2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

Combining High Level Scheduling and Low Level Control to Manage Fleets of Mobile Robots

Sabino Francesco Roselli[1] and Ze Zhang[1] and Knut Åkesson [1]

Abstract— **The deployment of mobile robots for material handling in industrial environments requires scalable coordination of large fleets in dynamic settings. This paper presents a two-layer framework that combines high-level scheduling with low-level control. Tasks are assigned and scheduled using the compositional algorithm ComSat, which generates time-parameterized routes for each robot. These schedules are then used by a distributed Model Predictive Control (MPC) system in real time to compute local reference trajectories, accounting for static and dynamic obstacles. The approach ensures safe, collision-free operation and supports rapid rescheduling in response to disruptions such as robot failures or environmental changes. We evaluate the method in simulated 2D environments with varying road capacities and traffic conditions, demonstrating high task completion rates and robust behavior even under congestion. The modular structure of the framework allows for computational tractability and flexibility, making it suitable for deployment in complex, real-world industrial scenarios.**

I. INTRODUCTION

Mobile robots have become an integral part of material handling in industrial environments, supporting the transport of goods across various areas of a plant [1]. In their simplest form, such systems consist of a small number of robots executing predefined tasks along fixed paths. These setups are relatively easy to manage but offer limited flexibility. More advanced systems involve larger fleets and allow for more flexibility; robots are able to adjust their trajectory in order to avoid obstacles, both static and dynamic. Moreover, they may change their path altogether if changes in the environment make the current one obsolete or inefficient [2]. Such advanced systems offer great advantages and can be deployed in complex scenarios. At the same time, their development poses a few different challenges.

The fleet as a whole needs to be scheduled to execute a list of tasks quickly, or within given time windows, depending on the system; robots need to be assigned to the tasks based on their features that need to match the tasks requirements; the paths computed to travel from and to different locations of the plant also form part of the problem. In fact, it can be assumed that encounters of two or more robots in some areas of the plant will result in deadlocks. Therefore, the choice of paths and the time in which each robot needs to travel them must ensure that such deadlocks do not occur.

Scheduling the fleet is a computationally hard problem, that can only be solved in reasonably short time for small instances limited to a few robots and tasks, as well as a small environment (more details in Section VI). In such cases, both approximate methods [3] and exact algorithms [4] can be used effectively. For larger systems, problem decomposition may help improve scalability. Specifically, iterating between first assigning robots to tasks and then schedule them along the paths to execute such tasks is a promising approach [5]. Alternatively, routes and schedules can be computed sequentially for one robot at the time [6]. While these approaches allow for faster computation and can solve larger problem instances, they can't guarantee global optimal solutions. Another approach to schedule the fleet is to use machine learning algorithms [7]; in particular, reinforcement learning has been applied successfully end-to-end [8, 9] to either drive a single robot, or the entire fleet.

While scheduling the fleet is necessary to meet the above-mentioned requirements, each robot also needs to be individually controlled to guarantee that it follows its schedule while avoiding collisions with obstacles and other robots. Model Predictive Control (MPC) has become a widely used approach for collision-free trajectory tracking, showing effectiveness across various applications [10, 11]. By incorporating obstacles and other robots as constraints and utilizing a receding horizon strategy, MPC can anticipate potential collisions within the planning horizon and generate actions that maintain safe trajectory tracking. By running MPC in a distributed manner [12], it can be integrated into other scheduling and coordination methods with good flexibility and scalability.

The combination of a high-level centralized fleet scheduler and a low-level distributed controller has proven to be a successful strategy [13, 14, 15]. In this paper, we present a sequential solution that involves high level scheduling first, i.e., the assignment of tasks to the available robots from the fleet, the design of routes from the robot's initial location, to the locations of the tasks assigned to it, and the scheduling of the robots along their routes, i.e., the relative point in time when the robot is supposed to reach a specific location along its route. This first phase is performed using the compositional algorithm ComSat [5]. The schedule for each robot becomes the input to the next phase, where a robot-specific local trajectory planner dynamically calculates a local reference trajectory and speed based on the urgency of the current task. Finally, a distributed MPC controller generates a kinematic control action given the real-time reference.

