Yakovlev., K., Andreychuk., A., Rybecký., T., and Kulich., M. (2020). **On the Application of Safe-Interval Path Planning to a Variant of the Pickup and Delivery Problem**. *Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO,*, 521–528 https://doi.org/10.5220/0009888905210528.

On the Application of Prioritized Safe-interval Path Planning with Kinematic Constraints to the Single-shot Pickup and Delivery Problem

Konstantin Yakovlev[1,2][a], Anton Andreychuk[1,3][b], Tomáš Rybecký[4][c] and Miroslav Kulich[4][d]

[1] *Federal Reseach Center for Computer Science and Control of Russian Academy of Sciences, Russia*

[2] *Moscow Institute of Physics and Technology, Russia*

[3] *Peoples' Friendship University of Russia (RUDN), Russia*

[4] *Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University, Czech Republic*

yakovlev@isa.ru, andreychuk@mail.com, Tomas.Rybecky@cvut.cz,kulich@cvut.cz

Keywords: Multi-robot Systems, Safe-interval Path Planning, Multi-agent Path Finding, Automated Warehouses.

Abstract: In this paper, we address the multi-agent pickup and delivery problem, a variant of multi-agent path finding. Specifically, we decouple the problem into two parts: task allocation and path planning. We employ the any-angle safe-interval path planning algorithm introduced in our recent work and study the performance of several task allocation strategies. Furthermore, the proposed approach has been integrated into a control system to verify its feasibility in deployment on real robots. A key part of the system is a visual localization system which is based on the detection of unique artificial markers placed in the working environment. The conducted experiments show that generated plans can be safely executed on a real system.

1 INTRODUCTION

Multi-robot systems are gaining much attention recently as they can effectively be used for search-and-rescue (Kulich et al., 2017), exploration (Faigl and Kulich, 2013), logistics (Wurman et al., 2008). The latter is a domain where robotic systems are already widespread, especially when it comes to the automated warehouses where the fleet of mobile robots replace human workers in order to increase the throughput, see (Roodbergen and Vis, 2009) for an overview.

One of the core problems in the context of automated warehouses is to ensure safe navigation of multiple robots w.r.t. both static (shelves, sorting stations etc.) and dynamic (other robots) obstacles. This problem is commonly tackled by introducing a centralized controller that plans collision-free trajectories beforehand and assuming that robots will follow them accurately enough. Thus the problem boils down to multi-robot path planning, which is known to be an NP-hard problem even in the simplest cases, e.g. when the disk-shaped robots move with constant speed among polygonal obstacles (Spirakis and Yap, 1984).

[a] https://orcid.org/0000-0002-4377-321X
[b] https://orcid.org/0000-0001-5320-4603
[c] https://orcid.org/0000-0002-5019-9260
[d] https://orcid.org/0000-0002-0997-5889

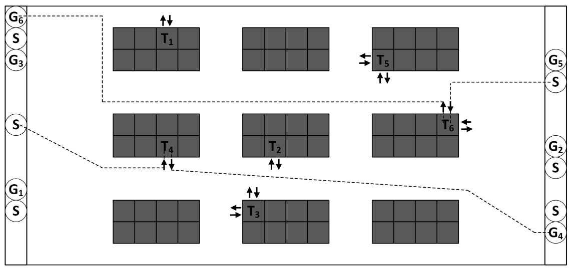

Figure 1: Multi-agent pickup and delivery instance. Storage areas are depicted in gray. They contain pallets (T) that have to be delivered to the sorting stations (G). Robots initial locations are marked with S. Arrows show the locations from which robots can go under a pallet to lift it. Two types of paths are shown by dashed lines: the one that contains only cardinal moves and the one that contains any-angle moves.

A discretized version of the problem is typically solved when the robots are confined to a graph and allowed to move only following the edges of this graph. Still, even the discretized version of the problem, often named multi-agent path finding (MAPF), is NP-hard to solve optimally for a range of critical objective functions, such as the *makespan* – the time by which the last agent reaches its goal, and the *flowtime* – the sum of execution times across the agents (Yu and LaValle, 2016). Indeed the planners that are able to find cost-optimal solutions for both of these objectives exist, but they scale poorly to a large number of robots.

From a practical point of view, it is reasonable to trade-off optimality for the runtime. A widespread approach to do this is to rely on prioritized planning (PP) (Erdmann and Lozano-Pérez, 1987), when the solver plans individual trajectories of the robots sequentially one after the other in some fixed order. This speeds up the search significantly as at each iteration previously planned trajectories are considered to be fixed. The main drawback is that prioritized planning is incomplete. However, for a large class of the MAPF problems, i.e. the ones that involve planning in the *well-formed infrastructures*, PP guarantees to find a solution (if one exists) (Čáp et al., 2015). Coincidentally, the warehouse environments are designed to obey the rules of the well-formed infrastructures in the vast majority of cases, so the prioritized planning fits very well to this domain.

