Real-time Optimized Rendezvous on Nonholonomic Resource-Constrained Robots

Sven Gowal and Alcherio Martinoli

Abstract In this work, we consider a group of differential-wheeled robots endowed with noisy relative positioning capabilities. We develop a decentralized approach based on a receding horizon controller to generate, in real-time, trajectories that guarantee the convergence of our robots to a common location (i.e. rendezvous). Our receding horizon controller is tailored around two numerical optimization methods: the hybrid-state A* and trust-region algorithms. To validate both methods and test their robustness to computational delays, we perform exhaustive experiments on a team of four real mobile robots equipped with relative positioning hardware.

1 Introduction

Since the 1960s, *consensus* problems have puzzled the minds of many researchers in various fields, ranging from computer science to information aggregation [23]. The term consensus describes the problem of reaching an agreement amongst different agents on a certain quantity or state. These agents can share information about their state either by means of communication or observations. In a network of robots, solving the consensus problem on the position of each agents refers to the task of controlling them as to reach a common *rendezvous* point. The ability to meet or to rendezvous has indeed many practical applications such as formation control [11], flocking [7], attitude alignment [25] or cooperative aerial surveillance [1]. Additionally, although this paper specifically addresses the rendezvous of differential-wheeled robots, its general concept may be applied the wider range of consensus problems.

Sven Gowal and Alcherio Martinoli
Distributed Intelligent Systems and Algorithms Laboratory, School of Architecture, Civil and Environmental Engineering, École Polytechnique Fédérale de Lausanne e-mail: svenadrian.gowal@epfl.ch, alcherio.martinoli@epfl.ch

1.1 Related Work

Solving the rendezvous with nonholonomic agents is complex, and proving the convergence property can be difficult. Many works employ *feedback linearization* to design relaxed control laws that recreate the holonomic properties [16, 26]; others create algorithms that are very specific to their application needs [6, 5]; but all of them rely on deterministic assumptions both in terms of actuation and sensing. Our previous work [13] incorporates insights from the *probabilistic consensus* problem [9] to guarantee that differential-wheeled robots can rendezvous under noisy measurements. However, this approach and all prior approaches to solve the rendezvous problem on nonholonomic mobile robots rely heavily on strict time-invariant controllers that yield poor trajectories without consideration to neither actuation constraints nor the energy spent.

On another front, a great body of literature starting with Meschler [19] in 1963 focuses on the optimization of the rendezvous maneuver, and although efforts to decentralize the optimization approach using communication between agents have been made [18], many works remain centralized [20, 4] and thus need global knowledge of the system. We can also observe that most work, including [18], use a predefined cost function and leave no design choices to the user. To tackle the problem of decentralization with an arbitrary user-defined metric, we rely on a receding horizon controller (RHC) [12]. This RHC needs to run in real-time on our platform, the Khepera III robot [24] (shown on Fig. 1(a)) equipped with an Intel XScale PXA-270 running at 624MHz without floating-point unit.

In particular, we use two distinct optimization strategies (within our RHC) that provide real-time capabilities to resource-constrained robots: *(i)* the hybrid-state A* algorithm [8] – an optimization strategy based on the A* search algorithm that generates quickly feasible trajectories for a wide range of cost functions, *(ii)* a subspace conjugate gradient trust-region method [2, 3] – a numerical optimization method that takes advantage of the differential flatness property of our robots. We compare both strategies in terms of performance and computational requirements. The purpose of this work is then twofold: first, to experimentally verify the convergence of our mobile robot team using our decentralized approach; second, to compare its efficiency with that of a centralized equivalent. We note that, to date, no contribution has addressed the generation of real-time, optimal rendezvous maneuvers on mobile robots performing noisy positioning observations — neither from a theoretical nor from an experimental point of view. In this work, we focus on the experimental aspect (the theory is covered in more depth in another concurrent publication [14]).

1.2 Problem Statement

We have a team of N differential-wheeled robots $\mathbf{R}_1, \ldots, \mathbf{R}_N$ driven by the kinematic equations:

$$\begin{cases} \dot{x}_i &= u_i \cos\theta_i \\ \dot{y}_i &= u_i \sin\theta_i \\ \dot{\theta}_i &= \omega_i \end{cases}, \tag{1}$$

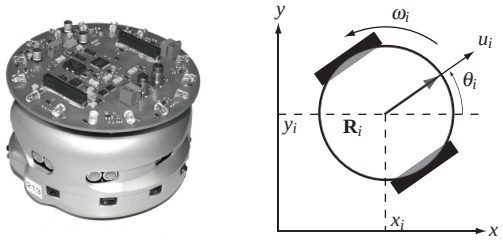


Fig. 1 (a) A Khepera III robot with a range and bearing module attached. (b) The kinematic model of a differential-wheeled robot $\mathbf{R}_i$.

where $\mathbf{u}_i = [u_i, \omega_i]^\mathsf{T}$ is the vector of control inputs, with u_i the linear translational speed and ω_i the rotational speed, and the vector $\mathbf{x}_i = [x_i, y_i, \theta_i]^\mathsf{T}$ defines the absolute pose or state of the robot $\mathbf{R}_i$, as shown on Fig. 1(b).

A robot $\mathbf{R}_i$ has a set of neighbors $\mathcal{N}_i$ containing all robots $\mathbf{R}_j$ such that it can measure the range e_{ij} and bearing α_{ij} to them. Its measurements are affected by noise such that each observation $z_{ij}(t)$ of $\mathbf{R}_j$ at time t is defined by

$$z_{ij}(t) = \begin{bmatrix} \tilde{e}_{ij}(t) \\ \tilde{\alpha}_{ij}(t) \end{bmatrix} = \begin{bmatrix} e_{ij}(t) \\ \alpha_{ij}(t) \end{bmatrix} + \varepsilon_z, \tag{2}$$

where ε_z is a random noise vector.