The remainder of the paper is as follows: the problem is formally introduced in Section II; Section III introduces the

[1]Division of Systems and Control, Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden{rsabino, zhze, knut.akesson}@chalmers.se

This work was supported by the Vinnova projects AIHURO (Intelligent människa-robot-samarbete) and CLOUDS (Intelligent algorithms to support Circular soLutions fOr sUstainable proDuction Systems)

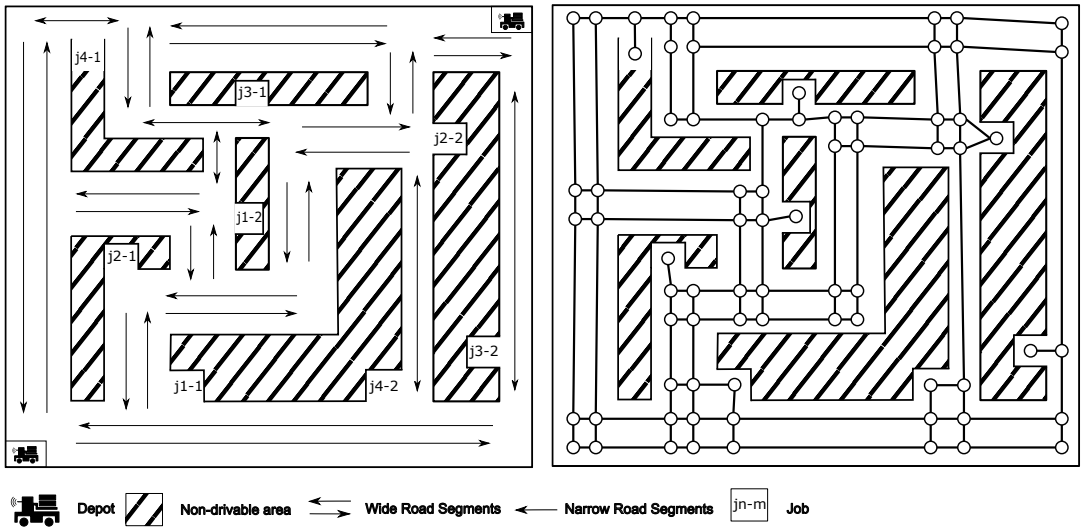

Fig. 1. Visualization of an illustrative environment, some roads are wide enough for bidirectional traffic or overtaking in the same direction, which is not possible for other narrow roads. The plant can be abstracted into a graph, as shown in the right sub-figure.

scheduler and Section IV connects it to the model predictive controller, discussed Section V; experiments setup and results are reported in Section VI and, finally, conclusions are drawn in Section VII.

II. PROBLEM AND SYSTEM DEFINITION

In this section, the fleet management system is defined by dividing it into the scheduler, local planner, and controller. We formalize the problem and provide the interface for each module. In this work, the workspace is assumed to be 2D. We use t for continuous time and k for discrete time, and $t = k \cdot \Delta t$ where Δt is the sampling time.

Scheduler: The schedule is computed over a finite time interval $[0, T]$, where the *Schedule Horizon T* is a fixed and continuous time instant when all tasks should be completed. As shown on the left side of Fig. 1, the workspace for mobile robots can be distinguished into drivable (blank areas) and non-drivable (dashed areas). The drivable areas are organized into *road segments* that can be either *narrow* or *wide*; this property of the segments is henceforth named *capacity*. In narrow segments, two robots traveling in the same direction cannot drive next to each other, i.e., overtaking is not possible, and two robots traveling in opposite directions end up in an impasse. In wide segments, the opposite is true. Therefore, as shown on the right side of Fig. 1, the environment is abstracted into a finite, strongly connected, weighted, directed graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, where nodes $n \in \mathcal{N}$ represent either tasks' locations, depots, or intersections; edges $e = (n, n') \in \mathcal{E}$ represent narrow road segments or lanes of wide road segments; the inverse of an edge $e = (n, n')$ is denoted $\bar{e} = (n', n)$, and the length of a road segment is represented by the length of the edge $|e|$.

In a set $\mathcal{V}$ of mobile robots, which contains $|\mathcal{V}|$ robots, each robot $v \in \mathcal{V}$ moves between locations in the plant at constant speed $\bar{\Omega}$ and performs its tasks. Robots start from their dedicated depot and return either after completing their tasks or intermittently between tasks for battery recharging. As battery-powered vehicles with a limited operating range, their energy consumption is linearly proportional to the distance traveled; they can recharge to full capacity [16] at their depot, but not at other depots. Each task $a \in \mathcal{A}$, where $\mathcal{A}$ is the set of tasks, represents a place to be visited exactly once by a vehicle to pick up or deliver material. A task a is always associated with a node where the pickup/delivery takes place, and has a time window $[t_a^-, t_a^+]$ indicating the earliest and latest time at which the node can be visited. Unless explicitly given, the time window is $[0, T]$. Moreover, each task has a precedence list that indicates other tasks that must be executed before the task itself. This is so since the tasks represent pickups and deliveries, hence a vehicle has to deliver the goods after it has picked them up.

