2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Macau, China, November 4-8, 2019

Multi Robot Route Planning (MRRP): Extended Spatial-Temporal Prioritized Planning

Benjamin Binder[1], Florian Beck[2], Felix König[3] and Markus Bader[3]

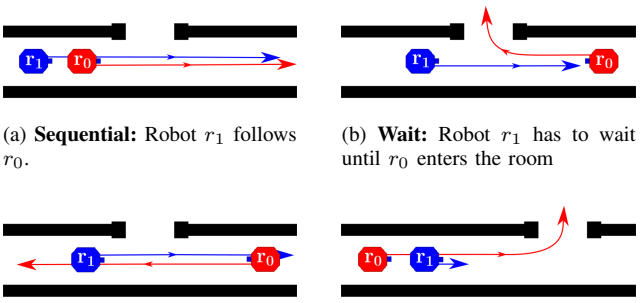
***Abstract*— Autonomous vehicles are, in contrast to classical automated guided vehicles (AGVs), less predictable in their behavior and drive time. Therefore, the issue of how to efficiently control these vehicles arises, because autonomous agents need to be coordinated and not controlled, to give autonomous behaviors and actions space. The scientific contribution of this paper is a novel approach, based on prioritized planning to target this issue as well as an open source framework for evaluation and comparison. Prioritized planning has the disadvantage of being neither optimal nor complete, however, it has the advantage of being computationally feasible. This work utilizes prioritized planning to significantly increase the set of feasible scenarios through collision prevention: by locally finding alternative routes and adding them to the search graph. The paper clearly formulates the extensions needed and delineates the approach's limits, as it is neither optimal nor complete. More importantly, however, our method calculates routes for each vehicle with inter-vehicle synchronization, enabling vehicles to execute the plan in a distributed fashion without centralized control, thereby allowing autonomous behavior. Finally, results are verified by comparing our Multi Robot Router (MRR) proposed in this work to classical approaches. The software developed as well as the test sets are publicly available for ROS and the simulation environment.**

I. Introduction

Automated guided vehicles (AGVs) are commonly used in warehouses and on assembly lines to transport goods [1]. This situation requires a dead-lock free coordination of these AGVs. Currently there are two main approaches used. One is a centralized approach, which can be optimal and complete, but complexity rapidly increases along with the number of robots. The other method follows a decentralized strategy, which splits the problem into multiple smaller tasks. Whereby every robot's route is individually planned and merged to a global plan. The disadvantage to this is that this algorithm is neither optimal nor complete. A well-known decentralized approach is prioritized planning [2], which plans every robot's path sequentially, taking care of already-existing paths.

In Fig. 1, four scenarios are shown which are not solvable by the prioritized planning approach without spatial-temporal planning; however, they require solving if robots move within narrow environments such as hallways or warehouse environments as shown in the Fig. 1e. The fist scenario Fig. 1a *sequential* is solvable by introducing spatial-temporal constraints but the scenarios such *wait*, *avoid* and *push* (Fig. 1b-c) do need some extra extensions discussed later.

This paper formalizes an approach which utilizes the

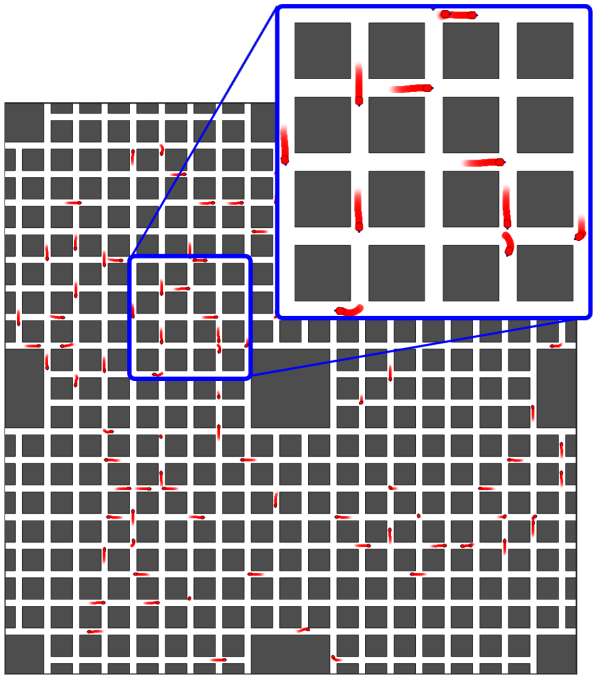


(a) **Sequential:** Robot r_1 follows r_0.

