Towards Autonomous Intralogistics: A Testbed Environment for the Coordination of a Robotic Fleet

Fabian Maas genannt Bermpohl and Andreas Bresser
German Research Center for Artificial Intelligence
Bremen, Germany

Abstract

In this paper we present a testbed implementation, in which a production plant for airplane wings is simulated. The simulated environment can be used for the evaluation of different approaches to multi-agent task allocation and path finding. We outline the relation to the relevant problems of job-shop scheduling and multi-agent path finding and describe our approach towards a system architecture and a baseline implementation of the required software components for a multi-agent system to tackle these challenges.

Our approach facilitates a centralized database where the state of the production environment is consolidated (a so called "digital twin"), a high-level planner that continuously dispatches the available agents to tasks, and a multi-agent simulation to simulate the task execution. In general our system is designed in a modular fashion, which allows the developer to extend or replace the distinct components easily. The modular design also makes it possible to factor out the multi-robot simulation in favor of a real-world test environment with actual robotic agents.

We describe the relevant parameters that are tracked about the environment, the design of the planner, and the interactions between the planner and the agents, that are executing the plans autonomously. We present first experiments conducted within this environment as a starting point for a series of runs of experimentation.

Introduction

This work is set in a simulated industrial production site dedicated to the assembly of airplane wings. While manufacturing processes in other industrial sectors already rely heavily on the support by robotic systems and fleets (e.g., Enright and Wurman 2011), it still seems to be infeasible to adopt the same for the production of large and expensive parts in relatively small numbers.

In order to address this issue, our overarching goal is to develop techniques to increase the degree of automation and efficiency of the shopfloor intralogistics. To achieve this, self-driving automatic transportation agents are employed for the just-in-time delivery of wing parts to assembly stations. The assembly workers can then focus on the actual assembly tasks and spend less of their time waiting for components or collecting the required parts themselves.

While there are other measures that can also positively contribute to an increase in efficency of the production plant, our focus is put on the intralogistics. Within the last decades, much relevant progress has been made, especially towards automation of warehouses or distribution centers (Enright and Wurman 2011; Hvilshøj et al. 2012), that we would like to see applied to other branches of industry too. Improvements to the facility's layout, to the mechanical design of the parts, or improvements to the execution of the actual manipulation tasks are interesting topics on their own, but we see those outside the scope of this work. Rather do we notice that these are studied elsewhere (Kulturel-Konak 2007; Fujita 2002; Jefferson et al. 2013).

Instead, we here investigate the specific applicability of models and approaches to warehouse automation around the aspects of planning, scheduling and multi-agent pathfinding in the domain of airplane wing equipping and assembly. We notice that there is a trend of more robotic systems becoming commercially available, that can be used for automating a plant's intralogistics. At the same time however, the management of robotic fleets and their support for automatic task assignment are still underdeveloped, which often prevents a successful integration of these systems.

In order to let autonomous agents take actions considering the actual state of the production environment, a sufficient level of digitization of the plant is required, i.e., the development of a digital twin representation, similar or in extension to a manufacturing execution system (MES). In order to focus on the business logic first, which is the autonomous intralogistics to and from the assembly stations, we propose a modular abstraction layer as simple as possible around the digital twin of the production environment. Working with a simulated plant we can concentrate on the algorithmic challenges first, and blend out the acquisition of real-world data, the API of a production-scale MES and the integration of actual robot hardware for now.

Our approach shows some overlap to the one presented in Yao et al. 2018. Based in a warehouse setting, they demonstrate how a centralized management system automatically schedules a single replenishing action based on a sensor value, that is then carried out by a simple robotic system. Overall the setting is a relatively narrow, single-agent use case, while the scenario we propose represents a multi-agent system designed to continuously optimize for an optimal production rate.

Our approach has obviously areas of overlap with the goals and challenges posed by the Robocup Logistics League (Niemueller, Lakemeyer, and Ferrein 2015). In contrast to the setting represented there, for our use case we assume full observability and high-level action execution. That is, the level of abstraction largely hides the aspects of exploration, localization and part manipulation. This enables us to put more focus on the actual problems of planning, scheduling and agent coordination. We do acknowledge that these now hidden aspects play a significant role in any logistics task. However, given recent advances towards centralized tracking and positioning infrastructure in highly digitized factories, we believe these topics are no longer the major road blocks towards automatic factory intralogistics. Also, as discussed above, we would like to focus on the algorithmic challenges of the multi-agent coordination first and separately from the challenges introduced by interactions with actual robotics hardware.

