A Comparison of Two Decoupled Methods for Simultaneous Multiple Robots Path Planning

Benjamin Bouvier and Julien Marzat

DTIS, ONERA, Université Paris Saclay, F-91123 Palaiseau, France
{firstname dot lastname at onera dot fr}

Abstract. Two path planning algorithms dedicated to multiple robots driving simultaneously in a cluttered workspace are defined and compared in this paper. Each robot is assigned an initial position, a goal position and a constant reference speed, and has to drive without colliding with its teammates and the environment. Both approaches are based on an implementation of the widely known A* algorithm and belong to the family of *decoupled path planning methods*, since robots are considered sequentially and with a predefined priority ranking. The first algorithm is called *Prioritized Planning* (PP), where the path of each robot is computed sequentially while taking into account static obstacles as well as the previously-planned robot positions. The second algorithm, called *Fixed-Path Coordination* (FPC), follows a two-step approach: (i) obtaining A* paths of all robots taking into account static obstacles only; (ii) applying a velocity-tuning procedure so that lower-priority robots can stop and restart along their paths to let the higher-priority robots move freely. Both algorithms have been applied on a variety of test-cases through a Monte Carlo setup to evaluate and compare their performances.

Keywords: Multi-robot motion coordination, Decoupled path planning, Prioritized Planning, Fixed-Path Coordination

1 Introduction

Coordinated path planning of a group of mobile robots is a major requirement for many cooperative tasks in interaction with a complex environment (*e.g.* unknown, cluttered, dynamical). A high-level description of the problems related to either collaborative or competitive robot coordination can be found in [1]. This coordination implies an inherent conflict of resources: in path planning the conflict is mostly a space conflict because the robots move around each other, but it may also be a conflict due to limited communication channels or a conflict of payload if various robots manipulate the same one. Task allocation is a problem in itself too, as well as the option of leaving the fleet management to a centralized authority or allowing every robot to take decisions on their own. It is considered here that the allocation of goal positions has already been performed and that the problem consists in coordinating the robots' simultaneous motion in the workspace. When managing more than one robot, a solution can consist in applying *motion coordination* [2] *i.e.* coordinating the robots online, as they are moving, to prevent collisions from occurring. This can be achieved either through a centralized authority sending

commands to the robots (*centralized approach*) or with the robots communicating with each other based on their future displacements (*decentralized approach*). Motion coordination can result in various solutions: (i) a system of traffic rules; (ii) reactive approaches which can be based on potential fields with attractive forces towards goals and repulsive ones to stay away from obstacles; (iii) the management of a swarm or fleet with a common objective. For example, coordinated control approaches have been previously used for several applications that can be addressed by a fleet of autonomous aerial or terrestrial robots [3]. To limit computational complexity, the online search of a model predictive control input for each vehicle can be reduced to a discretized set while still taking into account the predicted motion of the other robots, as in [4]. However this strategy can be limited for autonomous robots moving freely in a very cluttered environment, where in some configurations no feasible solution can be found. This is why graph-based and sampling-based path planning methods should also be considered for simultaneous robot coordination [5, 6]. In the latter framework, a simple approach could consist in considering only the current positions of the other robots when computing the path of each one, but collision avoidance with the other robots is then not guaranteed. The objective of the present work is to study path planning methods that can be applied to multiple robots moving simultaneously in the same cluttered environment, with an explicit handling of their motion interaction. For this purpose two decoupled sampling-based path planning algorithms derived from the A* procedure are described in Section 3, and their performances are evaluated in a Monte Carlo setup with a combination of map sizes, rate of static obstacles and number of robots (see Section 4).

2 Related Work

Simultaneous motion planning for multiple robots comes with various difficulties, such as the numerical capacity of planning for a high number of robots or large workspaces, the consideration of robot dynamics as well as sensing constraints, or the transition from numerical simulation to real robots experiments [2]. To deal with this variety of possible issues, several motion planning paradigms have been derived as described in the reference textbook [5], where in particular algorithms can be classified as either sampling-based or combinatorial methods. In *Sampling-based motion planning*, the idea is to sample the free space to conduct the search for a path relying on a collision detection module, hence the explicit construction of the obstacle space is avoided. In *Combinatorial motion planning*, also referred to as *exact* methods, paths are computed in the continuous configuration space without resorting to sampling or discretization. Two main categories emerge from the literature to address simultaneous path planning, namely coupled approaches [6–8] and decoupled approaches [8–11]. In *coupled planning*, the problem can be seen as handling a single robot which concatenates the states of the actually different robots. This increases the dimension of the configuration space and complexity is known to rise exponentially in this number. Classical algorithms may then fail to find a solution or require too much time (this has been verified in our evaluations, see Section 4.1). In *decoupled path planning*, two subcategories can be considered [12]:

1. Robots are treated one by one. For every one of them, a path is calculated considering all other robots as moving obstacles on top of the already-existing static obstacles.