(b) **Wait:** Robot r_1 has to wait until r_0 enters the room

(c) **Avoid at Crossing:** To switch positions, Robot r_1 has to avoid r_0 in the door above the hallway

(d) **Push:** r_1 blocks r_0's path. Therefore, r_1 is pushed into the room above the hallway.



(e) **Simulation:** 100 robots simulated in Stage executing a distributed plan without collisions and dead-locks

Fig. 1. Four sample scenarios the proposed router is able to solve as well as a simulated test scenario to show the salability of our approach.

The research leading to these results has received funding from the Austrian Research Promotion Agency (FFG) according to grant agreement 855409 (AutonomousFleet) and 854865 (TransportBuddy).

[1]Benjamin Binder is with DS AUTOMOTION GmbH, Lunzerstraße 60, 4030 Linz, Austria b.binder@ds-automotion.com

[2]Florian Beck is with the Automation and Control Institute, Vienna University of Technology, Vienna 1040, Austria beck@acin.tuwien.ac.at

[3]Felix König and Markus Bader are with the Institute of Computer Aided Automation, Vienna University of Technology, Treitlstr. 1-3/4. Floor/E183-1, 1040 Vienna, Austria [firstname.lastname]@tuwien.ac.at

prioritized planning approach in order to manage such scenarios. As the approach targets the computation of routes for multiple robots, one can imagine a graph search such as A-star on a pixel map. However, typical industrial systems with AGVs have handcrafted route segments like rails which are used as a basis for guiding vehicles. This simplifies the search graph, but our approach is also able to recompute pixel maps in order to derive segments on its own using a Voronoi distillation, as proposed in [3], and a segmentation algorithm. The segmentation algorithm splits the Voronoi path generated into segments used as vertices for the graph. Therefore our framework can work with either handcrafted route segments or with auto-generated segments.

As one can see in Fig. 1, each scenario is time-dependent. Thus the proposed Multi Robot Router (MRR) takes time into account and is, therefore, able to have robots wait in order to find feasible paths. This is done by introducing a collision resolver which is triggered when a higher-priority robot blocks a lower-priority one's path. The collision resolver tries to find a path that avoids the other robot's route, taking spatial and temporal constraints into account.

This paper is organized as follows: Related Work discusses papers published which deal with multi-robot planning problems. Section III describes the MRR, and in Results, the approach is compared to other approaches in terms of computation time and number of scenarios solved.

II. Related Work

The paper [4] describes an approach to organizing a large number of robots while allowing every robot to deviate from its given route. This allows for centralized coordination of all robots, preventing dead-locks and improving efficiency, and gives every vehicle sufficient autonomy to react flexibly to dynamic environments as needed in the project Transport-Buddy [5]. Those ideas are used to reduce the complexity of the MRR proposed here.

Other papers such as [6], [7] and [8] describe approaches to improving the planning algorithm directly.

[6] describes a prioritized planning algorithm which is complete for specific structures, but because time and robot priorities are not taken into consideration, it has disadvantages in random environments.

[7], on the other hand, describes an algorithm using a spatio-temporal A-star algorithm taking time into consideration by adding waiting steps. Therefore, a great improvement in the number of solvable scenarios can be observed.

[8] describes algorithms for finding optimal priority schedules for a number of robots. Furthermore, an algorithm is proposed here which swaps selected priorities on the schedule and retriggers the planning process. Another significant improvement in scenarios solved can be observed as a result of this.

All of these algorithms significantly improve upon the behavior of prioritized planning, but in tight environments, as shown in Fig. 1, these approaches lead to sub-optimal results. Therefore, an approach was developed for such scenarios. This approach is detailed in the next chapter.

III. Approach

The scenarios depicted in Fig. 1 are scenarios which are not solvable using commonly-used prioritized planning approaches. In this chapter, the algorithm for solving scenarios like those shown in Fig. 1 is discussed. First, the terminology used as well as the preconditions for the planner are explained. Second, the data structure the router uses to find a route is explained. Third, the main part of the planner is discussed and, finally, extensions for improving the planner's results and the execution of the routes generated are discussed.