Another advantage of the prioritized planning is that it can rely on the advanced single-agent path-finding algorithms, e.g. Safe interval path-planning (SIPP) (Phillips and Likhachev, 2011), that allow lifting numerous limiting assumptions, often adopted by the optimal MAPF solvers, e.g. wait actions of the uniform duration.

In this work, we extend previous research on prioritized multi-agent path planning with continuous time and kinematic constraints by considering the application of this approach to a variant of multi-agent pickup and delivery (MAPD) problem arising in the context of automated warehouses. In the studied setting robots have to pick up goods that are distributed over the specified zone and deliver them to the sorting stations. Thus, a combination of task allocation and multi-agent path-finding is needed to accomplish the mission. We suggest and evaluate a range of heuristics for task allocation. For path planning, we use a state-of-the-art prioritized planner – AA-SIPP(m) (Yakovlev et al., 2019) relying on its flexibility to produce different types of trajectories, i.e. the one that contain any-angle moves and the one that allows only movements into the cardinal direction.

We analyze the efficiency of the suggested approach in a simulation, where we run experiments with up to 164 robots. We also evaluate how the constructed plans can be executed on the real-word differential drive robots, commonly used in robotics research – Turtlebots 2 (Open Source Robotics Foundation, Inc., 2020). Unlike numerous works that rely on the external centralized navigation system, we show that the trajectories produced by the considered algorithm are accurately executed by Turtlebots relying on the local vision-based navigation system.

2 RELATED WORK

In the recent decade, a range of prominent multi-agent path finding algorithms has been proposed. Search-based approaches, such as as A*+ID+OD (Standley, 2010), M* (Wagner and Choset, 2011), CBS (Sharon et al., 2015) are typically aimed at finding optimal solutions. They do not scale well to the problems involving a large number of agents, though their sub-optimal variants, e.g. ECBS (Barer et al., 2014), mitigate this issue to a certain extent. When optimality is sacrificed, polynomial-time algorithms can be proposed, that treat multi-agent path planning as a pebble motion problem. Push-and-rotate (de Wilde et al., 2013) is a prominent algorithm of this kind. Its main drawback is that it assumes a sequential movement of pebbles (robots). thus the cost of the solution is very high in practice. In (Kulich et al., 2019) an extension to this algorithm was proposed that supports parallel movements. Overall, most of the aforementioned algorithms assume discrete time, uniform-cost transitions and do not take kinematic constraints into account. Most recent algorithms, introduced in (Cohen et al., 2019; Andreychuk et al., 2019; Hvězda et al., 2018a; Hvězda et al., 2019), lift these assumptions but are very computationally intensive.

Prioritized planning (Erdmann and Lozano-Pérez, 1987) is an appealing alternative, when kinematic constraints should be taken into account, as it is computationally less burdensome, scales well and is complete under certain conditions that often hold in practice (Čáp et al., 2015). In (Andreychuk et al., 2019) a road-map based planner supporting different moving speeds was suggested, in (Yakovlev and Andreychuk, 2017) grid-based planner capable of handling any-angle moves was proposed. The latter was extended in (Yakovlev et al., 2019) to handle kinematic constraints such as agents' size, rotation and translation speed etc. This work builds on the latter algorithm.

Among the works that study the application of multi-agent path finding algorithms to the problems arising in the context of automated warehouses, one can name the following. In (Ma et al., 2017) a lifelong variant of the multi-agent path finding problem with uniform-cost actions was investigated. In the subsequent work (Ma et al., 2019) it was extended to handle different moving speeds of the agents. However, they were allowed to move only in cardinal directions on a grid. In (Ma and Koenig, 2016; Hönig et al., 2018) both target assignment and path planning for teams of agents were considered. Time was discretized in these works, while we assume a continuous timeline.

3 PROBLEM STATEMENT

Consider a set of n robots, $1,2,...,n$, operating in the rectangular 2D workspace, W, discretized to a regular grid. Each robot is modelled as a disk of a predefined radius, r_i, and its state at any moment of time is a tuple (x,y,θ), where x, y are the coordinates, and θ is the heading angle. Robots are initially positioned at the centers of the grid cells. The following actions constitute a robot's action set: wait, rotate, move (translate). Robots are allowed to wait and rotate only at the centers of grid cells. Moving is allowed from the center of one cell to the center of the other following the straight line connecting them, hence – translate. The duration of rotate and move actions is defined by the endpoint configurations of a robot. E.g. the duration of a translate action is the distance between the move's endpoints divided by the robot's translation speed. Inertial effects are neglected, i.e. robots are assumed to accelerate/decelerate instantaneously. The duration of the wait action is arbitrary, that is a robot can wait for *any* amount of time at the center of a grid cell. Thus the timeline is continuous.