Two steps are required, namely: a planning algorithm and a prioritization rule. The planning algorithm is applied to obtain the path of each robot to its respective goal, and then prioritization should favor those with the highest cost (depending on time, distance, energy, etc.). For instance, robots with the least cost could be penalized by having to bypass or make detours.

2. Each robot has its own path, calculated independently of the others with only the static obstacles taken into account. Then a coordination diagram should plan the displacements to avoid collisions. Some authors mention a second phase of velocity tuning (*e.g.* in [13]) to coordinate robots in parallel: the relative speed between two robots can thus be tuned in the portion of path subject to a risk of collision.

Finally, it is possible to build *hybrid approaches* [12] which rely on both coupled and decoupled search. The idea is to divide the robots into subgroups depending on their level of dependency or degree of priority (the constitution of such groups is in itself a task). For each subgroup, which has a smaller configuration space, a coupled approach is applied. The groups are treated in a decoupled way with respect to one another. The authors in [14] proposed a planning algorithm based on satisfiability modulo theory, where suboptimal paths can be accepted if, in turn, this makes the problem solvable.

Coupled or *hybrid* approaches however still report computational loads that are prohibitive for embedded applications for realistic numbers of robots and workspace dimensions. It has thus been chosen to focus on *decoupled planning methods* in this work, since this category of approaches has the capacity of decomposing the problem into simpler ones, and the literature shows that feasible solutions can be found even when the number of robots, the workspace dimension or the degrees of freedom increase. In the context of managing intersections for autonomous vehicles, a priority-based coordination framework which combines a high-level priority graph with a feedback control law has been proposed in [15]. Under this framework, the proposed overall coordination system is proven to be collision-free and deadlock-free. In [16], a prioritized planning approach has been defined with a formal guarantee to provide a solution under strict conditions regarding the workspace and the robots initial conditions. An asynchronous decentralized implementation of this strategy has also been proposed and demonstrated reliable results via a simulation evaluation on real-world maps. Similar approaches relying on priority-based planning have also been proposed in [9, 10]. The main ideas for coordinating the speed of robots having independent goals along predefined paths have been presented in [8]. The so-called coordination diagram based on a bounding box representation of the obstacles has then been introduced in [17] to tackle the motion coordination of several robots moving along fixed independent paths while avoiding mutual collisions. Multi-robot path planning continues to be an active area of research with many different approaches proposed in the past few years, based *e.g.* on reinforcement learning [18, 19], particle swarm optimization [20, 21], graph neural networks [22] and with the design of novel heuristics [23].

Several decoupled multi-robot coordination strategies have been defined and evaluated in these previous works, relying either on priority-based successive planning or on velocity adjustment along fixed paths. Two such decoupled algorithms are defined in the present paper, one from each category, namely *Prioritized Planning* (PP) and *Fixed-Path Coordination* (FPC). They are both derived from the same A* framework with practical

implementations, which are detailed in pseudocodes. This allows to compare fairly the two approaches in the same simulation setup using extensive Monte-Carlo evaluations on randomly-defined workspace configurations, and as a result provide recommendations for the motion coordination of multiple robots having independent goals in cluttered environments.

3 Planning Algorithms

3.1 Problem formulation

The problem is illustrated in Figure 1. A two-dimensional workspace $W = \mathbb{R}^2$ is considered (but the methods can easily generalize to higher-dimension workspaces). This workspace can be separated into two parts: a first part where robots are free to move, C_{free}, and a second one occupied by static obstacles, C_{obs}, where robots are not allowed to drive. A robot A can undergo various transformations which result in reaching a particular configuration $q \in W$. An obstacle-free configuration such that $q \in C_{free}$ is considered valid, while a collision configuration $q \in C_{obs}$ is invalid. Let us now consider we have m robots, $m \in [\![2; +\infty[\![$. Every robot $A_i, i \in [\![1; m]\!]$, is assigned an initial configuration, q_I^i, and a final configuration or goal, q_G^i, defined in C_{free}. Contrary to the single-robot case where it is sufficient to consider robot-obstacle collisions, robot-robot collisions should be taken into account. Robots are moving with respect to each other, therefore a given robot A_i should consider all the other robots $A_j, j \neq i$, as moving obstacles. The set of configurations such that robot A_i collides with robot $A_j, j \neq i$, is defined as $C_{mov\,obs}^{i,j}$.