A. Terminology

The framework used is based on the ideas presented in [4] and has a *central multi robot route planner* as well as a *local path and trajectory controller* on each vehicle. The central multi robot route planner can be seen as the Multi Robot Router (MRR) presented in this paper and we will refer to the local path and trajectory controller just as *local controller*. The MRR computes dead lock free *routes* for each vehicle and a finished successful *plan* is a named *routing table*. Routes differ from classic paths in that routes contain synchronization points in the form of *check points*. Classical paths consist of *way points* and can be traversed by means of a trajectory controller such as a PID or MPC [9]. *Route candidates* are routes without synchronization points calculated by a Single Robot Router (SRR). All vehicles are publishing, at low frequency ($\sim$ 1Hz), their progress on the assigned route to realize synchronization between vehicles. Reaching a check point means entering an area described by a *vertex*, this is needed to enable autonomous behaviours like in [5]. The MRR uses a *graph* for planning which consists of vertices $V_0 ... V_n$ to describe the accessible environment. Every vertex can be mapped to an area on a map describing the environment and a *path segment* composed of way points traverses each area to connecting vertices as graph. A vertex also links to the occupation times for the corresponding area by vehicles. The graph in this implementation is generated by computing Voronoi distillation but it is also possible to define the graph by hand – (A program within the tuw_muli_robot framework is provided to process CAD files such as DXF to handcraft graphs.).

B. Preconditions

As the technology exists, automated guided vehicles will be replaced by autonomous vehicles in the future if the surroundings allow for it. Autonomous vehicles have the capability of localizing themselves in an environment and are able to find locally-optimal paths to a given destination [4]. This reduces the complexity of the planner, because exact paths can be reduced to way points and check points placed every few meters or where needed, e.g. at crossings. Therefore, the length of segments described by two way points mainly depends on three properties: (i) the environment, (ii) the vehicle's ability to successfully drive autonomously between check points and (iii) the MRR. To simplify the planner's task and prevent collisions, every segment needs

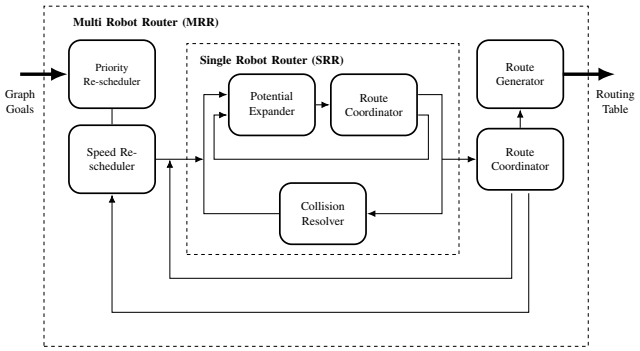


Fig. 2. Structure of the MRR, which includes a SRR and some extensions like a priority and speed rescheduler, root coordinator and route generator.

to be at least as long as the largest robot's radius. Therefore, a vehicle can only occupy space on a vertex and its predecessors or successors at a specific point in time, defined by a sufficiently-high resolution for the proposed MRR. A plan which is successful at dealing with the scenarios described in Fig. 1 must take time into consideration in order to synchronize vehicles' executions of commands. Therefore, the weights for the associated search graphs are initialized using the length of each segment. In this work we are proposing the use of check points to allow a vehicle to enter the next segment only after synchronization with dependent vehicles. This facilitates the use of autonomous vehicles in an evenly-distributed fashion.

C. Framework and Structure

The MRR builds upon a SRR, as shown in Fig. 2, which finds routes for every robot, avoiding vertices which are simultaneously occupied by other higher-priority vehicles. These route candidates are computed using an A-star algorithm [2] on a given search graph which relates to given or computed path segments in an environment. The potential of the algorithm is denoted by the time a vehicle spends on the traversed vertices. This time is calculated using the segment lengths and the vehicle's speed.

To restrict the A-star algorithm to spatial and temporal non-overlapping paths with higher-priority vehicles, a route coordinator is used. The route coordinator allows the algorithm to expand if the vertex is not occupied by another robot at the time span requested by the A-star algorithm.

If the A-star algorithm cannot expand into a vertex due to a potential collision with another robot, the issue is passed on to a collision resolver, as explained in Section III-D in order to avoid collisions by extending the search graph.

Additionally, a collision and a speed resolver, described in Section III-E, are added to the MRR to alter the priority and maximum speeds of certain robots if no routing table is found. This behavior is controlled by the route coordinator, since it holds a list of all collisions occurred during the routing process.

After a valid list of route candidates has been determined, it has to be converted into a synchronized list of routes (a routing table). This is done by using a route generator,

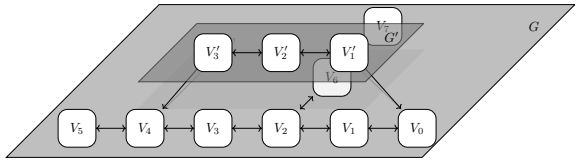


Fig. 3. Example of a three-dimensional graph extension. Vertices $V'_{1,2,3}$ are already assigned with different potentials as compared to their parent nodes ($V_{1,2,3}$).

as described in Section IV-A. A more detailed structure description can be found in [10].