The plan for a robot is a mapping from time to workspace: $\pi : [0,\infty) \rightarrow W$. It can also be represented as a sequence of the timed actions: $\pi = (a_1,t_1),(a_2,t_2),...,(a_k,t_k)$, s.t. $t_{j+1} - t_j = dur(a_j)$, where (a_j,t_j) represents an action – either rotate, move or wait, – that starts at time moment t_j and dur stands for the duration of an action. The cost of a plan is its duration: $c(\pi) = t_k + dur(a_k)$. When a robot finishes its plan it's assumed to stay at the last location until the mission ends. This dummy stay-at-target action does not contribute to a plan's cost.

Two plans are called collision free *iff* the robots that follow them never collide. Formally, π_i and p_j are collision free *iff* $\nexists t : dist(\pi_i(t),\pi_j(t)) < r_i + r_j$. A set of plans, $\Pi = \pi_1,...,\pi_n$, one for each robot, is called collision free if any pair of the constituent plans is collision-free. Cost for Π can be defined as either the *makespan*: $C(\Pi) = \max\{c(\pi_1),...,c(\pi_n)\}$, or the *flowtime*: $C(\Pi) = c(\pi_1) + ... + c(\pi_n)$.

Two types of designated areas are present in the workspace: sorting stations and storage areas. Technically, each area is a composition of adjacent grid cells. These cells are assumed to be distributed in such a manner that they form a *well-formed infrastructure* (Čáp et al., 2015). The later means that there always exists a path for a robot between any of those cells that avoids collision with other robots in case they are positioned in the other sorting/storage cells. An example of the well-formed infrastructure that resembles the one used in our experiments is shown in Fig. 1.

Cells that belong to the storage areas are considered to contain pallets with goods (one pallet per cell). Hence the robots can not move through them. However, to load a pallet, a robot must roll under a pallet from a free cell that is a cardinally-adjacent to the storage cell. n storage cells are distinguished and it's assumed that they contain pallets that need to be carried to the sorting stations. Each pallet has to be delivered to a particular sorting station, that is there is a one-to-one binding between the cells containing pallets and the sorting-stations cells. We will call the former the sub-goals, and the latter - the goals.

The multi-agent pickup and delivery (MAPD) problem we are considering in this work can now be formulated as follows. Given n starting locations, sub-goals and goals the task is to find n collision-free plans (one for each robot) from start to sub-goal to goal. It is not required to solve the problem optimally w.r.t. flowtime (or makespan). However, lower-cost solutions are, obviously, preferable.

4 METHOD

We suggest the following decoupled, two-step approach to solve the considered MAPD problem. At the first step, the sub-goals are distributed among the robots, then, at the second step, collision-free plans are constructed by a prioritized planner. By solving assignment and planning problems independently, we, obviously, can not provide optimality of the resultant solution. On the other hand, such an approach leads to a faster runtime, which is critical for the real-world applications. We discuss both steps in details next.

4.1 Task Allocation

Indeed there exist algorithms that solve the assignment problem optimally, e.g. the Hungarian method (Kuhn, 1956). However, these methods typically rely on the knowledge of the cost associated with each allocation. In our scenario to get this cost one actually needs to solve the multi-agent pathfinding problem for all n robots and sub-goals (pallets), which is computationally expensive. Instead, we suggest relying on the following greedy approach to assign sub-goals to robots. We iterate over the robots and at each step assign a sub-goal to the current robot by the virtue of a select function, `select`$(s_i,l_1,...,l_n)$. The input of the latter is the start position of the current robot and all the sub-goals' positions. The output is a sub-goal assignment, i.e. l_j that is assigned to robot i. We suggest the following `select` functions.

Basic implementation of a select function may simply return random sub-goal out of the sub-goals that have not been assigned so far. Indeed, such a strategy does not take the cost of the resultant plan into account. A more intelligent approach to implement `select` is to compute the Euclidean distances between all of the unassigned sub-goals and the current robot and choose the sub-goal with the minimal computed distance. Intuitively, by following this approach, we minimize the travel time of a robot to a sub-goal. However, in this case, static obstacles, i.e. the cells that constitute the storage area, are not taken into account. To mitigate this issue a more involved implementation of a select function can be proposed that relies on invoking a path planning algorithm that finds the shortest path between the start location of the robot and each of the sub-goals. Then the sub-goal with the minimal computed cost is selected. Indeed, this approach is more computationally expensive, compared to the previous ones. On the other hand, it's likely to be more accurate in estimating the actual time needed for each robot to reach a respective sub-goal and, as a result, the overall mission completion time is likely to be lowered down. The experiments that we carried out confirm this hypothesis.