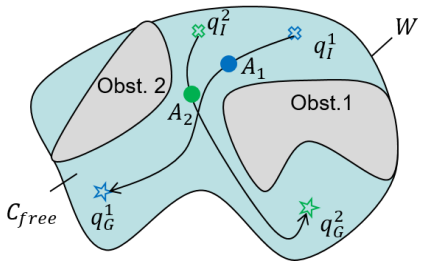


Fig. 1: Multiple robots path planning: robots A_1 and A_2 drive from their initial to final configurations while avoiding two static obstacles and ensuring they do not collide.

The path planning problem consists in finding, for every robot A_i, a continuous path $\tau_i : \{[0; 1] \rightarrow C_{free}\ ;\ s \mapsto \tau_i(s)\}$ where s parameterizes the path such that $\tau(0) = q_I^i$ and $\tau(1) = q_G^i$, and with an empty intersection with both C_{obs} and $C_{mov\,obs}^{i,j}$ for all $j \neq i$. The notion of *moving* obstacles requires to have a parameter that can be used to determine where a given robot is located at some instant. If all robots have the same speed, the cumulative distance could be used and s would be sufficient. However, for robots with different speeds v_i, time should be considered explicitly in the problem definition. Therefore, a date is associated to every configuration of the path. Instead of

looking for the parameterized path τ_i defined previously, we now look for a trajectory $\phi_i = \tau_i \circ \sigma_i$ considering a timing function defined as $\sigma_i : \{T \rightarrow [0;1]\ ;\ t \mapsto s\}$, where t denotes time.

3.2 Decoupled Prioritized Planning (PP)

The first algorithm is a decoupled two-dimensional algorithm derived from A* using prioritization to sequentially obtain the path of every robot without collisions. The method requires that the m robots have been ranked from highest to lowest priority. For the first one, A_1, a standard A* path is calculated from its initial position to its goal position, taking into account the static obstacles present in the workspace. Then, the second robot is treated: besides static obstacles, A_2 has to consider the previously computed motion of robot A_1 along its path with a given constant speed v_1. In case of collision with A_1 at a given node, A_2 will have to reach another configuration, therefore bypassing A_1 which has priority. This procedure is applied iteratively for the next robots until all have been treated. The idea is illustrated in Figure 2. In this algorithm, every robot is assumed to drive at its own constant nonzero speed, meaning that a robot is unable to stop. All it can do is bypass, which requires to have sufficient free space around the static and moving obstacles.

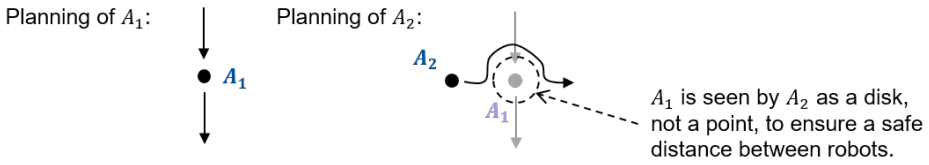


Fig. 2: PP Algorithm: A_2's path is obtained by bypassing the higher-priority robot A_1.

Each robot is assigned a vector of dates associated to its A* path, given its nominal velocity v_i specified by the user. For robot A_i, it will allow to interpolate the positions of all higher-priority robots $A_j, 1 \leq j < i$, to detect possible collisions. Collisions are defined as follows: the current robot A_i is considered as a point, while all the robots with higher priorities $A_j, 1 \leq j < i$, are considered to occupy a disk of parameterized safety radius centered at the robot position at the current date. If A_i finds itself inside or on the outer circle of any disk representing A_j, a collision is detected and the corresponding node is declared unfeasible. The pseudocode of the algorithm is given in Algorithm 1. It globally consists in a loop with a modified A* procedure applied for each robot (modifications are indicated in blue). The dates in the list $time\_vec$ (which is of same length as the heuristic scores g_{scores}) are defined for the nodes present either in the $closedSet$ (which progressively contains the path) or in the $openSet$ (which contains nodes considered to be part of the path but eventually not selected). The considered heuristic dist(nbg, n) is based on the Euclidean distance. In the end, a trajectory $(path^i; dates^i)$ is obtained for each robot i. If the $while$ loop is exited with $success = false$, the path is extracted from the node of minimal heuristic value instead of q_G^i, therefore an incomplete path is obtained to determine how far the robot can go before having to stop.