Our goal will be to drive all robots to the same meeting point. For each robot $\mathbf{R}_i$, this *rendezvous maneuver* should be performed *optimally* in *real-time* under a local user-defined metric $\mathscr{J}_i(\mathbf{u}_i)$ which should only depend on values directly measurable (either through sensors or communication) or calculable by each individual robot $\mathbf{R}_i$. Without loss of generality, throughout this paper, we will use the Bolza form $\mathscr{J}_i(\cdot) = \int L_i(\cdot)\mathrm{d}t + V_i(\cdot)$ where $L_i(\cdot)$ is a cost rate and $V_i(\cdot)$ is a terminal cost (also called *salvage term*).

2 Technical Approach

In this section, we explain in brevity how RHC can guarantee rendezvous with the addition of optimization constraints, and how to perform each optimization cycle, on-board, in real-time, using, on one hand, the hybrid-state A* algorithm and, on the other, a subspace conjugate gradient trust-region method. Additionally, we introduce a closed-loop control that follows the resulting RHC trajectories.

2.1 Decentralized Receding Horizon Control

To solve our optimization problem (i.e., minimizing $\mathscr{J}_i(\cdot)$ whilst guaranteeing the rendezvous, as seen in Section 1.2), we will rely on RHC. RHC carries many names such as model predictive control (MPC) or real-time optimization (RTO). It is an advanced method, widely used in industry, that has the ability to use the available

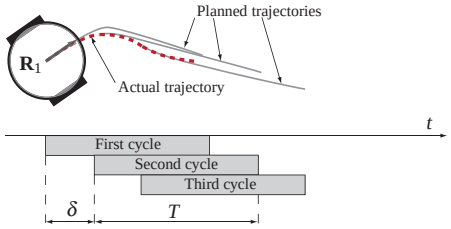


Fig. 2 Receding horizon trajectories: $\mathbf{R}_1$ plans an initial trajectory for the next T seconds and executes that trajectory *blindly* for the first δ seconds; at that point, $\mathbf{R}_1$ plans a new trajectory (and so on).

information on the system at hand to control it optimally under a user-defined cost. Although its requirements in terms of computing power are high [21], it has found many successful applications, in particular when the underlying system to control has slow dynamics (i.e., in the order of minutes or seconds). The recent advances in computing power have in part alleviated this issue, but RHC reaches its limits when the underlying system is nonlinear, changing fast and has to run on a simple mobile platform.

RHC is an optimization-based control that uses online, optimal trajectory generation. The general idea is to plan a feasible and sub-optimal trajectory over a finite time T horizon and control the system (i.e. the robots) to follow this trajectory over a sampling time δ ($0 < \delta \leq T$). After δ seconds, a new trajectory is recomputed from the current position until time $\delta + T$ and this trajectory is again followed until time 2δ. This cycle is repeated until the goal is reached. We will denote such a receding horizon control with the symbol $\mathscr{RH}(T,\delta)$. This process is schematized in Fig. 2, where robot $\mathbf{R}_1$ plans during three cycles three trajectories that are tracked sequentially.

Theorem 1. *Given a* **symmetric** *and* **connected** *group of N differential-wheeled robots $\mathbf{R}_1, \ldots, \mathbf{R}_N$, the decentralized receding horizon control $\mathscr{RH}(T_i,\delta_i)$, with $T_i > 0$ and $0 < \delta_i \leq T_i$, that solves the following optimization problem on each robot $\mathbf{R}_i$ at time τ:*

$$\textit{minimize} \quad \mathscr{J}_i(\mathbf{u}_i) = \int_{\tau}^{\tau+T} L_i(t,\mathbf{x}_i,\hat{\mathbf{x}}_i,\mathbf{u}_i)\,dt + V_i(\tau+T,\mathbf{x}_i,\hat{\mathbf{x}}_i,\mathbf{u}_i) \tag{3}$$

$$\textit{subject to} \quad \textit{Eq. 1}, \mathbf{u}_i \in \mathscr{U}_i, \mathbf{x}_i \in \mathscr{X}_i \tag{4}$$

$$\textit{such that} \quad \exists k_{ij} = k_{ji} > 0 \textit{ satisfying } u_i = \sum_{\mathbf{R}_j \in \mathscr{N}_i} k_{ij}\hat{x}_{ij} \tag{5}$$

$$\exists t \geq \tau \textit{ satisfying } \omega_i(t) \neq 0, \tag{6}$$

where $\mathscr{X}_i$ and $\mathscr{U}_i$ are user-defined admissible sets, drives the group almost surely *to a common rendezvous point if $\hat{\mathbf{x}}_i(t) = \{\hat{\mathbf{x}}_{ij}(t) = [\hat{x}_{ij}(t), \hat{y}_{ij}(t)]^{\mathsf{T}} | \mathbf{R}_j \in \mathscr{N}_i\}$ and the estimation $\hat{x}_{ij}(t)$ of $x_{ij}(t) = e_{ij}\cos\alpha_{ij}$ is unbiased in the time interval $t \in [\tau, \tau+\delta_i]$.*

Proof. The proof is omitted for conciseness, but its complete derivation is available in [14]. □

Remark 1. As we will see in Section 3, the constraints (5) and (6) can in practice be ignored when an adequate salvage term $V_i(\cdot)$ is used. In particular, it is sufficient to penalize inter-robot distances that increase.

Remark 2. Theorem 1 assumes for each robot $\mathbf{R}_i$ the presence of a prediction function $\hat{\mathbf{x}}_i(t)$ capable of estimating the position of neighboring robots. As explained in [14], this function can be implemented using an extended Kalman filter based on the observations made through the relative positioning hardware.