4.2 Multi-agent Path Planning

To find collision-free plans from the robots' start locations to the assigned sub-goals (pallets) and then to the corresponding goal locations (sorting stations) we suggest using the prioritized planning (PP) approach as it is known to work fast and scale well to a large number of robots (up to hundreds).

In PP, each robot is assigned a priority first and then plans are constructed sequentially in accordance with the imposed priority ordering. Different ways to assign priorities can be proposed. One of the common ways to do so is based on computing the distances between the robots and their goals. It is known from previous research that assigning higher priorities to the robots with lower distances positively influences the flowtime, and doing the same for the robots with higher distances – the makespan (Andreychuk and Yakovlev, 2018). When time permits, one can also try running the planning algorithm with a range of various assignments in an attempt to decrease the cost of the resultant solution (Bennewitz et al., 2002). A more sophisticated approach is presented in (Hvězda et al., 2018b), where priorities are dynamically recomputed based on how the robot influences the trajectories of already planned robots.

When priorities are fixed, a planner finds a collision-free plan for each agent one by one. Once the plan for agent i is found it's considered to be fixed, and the agents $i+1$, $i+2$, ... n have to avoid collision with i by modifying their own plans, not ith. In general, this approach does not provide completeness guarantees. However, for the well-formed infrastructures, PP is guaranteed to find a solution, if one exists, provided that the individual planner explicitly avoids start locations of all robots (Čáp et al., 2015). In this work, we consider only well-formed infrastructures, thus in our implementation, we, indeed, follow this rule.

The task of finding a valid plan for a robot that has to avoid collisions with previously planned robots can be solved by any planner that takes time dimension into account. In the simplest case, when time is discretized and only cardinal translations and rotation on 90° are allowed, A* (Hart et al., 1968) can be adopted to solve the problem. The search state, in this case, should consist of the configuration-time pairs. At each step of the search, the planner should reason whether the adjacent configurations are reachable in the next time step, i.e. whether such transitions do not conflict with the given set of constraints induced by the higher-priority robots. An alternative to staying in the same configuration, i.e. to wait, should also be considered. Obviously, searching in such a search-state is a computationally intensive task as the number of states is huge.

In (Silver, 2005) a modification of the described approach, named WHCA*, was suggested that introduces a time-window and considers time dimension only when planning inside this window. A more advanced approach, known as Safe-interval path planning (SIPP), was suggested by Phillips and Likhachev in (Phillips and Likhachev, 2011). The idea of SIPP is to group for each configuration the time steps at which no collision happens and to make search-nodes out of these intervals. I.e. each search node is a tuple $\langle cfg, [t_b, t_e] \rangle$, where cfg is the configuration of the robot and $[t_b, t_e]$ – is the safe interval for that configuration. Endpoints of the interval are computed taking the trajectories of the dynamic obstacles (high-priority robots) into account. SIPP significantly reduces search effort as now the number of the search-states corresponding to a single configuration is proportional not to the total number of time-steps, but to the number of dynamic obstacles that interfere with that configuration.

A nice property of SIPP is that it can be naturally extended to handle non-discretized, i.e. continuous, time as the endpoints of the intervals are not restricted by the algorithm to be integers. This, in turn, provides a pathway to handle any-angle moves (Yakovlev and Andreychuk, 2017), translations and rotations of arbitrary duration etc. In this work, we employ one of the latest modifications of SIPP, suggested in (Yakovlev et al., 2019), that supports planning with a wide range of kinematic constraints by explicitly reasoning about the velocities of the agents, their sizes, etc.

5 CONTROL SYSTEM

The proposed planning approach has been integrated into a control system to allow testing and evaluation of the planner with real robots. Its first component is the planner which is given a set of assignments for the agents, a map and a configuration. The planner then provides a set of timed plans for the agents – robots. These plans are processed by the *FleetControl* module that transforms them into messages to be sent to the robots at a specified time. The *FleetControl* handles the communication with the robots. The robots localize themselves by fusing data from a camera and odometry. Based on the knowledge of their current location, they navigate to a target given by the last command.

5.1 FleetControl system

The FleetControl consists of four blocks (Main, Server, Timer and the Data) and its architecture is indicated in Fig. 2. It consists of four blocks (*Main*, *Server*, *Timer*, and *Data*).