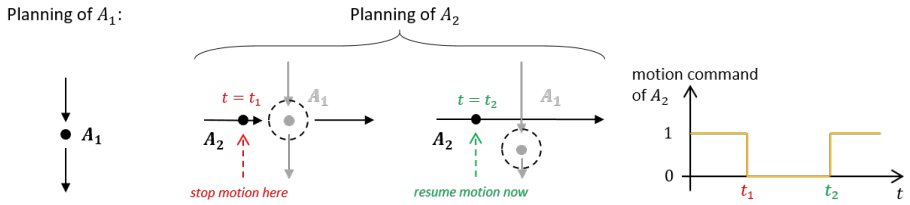
The core of the multi-robot layer lies in the robot-robot collision detection. When managing static obstacles, it is sufficient and straightforward to check if the intended configuration is inside or outside C_{obs}. When it comes to moving obstacles (higher-priority robots), one has to consider the relative speed between the robot and the obstacle as well as the obstacle width. Collision checks are performed every dT seconds, given the relative speed vector and the obstacle width. When moving from a current node position n to the intended one ngb, the robot does not change its direction but the moving obstacle might. As a consequence, care should be taken in selecting the dates and in calculating the relative speed vector. Moreover, collision checks are performed with all robots of higher priority (the sweeping is nevertheless stopped when a collision is detected because one is sufficient). All these additional calculations can be costly so they are run only if a static obstacle has not already been detected. Finally, a post-check is conducted on the obtained paths and dates such that the coordination is considered to have failed in the particular case where a lower-priority robot has reached its destination and is blocking the path of a higher-priority robot.

foreach *robot* $i \in [\![1; m]\!]$ [ranked from highest to lowest priority] **do**
  Initialize $closedSet$ to empty and $openSet$ to the node containing q_I^i;
  Initialize lists g_{scores} (cumulated distances from q_I^i) and $time\_vec$ (time instants);
  while $openSet$ *not empty* **do**
    Find node n in $openSet$ with minimal heuristic value mhv;
    if $mhv = \infty$ **then** Assign $success$:= false; *Break Loop*; **end**;
    if $n = q_G^i$ **then** Assign $success$:= true; *Break Loop*; **end**;
    Add n in $closedSet$; Remove n from $openSet$;
    for *all neighbors* ngb *of "n" inside the workspace* **do**
      if $ngb \notin closedSet$ **then**
        Compute candidate cost $alt := g_{scores}(n) + \text{dist}(ngb, n)$;
        if *"ngb" collides with a static obstacle or with previous robots on* $\{path^j, dates^j, \forall j < i\}$ **then**
          $alt := \infty$;
        end
        if $alt < g_{scores}(ngb)$ **then** Update g_{scores}, $time\_vec$ **end**;
      end
    end
  end
  if $success = true$ **then**
    Get full $path^i$ from start q_I^i to goal q_G^i and vector $dates^i$ from $time\_vec$;
  else
    Get incomplete $path^i$ and vector $dates^i$ from start to best reachable node;
  end
  Return $path^i$ and $dates^i$;
end

Algorithm 1: Prioritized Planning (PP) pseudocode

3.3 Decoupled Fixed-Path Coordination (FPC)

The second algorithm is also decoupled, two-dimensional and A*-based. It differs from the PP algorithm in that it allows robots to stop on their paths and resume their motions. Therefore, robot-robot collisions are not avoided by bypassing but simply by stopping and starting again (see Figure 3a), which is one possible choice for modulating the robot velocities along their trajectories [5].



(a) Principle of FPC Algorithm (b) Motion command

Fig. 3: A_2's trajectory is obtained by pausing until higher-priority robot A_1 has gone by.

First, an A* path is obtained independently for each of the m robots by considering static obstacles only (referred to as *single-robot planning*). The input requirements are the same as the previous algorithm: the robots should be ranked from highest to lowest priority and each is assigned a nominal velocity v_i. A speed profile is then initialized for each robot along the A* paths. The next step consists in tuning these velocity profiles to ensure collision-free paths, which is achieved by inserting pauses every time a robot-robot collision is detected. Note that these breaks can be inserted anywhere in the path, which means not necessarily on the nodes of the A* grid. If a collision is detected (with the same mechanism as in PP), the robot with the lowest priority will pause as long as necessary for the higher-priority robots to go past. Another less conservative option would be to optimize the robot motions given some global criterion to select which robot should stop for given collision conditions, which might result in different prioritization rules for similar collision configurations at different time instants.