2.2 Cost Function

To ease our discussion on the algorithmic details, we describe first the cost function used in our experiments. Given a continuous trajectory for next T seconds, we discretize it by splitting it into N linear segments of $\Delta t = T/N$ seconds each, thus generating a sequence of $N+1$ vertices $\mathbf{p}_i \in \mathbb{R}^2$ with $i \in \{0,\dots,N\}$. Additionally, we assume that there are N_o obstacles denoted $\mathscr{O}_j$ with $j \in \{1,\dots,N_o\}$. Each obstacle has a position $\mathbf{o}_i^{(j)} \in \mathbb{R}^2$ at time $i\Delta t$ and an associated uncertainty $R_i^{(j)} \in \mathbb{R}^{2\times 2}$. We denote by $\Delta\mathbf{p}_i = \mathbf{p}_i - \mathbf{p}_{i-1}$ the displacement vector at a vertex and by $\mathbf{p}_f$ the final position that the trajectory aims to reach. Our cost function is then:

$$f(\mathbf{p}_{0\dots N}) = \overbrace{w_s \sum_{i=1}^{N-1} (\Delta\mathbf{p}_{i+1} - \Delta\mathbf{p}_i)^{\mathsf{T}} (\Delta\mathbf{p}_{i+1} - \Delta\mathbf{p}_i)}^{f_1} + \underbrace{w_e \sum_{i=1}^{N} \|\Delta\mathbf{p}_i\|_2^2}_{f_2} + \underbrace{w_o \sum_{i=0}^{N} \sum_{j=1}^{N_o} \Phi(\mathbf{p}_i; \mathbf{o}_i^{(j)}, R_i^{(j)})}_{f_3} + \underbrace{w_f \|\mathbf{p}_f - \mathbf{p}_N\|_2^2}_{f_4}, \quad (7)$$

where w_s, w_e, w_f, w_o are positive weights and $\Phi(x;\mu,\Sigma)$ is the multi-variate normal probability density function with mean μ and covariance Σ.

The first term f_1 of the cost function forces the trajectory to be smooth: the forward acceleration and rotational speed should be small. The second term f_2 penalizes fast motion and ensures that minimal energy to spend in actuation. The third term f_3 guides the trajectory away from obstacles and corresponds roughly to a scaled probability of hitting any of them. The fourth term f_4 steers the trajectory towards a goal position by penalizing an excessive distance to it. The first three terms correspond to the sum of all cost rates over the trajectory, whereas the last term is the salvage term. To ensure the rendezvous in practice, it is enough to set the goal position $\mathbf{p}_f$ to the estimated center of mass of all neighboring robots at time T,

$$\mathbf{p}_f = \frac{1}{|\mathscr{N}_i|} \sum_{\mathbf{R}_j \in \mathscr{N}_i} \hat{\mathbf{x}}_{ij}(T). \quad (8)$$

2.3 Optimization Strategies

The sampling time δ is related to the computational time required by the optimization of Eq. (3) [21]. Hence, it is important that this optimization takes as little time as possible (to guarantee, in practice, the unbiasedness of the estimator of $\hat{x}_{ij}$). In this section, we provide two alternatives capable of efficient real-time optimization.

2.3.1 Hybrid-state A*

The first alternative, explained in [8], is a continuous optimization method derived from a discrete heuristic search method, the A* search algorithm. The general idea is to discretize into cells the three-dimensional search space $\langle x_i, y_i, \theta_i \rangle$ representing the robot's state. Whereas A* explores the center of those discrete cells and generates paths that may not be feasible with respect to the kinematic constraints of Eq. (1), hybrid-state A* associates with each cell a five-dimensional continuous state $\langle x_i, y_i, \theta_i, u_i, \omega_i \rangle$. Hence the transitions from a cell to the next may change according to the stored continuous state. To determine those transitions, we simply discretize the action that the robot can take during the next Δt seconds and perform an Euler integration of the kinematic equations. In particular, we allow the robot to either keep, increase, or decrease its forward or rotational speed by a constant increment Δu or $\Delta \omega$ respectively. It is clear that hybrid-state A* is not guaranteed to find the minimal-cost solution because of the discretization of controls and time, as well as the pruning of all but one continuous-state branches that enter a cell. Finally, to use hybrid-state A* with RHC, we stop the search when the number of cells explored on the current branch reaches the number of points $N+1$ required by the trajectory. We note that although hybrid-state A* is memory hungry, it will always generates feasible trajectories and can be easily modified to include dynamics and additional constraints with little overhead in terms of computational time.

For completeness, we show through Algorithm 1 the complete routine, where $g(c)$ represents the real cost of the current path from the starting cell to cell c, $h(s, \mathbf{p})$ is the heuristic cost to reach position $\mathbf{p}$ from a continuous state s and $s(c)$ is the continuous state associated with cell c. Note that the cost $g(c)$ can be computed by adding up the first three terms of our cost function ($f_1 + f_2 + f_3$) until cell c on the current branch and the heuristic $h(c, \mathbf{p})$ is the last term of this same cost function. In the context of our experimental test-bed, we observe that this algorithm can easily make use of fixed-point arithmetic as all variable ranges are known a priori. Combined with a proper implementation (i.e., efficient priority queue), we obtain a procedure on our platform, the Khepera III robot. In the experiments of Section 3, the cell discretization is done by a $64 \times 64 \times 52$ grid on an area of $2\text{m} \times 2\text{m} \times 360^\circ$ centered around the robot. The speed increments Δu and $\Delta \omega$ are set to 0.125m/s and 1.5rad/s respectively and Δt is set to 0.1s. These values are selected to reach the best compromise between optimality and computational/memory requirements (the overall memory usage is 14.8 MB which can easily fit on-board the Khepera III).