The scheduler processes the graph $\mathcal{G}$, the task set $\mathcal{A}$, and the vehicle set $\mathcal{V}$, and produces a schedule $\mathcal{S}$, an ordered set of triplets, each specifying the time at which a robot reaches a particular node along its route.

Local planner: Each robot has its local planner to generate a local reference trajectory at every step. The local planner for robot v_i queries the overall schedule $\mathcal{S}$ for the specific schedule s_i and generates a local reference trajectory $\mathcal{T}_k^{(i)}$ at each time step k, consisting of a sequence of N points, where N is the predictive horizon for the MPC controller.

Controller: For each robot v_i, the controller takes the reference trajectory $\mathcal{T}_k^{(i)}$ generated from the corresponding local planner and computes a collision-free action $\boldsymbol{u}_k^{(i)}$ accordingly at time step k. The controller should avoid collisions with both environmental obstacles and other robots.

III. FLEET SCHEDULER

In this section, we introduce the compositional algorithm ComSat [5] used for the high-level scheduling and coupled with the local planner and controller in the proposed method. To show how the scheduler computes the output $\mathcal{S}$ based on

the input $(\mathcal{G}, \mathcal{A}, \mathcal{V})$, we provide the following definitions:

- *Depots*: nodes at which vehicles start and must return to after completing the assigned tasks.
  $\emptyset \subset \mathcal{D} \subseteq \mathcal{N}$: the set of depots.
- *Route*: a sequence, starting and ending at the same depot, of unique tasks in-between that may have the same depot embedded:
  $R_j = \langle d, a_1, a_2, \ldots, d, \ldots, a_{n-1}, a_n, d \rangle$ where $d \in \mathcal{D}, a_i, a_j \in \mathcal{A}$, $a_i \neq a_j$ for $i \neq j$, and $n \leq |\mathcal{A}|$ since a route can at most include all tasks.
- *Path*: ordered set of unique nodes, used to keep track of how vehicles travel among tasks of routes, since each pair of consecutive tasks is connected by a path.
  $\theta_{a_i,a_j} = \langle n_1, \ldots, n_m \rangle,\ a_i, a_j \in R, j = i+1,$
  $m \leq |\mathcal{N}|,\ n_i \in \mathcal{N},\ i = 1, \ldots, m$
- *Edge sequence*: ordered set of unique edges for a given path θ_{a_i,a_j}.
  $\delta_{a_i,a_j} = \langle e_1, \ldots, e_m \rangle,\ a_i, a_j \in R, j = i+1,$
  $m = |\theta_{a_i,a_j}| - 1,\ e_i \in \mathcal{E},\ i = 1, \ldots, m$

Initially, a set of paths to connect each two tasks/depots in the plant is computed using Dijkstra's algorithm [17]. Because these are the shortest paths, they can be traveled in the shortest time. Using these paths, a set of routes $\mathcal{R}$ is computed by solving the *E-Routing* sub-problem; these routes are designed accounting for the tasks' time windows (if any), their requirements on the type of robot to execute them, as well as the precedence constraints among tasks. Moreover, they take into account the limited operating range of the robots and, if necessary, they involve a visit to the depot to recharge the battery. Because the *E-Routing* sub-problem uses the shortest paths between tasks/depots to compute $\mathcal{R}$, if this problem is infeasible, we know that the overall scheduling problem is infeasible. On the other hand, if the *E-Routing* sub-problem is feasible, the set of routes $\mathcal{R}$ constitutes the input for the *Capacity Verification*, in which a schedule for each robot to travel along its route is computed; the schedule marks at which time each node in the route should be reached, and accounts for the road segment width, to avoid deadlocks.

In order to adapt the *Capacity Verifier* to the presented framework, one minor adjustment was required. To discuss such an adjustment, we introduce some constraints of the problem, defined using the following variables:

x_{Rn}: non-negative real variable that models the time when a vehicle executing route R arrives at node n;
y_{Re}: non-negative real variable that models the time when a vehicle executing route R starts traversing edge e.