In the PP algorithm, a list of dates has been added as additional information to handle collision-checking. The same applies in this second algorithm, with the addition of a binary *motion command* state associated to each date, to determine if the robot is driving or waiting (see Figure 3b). Another difference with Algorithm 1 is the calculation of dT. In PP it is calculated from the relative speed vector between two given robots at a certain date, while a unique dT is now used in FPC for all couples of robots on the basis of the highest possible relative speed. The pseudocode is given in Algorithm 2. Every time a collision is detected, a pause is inserted, which requires to check again the collisions of all pairs of robots at the date of collision, in case the pause insertion would induce a collision that did not exist before. Also, when two robots are moving exactly in opposite directions, that stop may need to be inserted several times dT before the collision date, which requires going backwards in time to check all pairs and can turn out to be costly. Although not explicit in Algorithm 2, the main *while* loop (line 7) can be interrupted

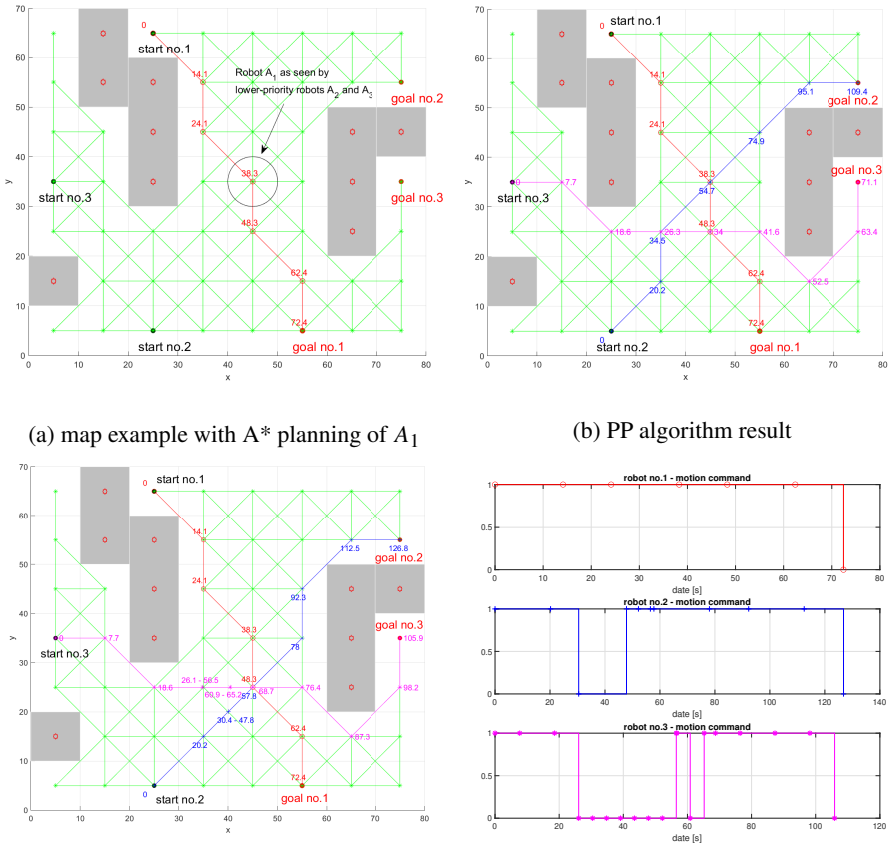
on a maximum number of collision checks in the sub-routines, which indicates a likely absence of solution and is thus considered as a failure to find coordinated paths.

for *all robots* $i \in [\![1; m]\!]$ **do**
  get A* $path^i$ (with static obstacles only), initial $dates^i$ and $motion^i$ vectors;
end
$T_{max} \coloneqq$ maximum final date among all robots;
Calculate dT given nominal speeds $v_i, i \in [\![1, m]\!]$;
Initialize $K \coloneqq \text{ceil}(T_{max}/dT)$ and time index $k \leftarrow 1$;
while $k \leq K$ **do**
  Initialize boolean $any\_collision\_detected$ with value $true$;
  Initialize integer $k\_candidate$ with value k;
  while $any\_collision\_detected$ **do**
    Set $collision\_counter$ to 0;
    for *robot r from* 1 *to* $m - 1$ [loop over priority robots] **do**
      for *robot s from r to m* [loop over subordinate robots] **do**
        if *robot 's' collides with robot 'r'* **then**
          $collision\_counter \leftarrow collision\_counter + 1$;
          Insert a stop at last date d_{stop} when a motion command had been applied before the collision, and update $dates^s$ and $motion^s$;
          $k\_candidate \leftarrow \min\{k\_candidate; \text{ceil}(d_{stop}/dT)\}$;
          if *final date of robot 's' > final date of all other robots* **then**
            $K \leftarrow K + 1$;
          end
        end
      end
    end
    if $collision\_counter == 0$ **then** $any\_collision\_detected \coloneqq false$;
    end
  end
  if $k\_candidate < k$ **then**
    $k \leftarrow k\_candidate$; [start over the collision check in the subinterval containing or ending by the smallest d_{stop} used in the latest collision while loop]
  else
    $k \leftarrow k + 1$ [go to next time subinterval];
  end
end
For all robots $i \in [\![1; m]\!]$, return $dates^i, motion^i$ ($path^i$ is fixed by design);