Algorithm 1 Hybrid-state-A*$(x_i, y_i, \theta_i, u_i, \omega_i, \mathbf{p}_f)$

```
closedSet ← ∅
start ← getCell(x_i, y_i, θ_i)
s(start) ← ⟨x_i, y_i, θ_i, u_i, ω_i⟩
g(start) ← 0
openSet ← {start}
while openSet ≠ ∅ do
    cell ← argmin_{c∈openSet} g(c) + h(s(c), p_f)
    if isGoal(cell) or depth(cell) = ⌈T/Δt⌉ then
        return generatePathTo(cell)
    end if
    openSet ← openSet \ cell
    closedSet ← closedSet ∪ cell
    ⟨x, y, θ, u, ω⟩ ← s(cell)
    for all [u', ω'] ∈ {[u, ω] ± [Δu, Δω]} do
        ⟨x', y', θ'⟩ ← eulerIntegration([u', ω'], ⟨x, y, θ⟩, Δt)
        dest ← getCell(x', y', θ')
        if dest = cell then
            continue
        end if
        newCost = g(cell)+ edgeCost(cell, ⟨x', y', θ', u', ω'⟩)
        if newCost + h(⟨x', y', θ', u', ω'⟩, p_f) > g(dest) + h(s(dest), p_f) then
            continue
        end if
        openSet ← openSet ∪ dest
        closedSet ← closedSet \ dest
        s(dest) ← ⟨x', y', θ', u', ω'⟩
        g(dest) ← newCost
    end for
    return trajectory impossible
end while
```

2.3.2 Subspace Conjugate Gradient Trust-Region

This second alternative is an efficient non-convex optimization method that uses a preconditioned conjugate gradient to define a two-dimensional subspace on which a trust-region method is applied.

Let us consider a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, which we want to minimize. We currently have an estimate x of the solution, which we wish to improve. The basic idea behind the trust-region approach is to approximate the function f with a function q reflecting the behavior of f in a neighborhood Ω around the point x. This neighborhood is the trust-region. Hence the problem is to find a step s that minimizes q: $\min_s\{q(s)|s \in \Omega\}$. If the vector $x+s$ is a better estimate of the solution (i.e., $f(x+s) < f(x)$), x is set to $x+s$; otherwise, it is unchanged and the trust-region is shrunk. In practice, the approximate function q is defined by the first two term of the Taylor expansion of f around x and the trust-region is often circular. The trust-region step then becomes

$$\min_s \left\{ \frac{1}{2} s^\mathsf{T} H s + g^\mathsf{T} s \quad | \quad \|s\|_2 \leq \Delta \right\}, \tag{9}$$

where g and H are the gradient and Hessian of f respectively, and Δ is positive. Good algorithms to solve Eq. (9) based on the eigenvalues of H exist [22]. However, they become inefficient when H becomes large. Hence, a good heuristic is to reduce the original problem into a two-dimensional subspace spanned by an approximate Newton direction (given in our case by a preconditioned conjugate gradient method) and the gradient direction.

The method of conjugate gradient (CG) [15] is an effective way to iteratively solve large-scale linear equations such as $Hv = -g$ (note that, here v is the Newton direction) without calculating the inverse of H. Using a preconditioned variant (PCG) allows for faster convergence by altering the original problem to $M^{-1}Hv = -M^{-1}g$, where M is called the preconditioner. Finally, the only costly operation that PCG needs to perform is the multiplication of H with a vector. Thus, PCG is very efficient when H is sparse. If the number of points is small (i.e., $N < 50$), PCG can be replaced with Newton's method for greater efficiency; but Newton's method will need more memory as the inverse of the Hessian needs to be stored.

For a fast implementation, it is important that the function f be twice differentiable and that both an analytical gradient and Hessian can be computed. In our case, for the gradient, we have

$$\begin{aligned}\frac{\partial f}{\partial \mathbf{p}_i} &= w_s(2\Delta\mathbf{p}_{i+2} - 6\Delta\mathbf{p}_{i+1} + 6\Delta\mathbf{p}_i - 2\Delta\mathbf{p}_{i-1}) \\ &\quad - w_e(2\Delta\mathbf{p}_{i+1} - 2\Delta\mathbf{p}_i) \\ &\quad + w_o \sum_{j=1}^{N_o} (R_i^{(j)})^{-1} \Phi(\mathbf{p}_i; \mathbf{o}_i^{(j)}, R_i^{(j)})(\mathbf{o}_i^{(j)} - \mathbf{p}_i) \\ &\quad + w_f \mathbf{1}_{i=N}(\mathbf{p}_f - \mathbf{p}_N), \end{aligned} \tag{10}$$

where $\mathbf{1}_A$ is the indicator function of A. The Hessian is then simply the sum of two sparse matrices: a constant banded matrix H_1 representing the first, second and fourth term of f and a block diagonal matrix H_2 composed of 2×2 blocks $B_0, \ldots, B_N$ (if we interleave the coordinates of every point $\mathbf{p}_i$), with

$$B_i = \frac{\partial^2 f_3}{\partial \mathbf{p}_i^2} = \left(R^{-1}(\mathbf{p}_i - \mathbf{o}_i^{(j)}) \cdot (\mathbf{p}_i - \mathbf{o}_i^{(j)})^\mathsf{T} - I\right) R^{-1} \Phi(\mathbf{p}_i; \mathbf{o}_i^{(j)}, R), \tag{11}$$

where R means $R_i^{(j)}$. If no collisions are possible (i.e., $H_2 = \mathbf{0}$), it is beneficial to use Newton's method to minimize f (if memory allows). Indeed the function f becomes quadratic and Newton's method converges in one iteration. Also, as H_1 is constant, its inverse only needs to be calculated once.