The *Main* block loads the file with plans for the agents and directly transforms them to timed commands for robots. The format of all commands is the same. They contain the start and target locations and orientations, execution time and an ordinal number. The expected execution time of the command is used to maintain average speed during motion control, which is vital to fulfill the plan as expected by the collision-avoiding planner.

The time when a command should be sent is determined by the duration of the preceding commands. The commands for each robot are put in a queue that is sorted by their execution start. The queues and other state variables are stored in *Data*.

After the plans are loaded, the *Server* and *Timer* threads are started. *Main* now only initiates checks of remaining commands by notifying *Server*. After all commands are depleted, other threads are stopped, and the program quits.

The *Timer* thread maintains a given frequency of 100 Hz in increasing the timestamp of the system. The timestamp is also stored in Data and is then used to determine the time to send new commands to robots.

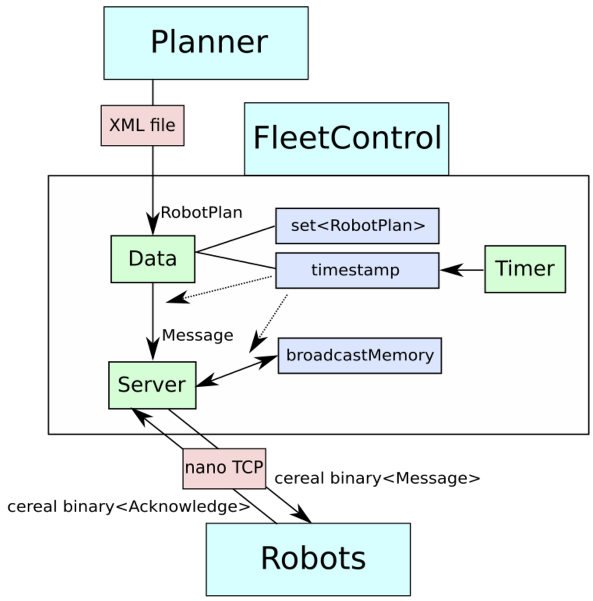

Figure 2: FleetControl scheme.

The *Server* thread finally maintains the communication with the robots, which is performed over the TCP network protocol. The TCP network is realized making use `nanomsg` library (nanomsg - A C socket library aiming to make the networking layer fast, scalable and easy to use, 2020). *Server* maintains broadcast of new commands to robots, as well as the reception when robots announce the completion of a command or confirm the reception of a new one.

The time-ordered queue of commands for robots is also checked in *Server* and if the time reaches the point when the current first command has to be sent, it is sent to the correct robot. The commands are serialized to a binary format to minimize the size of messages sent over the network. The `cereal` library is used for this purpose (Grant and Voorhies, 2020).

Correct reception of commands by robots is ensured by a broadcast memory and timer mechanism. This remembers the last command sent to each robot and if the robot does not confirm its reception within a set time, the command is sent again.

5.2 Robot localization and control

The robots are entirely driven by ROS (Quigley et al., 2009). Several standard ROS packages together with the ones designed for the described scenario are used. The central ROS node is used for navigation and communication with *FleetControl*. It listens on a TCP client socket to any new commands from *FleetControl*. If the previous command was completed, meaning the robot is expecting a new one, it stores the new deserialized command for execution and acknowledges the reception to *FleetControl*. If a command is completed, another message is sent to *FleetControl*.

The localization of the robots in the map is carried out in two ways. The odometry is used for continuous localization, although it is prone to inaccuracy and growing errors. To provide absolute localization, robots are equipped with cameras, and a system based on detection of fiducial markers is employed. Specifically, a set of visually detectable and unique markers - AprilTags (Olson, 2011; Wang and Olson, 2016) is placed in the environment and the positions of these markers are stored. Whenever some marker is detected in a camera image and its ID is recognized, the absolute position of the robot is retrieved from the stored absolute position of the marker, a position of the marker in the image, and the relative position of the camera to the robot. Decoding of AprilTag information from a camera image is done the `apriltag_ros package` (http://wiki.ros.org/apriltag_ros).

The navigator/controller part receives updates of relative robot position from the translator node and determines the current absolute robot location and orientation in the map. This information is used as the robot pose when no tag is visible and thus AprilTag-based localization does not provide data.

From the robot position, the regulation errors are computed – the deviation in the desired orientation to the current goal and the expected and real traveled distance based on expected average speed. A proportional controller is used to control robot speed:

$$u_{speed}(t) = K_P e_{speed}(t), \quad (1)$$

while a PI controller is employed to control robot steering:

$$u_{steer}(t) = K_P e_{steer}(t) + K_I \int_0^t e_{steer}(t')dt'. \quad (2)$$

In the equations, K_P and K_I denote the proportional and integral coefficients respectively, e_x are the regulation errors. The sum of the resulting values $u_x(t)$ is the input value for the differential engines of the TurtleBot2. The controllers ensure navigation towards the goal with the defined precision and its attainment in the desired time.