Algorithm 2: Fixed-Path Coordination (FPC) pseudocode

4 Numerical Experiments

Extensive simulations have been run based on the following setup. The workspace is a bounded subset of $\mathbb{R}^2$ with translational degrees of freedom. As shown in Figure 4a, the free space is sampled at regularly-spaced points (in green) which form a grid with at most eight neighbors per node. Robots move along the edges with the aim of traveling from their starting positions (black) to their final ones (red). The number indicated next to every node is the date at which the robot reaches the corresponding position.



(a) map example with A* planning of A_1

(b) PP algorithm result

(c) FPC algorithm result

(d) FPC motion commands

Fig. 4: Illustrative test case with 3 robots successfully solved by PP (b) and FPC (c-d). Arrival times are indicated around the nodes and motion command profiles (d) show the stop-and-go behaviour of FPC.

4.1 Illustrative test cases

Coupled planning A coupled A* strategy has been considered as a baseline, where all the positions of the robots in the fleet were aggregated into a single state vector. However the combinatorial complexity grows exponentially with the number of nodes and the number of robots, making it impractical for the simultaneous path planning of teams with more than two or three robots. This has been verified on a simple example with around 100 nodes where computing the paths takes a few milliseconds for one robot, several seconds for two robots, and several hours for three robots. Therefore, the focus has been put on the comparison of the proposed decoupled algorithms.

Decoupled methods As an illustrative example, a simple test case for three robots with different velocities is presented in Figures 4b to 4d. Both decoupled algorithms are successful in coordinating the three agents. With PP, robot A_2 finds an alternative node to avoid a collision with A_1 at node $(45; 25)$. With FPC, robots A_2 and A_3 stop once and twice respectively, increasing their motion duration by 16 and 89% compared to the constant-speed travel time of the initial independent A* paths.

4.2 Monte Carlo simulations

In order to evaluate and compare the algorithms performances, a Monte Carlo simulation was set up. It consisted in (i) creating random scenarios (following uniform distributions) as two-dimensional maps with randomly-placed static obstacles and random initial and final configurations of robots and (ii) applying the two algorithms. The following parameters are selected for each experiment: (a) a number of nodes (related to the map size), (b) a rate of static obstacles and (c) a number of robots. The static obstacles are placed so that a particular percentage of the map (called *occupancy rate*) is covered with them, *e.g.* 10% means that 10% of the nodes are occupied by static obstacles. Even though both algorithms allow the robots to have different nominal speeds, they were all set to the same value in these evaluations for a fair comparison since path lengths and travel times are handled differently by PP and FPC. Two values were respectively chosen for the map size (about 10^3 and 10^4 nodes), the occupancy rate (10 and 30%) and the number of robots (5 and 10), and all eight combinations evaluated. Any map where single-robot A* planning failed was dismissed, since that means at least one destination cannot be reached. The expression *single-robot planning* refers to the superposition of the paths obtained for every robot considering only the static obstacles and ignoring the other agents. Two main metrics were computed for all the test cases and algorithms: the success rate and the increase of travel duration with respect to the single-robot reference (see Table 1).

Table 1: Monte-Carlo comparison (mean results on 1000 runs) for PP and FPC algorithms

Number of Nodes		9.10^2				10^4			
Occupancy Rate		10%		30%		10%		30%	
Number of Robots		5	10	5	10	5	10	5	10
Single-robot success rate (%)		72	22	57	9.3	90	64	86	49
Multi-robot success rate (%) (single-robot failures)	PP	**89**	**75**	**86**	**64**	**90**	**85**	**96**	**86**
	FPC	75	62	74	51	81	80	80	77
Travel duration increase (%)	PP	**1.4**	**1.6**	**6.1**	**5.4**	**0.75**	**0.34**	**0.80**	**0.71**
	FPC	45	41	48	57	18	33	32	34