Although, less hungry than hybrid-state A* in terms of memory, the subspace conjugate gradient trust-region method (which we denote from hereon as PCG-TR) may not generate a feasible trajectory. However, our specific choice of the cost function f, which penalizes non-smooth trajectories, mitigates this issue. Additionally, it can be shown that differential-wheeled robots are differentially flat and thus can follow a *sufficiently* smooth trajectory. Algorithm 2 shows the complete routine.

Lines (5-11) compute the two-dimensional subspace while lines (12) and (14-20) perform a trust-region step.

Algorithm 2 PCG-TrustRegion($x_i, y_i, \theta_i, u_i, \omega_i, \mathbf{p}_f$)

1: $x_{\text{old}} \leftarrow$ generateInitialTrajectory($x_i, y_i, \theta_i, u_i, \omega_i, \mathbf{p}_f$)
2: $f_{\text{old}} \leftarrow f(x_{\text{old}})$
3: $\Delta \leftarrow$ InitialTrustRegionRadius()
4: **repeat**
5: $\quad \langle g, H \rangle \leftarrow$ computeGradientAndHessian(x_{old})
6: $\quad v_1 \leftarrow$ preconditionedConjugateGradient(g, H)
7: $\quad v_1 \leftarrow v_1 / \|v_1\|_2$
8: $\quad v_2 \leftarrow g - v_1(v_1^{\mathsf{T}} g)$
9: $\quad v_2 \leftarrow v_2 / \|v_2\|_2$
10: $\quad g' \leftarrow [v_1 v_2]^{\mathsf{T}} g$
11: $\quad H' \leftarrow [v_1 v_2]^{\mathsf{T}} H [v_1 v_2]$
12: $\quad s' \leftarrow \text{argmin}_s \left\{ \frac{1}{2} s^{\mathsf{T}} H' s + g'^{\mathsf{T}} s \quad | \quad \|s\|_2 \leq \Delta \right\}$
13: $\quad s \leftarrow [v_1 v_2] s'$
14: $\quad x \leftarrow x_{\text{old}} + s$
15: $\quad f \leftarrow f(x)$
16: $\quad$ **if** $f < f_{\text{old}}$ **then**
17: $\quad\quad f_{\text{old}} \leftarrow f$
18: $\quad\quad x_{\text{old}} \leftarrow x$
19: $\quad$ **end if**
20: $\quad \Delta \leftarrow$ updateTrustRegionRadius(Δ)
21: **until** convergence
22: **return** x_{old}

2.4 Computational Delays

In RHC, the optimized trajectory is followed during a time δ during which no feedback from the environment is observed. After δ seconds, feedback from the environment is incorporated to re-optimize the trajectory. In practice, the amount of time δ dedicated to follow the trajectory is not fixed. Indeed, one often prefers to optimize the trajectory as fast as possible and use the result as early as possible. The sampling time δ then directly relates to the computation time needed to optimize the new trajectory.

It is clear that while the optimization takes place, the robot continues to move according to the old trajectory which may result in a mismatch between the optimized position and the current position at the time when the optimization completes. Hence, the robot $\mathbf{R}_i$ needs to reacquire (and track) the optimized trajectory. To do so, it needs to know its current position with respect to the desired new position, hereafter denoted by the coordinates (x_d, y_d). This can easily be achieved by integrating the open-loop controls or, for more precision, by using odometry measurements (in our case given by wheel encoders which are deployed on most differential-wheeled robots). Fig. 3 shows a robot with its desired trajectory.

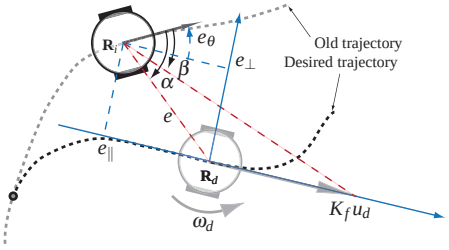


Fig. 3 Schema of the quantities used by the control law in Eq. 12 that enable $\mathbf{R}_i$ to reach a trajectory given by the virtual robot $\mathbf{R}_d$ (desired trajectory) after having followed the old trajectory for *too long*.

According to the desired trajectory, robot $\mathbf{R}_i$ should be located at the position indicated by the virtual robot reference $\mathbf{R}_d$. $\mathbf{R}_i$ is able to calculate the range e and the bearing α to $\mathbf{R}_d$. It can identify the orientation $-e_\theta$ (with respect to itself), the forward motion u_d and rotational motion ω_d of $\mathbf{R}_d$. Note that β is the bearing to the point located at a distance $K_f u_d$ in front of $\mathbf{R}_d$. We propose the following control law when $\mathbf{R}_d$ moves forward:

$$\begin{cases} u_i &= K_u e \cos\alpha + u_d \\ \omega_i &= K_\omega e \sin\alpha + K_b \beta + \omega_d \end{cases} \tag{12}$$

with K_u, K_ω, K_b and K_f all positive constants. An equivalent control law can be found when $\mathbf{R}_d$ moves backward. Although omitted here for conciseness, it can be shown that this control law is stable and converges to the desired trajectory.

This strategy bears resemblance to the third strategy proposed by Milam et al. [21] to account for computation delays, with the exception that, instead of blindly applying the optimized control inputs (open-loop), we compute corrected control inputs based on the optimized trajectory using a tracking layer (closed-loop).