Also, let e_n be the edge visited before node n and, with some abuse of notation, let θ_R and δ_R be a path and an edge sequence to travel along route R respectively. Then the relevant constraints for the *Capacity Verification* are:

$$x_{Rn} \geq y_{R'e^n} + \mu \ \vee \ x_{R'n} \geq y_{Re^n} + \mu, \quad \forall R, R' \in \mathcal{R},\ R \neq R',\ n \in \theta_R \cap \theta_R \quad (1)$$

$$y_{Re} \geq y_{R'e} + \mu \vee y_{R'e} \geq y_{Re} + \mu, \quad \forall R, R' \in \mathcal{R},\ R \neq R',\ e \in \delta_R \cap \delta_{R'} \quad (2)$$

$$y_{Re} \geq y_{R'e'} + |e'|/\bar{\Omega} + \varphi \ \vee y_{R'e'} \geq y_{Re} + |e|/\bar{\Omega} + \varphi, \quad \forall R, R' \in \mathcal{R},\ R \neq R',\ e \in \delta_R,\ e' \in \delta_{R'},\ e = \bar{e}' \quad (3)$$

Constraints (1)-(3) prevent robots from occupying the same physical location in space at a node or along an edge (see [5] for the complete model). In the original work, the parameter μ is a small positive value used to prevent swapping of robots' positions in (1) and to forbid robots to traverse the same edge at exactly the same time in (2); φ in (3) is equal to zero. However, in this work, $\mu = \varphi$ is a safety margin empirically set to 20 seconds to account for delays in the schedule and prevent robots from occupying the same physical location too close in time.

If the *Capacity Verification* problem is feasible, a schedule $\mathcal{S}$ is returned by ComSat; if it is infeasible, new paths (other than the shortest ones computed using Dijkstra's) may be found by solving the *Paths Changing* sub-problem.

The scheduler iterates between the above-mentioned sub-problems until either a solution $\mathcal{S}$ is found when the *Capacity Verification* sub-problem is feasible, or the problem is deemed infeasible when the *E-Routing* sub-problem is infeasible. For more details on the workings and performance of this scheduling algorithm, we refer the reader to [5]. Fig. 2 shows how the sub-problems are connected.

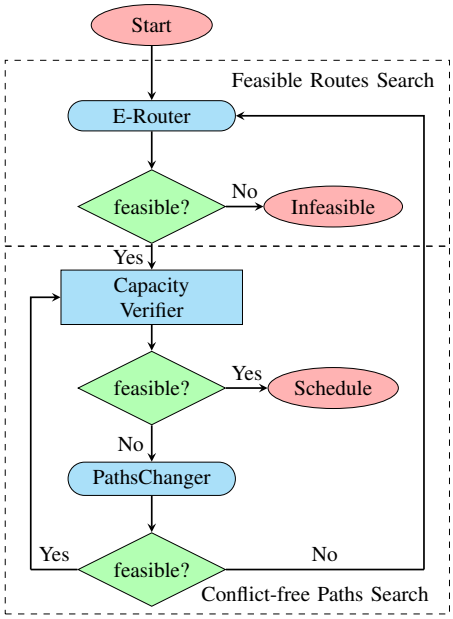

Fig. 2. Flowchart of ComSat [5].

IV. BRIDGING SCHEDULER AND CONTROLLER

The schedule $\mathcal{S}$ focuses on task allocation and assumes a constant speed of mobile robots to simplify the optimization problem. While in real-world execution, robots

may encounter delays due to longer task execution times or traffic obstructions, making it impractical to directly follow the schedule with basic schedule-tracking controllers. To address this, a local planner is employed to bridge the global scheduler with the local control, dynamically adjusting the reference speed and reference trajectory of each robot according to the given schedule.

For each robot v (index i is omitted), its schedule s is a reference path from the start point, passing all intermediate waypoints, to the end point. To achieve real-time performance, a global reference trajectory is precomputed upon receiving the schedule by sampling the reference path under the assumption of maximal velocity $\Omega_{\max}$, rather than generating a local trajectory from scratch at each time step. During operation, an initial reference trajectory $\bar{\mathcal{T}}_k$ is extracted from the global trajectory based on the current location of the robot. Assuming the current time is t and the scheduled arriving time for the next node is t_n, the desired speed for the robot at time step $k = t/\Delta t$ will be

$$\Omega_k = \min\left(\Omega_{\max},\, L_{\text{next}}/\max(0, (t_n - t))\right),$$

where L_{next} is the distance to the next node. Then, a new reference trajectory $\mathcal{T}_k$ is generated by resampling the initial one $\bar{\mathcal{T}}_k$ with the ratio $\Omega_k/\Omega_{\max}$.