D. Collision Resolver

In Fig. 1b a scenario is depicted in which Robot r_1 has to wait until Robot r_0 has entered the room at the top of the image. This scenario can be resolved by expanding the graph with edges of higher potential weight which relates to a longer travel time/or causes the vehicle to wait. In our framework a module called *potential expander* does this graph modifications but then the issue rises at which edge and how much should be the time to wait increased.

Assume that r_0 is higher priority and its path is fixed. When the expander starts processing the graph, it has no idea if r_1 should enter the room before r_0 or after it, because at this point in time the potential expander has no idea where the goal of r_1 is located.

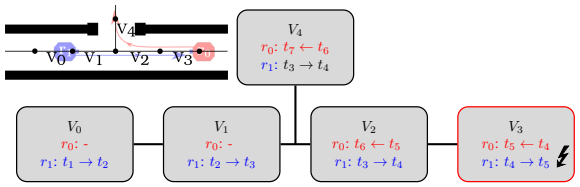
Therefore, the collision resolver is allowed to copy vertices onto another graph which is then added to the main search graph, thus providing a solution to both scenarios (those of entering the room before r_0 and after it). Additionally, the collision resolver assigns certain vertices with the appropriate amount of time to wait. Such a graph extension can be seen in Fig. 3. The vertex used for the location of the wait and the length thereof is determined by using three different strategies. These three strategies are used in combination to find a path for the robot currently being planned and all higher-priority ones.

1) Wait: The wait strategy is used in order to have robots wait until another one has passed a specific area. This is done by the collision resolver backtracking along already-expanded vertices until a crossing with a vacant vertex is found.

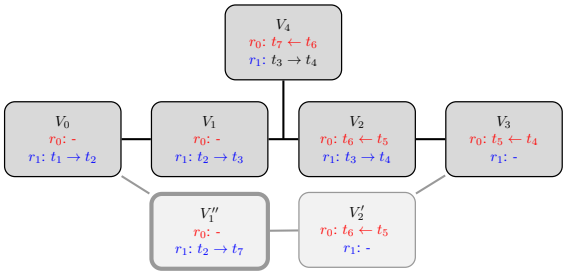
In Fig. 1b, a wait scenario is shown. In it, r_0 plans its route candidate first and its route is thus fixed. The potential expander then tries to find a route candidate for r_1.

Fig. 4a shows the initial attempt by the potential expander at finding a route candidate covering a route over V_2 and V_3. In each vertex, the time t_x a robot spends on this segment is depicted. These time steps are saved with a slight overlap in order to prevent robots from swapping places where it is not permitted.

As one can see, r_1's path is expanded until a collision is detected in V_3. At this point, the collision resolver is triggered. It starts to backtrack along expanded vertices until a vertex is found which is not occupied by r_0. The vertex



(a) The collision found in V_3 when expanding the potential of r_1.



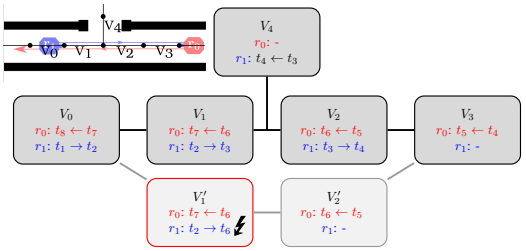
(b) Graph extension (V_1', V_2') for the wait scenario

Fig. 4. Graph view with the extension graph G Fig. 4 corresponding to the **wait scenario** Fig 1b.

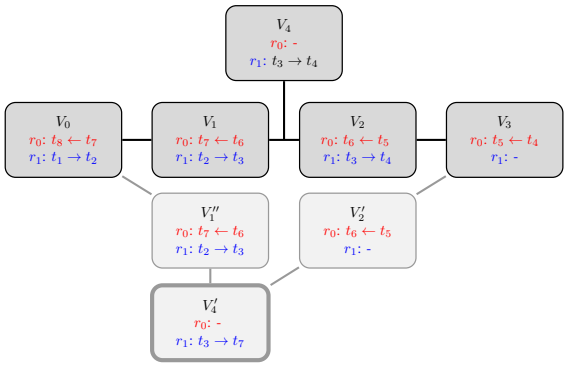
is selected as a wait vertex and delineated with the time the robot has to wait until the other robot has passed the area defined by the vertex. Finally, all vertices that have been occupied are copied onto a new path. Only the necessary preceding and succeeding vertices are connected, and the wait vertex is added to the potential expander's queue. This is shown in Fig. 4b.