3 Experiments

Experiments are conducted using Khepera III robots in a $3\times3\text{m}^2$ arena. This robot has a diameter of 12cm, making it appropriate for multi-robot indoor experiments. As shown on Fig. 1(a), we equip each robot with a range and bearing module allowing for inter-robot positioning. A measurement campaign performed in [13] showed that the observation noise ε_z is normally distributed with a covariance $\Sigma \approx [0.0221 \;\; -0.0011; \;\; -0.0011 \;\; 0.0196]$. The ground truth position and orientation of each robot is monitored using an overhead camera with SwisTrack [17], an open-source tracking software. The experiments are designed to analyze four different controllers:

Reactive This controller, on top of which we add an obstacle avoidance control as explained in [10], was presented in [13]. It is a standard reactive controller, which does not optimize trajectories, nor predicts the future positions of neighboring robots or obstacles. However, it guarantees the rendezvous mathematically and was shown to perform under noisy perception particularly well.

Hybrid-state A* This controller implements Algorithm 1.

PCG-TR This controller implements Algorithm 2.

Centralized PCG-TR This controller implements Algorithm 2, but optimizes simultaneously the trajectories of all robots. In particular, the new cost function is the sum of the individual costs of all robots: $\sum_{\mathbf{R}_i} f(\mathbf{p}^{(i)}_{0\ldots N})$, where $\mathbf{p}^{(i)}_{0\ldots N}$ are vertices of the trajectory of robot $\mathbf{R}_i$. We note that even if inter-dependencies between robots avoiding each other arise, the new Hessian matrix stays sparse and the optimization stays efficient. While all of the three above controllers use exclusively on-board resources, this centralized optimization is run off-board on a desktop computer and uses the information of the tracking system as input. It serves as an upper-bound on performance.

All controllers are tuned such that the average speed of the robots is about 15cm/s (i.e., f_2 is about the same across all controllers). Four scenarios are selected to provide a wide-range of situations upon which the different controllers can be tested:

Scenario (a) Four robots are randomly placed in the arena and form a complete graph (all robots are neighbors). Their task is to perform the rendezvous.

Scenario (b) Two robots $\mathbf{R}_1$ and $\mathbf{R}_2$ are placed 2 meters apart, facing each other. Each robot has to reach the initial location of the other robot. These locations are represented by 2 additional motion-less robots $\mathbf{R}_3$ and $\mathbf{R}_4$ (whose relative positions are artificially fed to the robots). Also, $\mathbf{R}_1$ and $\mathbf{R}_2$ have to avoid each other. Formally, we have $\mathscr{O}_j = 1$ and $\mathbf{o}^{(1)}_k = \hat{\mathbf{x}}_{12}(k\Delta t)$ for $\mathbf{R}_1$ and $\mathbf{o}^{(1)}_k = \hat{\mathbf{x}}_{21}(k\Delta t)$ for $\mathbf{R}_2$. This scenario not only tests collision avoidance, but also how each robot is able to rendezvous with a fixed goal position.

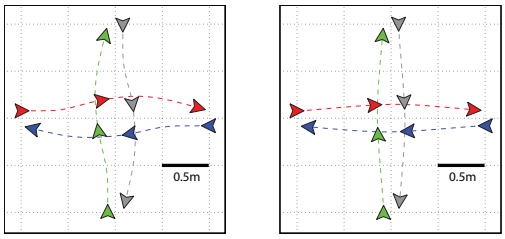
Scenario (c) Like the previous scenario but with four robots. This is a complex crossing and is an effective test-bed for analyzing the ability to optimize the trajectories quickly. Examples of trajectories obtained by the robots are shown in Fig. 4.

Scenario (d) This scenario involves two robots having to rendezvous and two other robots disturbing this rendezvous maneuver by crossing the arena.

Finally, across all scenarios and controllers, we perform two sets of experiments:

Set I The first set tests the performance in terms of smoothness f_1 of the resulting ground-truth trajectories of all controllers. The smoothness is a valid performance indicator, since the average forward speed was the same across controllers and scenarios, *all runs were collision-free* ($f_3 \ll f_1$) and *all rendezvous maneuvers succeeded* ($f_4 \ll f_1$). In this set, we did 10 runs per scenario per controller, resulting in a total of 160 experimental runs.

Set II The second set aims to test the degradation of performance in **Scenario (a)** when the computational time of the controllers is increased by a fixed additional delay of 0, 200 and 400ms. Additionally, we test our approach to mitigate computational delays (Section 2.4, closed-loop) against the third strategy proposed in [21] (open-loop). The performance is measured both through the smoothness and the convergence speed towards the rendezvous point. We did 5 runs per delay per mitigation strategy per controller, resulting in a total of 90 experimental runs.



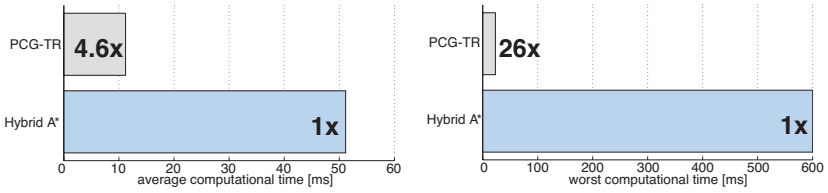
(a) (b)

Fig. 4 Runs performed on **Scenario (c)** with (a) the decentralized PCG-TR variant and (b) its centralized equivalent.