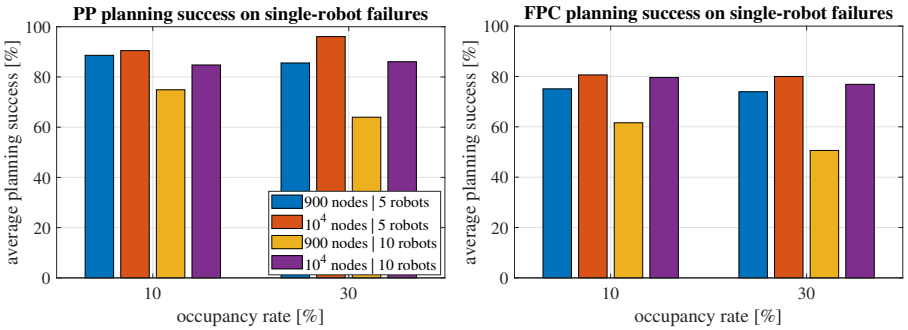


Fig. 5: PP and FPC success rates show little dependency on occupancy rate

Rate of planning success Single-robot success refers to cases where independent single-robot planning happens to generate no collisions with constant-speed travel. As can be seen in Table 1 (line 1), the success rates ranging from 10 to 90% decrease by 4 to 15 points when the occupancy rate is tripled and by 26 to 67 points when the number of robots is doubled. Available nodes become progressively less numerous so the probability of path overlap and/or crowded areas rises, which produces more collisions. On the contrary, growing the number of nodes has a positive impact on the success rate because more free space becomes available: paths are less likely to overlap and, even if they do, robots are less likely to take these common portions at the same dates.

Considering now the cases where single-robot planning fails and multi-robot planning becomes necessary, we may compare the efficiency of PP and FPC (lines 2-3 in Table 1, also shown in Figure 5). It should be noted that this metric is calculated only on the subset of single-robot failures. PP has a success rate between 64 and 96% and FPC from 51 to 81%. PP always performs better than FPC because FPC fails when an initial or final position is on another robot's path. A goal position acts as an additional static obstacle from the perspective of a lower-priority robot; an initial position is also seen as a static obstacle but by a higher-priority robot because this is the only position that cannot be freed by holding up the robot somewhere else in the map. In a nutshell, FPC handles path overlaps less efficiently. A check on initial and final positions as well as priority reordering could help avoiding such cases. Finally, the number of nodes has a positive impact on the planning status because it offers more possibilities for PP to find alternative paths and because FPC is less affected by the previously described issues.

Travel duration increase It is worth reminding the difference between PP and FPC path alteration. PP will change portions of paths to bypass higher-priority robots while driving continuously so both travel duration and distance may increase. FPC, however, can make smaller-priority robots wait but cannot make them divert from single-robot paths, which means distance remains unchanged. This is why Table 1 only mentions *travel duration* increase. Path lengthening would be zero for FPC and equal to duration increase for PP because all speeds are equal. The metric is computed only on multi-robot configurations for which the paths were successfully modified by the corresponding algorithm. The results show that FPC delays the robots much more than PP, with 18 to 57% of time

increase on average compared to less than 6.1% for the smaller map and always less than 1% on the bigger one. In many cases, bypassing a robot has a limited impact on arrival time because various slightly different paths (and sometimes ones of exact same length) lead to the goal. If two robots drive in opposite directions, one may simply shift to a parallel lane with PP whereas in FPC the smaller-priority agent will have to stop to give way.

5 Conclusion

Two path planning algorithms dedicated to multiple robots driving simultaneously in a cluttered workspace have been defined as variations of A* procedures and compared in this paper. Both are based on a predefined order of priority and assumptions of constant nominal velocities. The first one named *Prioritized Planning* (PP) consists in bypassing higher-priority agents while continuously driving. The second one named *Fixed-Path Coordination* (FPC) handles collision risks along fixed paths by inserting pauses in order to let higher-priority robots move past. The Monte Carlo simulation setup showed that directly superimposing single-robot planning paths can be successful in a number of configurations with limited occupancy rates and fleet sizes. This could thus be a first step when addressing a multi-robot coordination problem, where the new strategies can then be called upon when this straightforward approach has failed. PP showed good performances thanks to its flexibility in adapting the paths by following alternative nodes. Satisfactory success rates and very limited travel duration and distance increases were obtained. The proposed FPC version proved less efficient because of the incapacity of diverting from the initial single-robot paths. This strategy may only prove more efficient in extremely cluttered environments where alternative nodes would be harder to find for PP. Most of the FPC failed cases were due to initial and final positions conflicts, as they may prevent other robots from reaching their goals. Moreover, although distance remains minimal with FPC, travel duration grows significantly more than with PP. Free space sampling maximization is of interest to improve the success rate of both single- and multi-robot algorithms.