V. MODEL PREDICTIVE CONTROL

For robot v_i, after obtaining the desired speed $\Omega_k^{(i)}$ and reference trajectory $\mathcal{T}_k^{(i)}$, the MPC problem can be solved. The objective of the MPC problem consists of three parts: reference tracking, obstacle avoidance, and fleet collision avoidance. For reference tracking, at time step k,

$$J_R(k) = ||\boldsymbol{s}_k - \tilde{\boldsymbol{s}}_k||^2_{\boldsymbol{Q}_s} + ||\boldsymbol{u}_k - \tilde{\boldsymbol{u}}_k||^2_{\boldsymbol{Q}_u} + ||\boldsymbol{u}_k - \boldsymbol{u}_{k-1}||^2_{\boldsymbol{Q}_a}, \quad (4)$$

where $\boldsymbol{s}_k$ is the state, $\tilde{\boldsymbol{s}}_k$ is the reference state from $\mathcal{T}_k^{(i)}$, $\boldsymbol{u}_k$ is the action, $\tilde{\boldsymbol{u}}_k$ is the reference action composed by $\Omega_k^{(i)}$, and all $\boldsymbol{Q}$ variables are corresponding weight matrices. The obstacle avoidance term $J_{\mathcal{O}}(k)$ is based on the indication of the robot being inside or outside of a polygonal obstacle [12]. The fleet collision avoidance is defined as

$$J_F(j) = \max\left[0, Q_f \cdot \left(d_{\text{fleet}} - ||\boldsymbol{z}_k^{(i)} - \boldsymbol{z}_k^{(j)}||\right)^2\right], \quad (5)$$

where $\boldsymbol{z}^{(i)}$ is the position of robot v_i, d_{fleet} is the safe distance between robots, and Q_f is the weight parameter.

Assuming the current time step $k = 0$, the complete MPC problem can be formulated as

$$\min_{\boldsymbol{u}_{0:N-1}} \sum_{k=0}^{N-1} \left[J_R(k) + J_{\mathcal{O}}(k) + \sum_{j=1, j\neq i}^{|\mathcal{V}|} J_F(j)\right], \quad (6)$$

$$\text{s.t.} \quad \boldsymbol{s}_{k+1} = f(\boldsymbol{s}_k, \boldsymbol{u}_k),\ \forall k = 0, \ldots, N-1, \quad (7)$$

$$\boldsymbol{u}_k \in [\boldsymbol{u}_{\min}, \boldsymbol{u}_{\max}],\ \forall k = 0, \ldots, N-1, \quad (8)$$

$$\dot{\boldsymbol{u}}_k \in [\dot{\boldsymbol{u}}_{\min}, \dot{\boldsymbol{u}}_{\max}],\ \forall k = 0, \ldots, N-1, \quad (9)$$

$$\boldsymbol{z}_k \notin \mathcal{O},\ \forall k = 0, \ldots, N-1, \quad (10)$$

where $f(\cdot)$ is the motion model, $\dot{\boldsymbol{u}}_k$ is the derivative of the action, and $(\boldsymbol{u}_{\min/\max}, \dot{\boldsymbol{u}}_{\min/\max})$ are physical limits of the robot's action.

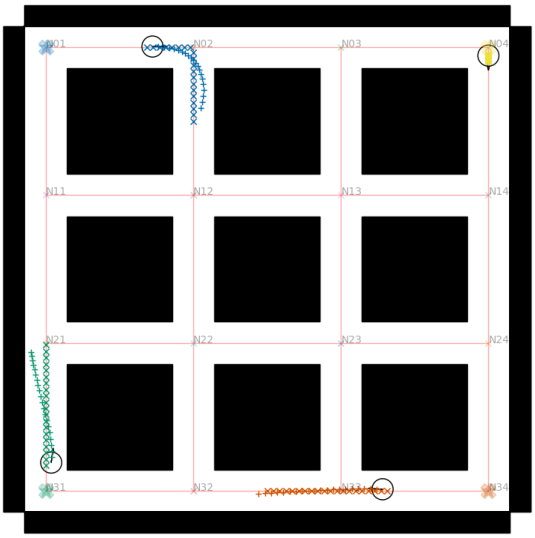

Fig. 3. Map of the small environment with four horizontal and four vertical narrow road segments. The map is abstracted into a graph (nodes from *N01* to *N34* and orange edges), and the robots are represented by circles, with their reference trajectories marked by crosses ($\times$) of different colors and predicted trajectories marked by pluses ($+$).