4 Results

Before diving into the core results, we show that both algorithms are capable of running in real-time on board of our miniature robots. The average computational time over all robots and scenarios of **Set I** is shown in Fig. 5(a). We observe that PCG-TR is 4.6× faster than hybrid-state A*, averaging 11.2ms per optimization cycle against 51.15ms (both algorithms run faster than 19.5Hz in average). The difference in performance is even more stagering when looking at the worst-case performance, in Fig. 5(b), yielding 23.12ms for PCG-TR and maxing out at 600ms for hybrid-state A* (600ms is a hard computational time limit imposed on both optimization strategies to keep real-time capabilities). Indeed, hybrid-state A* may have to explore many cells when the heuristic does not match the current situation. However, hybrid-state A* guarantees that the optimized trajectory is feasible. Note that when run on a single core on a standard desktop computer (Intel® Core™ i7 2.93GHz), PCG-TR averages 0.32ms and hybrid-state A* 1.3ms.

Fig. 6 shows in the form of boxplots, the distribution of the smoothness of each trajectory of each robot in **Set I**. Low smoothness values indicate smooth trajectories, whereas high values indicate rough trajectories with many speed changes. Incidentally, a low value means a better minimization of the cost function. We observe that PCG-TR and hybrid-state A* perform equally well and provide an improvement of about 300% over the standard state-of-the-art reactive controller. Both algorithms show their capability to minimize the objective function across the wide range of proposed scenarios. Their performance with respect to the centralized PCG-TR also suggests that our decentralized approach is competitive (about 74% worse). Remember that the centralized algorithm uses ground-truth positioning information



(a) (b)

Fig. 5 (a) Average and (b) worst computational time across all scenarios for PCG-TR and hybrid-state A* on the real robots on **Set I**.

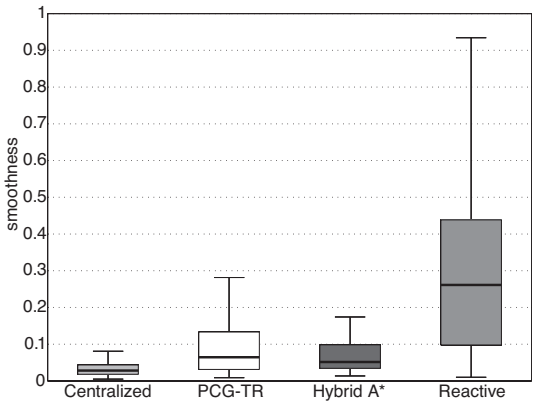


Fig. 6 Boxplot of the resulting smoothness of each controller across all scenarios and all robots. Smaller values indicate that trajectories are more smooth (i.e., smaller is better). We observe that the centralized controller performs best as expected, PCG-TR and hybrid-state A* perform slightly worst but are much better than the reactive controller.

and requires the synchronization of our robots, whereas the decentralized variants only require local observations and no explicit communication. One can qualitatively compare trajectories obtained with PCG-TR and its centralized equivalent in Fig. 4.

The results of the second set of experiments are shown in Fig. 7. They concern only the pure rendezvous scenario, **Scenario (a)**, with four robots. On the first row, we show the smoothness degradation for both the closed-loop and open-loop control as we increase the computational delay. On one hand, the hybrid-state A* seems to perform slightly worse and the resulting smoothness degrades more rapidly. This simple scenario may indeed exacerbate the computational time required by hybrid-state A*. On the other hand, the benefit of our closed-loop control is to keep the smoothness almost constant even when computational delay reaches up to $35\times$ the original computation time (for PCG-TR). The same conclusion can be made when looking at the second row of Fig. 7. This row shows the average convergence speed of the inter-robot distances (the higher, the faster the robots converge to the same rendezvous point). We observe that the performance of the open-loop control worsen as the one of the closed-loop control stays constant. Overall, the closed-loop control provides an efficient alternative when the optimization process is slow. However, it may only be implementable on robots equipped with accurate proprioceptive sensors. Hence, when one can only use the open-loop control, it is important to provide fast optimization methods such as PCG-TR or hybrid-state A*.

5 Conclusion

In this work, we proposed a RHC capable of performing the rendezvous on a team of differential-wheeled robots equipped with noisy relative positioning hardware. This RHC is tested with two complementary optimization procedures and showed, in both cases, its ability to rendezvous on a wide range of experimental scenarios. The two optimization procedures are the hybrid-state A* algorithm and a subspace

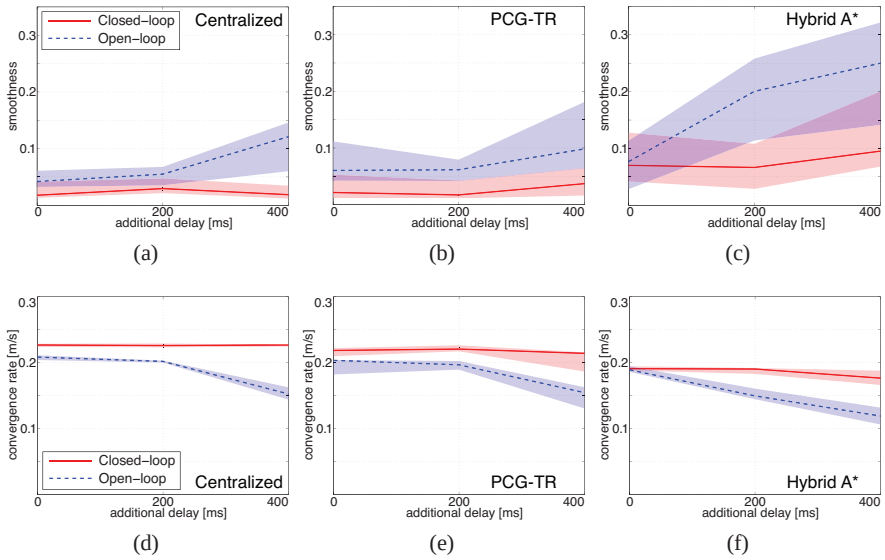


Fig. 7 Plots of the smoothness (top row, smaller is better) and convergence rate (bottom row, higher is better) for the centralized (left column), PCG-TR (middle column) and hybrid-state A* (right column) controllers for different additional computational delays and different tracking strategies (open versus closed-loop). The solid lines represent the median while the shaded region show the 25th and 75th percentiles. The performance worsen as the computational delay increases although the closed-loop controller (see Section 2.4) mitigates the added delays and performs better.

trust-region method based on PCG. Both algorithms were successfully deployed on miniature robots, the Khepera III, and are able to run in real-time on-board. Finally, we developed a closed-loop control that follows the optimized trajectories and showed superior performance than its open-loop variant. This work provides an exhaustive analysis of two fast, numerical, optimal approaches to nonholonomic rendezvous for differential-wheeled robots.