2) Avoid: This strategy covers scenarios in which a lower-priority robot has to diverge from its route in order to avoid higher-priority ones. Such a scenario is shown in Fig. 1c: both robots move along a hallway in opposite directions. Again, r_1's route is planned first, thus its route is fixed. In Fig. 5a, the result after applying the backtracking strategy is shown. As one can see, backtracking would fail, because r_0 moves along r_1's route in the opposite direction, which would lead to another collision in V_1'. To solve this issue, the collision resolver is allowed to choose any free segment at a crossing for the wait. This generates the extension graph shown in Fig. 5b covering V_1'', V_4' and V_2'. Vertex V_4' is again added to the potential expander's queue. Moreover, in order to generate additional, more efficient (shorter-path) solutions, the collision resolver continues backtracking until the starting vertex is found. This is also done if, due to other robots' paths, the wait vertex determined via the avoidance strategy is of no use in finding a route to the destination.

3) Push: The push strategy is used for scenarios where lower-priority robots are sitting at an end point or a waiting point on their path and blocking a higher-priority robot's path. An example is shown in Fig. 1d, where r_1 starts right in front of a crossing, blocking r_0's path. Another scenario is if a robot avoids another one at a crossing and a third robot attempts to progress through the waiting spot. As before, the graph view of the scenario in Fig. 1d is shown in Fig. 6. r_1 is higher-priority and has its route candidate fixed, and the blue robot's path has to be determined. In none of the cases covered by the push strategy is backtracking possible. A solution can only be found by moving along



(a) Collision found in V_1' in the extension graph generated by the avoid strategy.



(b) Graph extension (V_1'', V_4', V_2').

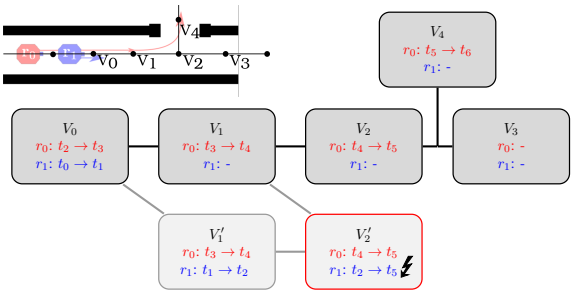
Fig. 5. Graph view with the extension graph corresponding to the **avoid scenario** shown in Fig 1c.

a higher-priority path until a waiting spot is found. For the scenario shown in Fig. 6a, the first step is taken by expanding into a neighboring vertex. This is done by copying the wait/destination vertex and adding the successor of this vertex as a new wait vertex. The vertex copied is used to move away from the other robot, and the original vertex is used to move backwards on the path. If a crossing is detected, the collision resolver is allowed to expand into all free branches of the crossing. In Fig. 6b, the final extension graph can be seen, in which a waiting spot has been found in V_3', which is then added to the potential expander's queue. The algorithm combining these strategies is shown in Fig. 7. As one can see the collision resolver starts expanding into all free branches while backtracking until a solution is found. Hereby *ExpandToFreespace* expands into a branch of a crossing and saves the extension graph like shown in Fig. 6, representing the push and avoid scenario. The wait scenario is covered by the backtracking attempt executed each loop iteration.

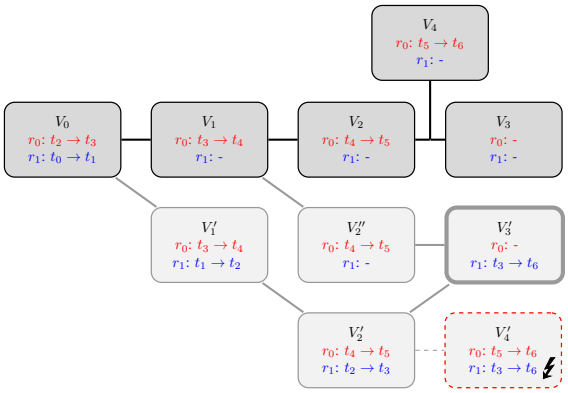
E. Extensions to the Router

In addition to the scenarios in Fig. 1, there are two more possible scenarios, shown in Fig. 8.

1) Priority Re-scheduling: In Fig. 8a a scenario can be seen in which robot r_0 has a higher priority and blocks robot r_1's route. This cannot be solved using the proposed strategies for the SRR. Instead, the MRR has to be altered. This can be done by adding an improved version of the priority re-scheduler, similar to the one in [8]. The paper [8] proposes randomly exchanging robot priorities. Thus, it could be that two robots which are not involved in the



(a) First attempt to move away from the higher-priority robot.