VI. EXPERIMENTS

In this section we present the experimental setup developed to test the proposed framework, as well as the results achieved by it. The *small* setup is a grid-like environment with four horizontal and four vertical narrow road segments, abstracted into a graph with 16 nodes, as shown in Fig. 3. In this setup, four robots are initially located at the four corners of the map (the depots), and each of them has to travel to three other locations before returning to its depot. For instance, robot *A4*, represented in yellow and initially located in the top-right corner, corresponding to Node *N04* has to travel to the locations *N23*, *N21*, and *N24*, in this order, before going back to *N04*.

A second experiment, shown in Fig. 6 (last page) is set up using a map that mimics a factory layout, in which an assembly line is located in the middle and on each side of it there is a set of kitting areas where the robots, dispatched from two depot areas, pick up the material to deliver to the assembly line. All road segments are narrow, with the exception of the two main ones running parallel to the assembly line. The map is abstracted into a graph with 106 nodes and 292 edges. In this set up, ten robots are initially located in the two depot areas, five in each; each robot starts and returns to its own depot, after visiting one of the kitting areas (pickup) and one of the work stations in the assembly line (delivery).

The scheduler uses the state-of-the-art MILP solver Gurobi 12.0.1 [18] to solve the *E-Routing* and *Path Changing* sub-problems and the state-of-the-art SMT solver Z3 4.14.1 [19] to solve the *Capacity Verification* sub-problem. The MPC problem is solved via the PANOC algorithm [20], which is a real-time nonlinear optimal control solver. Experiments are run on a MacBook Pro with 18GB of RAM and M3 Pro core

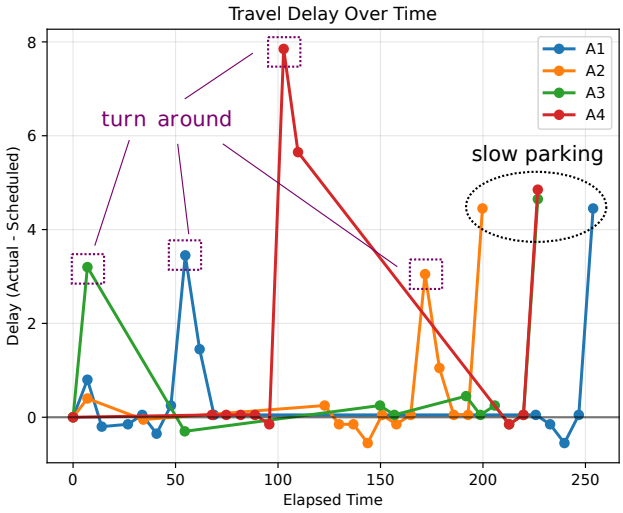

Fig. 4. Execution delays (in Seconds) of the four robots in the small environment. The delay, positive when the robot is late or negative otherwise, is the difference between the actual arrival time at a node versus the scheduled arrival time.

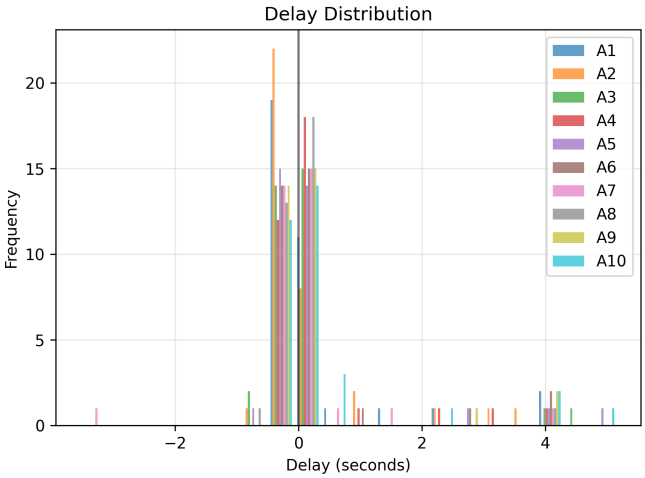

Fig. 5. Delay distribution for ten robots in the large environment.

running Sequoia 15.5.[1]

Both experiments show that the robots are able to accomplish their tasks and return to their depots in due time. In Fig. 4 is reported the delay of each robot to reach each node of its route for the experiment in the small environment; this delay is computed by logging when the robot actually reaches a certain node during the simulation and comparing it to the scheduled time in which the robot should reach such node. The delay for all robots ranges from 0 to 5 seconds on average, with a peak of almost 8 seconds for robot *A4*. Delays happen when robots need to perform 180 degrees changes of directions, which are non instantaneous; however, the controller increases the speed of the robot when falling behind schedule on a road segment and is able to compensate. The delay all robots experience when reaching the last node of their route is instead due to an intended slow parking behaviour.