Problem Description

System Resources

On a theoretical level the overall system is quite similar to an order picking scenario in a retail logistics distribution center. As will be shown it seems to be a simplified version form of it. Compared to the *Kiva system*[1] described in Enright and Wurman 2011, inventory takes on the shape of wings, wing movables and their respective support structures on the input side, and readily assembled wings on the output side. Due to the size of these components they are usually not stored in bins of inventory or order pods. Each one of them rather forms a unit to be transported individually, thereby making use of specific transportation support structures to enable mounting the different components on top of the transport robots efficiently. Assembly tasks can be represented as orders that are being worked on at multiple stations simultaneously. For each assembly station, there is only one assembly task going on at any time.

The required wing parts for the assembly need to be fetched from an induction station somewhere on the shop floor and delivered to the assembly station. This task can be perfomed by any agent from the fleet of transportation robots. Similar to the *OrderFetch* configuration (c.f., Enright and Wurman 2011), fully assembled wing modules are then scheduled to be picked up by a transportation agent again to be delivered to the shipping station. The overall system is to be coordinated in an intelligent way, in order to avoid agents blocking each other possibly indefinitely and to maximize assembly stations' utilization.

Optimization Problems

In Enright and Wurman 2011, the authors describe several *allocation problems*, which are different domain specific aspects that each can affect the overall performance of the system in different ways. These range from the allocation of orders to stations, of inventory pods to picking stations, to the allocation of missions to one or multiple transportation robots. While optimizing the system behavior we need to consider many factors including the deadlines for shipping, synergies of multiple items for different orders that can be picked from the same pod, spatial distances between storage locations and stations, the expected waiting time at the picking station and many more. The application domain facilitates temporal planning for multiple agents and concurrent task execution.

[1]*Kiva Systems* was rebranded to *Amazon Robotics* in 2015.

Some of these requirements can be translated in a straightforward way, others can be neglected, as they don't apply to the domain described above. This makes the following domain specific aspects the main subjects of our further consideration.

- Assembly stations need to be provided with wings to assemble from the list of orders.
- Robots need to be selected to fetch and deliver the required parts to the right assembly stations. Thereby the distance and the route a robot needs to travel in order to fulfill a transportation task needs to be considered. Especially as the robot will travel at a rather slow pace while mounted with a large wing part on top. Ideally the task execution will have to start before the assembly station is idle waiting for the part. If the part however arrives before the assembled part that was being worked on before has been taken away, it may lead to congestion and possibly delay the overall process.
- After the assembly is completed, a robot needs to fetch and deliver the wing module to the outgoing storage to make it ready for shipment. This needs to be prioritized, as arriving parts for the subsequent assembly task might block the pathways if it is being done too late.

Translating this into research topics, we can identify a *Job-Shop Scheduling Problem* (JSSP), intermingled with *Multi-Agent Path Finding* (MAPF). Also, the recently formulated *Simultaneous Task Allocation and Planning* (STAP) problem can be seen as a joint instance of the former standard problems (Schillinger, Bürger, and Dimarogonas 2018).

Despite JSSP and its variations haven been discussed for decades, transfer into practical applications still is difficult (Mohan, Lanka, and Rao 2019). Often the scenario is overlapped with instances of other (optimization) problems that affect the viability and performance of candidate solutions. Also the added complexity of variations with stochastic durations, deadlines etc. is often accounted for by relaxing other constraints to keep it tractable. While the methods proposed deliver optimal solutions in theory, real world applications can rarely be represented as a purely combinatorial optimization problem.

The interest in solutions to MAPF has grown in the past decade, and due to its NP-hard nature it is still not possible to find an optimal solution for non-trivial problem sizes in real time. Work is being done to find approaches that give approximate results within acceptable error-bounds (Sharon et al. 2015), and approaches that avoid replanning in the face of deviations from the original plan to minimize execution time (Ma et al. 2017).

Approaches that aim for simultaneous task assignment and path planning are already gaining traction (Bellingham

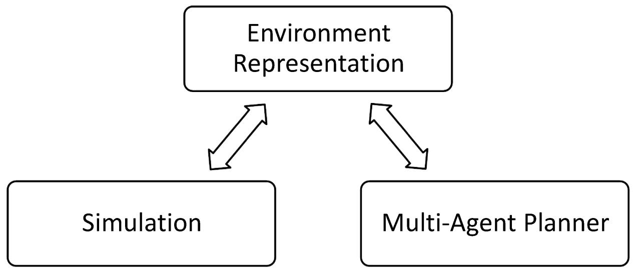

Figure 1: Modular components of the system architecture. Both the multi-agent simulation and the multi-agent planner directly interface only with the database used to track the state of the environment.

et al. 2003; Biswas, Anavatti, and Garratt 2017; Yang and Chakraborty 2019). Most of the time, when dealing with real world conditions, only approximate approaches seem viable, in order to keep computation times within a reasonable range.