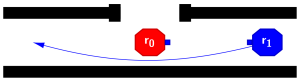


(b) Solution to the scenario: adding two new vertices to the graph.

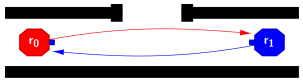
Fig. 6. Graph view with the extension graph corresponding to the **push scenario** shown in Figure 1d.

procedure EXTENDGRAPH(Graph G, PathVertex v_p)
  Graph G_{EB} ▷ empty graph for backtracked vertices
  repeat
    for all $v_n \in$ NotExpandedNeighbours(G, v_p) **do**
      Graph $G_E \Leftarrow$ ExpandToFreespace(G,v_n)
      if IsValid(G_E) **then**
        AddGraphExtension(G,G_E)
        AddVertexToQueue(Front(G_E))
      end if
    end for
    if IsStart(v_p) **or** IsGoal(v_p) **then**
      return
    end if
    if BacktrackingPossible(v_p) **then**
      $v_p \Leftarrow$ BacktrackOnce(v_p)
      AddVertex(G_{EB}, v_p)
      if !IsCollision(v_p) **and**
      !IsGoalOfCollidingRobot(Successor(v_p)) **then**
        AddGraphExtension(G,G_{EB})
        AddVertexToQueue(Front(G_{EB}))
        return
      end if
    end if
  end repeat
end procedure

Fig. 7. The collision resolver algorithm. The algorithm starts extending into free branches and backtracking simultaneously until a solution is found or the start of the route candidate is reached.



(a) A higher-priority robot blocks a lower-priority one's path.



(b) Two robots are equidistant from the crossing and want to switch places.

Fig. 8. Two scenarios which are not solvable with standard prioritized planning.

problem's priorities get exchanged. In selecting the two robots whose priorities are to be exchanged, we can take advantage of the fact that we know from the path coordinator which robots have had collisions. Obviously, one of those robots is the robot for which the SRR has failed to find a feasible route candidates. We refer to this robot as r_f. The second robot selected is the one with the most route intersections concerning r_f. It is referred to as r_c. To avoid ending up in a loop, two robots' priorities can only be exchanged once.

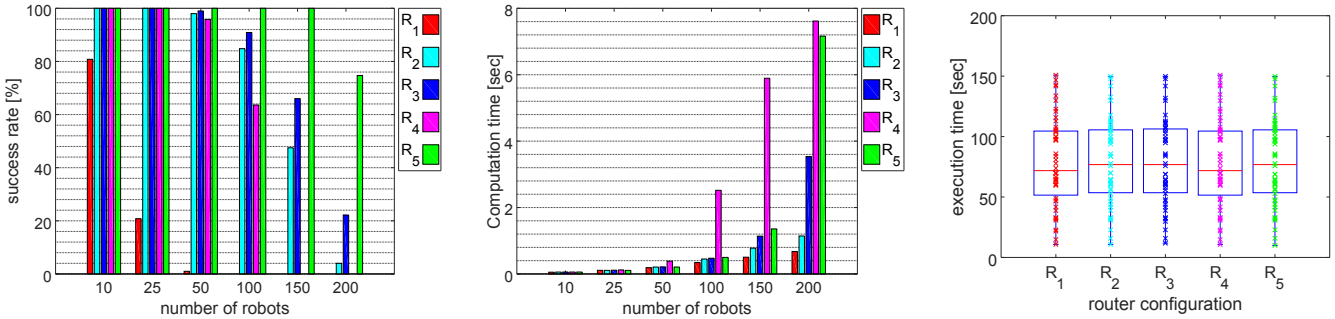
2) Speed Re-scheduling: In Fig. 8b a scenario is depicted in which two robots are equidistant from a crossing. In this scenario, the priority re-scheduler will not find a solution to this scenario because neither of the robots has enough time to avoid the other one. The approach to this situation is to reduce the maximum speed of one robot. For example, if r_0's maximum speed is halved, r_1 has enough time to move into the space above to avoid r_0.

Similar to in priority re-scheduling, r_c can be identified and its maximum speed reduced. Since r_c already exists, when the SRR attempts to plan r_f's route, it has higher priority and its route is planned prior to that of r_c with reduced speed, which gives r_f more freedom to find a path. Here as well r_c's maximum speed is only allowed to be modified for a certain number of retries in order to avoid loops. In our approach these two extensions are only used if the SRR fails to find a plan for each vehicle. However, these two extensions could be used as well if a valid plan is found to improve the overall execution time of the routing table.