A similar result is achieved in the large environment with ten robots. Fig. 5 shows the delay distribution for each robot. The vast majority of arrival times at the node fall within 1 second before or after the expected arrival time, with a maximum lateness below five seconds. This consistency across environments of varying scale suggests that the control strategy scales effectively, maintaining tight synchronization with scheduled plans even as the number of agents and route complexity increase.

We compared the presented method with two alternatives; one, in which the MPC on each robot is replaced with a simple *reference tracker* that does not account for the other robots' presence, but only for static obstacles. In this case, collisions happened if two robots were scheduled to travel through an intersection at similar time. The other alternative used the MPC presented in Section V and the routes computed by the scheduler, but no time-schedule, i.e., the robots are traveling along the routes as fast as they can. In this case, the lack of centralized scheduling resulted in collisions for every single simulation.

[1]The implementation of the proposed framework, including the scheduler based on ComSat, the MPC, and the problem instances, is available at `https://github.com/Woodenonez/TrajPlan-ScheMPC`

VII. CONCLUSIONS AND FUTURE WORK

This work presents an integrated framework that bridges high-level scheduling and low-level control to manage fleets of mobile robots operating in dynamic and constrained environments. By combining the compositional scheduling algorithm ComSat with a distributed Model Predictive Control (MPC), we demonstrate a scalable and responsive system capable of handling both long-term planning and real-time execution challenges. The proposed method first computes conflict-free schedules, then adapts these at runtime through a local planner and MPC-based controller, ensuring safe and timely task completion despite disturbances such as delays or congestion.

The experimental results validate the practicality of our approach. In both small-scale and large-scale scenarios, the robots successfully complete their assigned tasks and return to their depots with minimal deviation from the schedule. Importantly, the use of MPC enables robots to recover from potential delays by dynamically adjusting their trajectories, highlighting the method's robustness and flexibility.

Moreover, the framework is modular, allowing for future extension with additional constraints, such as energy optimization, learning-based adaptations, or human-robot interaction models. As mobile robot fleets become increasingly common in industrial and logistics settings, this work provides a solid foundation for building efficient, adaptable, and scalable robotic management systems

The obvious next step is to implement an experimental setup with real robots to validate the proposed method under realistic conditions. This will allow evaluation of its robustness against uncertainties such as sensor noise, communication delays, and environmental interferences, providing insight into its practical feasibility beyond simulation.