References

1. Beard, R., McLain, T., Nelson, D., Kingston, D., Johanson, D.: Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs. Proceedings of the IEEE **94**(7), 1306–1324 (2006)
2. Branch, M., Coleman, T., Li, Y.: A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. In: SIAM Journal on Scientific Computing, vol. 21, pp. 1–23 (1999)
3. Byrd, R., Schnabel, R., Shultz, G.: Approximate solution of the trust region problem by minimization over two-dimensional subspaces. In: Mathematical Programming, vol. 40, pp. 247–263 (1988)
4. Chyung, D.H.: Time optimal rendezvous of three linear systems. Journal of Optimization Theory and Applications **12**, 242–247 (1973)

5. Dimarogonas, D., Johansson, K.: Further results on the stability of distance-based multi-robot formations. In: American Control Conference, pp. 2972–2977 (2009)
6. Dimarogonas, D., Kyriakopoulos, K.: On the rendezvous problem for multiple nonholonomic agents. IEEE Transactions on Automatic Control **52**(5), 916–922 (2007)
7. Dimarogonas, D.V., Loizou, S.G., Kyriakopoulos, K.J., Zavlanos, M.M.: A feedback stabilization and collision avoidance scheme for multiple independent non-point agents. Automatica **42**(2), 229–243 (2006)
8. Dolgov, D., Thrun, S., Montemerlo, M., Diebel, J.: Path planning for autonomous driving in unknown environments. In: Experimental Robotics, *Springer Tracts in Advanced Robotics*, vol. 54, pp. 55–64. Springer Berlin / Heidelberg (2009)
9. Eisenberg, E., Gale, D.: Consensus of subjective probabilities: The pari-mutuel method. The Annals of Mathematical Statistics **30**(1), 165–168 (1959)
10. Falconi, R., Sabattini, L., Secchi, C., Fantuzzi, C., Melchiorri, C.: A graph–based collision–free distributed formation control strategy. In: 18th IFAC World Congress (2011). DOI 10.3182/20110828-6-IT-1002.02450
11. Fax, J., Murray, R.: Information flow and cooperative control of vehicle formations. IEEE Transactions on Automatic Control **49**(9), 1465–1476 (2004)
12. Findeisen, F., Allgöwer, F.: An introduction to nonlinear model predictive control. Benelux Meeting on Systems and Control pp. 119–141 (2002)
13. Gowal, S., Martinoli, A.: Bayesian rendezvous for distributed robotic systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2765–2771 (2011)
14. Gowal, S., Martinoli, A.: Real-time optimization of trajectories that guarantee the rendezvous of mobile robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2012, to appear)
15. Hestenes, M.R., Stiefel, E.: Methods of Conjugate Gradients for Solving Linear Systems. Journal of Research of the National Bureau of Standards **49**, 409–436 (1952)
16. Lawton, J., Beard, R., Young, B.: A decentralized approach to formation maneuvers. IEEE Transactions on Robotics and Automation **19**(6), 933–941 (2003)
17. Lochmatter, T., Roduit, P., Cianci, C., Correll, N., Jacot, J., Martinoli, A.: Swistrack - a flexible open source tracking software for multi-agent systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4004–4010 (2008)
18. McLain, T., Chandler, P., Rasmussen, S., Pachter, M.: Cooperative control of UAV rendezvous. In: American Control Conference, 2001. Proceedings of the 2001, vol. 3, pp. 2309–2314 (2001)
19. Meschler, P.: Time-optimal rendezvous strategies. IEEE Transactions on Automatic Control **8**(4), 279–283 (1963)
20. Miele, A., Weeks, M., Ciarcià, M.: Optimal trajectories for spacecraft rendezvous. Journal of Optimization Theory and Applications **132**, 353–376 (2007)
21. Milam, M., Franz, R., Hauser, J., Murray, R.: Receding horizon control of vectored thrust flight experiment. IEE Proceedings on Control Theory and Applications **152**(3), 340–348 (2005)
22. Mor, J., Sorensen, D.: Computing a trust region step. In: SIAM Journal on Scientific and Statistical Computing, vol. 3, pp. 553–572 (1983)
23. Olfati-Saber, R., Fax, J., Murray, R.: Consensus and cooperation in networked multi-agent systems. Proceedings of the IEEE **95**(1), 215–233 (2007)
24. Prorok, A., Arfire, A., Bahr, A., Farserotu, J., Martinoli, A.: Indoor navigation research with the Khepera III mobile robot: An experimental baseline with a case-study on ultra-wideband positioning. In: 2010 International Conference on Indoor Positioning and Indoor Navigation (2010). DOI 10.1109/IPIN.2010.5647880
25. Ren, W.: Distributed attitude consensus among multiple networked spacecraft. In: American Control Conference (2006). DOI 10.1109/ACC.2006.1656474
26. Ren, W., Beard, R.: Distributed Consensus in Multi-vehicle Cooperative Control: Theory and Applications. Springer Publishing Company, Incorporated (2007)