IV. RESULTS

A. Execution

Since the proposed MRR takes time into account for the route candidates it generates, one can imagine that these paths have to be executed synchronously in order to avoid dead-locks. This can be done by generating a velocity profile for every robot which must be closely adhered to. The advantages to this approach are that an energy-optimal profile can be calculated and no communication among robots is needed. However, if a robot fleet works in dynamic environments, it is most likely that a robot fails to adhere to a velocity profile and causes eventually a dead-lock. Furthermore, the model of every robot must be known. Another idea is to include path preconditions in segments to allow robots only to move through segments in a certain order. This has the disadvantage of necessitating communication among robots, but if the routing table generated is dead-lock free, the path execution will not cause any further dead-locks. To combine



(a) Success rate of a planner configuration to find a valid routing table.

(b) Computation time of a planner configuration to find a valid routing table.

(c) Travel time of 100 vehicles evaluated using the simulation on one specific test set.

Fig. 9. Comparison between different multi robot router configurations.

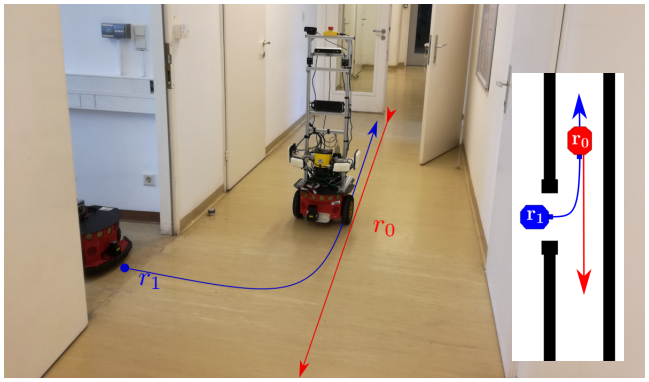


Fig. 10. Real world test scenario. Robot r_1 has to wait until robot r_0 has vacated the space in front of the door.

the best of both worlds, a route generator is used which receives input in the form of a list of route candidates and produces a velocity profile as well as a routing table. In order to move energy optimally and dead-lock free, every robot can now adhere to a velocity profile until a precondition of its path is impaired. The path generator creates this routing table by simulating the route candidates generated and simultaneously assigning a robot order to a segment. This order and the length of the routes is used to calculate a velocity profile and a routing table.

To verify the MRR, it is compared to other commonly-used prioritized planning approaches. The approaches used for comparison are integrated into the MRR to become independent of implementation. Therefore we where able to compare five meaningful router configurations ($R_1 ... R_5$) of the implemented underlying spatial-temporal prioritized planner by activating options such as Priory Re-scheduling (PR), Speed re-scheduling (SR) and Collision Resolving (CR).

- R_1 : PR = off, SR = off, CR = off
- R_2 : PR = off, SR = off, CR = on
- R_3 : PR = off, SR = on, CR = on (SRR)
- R_4 : PR = on, SR = off, CR = off
- R_5 : PR = on, SR = on, CR = on (MRR)

The environment, a 80m x 80m warehouse is simulated using Stage [11] and shown in Fig. 1e. Since this environment is organized as a grid, a solution to a given problem exist as long as one segment is empty and all start- and endpoints don't overlap. The underlying route segments and the related graph vertices are auto-generated, using a Voronoi distillation. Each test case was executed 100 times with generated random goals for each vehicle on different start location. We reused the this goal lists to evaluate the planner with the different configurations as stated on a Intel Core i7-6820 @2.7GHz. Each configuration was test using 10, 25, 50, 100, 150 and 200 robots. The stage environment configuration as well as the realted 500 genarted goal lists are public available on github within the tuw_multi_robot project.

Fig. 9a shows a comparison of the number of test cases solved. The most simplest router configuration R_1 start to fail even with 10 robots this configuration allows spatial-temporal routing but since all re-scheduler are deactivated a invalid plan can not be corrected. R_2 allows the re-planning of invalid plans by the use of the collision resolver and performs therefore better. The configuration R_3 is the same as used internally for the proposed SRR and uses speed re-scheduling. The configuration R_4 relates to the configuration proposed in [8] as one can see it is computational expensive and not as successful as R_3. R_5 relates to our proposed MRR framework. It uses all discussed extensions which results in a decent computation time to find valid dead-lock free routing tables as shown in Fig. 9b in less than eight seconds. In general it can be seen here that a standard prioritized planner doesn't perform very well with a large number of robots in dense environments. To solve this, waiting steps were included in the planner, which results in a significant improvement. The proposed SRR can improve upon these results even further. The Fig. 9c shows the travel time of all vehicles related to a specific data set which happened to be solvable for all five configurations with 100 vehicles. The distribution shows that no optimization is implemented and the results are similar if a path exists only the priority re-scheduling causes some minor derivations.