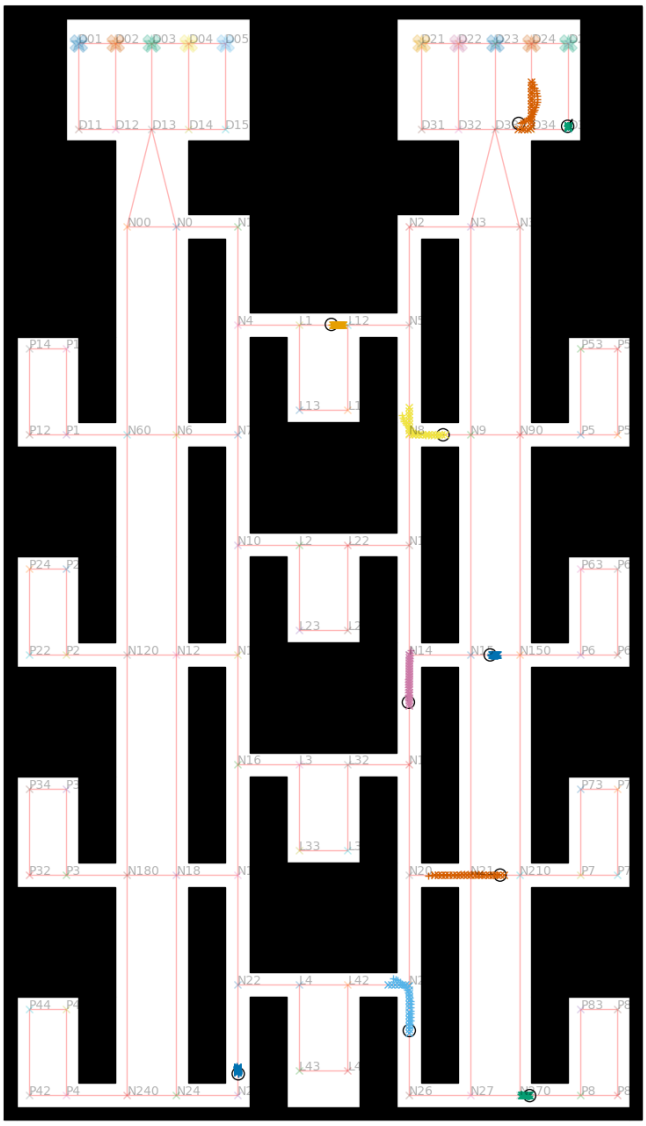

Fig. 6. Map of the large environment mimicking a manufacturing/assembly plant layout. The main line where goods need to be shipped is in the middle, with four workstations, while four kitting areas on each side are where material is picked up. The robots' depots are located in two areas at the top. The map is abstracted into a graph, and the robots are represented by the circles, with their planned trajectories marked by x of different colors.

References

[1] Sungbum Jun, Seokcheon Lee, and Yuehwern Yih. "Pickup and delivery problem with recharging for material handling systems utilising autonomous mobile robots". In: *European Journal of Operational Research* 289.3 (2021), pp. 1153–1168.

[2] Giuseppe Fragapane et al. "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda". In: *European Journal of Operational Research* 294.2 (2021), pp. 405–426.

[3] Sungbum Jun, Chul Hun Choi, and Seokcheon Lee. "Scheduling of autonomous mobile robots with conflict-free routes utilising contextual-bandit-based local search". In: *International Journal of Production Research* 60.13 (2022), pp. 4090–4116.

[4] Sabino Francesco Roselli, Martin Fabian, and Knut Åkesson. "Solving the conflict-free electric vehicle routing problem using SMT solvers". In: *Mediterranean Conference on Control and Automation (MED)*. IEEE. 2021, pp. 542–547.

[5] Sabino Francesco Roselli, Martin Fabian, and Knut Åkesson. "Conflict-free electric vehicle routing problem: an improved compositional algorithm". In: *Discrete Event Dynamic Systems* 34.1 (2024), pp. 21–51.

[6] Francesco Popolizio et al. "Online Conflict-Free Scheduling of Fleets of Autonomous Mobile Robots". In: *IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2024, pp. 3063–3068.

[7] Aigerim Bogyrbayeva et al. "Machine learning to solve vehicle routing problems: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 25.6 (2024), pp. 4754–4772.

[8] Maria Amélia Lopes Silva et al. "A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems". In: *Expert Systems with Applications* 131 (2019), pp. 148–171.

[9] Ke Zhang et al. "Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach". In: *Transportation Research Part C: Emerging Technologies* 121 (2020), p. 102861.

[10] Ze Zhang et al. "Future-Oriented Navigation: Dynamic Obstacle Avoidance With One-Shot Energy-Based Multimodal Motion Prediction". In: *IEEE Robotics and Automation Letters* 10.8 (2025), pp. 8043–8050.

[11] Ardalan Tajbakhsh, Lorenz T. Biegler, and Aaron M. Johnson. "Conflict-Based Model Predictive Control for Scalable Multi-Robot Motion Planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 14562–14568.

[12] Filip Bertilsson et al. "Centralized versus Distributed Nonlinear Model Predictive Control for Online Robot Fleet Trajectory Planning". In: *IEEE International Conference on Automation Science and Engineering (CASE)*. 2022, pp. 701–706.

[13] Maximilian Berndt et al. "Centralized robotic fleet coordination and control". In: *Mobile Communication-Technologies and Applications; 25th ITG-Symposium*. VDE. 2021, pp. 1–8.

[14] Diogo Miguel Matos et al. "Efficient multi-robot path planning in real environments: a centralized coordination system". In: *International Journal of Intelligent Robotics and Applications* 9.1 (2025), pp. 217–244.

[15] Elzbieta Roszkowska, Piotr Makowski-Czerski, and Lukasz Janiec. "Multi-level control for multiple mobile robot systems". In: *Discrete Event Dynamic Systems* 33.4 (2023), pp. 425–453.

[16] Michael Schneider, Andreas Stenger, and Dominik Goeke. "The electric vehicle-routing problem with time windows and recharging stations". In: *Transportation science* 48.4 (2014), pp. 500–520.

[17] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[18] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[19] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[20] Lorenzo Stella et al. "A simple and efficient algorithm for nonlinear model predictive control". In: *IEEE Annual Conference on Decision and Control (CDC)*. 2017, pp. 1939–1944. DOI: 10.1109/CDC.2017.8263933.