: $\rho_k^{L\alpha} \leftarrow \max_{k'} \rho_{k'}^{R\alpha}$
6. Update the disagreement variables u on each edge: $u_k^\alpha \leftarrow u_k^\alpha + \beta\,(x_k^\alpha - z_k^\alpha)$
7. Compute the new left-going messages: $n_k^\alpha \leftarrow z_k^\alpha - u_k^\alpha$

B Exact solution of two body problem



(a) (b) (c)

Figure B.1: Two robot problem. (a) Robot 1, of radius R_1, must start at $\vec{S}_1$ and end at $\vec{E}_1$. Robot 2, or radius R_2, starts at $\vec{S}_2$ and ends at $\vec{S}_1$. The robots must not collide at any time, and must minimize $\frac{\gamma}{2}(\int_0^T (\dot{\vec{r}}_1(t) + \dot{\vec{r}}_2(t))^2 dt)$, where $\vec{r}_i(t)$ is the position of robot a at time t. (b) Subsampling of the exact solution. (c) The problem in the reference frame of robot 1.

We take two robots, 1 and 2, of radii R_1 and R_2. Start positions for robots 1 and 2 are $\vec{S}_1$ and $\vec{S}_2$, and the end positions are $\vec{E}_1$ and $\vec{E}_2$. We can exactly solve the problem of minimizing

$$\frac{\gamma}{2}\int_0^T (\dot{\vec{r}}_1(t) + \dot{\vec{r}}_2(t))^2 dt, \tag{2}$$

where $\vec{r}_i(t)$ is the position of robot a at time t, and $||\vec{r}_1(t) - \vec{r}_2(t)|| \geq R_1 + R_2 \forall t \in [0, T]$. We separate out center of mass (CM) and relative coordinates (Rel): $f = f_{CM} + f_{Rel}$ with

$$f_{CM} = \frac{\gamma}{2}\int_0^T (\dot{\vec{r}}_1(t) + \dot{\vec{r}}_2(t))^2 dt \tag{3}$$

and

$$f_{Rel} = \frac{\gamma}{2}\int_0^T (\dot{\vec{r}}_1(t) - \dot{\vec{r}}_2(t))^2 dt. \tag{4}$$

We minimize the first term by having the CM move in a straight line

$$\vec{r}_1(t) + \vec{r}_2(t) = \vec{S}_1 + \vec{S}_2 + \frac{t}{T}(\vec{E}_1 + \vec{E}_2 - \vec{S}_1 - \vec{S}_2), \tag{5}$$

which contributes

$$f_{CM} = \frac{1}{2T}(\vec{E}_1 + \vec{E}_2 - \vec{S}_1 - \vec{S}_2)^2. \tag{6}$$

For $\vec{r}_2(t) - \vec{r}_1(t)$ see fig. B.1(c) drawn in the frame of robot 1. Robot 2 starts at $\vec{S}_2 - \vec{S}_1$ and ends at $\vec{E}_2 - \vec{E}_1$ w.r.t. robot 1, and must be at least a distance $R = R_1 + R_2$ away from the origin O. The fastest path first goes straight, hits the circle of radius R tangentially, moves along the circle's perimeter until a straight path is available to $\vec{E}_2 - \vec{E}_1$, and then takes that straight path. The first straight line has length

$$L_1 = \sqrt{(\vec{S}_2 - \vec{S}_1)^2 - R^2}, \tag{7}$$

the circular segment has length

$$L_2 = R\left(\arccos\Big(\frac{(\vec{S}_2 - \vec{S}_1)\cdot(\vec{E}_2 - \vec{E}_1)}{||\vec{E}_2 - \vec{E}_1||||\vec{S}_2 - \vec{S}_1||}\Big) - \arccos\Big(\frac{R}{||\vec{S}_2 - \vec{S}_1||}\Big) - \arccos\Big(\frac{R}{||\vec{E}_2 - \vec{E}_1||}\Big)\right) \tag{8}$$

and the second straight line has length

$$L_3 = \sqrt{(\vec{E}_2 - \vec{E}_1)^2 - R^2}. \tag{9}$$

Moving at constant velocity, the relative coordinates contribute

$$f_{\text{Rel}} = \frac{1}{2T}(L_1 + L_2 + L_3)^2. \tag{10}$$

To obtain the lower bound LB used in fig. 3, we add the objective function just derived for all pairs of robots on antipodal points of the circle exchanging places. Because this is the minimum of an objective function that has positive terms removed (interaction terms between circles not on antipodal points), it is a lower bound on any finite value of the full objective function.