Future work directions include the evaluation of priority shuffles for both PP and FPC algorithms and their extension to higher-dimensional workspaces, which will allow to handle heterogeneous teams of robots. Additional comparison criteria can also be considered, such as energy efficiency. Finally, the deployment on embedded computers of real-world robots and the interaction with actual perception and control modules would also make it possible to estimate the computational load and performances in full autonomy loops on the field.

Acknowledgement This work was supported by ONERA project PR PARADIS.

References

1. Yan, Z., Jouandeau, N., Cherif, A.A.: A survey and analysis of multi-robot coordination. International Journal of Advanced Robotic Systems 10(12), 399 (2013)
2. Parker, L.E.: Path planning and motion coordination in multiple mobile robot teams. Encyclopedia of complexity and system science pp. 5783–5800 (2009)

3. Hoy, M., Matveev, A.S., Savkin, A.V.: Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. Robotica 33(3), 463–497 (2015)
4. Bertrand, S., Marzat, J., Piet-Lahanier, H., Kahn, A., Rochefort, Y.: MPC strategies for cooperative guidance of autonomous vehicles. AerospaceLab Journal (8), 1–18 (2014)
5. LaValle, S.M.: Planning algorithms. Cambridge university press (2006)
6. Švestka, P., Overmars, M.H.: Coordinated path planning for multiple robots. Robotics and autonomous systems 23(3), 125–152 (1998)
7. Yu, J., LaValle, S.M.: Planning optimal paths for multiple robots on graphs. In: 2013 IEEE International Conference on Robotics and Automation. pp. 3612–3617. IEEE (2013)
8. LaValle, S.M., Hutchinson, S.A.: Optimal motion planning for multiple robots having independent goals. IEEE Transactions on Robotics and Automation 14(6), 912–925 (1998)
9. Van Den Berg, J.P., Overmars, M.H.: Prioritized motion planning for multiple robots. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 430–435. IEEE (2005)
10. Zheng, T., Liu, D., Wang, P.: Priority based dynamic multiple robot path planning. In: International Conference on Autonomous Robots and Agents. Massey University, New Zealand (2004)
11. Kala, R.: Rapidly exploring random graphs: motion planning of multiple mobile robots. Advanced Robotics 27(14), 1113–1122 (2013)
12. van Den Berg, J., Snoeyink, J., Lin, M.C., Manocha, D.: Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In: Robotics: Science and systems. vol. 2, pp. 2–3 (2009)
13. Sánchez, G., Latombe, J.C.: On delaying collision checking in PRM planning: Application to multi-robot coordination. The International Journal of Robotics Research 21(1), 5–26 (2002)
14. Saha, I., Ramaithitima, R., Kumar, V., Pappas, G.J., Seshia, S.A.: Implan: Scalable incremental motion planning for multi-robot systems. In: 2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS). pp. 1–10. IEEE (2016)
15. Gregoire, J.: Priority-based coordination of mobile robots. Ph.D. thesis, Ecole Nationale Supérieure des Mines de Paris (2014)
16. Čáp, M., Novák, P., Kleiner, A., Seleckỳ, M.: Prioritized planning algorithms for trajectory coordination of multiple mobile robots. IEEE Transactions on Automation Science and Engineering 12(3), 835–849 (2015)
17. Siméon, T., Leroy, S., Lauumond, J.P.: Path coordination for multiple mobile robots: A resolution-complete algorithm. IEEE Transactions on Robotics and Automation 18(1), 42–49 (2002)
18. Bae, H., Kim, G., Kim, J., Qian, D., Lee, S.: Multi-robot path planning method using reinforcement learning. Applied sciences 9(15) (2019)
19. Wen, S., Wen, Z., Zhang, D., Zhang, H., Wang, T.: A multi-robot path-planning algorithm for autonomous navigation using meta-reinforcement learning based on transfer learning. Applied Soft Computing 110 (2021)
20. Das, P.K., Jena, P.K.: Multi-robot path planning using improved particle swarm optimization algorithm through novel evolutionary operators. Applied Soft Computing 92 (2020)
21. Tang, B., Xiang, K., Pang, M., Zhanxia, Z.: Multi-robot path planning using an improved self-adaptive particle swarm optimization. International Journal of Advanced Robotic Systems 17(5) (2020)
22. Li, Q., Gama, F., Ribeiro, A., Prorok, A.: Graph neural networks for decentralized multi-robot path planning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 11785–11792 (2020)
23. Han, S.D., Yu, J.: Effective heuristics for multi-robot path planning in warehouse environments. In: IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS). pp. 10–12 (2019)