The MRR significantly outperforms the other planners. This is because the SRR finds more solutions than the simple planner and therefore the MRR needs less reschedule iterations in order to determine a plan. In conclusion, the

proposed SRR significantly outperforms the planner with no re-scheduling strategy, but it is not able to outperform those with a re-scheduling strategy. Thus, priority and speed re-scheduling can be added, which helps to find results to more test scenarios and greatly improves computation time.

To verify the planner's proper function in real-world scenarios, a test was performed using two Pioneer P3DX robots coordinated by the MRR (shown in Fig. 10). These robots are controlled using the approach from [12] and [9].

V. Conclusion and Further Work

The aim of this work was to coordinate a large number of robots and derive a plan for a distributed execution without centralized control. This is done utilizing the prioritized planning approach and introducing a collision resolver, which extends the search graph with solutions to time-dependent issues. As one can see in Section IV, this planner outperforms current approaches, but it is not able to outperform planners with re-scheduling strategies if the SRR is used alone. To compensate for this, the MRR includes a speed and a priority re-scheduling strategy, which drastically improves the planner's behavior. Additionally, tests are done with two real robots, including a controller from [9] to demonstrate a real-world scenario. Since we are using a prioritized planning approach, this planner is not yet complete. Therefore, one possible avenue of investigation would be to switch out the collision resolver with a complete planner, which starts looking for a solution to multiple robots in the local vicinity and increases its search radius after a failed attempt. Another unresolved issue is to synchronize the algorithm to increase computation speed. This could, for example, be done similarly to the approach presented in the paper [6]. The speed re-scheduler works well as a supplement, solving specific test cases. Therefore, it would be of interest to examine whether there is a heuristic for the speed scheduler that works well enough to replace the priority scheduler or waiting steps. As one can see, there are many areas for further investigation of the MRR which could assist in finding a fast and complete approach to prioritized planning. The framework developed is public available with samples data, simulation and demos for ROS[13] and documented on the related ROS-Wiki page `http://wiki.ros.org/tuw_multi_robot`.

References

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," in *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence - Volume 2*, IAAI'07, pp. 1752–1759, AAAI Press, 2007.

[2] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[3] Q. Wang, M. Langerwisch, and B. Wagner, "Wide Range Global Path Planning for a Large Number of Networked Mobile Robots based on Generalized Voronoi Diagrams," *IFAC Proceedings Volumes*, vol. 46, no. 29, pp. 107 – 112, 2013.

[4] M. Bader, A. Richtsfeld, M. Suchi, G. Todoran, W. Holl, and M. Vincze, "Balancing Centralised Control with Vehicle Autonomy in AGV Systems for Industrial Acceptance," in *Proceeding of the Eleventh International Conference on Autonomic and Autonomous Systems (ICAS 2015)*, 2015.

[5] M. Bader, G. Todoran, F. Beck, B. Binder, and K. Buchegger, "TransportBuddy: Autonomous Navigation in Human Accessible Spaces," in *Proceedings of 7th Transport Research Arena TRA 2018*, (Vienna, Austria), April 2018.

[6] M. Cap, P. Novak, A. Kleiner, and M. Selecky, "Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, pp. 835–849, July 2015.

[7] W. Wang and W.-B. Goh, "Multi-robot Path Planning with the Spatio-temporal A* Algorithm and Its Variants," in *Proceedings of the 10th International Conference on Advanced Agent Technology*, AAMAS'11, (Berlin, Heidelberg), pp. 313–329, 2012.

[8] M. Bennewitz, W. Burgard, and S. Thrun, "Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems," in *ICRA*, 2001.

[9] H. G. Todoran and M. Bader, "Just-in-Time Emergency Trajectories: A Formulation Towards Safety in Autonomous Navigation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[10] B. Binder, "Spatio-temporal prioritized planning," Master's thesis, Vienna University of Technology, 2018.

[11] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, pp. 189–208, Dec 2008.

[12] H. G. Todoran and M. Bader, "Expressive navigation and local path-planning of independent steering autonomous systems," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4742–4749, Oct 2016.

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