C Mathematical details behind the factors

Spline overlap factor

We use translational invariance to reduce this to a $3D$-dimensional problem. Define CM and relative coordinates :

$$\vec{Y}^{Sa,b}_{CM,i} = (\vec{Y}^{Sa}_i + \vec{Y}^{Sb}_i)/2 \tag{11}$$

$$\vec{Y}^{Sa,b}_{Rel,i} = \vec{Y}^{Sa}_i - \vec{Y}^{Sb}_i \tag{12}$$

where $\vec{Y} = \vec{x}$ or $\vec{n}$. Remember that $\vec{Y}^{Sa}_i = \{\vec{Y}^{Pa}_i, \vec{Y}^{Ca}_i, \vec{Y}^{Pa}_{i+1}\}$, so we define $\vec{Y}^{Pa}_i = \vec{Y}^{Pa}_i - \vec{Y}^{Pb}_i$ and $\vec{Y}^{Ca}_i = \vec{Y}^{Ca}_i - \vec{Y}^{Cb}_i$.

We rewrite the objective function in terms of CM and relative coordinates:

$$\frac{\rho}{2}(\vec{x}^{Sa}_i - \vec{n}^{Sa}_i)^2 + \frac{\rho}{2}(\vec{x}^{Sb}_i - \vec{n}^{Sb}_i)^2 = \rho(\vec{x}^{Sa,b}_{CM,i} - \vec{n}^{Sa,b}_{CM,i})^2 + \frac{\rho}{4}(\vec{x}^{Sa,b}_{Rel,i} - \vec{n}^{Sa,b}_{Rel,i})^2. \tag{13}$$

The constraint that $||P^{\vec{x}a}_i(t) - P^{\vec{x}b}_i(t)|| \geq R_a + R_b \forall t \in [t_i, t_{i+1}]$ depends on $\vec{x}^{Sa,b}_{Rel,i}$:

$$P^{\vec{x}a}_i(t) - P^{\vec{x}b}_i(t) = \left(\vec{x}^{Pa,b}_{Rel,i}(t_{i+1} - t^*) + \vec{x}^{Pa,b}_{Rel,i+1}(t^* - t_i) + \vec{x}^{Ca,b}_{Rel,i}(t^* - t_i)(t^* - t_{i+1})\delta t_i^{-1}\right)\delta t_i^{-1} \tag{14}$$

the solution will satisfy $\vec{x}^{Sa,b}_{CM,i} = \vec{n}^{Sa,b}_{CM,i}$ and we have SLSQP solve:

$$\begin{aligned} \underset{\vec{x}^{Sa,b}_{Rel,i}}{\text{minimize}} \quad & \frac{\rho}{4}(\vec{x}^{Sa,b}_{Rel,i} - \vec{n}^{Sa,b}_{Rel,i})^2 \\ \text{subject to} \quad & ||P^{\vec{x}a}_i\left(t^*(\vec{x}^{Sa,b}_{Rel,i})\right) - P^{\vec{x}b}_i\left(t^*(\vec{x}^{Sa,b}_{Rel,i})\right)|| \geq R_a + R_b \end{aligned}$$

where $t^*(\vec{x}^{Sa,b}_{Rel,i})$ is the time in $[t_i, t_{i+1}]$ at which $||P^{\vec{x}a}_i(t) - P^{\vec{x}b}_i(t)||$ is minimized.

Derivative continuity factor

For all robots a, for all time steps except the last one $i < N$, we require that the final solution satisfy $\dot{\vec{P}}^{\vec{x}a}_i(t_{i+1}) = \dot{\vec{P}}^{\vec{x}a}_{i+1}(t_{i+1})$. Thus we add factors that solve the following problem:

$$\underset{\vec{x}^{Sa}_i, \vec{x}^{Sa}_{i+1}}{\text{minimize}} \quad \frac{\rho}{2}(\vec{x}^{Sa}_i - \vec{n}^{Sa}_i)^2 + \frac{\rho}{2}(\vec{x}^{Sa}_{i+1} - \vec{n}^{Sa}_{i+1})^2 \quad \text{subject to} \quad \dot{\vec{P}}^{\vec{x}a}_i(t_{i+1}) = \dot{\vec{P}}^{\vec{x}a}_{i+1}(t_{i+1})$$

This can be solved exactly. Each dimension D is independent, so we have D 5-dimensional problems of the form:

$$\underset{\vec{v}}{\text{minimize}} \quad \frac{\rho}{2}\sum_j(\vec{v}_j - \vec{n}_j)^2 \quad \text{subject to} \quad \sum_j \mu_j \vec{v}_j = 0;$$

where

$$\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4, \vec{v}_5\} = \{\vec{x}^{Pa}_i, \vec{x}^{Ca}_i, \vec{x}^{Pa}_{i+1}, \vec{x}^{Ca}_{i+1}, \vec{x}^{Pa}_{i+2}\}. \tag{15}$$

and

$$\{\mu_1, \mu_2, \mu_3, \mu_4, \mu_5\} = \{-\delta t_i^{-1}, \delta t_i^{-1}, \delta t_i^{-1} + \delta t_{i+1}^{-1}, \delta t_{i+1}^{-1}, -\delta t_{i+1}^{-1}\}, \tag{16}$$

where $\delta t_i = t_{i+1} - t_i$, $\delta t_{i+1} = t_{i+2} - t_{i+1}$. Note that each $\vec{v}_j$ is a vector, while each μ_j is a scalar. Using Lagrange multipliers, the solution is found to be

$$\vec{v}_j = \vec{n}_j - \frac{\mu_j}{\sum_k \mu_k^2}\sum_{k'} \mu_{k'}\vec{n}_{k'}. \tag{17}$$