6 EXPERIMENTAL EVALUATION

6.1 Simulation Experiments

For simulation experiments, we used a warehouse grid map from the Moving AI repository, commonly

Figure 3: TurtleBots operating in an environment with AprilTags.

used in MAPF community (Stern et al., 2019). This map, `warehouse-10-20-10-2-2`, is constituted of 170×84 cells out of which 4 000 represent storage areas. These storage areas are organized into the racks of size 10×2. There are 200 racks in the central part of the map, the passages between the racks have a width of 2 cells. We generated 100 scenarios, each containing 164 unique start, sub-goal and goal locations. Start and goal were placed randomly in the right and left column of the grid that represent sorting areas. Sub-goals were randomly distributed across the racks. To generate instances involving n number of robots we took first n start-subgoal-goal triplets out of the generated scenarios. Overall, 100 different instances for the number of agents 32, 64, 128 and 164 were generated and used in the evaluation.

We considered two different action models for the robots. The first one allowed only cardinal moves, while the second one allowed any-angle moves on a grid as well. In both action models translation speed of all robots was set to be one cell per time unit, e.g. it took $\sqrt{2}$ time units to perform a translation between diagonally adjacent cells. Rotation speed was set to be π radians per time unit. In each experiment we measured the following indicators: algorithm's runtime, resultant flowtime and makespan.

Fig. 4 depicts the results of the first series of the experiments when only cardinal moves were allowed. Three lines on the three left-most plots (blue, orange, grey) correspond to different task allocation strategies: `random`, when the sub-goals were assigned randomly to the robots; `without obs`, when the sub-goals were assigned relying on computation of the Euclidean distance between them and the robots; `with obs`, when sub-goals assignment involved path planning with A* algorithm. In the latter case, the task allocation phase may take a significant amount of time, so we measured the breakdown between the

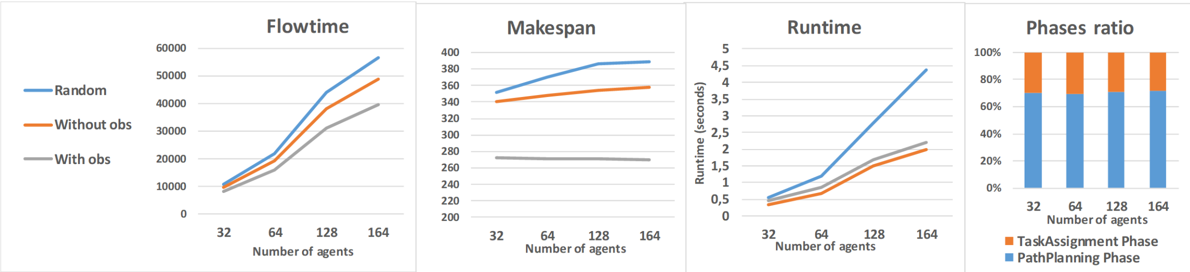

Figure 4: Experimental results when cardinal-only moves were allowed.

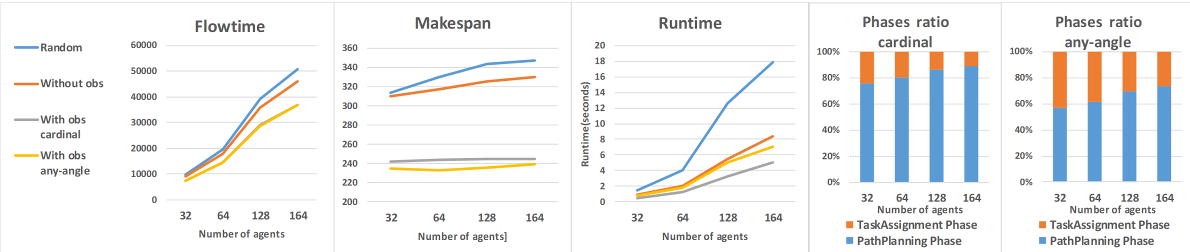

Figure 5: Experimental results when any-angle moves were allowed.

time spent on assigning sub-goal and time spent for path finding – it is shown in Fig. 4 on the right.

As one can see, the task assignment strategy has a significant impact on all of the estimated indicators. For examplem the difference in flowtime between `random` and `without obs` strategy is 15% when 164 agents are involved, the difference in between `random` and `with obs` is even more articulated – up to 30% for 164 agents. Trends for the makespan are similar. In terms of runtime, the best results were achieved by `without obs`. However, the difference between `without obs` and `with obs` in runtime is not significant (10% on average). Please also note, that a significant portion of the algorithm's runtime is spent on task allocation when `with obs` strategy is utilized (see the right-most plot on Fig. 4.)