Local, agent-based, reactive approaches for multi-robot navigation exist, that allow for limited coordination among the agents to help resolve conflict situations by mutual adjustment of the desired paths (van den Berg et al. 2011; Godoy et al. 2016). However, these approaches tend to end in deadlock configurations. Local optima might prevent agents to move out of their way in order to let another agent pass (e.g., in configurations like *Towers of Hanoi*) (Wang and Botea 2011; Janovský, Cáp, and Vokrínek 2014). Scenarios like this require a central authority that mediates the individual agents' paths in order to optimize the global plan, e.g. Le and Plaku 2017 or Pecora et al. 2018.

System Architecture

We implemented a modular testbed environment that allows us to tackle those research problems, both jointly and individually. In this testbed environment, three main components interact with each other continuously, that will be described in the following sections. There is a database to track the state of the environment, a multi-agent planner to assign tasks to agents based on the state of the environment and a 2D simulator for the task execution, which are cyclicly updating the database accordingly. The components and their interactions are depicted in Figure 1.

Environment Representation

Central part is the environment representation. Its role is to track the state of the environment and provide an interface for other modules, like visualizations or scripts that initialize, update or analyze the state of the environment.

We chose to build our implementation on top of a MongoDB database, because capabilities to stream changes to clients directly and its built-in support for representing GeoJSON and geospatial queries are readily available. However, we acknowledge that other database systems or even RDBMS might just work as good. In fact the data model already resembles a relational model and we even added an optional schema validation procedure to detect deviations from the models automatically for debugging purposes.

The database can be interfaced from Python code via an asynchronous API that provides the required domain-specific functionalities, e.g., regarding reading and updating properties of agents, their assigned tasks and the environment structure itself. Specific functions allow the planner to update the agents' task plans and others enable the simulation to query, execute and cycle to the next task, once the current goal has been completed. The data model allows the representation of several separate environments referred to via an identifier.

Each environment can contain agents, described by their pose and type, their current and subsequent tasks and the currently transported payload. The agent type can be selected from a number of predefined agent types, that can be configured separately. The agent type is used to specify the capabilities of the agent, like whether or not it can transport or assemble certain parts, and also to select the graphical model that is used in the visualization of the environment.

We currently distinguish the task types *idle*, *move-to*, *pick-up*, *put-down* and *assemble*. Each has an expected duration, which can be infinite. Tasks of type *move-to* also contain the target coordinates. Tasks of the types *pick-up*, *put-down* and *assemble* take the identifiers of the components that are supposed to be affected by the given action.

Tasks have certain preconditions. Only agents that are capable to move can execute a *move-to* task. For picking up, the agent needs to have the capability to carry that component, and it needs to be in its vicinity to pick it up. For putting down, the specified component needs to be in the payload list of the agent. For assembling, the agents needs to have the components in its payload list, and it needs to have the capability to assemble them.

The tasks that are assigned to an agent will be added to its task list. From there they will be executed one after another. When the agent's current task is completed it is moved to the task log, then the next task in the task list is assigned.

Within each environment an arbitrary number of stationary obstacles can be specified, to function as walls that block the agents' paths. Obstacles are defined as a polygon by the list of vertices along their contour path. Similarly areas for storage and assembly of parts can be defined as polygons. These indicate places where certain operations are performed, like assembly of wing modules or the supply of parts for the assembly.

For the simulation also supply and shipping of wings and wing components are modelled via so called *spawn* and *despawn points* for each component type. These are locations that either generate or consume components of the specified type, depending on whether or not components of the respective type are nearby. Available component types are also selected from a set of predefined types, that carry the information what parts are required to assemble them, or what modules they can be assembled to. Currently we only have three component types *wing*, *wing-movable*, which build into *wing-assembled*. Spawn points keep track of the number of components were supplied. Despawn points keep track of the number of wing modules that were shipped, that

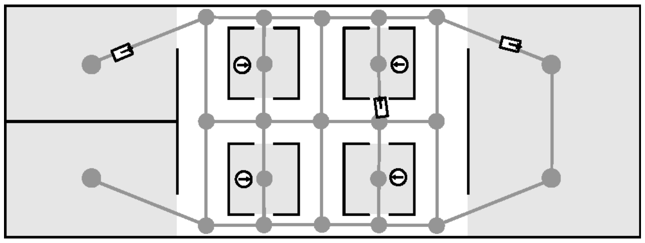

Figure 2: Visualization of an example environment configuration. Displayed are the areas for storage (left and right) and assembly (center), that are partially enclosed by obstacles (walls). Also a number of agents are shown (rectangles and circles with arrows indicating the orientation) and the edges and vertices of the navigation graph, that are used to support the agents' navigation.

can be used to calculate an average frequency of shipped modules.

Figure 2 shows a visualization of the most relevant aspects of the environment for a simple example configuration. A similar visualization is presented to a human operator by the web-based visualization frontend we developed, to monitor the environment state and the simulation progress. Updates to the state of the environment are continuously streamed to the frontend, which is thereby able to update the visualization instantly.