Soft objective factor

The optimization problem for the factor associated with the soft objective is:

$$\underset{\vec{x}^{Sa}_i}{\text{minimize}} \quad \frac{\rho}{2}(\vec{x}^{Sa}_i - \vec{n}^{Sa}_i)^2 + \gamma\int_{t_i}^{t_{i+1}} (\dot{\vec{P}}^{\vec{x}a}_i(t) - \dot{\vec{r}}^{Da}(t))^2 dt$$

There are two versions of this problem: one has no constraints on the beginning and endpoints, while the other does. For the splines of the first time interval we fix $\vec{P}^{\vec{x}a}_1$ to be the robot starting point, and for the last time interval $\vec{P}^{\vec{x}a}_N$ is set to the endpoint, if it is set.

The function to minimize is a quadratic function of $\vec{x}^{Pa}_i$, $\vec{x}^{Ca}_i$ and $\vec{x}^{Pa}_{i+1}$, which can be minimized quickly by solving a set of linear equations. Each dimension is independent, so we have D 3-dimensional linear problems. Calling $\vec{v}_j = \{(\vec{x}^{Pa}_i)_j, (\vec{x}^{Ca}_i)_j, (\vec{x}^{Pa}_{i+1})_j\}$ where j indexes a dimension, the objective can be written as a sum of quadratic forms $\frac{1}{2}\sum_{jkk'} A_{jkk'}(\vec{v}_j)_k(\vec{v}_j)_{k'} + \frac{\rho}{2}((\vec{v}_j)_k - (\vec{n}_j)_k)^2$, where we define $\vec{n}_j = \{(\vec{n}^{Pa}_i)_j, (\vec{n}^{Ca}_i)_j, (\vec{n}^{Pa}_{i+1})_j\}$. For each j independently we solve for $\vec{v}_j$ by taking the derivative and solving $(\sum_{kk'} A_{jkk'} + \rho\delta_{kk'})(\vec{v}_j)_k = \rho(\vec{n}_j)_k$. Since $A_{jkk'} + \rho\delta_{kk'}$ is constant it can be LU decomposed once, thus making this linear solve very fast. If starting points or end points are set, those variables are frozen and we can still use the same technique with the appropriate lower dimensional $A_{jkk'}$.

Line segment obstacle factor

Consider a line segment going from $\vec{v}_1$ to $\vec{v}_2$. One has to first check if the spline is overlapping with the line segment, in which case any overlap point is obviously the closest. If it is, we first translate the spline so that it no longer overlaps with the line segment; otherwise SLSQP would not be able to move out of overlap, since for any small motion of the spline the minimum distance will still be 0 and the linearization of the constraint which is inherent in SLSQP would be 0. If the spline and line segment are not overlapping we find the time at which they are closest to each other and ask that the distance at that time be treater or equal than R_a.

If the spline and line segment are not overlapping, we need to check several cases: the closest point on the spline could be internal or an endpoint, and the same goes for the line segment. Still, all the computations are straightforward.

Maximum velocity factor

As discussed in the main text, this problem has a quadratic function with a quadratic constraint. It is of the form

$$\underset{\vec{x}}{\text{minimize}} \quad \frac{\rho}{2}(\vec{x}-\vec{n})^2 \quad \text{s.t.} \quad (\vec{\mu}_i\vec{x}_i)^2 = C^2$$

where we assume the constraint is active, and C is a constant. By introducing a Lagrange multiplier λ, we have to minimize $\frac{\rho}{2}(\vec{x}-\vec{n})^2 + \lambda((\vec{\mu}_i\vec{x}_i)^2 - C^2)$, which gives

$$\vec{x} = \vec{n} - 2\lambda\rho^{-1}(\vec{\mu}\cdot\vec{x})\vec{\mu}. \tag{18}$$

Taking the inner product with μ on both sides and rearranging:

$$\vec{\mu}\cdot\vec{x}(1+2\lambda\rho^{-1}\vec{\mu}^2) = \vec{\mu}\cdot\vec{n} \tag{19}$$

we see that $\vec{\mu}\cdot\vec{x}$ is proportional to $\vec{\mu}\cdot\vec{n}$, which implies since the constraint is active that $\vec{\mu}\cdot\vec{x} = (C/||\vec{\mu}\cdot\vec{n}||)\vec{\mu}\cdot\vec{n}$. From this and eq. 19 we get $(1+2\lambda\rho^{-1}\vec{\mu}^2) = ||\vec{\mu}\cdot\vec{n}||/C$, which gives us λ and therefore the solution, using eq. 18.