Fig. 5 depicts the results of the second series of the experiments when any-angle moves were allowed. In this case, we evaluated one more task allocation strategy in addition to the previously used ones. It is marked as `with obs any-angle` and stands for assigning sub-goals to the robots relying on finding any-angle paths with Theta* (Nash et al., 2007) not A*. As one can see the most trends are similar to the ones that were obtained during the first experiment: the more involved strategy to assign sub-goals is used – the better results are in terms of flowtime and makespan. Interestingly the algorithm's runtime is now lower for the `with obs any-angle` and `with obs cardinal` compared to `without obs`. It means that investing time in path planning at the stage of task allocation (instead of using a closed-loop formula to assign the sub-goals) pays off in this setting. Two right plots in Fig. 5 with the runtime breakdown show that any-angle planning for task assignment is, indeed, more time-consuming.

Comparing the results of both series of experiments, the one can note that the difference in flowtime and makespan between the considered action models is about 10%. This is due to the presence of the open areas to the left and right of the storage racks in which robots can freely move into arbitrary directions shortening their overall paths.

6.2 Verification on Real Robots

The ability of the proposed planner integrated to control a fleet of real robots has been verified in a laboratory setup. The robots used were Kobuki TurtleBot2 equipped with an Intel NUC PC and Orbbec Astra camera. The camera provides 1280×960 RGB images at the frequency of 10FPS. Intrinsic/extrinsic parameters of the camera have been identified, and camera distortion was removed before processing by the AprilTag detector.

The laboratory setup serving for the demonstrator has been populated with a grid consisting of AprilTags (type Tag36h11) on the floor as shown in Fig. 3. The unique tags were placed equidistantly with 70 cm gaps, and their positions were stored in an XML map.

The planner was given a situation with five agents in a narrow area as depicted in Fig. 6. Two pairs of agents had switch their positions while the fifth agent had to cross the area perpendicularly. The crossing agent received the highest priority and quickly completed its plan. Also one of the position-exchanging

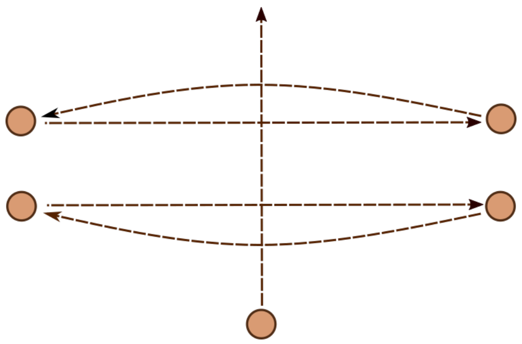

Figure 6: The experiment setup.

agents got a fully straight plan and reached the goal quickly. The other three agents had their routes split into two parts that ensured spatial clearance from the others. The speed of the agents was as expected by the planner, so they were able to keep safe distance without being able to perceive their peers. The location and orientation deviations did not exceed the set thresholds during the entire experiment, which were set to 5cm for the position error tolerance and 0.02 radians for the orientation error tolerance.

Generally, the proposed planner proved to be a flexible and versatile tool in practice. After appropriate tuning, it was capable of providing robust solutions that were safely executed by real robots.

7 CONCLUSIONS

In this work, we examined a variant of the multi-agent pickup and delivery problem that often arises in the context of automated warehouses. We suggested and evaluated a range of simple yet efficient task allocation strategies paired with state-of-the-art multi-agent path finding algorithm that takes kinematic constraints of the robots into account. We also conducted experiments on real-robots that rely only on a local vision-based localization system for navigation and plan execution. One of the appealing directions of future research is examining a lifelong variant of the problem (when sub-goal and goal appear on an ongoing basis). Another one is conducting experiments with a large number of real robots in various settings.

ACKNOWLEDGEMENTS

The work of Tomáš Rybecký and Miroslav Kulich has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 688117 and by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15 003/0000470). Konstantin Yakovlev and Anton Andreychuk were supported by the Russian Foundation for Basic Research (project 18-37-20032). Anton Andreychuk is also supported by the "RUDN University Program 5-100".

REFERENCES

Andreychuk, A. and Yakovlev, K. (2018). Two techniques that enhance the performance of multi-robot prioritized path planning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, pages 2177–2179.

Andreychuk, A., Yakovlev, K., Atzmon, D., and Stern, R. (2019). Multi-agent pathfinding with continuous time. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 39–45.

Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS 2014)*, pages 19–27.

Bennewitz, M., Burgard, W., and Thrun, S. (2002). Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and autonomous systems*, 41(2):89–99.

Čáp, M., Novák, P., Kleiner, A., and Selecký, M. (2015). Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering*, 12(3):835–849.

Cohen, L., Uras, T., Kumar, T. S., and Koenig, S. (2019). Optimal and bounded-suboptimal multi-agent motion planning. In *Proceedings of the 12th Symposium on Combinatorial Search (SoCS 2019)*.

de Wilde, B., ter Mors, A. W., and Witteveen, C. (2013). Push and rotate: cooperative multi-agent path planning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, pages 87–94.

Erdmann, M. and Lozano-Pérez, T. (1987). On multiple moving objects. *Algorithmica*, 2:1419–1424.

Faigl, J. and Kulich, M. (2013). On determination of goal candidates in frontier-based multi-robot exploration. In *Proceedings of 6th European Conference on Mobile Robots*, Barcelona, ES. Institut de Robotica i Informatica Industrial.

Grant, W. S. and Voorhies, R. (accessed April 6, 2020). cereal - a C++11 library for serialization. http://uscilab.github.io/cereal/.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hönig, W., Kiesel, S., Tinka, A., Durham, J. W., and Ayanian, N. (2018). Conflict-based search with optimal task assignment. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2018)*, pages 757–765.

Hvězda, J., Kulich, M., and Přeučil, L. (2018a). Improved discrete rrt for coordinated multi-robot planning. In *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics - (Volume 2)*, Madeira, PT. SciTePress.

Hvězda, J., Kulich, M., and Přeučil, L. (2019). *On Randomized Searching for Multi-robot Coordination*, pages 364–383. Lecture Notes in Electrical Engineering. Springer, Cham, CH.

Hvězda, J., Rybecký, T., Kulich, M., and Přeučil, L. (2018b). Context-aware route planning for automated warehouses. In *Proceedings of 2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui. IEEE Intelligent Transportation Systems Society.

Kuhn, H. W. (1956). Variants of the hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258.

Kulich, M., Novák, T., and Přeucil, L. (2019). Push, stop, and replan: An application of pebble motion on graphs to planning in automated warehouses. In *Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC 2019)*, pages 4456–4463.

Kulich, M., Přeučil, L., and Miranda Bront, J. J. (2017). On multi-robot search for a stationary object. In *Proceedings of ECMR 2017*, Marseille, FR. IEEE.

Ma, H., Hönig, W., Kumar, T. K. S., Ayanian, N., and Koenig, S. (2019). Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, page in press.

Ma, H. and Koenig, S. (2016). Optimal target assignment and path finding for teams of agents. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, pages 1144–1152.

Ma, H., Li, J., Kumar, T. S., and Koenig, S. (2017). Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2017)*, pages 837–845.

nanomsg - A C socket library aiming to make the networking layer fast, scalable and easy to use (accessed April 6, 2020). https://nanomsg.org/.

Nash, A., Daniel, K., Koenig, S., and Felner, A. (2007). Theta*: Any-angle path planning on grids. In *Proceedings of The 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 1177–1183.

Olson, E. (2011). AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE.

Open Source Robotics Foundation, Inc. (accessed April 6, 2020). Turtlebort2. https://www.turtlebot.com/turtlebot2/.

Phillips, M. and Likhachev, M. (2011). SIPP: Safe interval path planning for dynamic environments. In *Proceedings of The 2011 IEEE International Conference on Robotics and Automation (ICRA 2011)*, pages 5628–5635.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European journal of operational research*, 194(2):343–362.

Sharon, G., Stern, R., Felner, A., and Sturtevant., N. R. (2015). Conflict-based search for optimal multiagent path finding. *Artificial Intelligence Journal*, 218:40–66.

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2005)*, pages 117–122.

Spirakis, P. and Yap, C.-K. (1984). Strong NP-hardness of moving many discs. *Information Processing Letters*, 19(1):55–59.

Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 173–178.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K. S., Boyarski, E., and Bartak, R. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158.

Wagner, G. and Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 3260–3267.

Wang, J. and Olson, E. (2016). AprilTag 2: Efficient and robust fiducial detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–19.

Yakovlev, K. and Andreychuk, A. (2017). Any-angle pathfinding for multiple agents based on SIPP algorithm. In *Proceedings of The 27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pages 586–593.

Yakovlev, K., Andreychuk, A., and Vorobyev, V. (2019). Prioritized multi-agent path finding for differential drive robots. In *Proceedings of The 2019 European Conference on Mobile Robots (ECMR 2019)*, pages 1–6.

Yu, J. and LaValle, S. M. (2016). Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177.