The frontend can also be used to edit an environment, i.e., it is possible to add or move agents, walls or areas in edit mode, or to update the navigation graph. When the simulation is then restarted, the updated configuration is used as the new starting point.

All coordinates in the database are three dimensional in *latitude*, *longitude* and *elevation*. However, most parts of the system currently assume two dimensional cartesian coordinates $p \in \mathbb{R}^2$ and orientations $\theta \in [-\pi, \pi]$. So far our experiments were structured in a way that this simplification was sufficient, but we think this can and needs to be extended at a later point.

Multi-Agent Simulation

We implemented a simulator that executes tasks and handles the coordination of the agent movement.

Task Handling The simulator needs to take care of the following key responsibilities during each iteration step.

1. Inspect the agent's current task and act accordingly:
   - If the agent has the *Idle* task, pause the agent.
   - If the agent has a *Move-To* task, initiate agent movement towards the goal position until the threshold on proximity is satisfied.
   - If the agent has any of the stationary durative tasks, i.e., *Pick-Up*, *Put-Down* or *Assemble*, pause the agent until the simulated duration of the task has expired. Also add the target components for the specific task to the agent's inventory as required. In case of an assembly task, create a new assembled component out of the provided parts.
2. If the agent's current task's termination criterion is fulfilled, terminate the current task and assign the next task in the agent's schedule. Thereby the end time of the former current task is set, before it is moved to the agent's task log.

At the end of each iteration all changes are transferred to the database for the other modules to operate on the updated state of the environment.

Agent Movement and Agent Coordination Agent movement is simulated using a differential drive kinematics model. In order to move towards their goals, they can adapt their velocity. Pose updates are calculated accordingly and communicated to the centralized environment representation.

As we assume robust obstacle avoidance to be provided by the robotic platforms that are employed, we currently omit the simulation of collision events. For the purpose of this work, we have integrated the collision-free agent navigation scheme provided by the RVO2 library (van den Berg et al. 2016). The theoretical background is described in Snape et al. 2010 and van den Berg et al. 2011.

The basic principle is the continuous adaptation of the agent's linear and angular velocity in order to avoid collisions with mobile and stationary obstacles along their predicted trajectories. Thereby each agent slightly adjusts its own velocity towards a value outside the so called *Velocity Obstacles* (c.f., Fiorini and Shiller 1998) the other objects represent. Depending on the parameterization of time step, temporal horizons and safety margins we can rely on a collision-free multi-agent navigation within the simulation.

We define a navigation graph within the environment in order to guide agents around static obstacles, i.e., around the walls surrounding the storage and assembly areas. When the assigned goal position for the agent's current task is not in direct line of sight to the agent, i.e., when an obstacle needs to be circumnavigated, it will use a graph search algorithm (here: Dijsktra) to find the shortest path along the navigation graph to the goal position. It will then use the vertices along this path as waypoints for the navigation towards the goal position. A probabilistic roadmap algorithm can be used to create the navigation graph (Kavraki et al. 1996).

Multi-Agent Planner

We implemented a planner software that takes care of assigning tasks to the robots. As a first integration test, and to deliver a baseline performance for the planning system, we defined a set of heuristics about the state of the environment, based on which tasks would be assigned to any idle agents capable of doing that specific task.

The heuristics we have defined are described in the following paragraphs.

- Within this system, the first priority is to fetch readily assembled wing modules from the assembly stations and deliver them to the shipping station, which is modeled as a despawn point. The despawn point would remove nearby components after a specified time and calculate a number of statistics about the rate of products it has received in the past. These statistics are useful for a later integration into a more sophisticated, possibly learning or self-adapting planning algorithm.

- The next priority is maintaining the supply of input parts for the assembly stations in order to minimize their idle time. Currently the heuristic as implemented makes the check about whether new parts for another assembly procedure are required based on the current idle status. However, it is possible to use a threshold on the duration until the station would be idle again, measured against the expected time of arrival of the required part for each transportation agent, to indicate this status. If there is a soon-to-be idle assembly station and there are soon-to-be idle transportation agents to deliver the required parts, new tasks are added to their task lists to fetch and deliver the parts to the respective assembly station.

Using this approach, we set up an experiment to compare the effect of different numbers of transportation agents on the overall system performance, which is described in the following section. In the future, we will integrate more sophisticated approaches to the planning and coordination of multiple agents. We are especially curious how these will perform in relation to the approach presented here.

Experiments

In order to demonstrate the system's basic functionality, we set up the test environment depicted in Figure 2 with four assembly stations and for each run different numbers of transportation agents. With this setting, we evaluate how the system performance, i.e., the production rate, is affected by the number of agents that are employed for the logistics to and from the assembly stations.

We generated multiple configurations for different numbers of transportation agents in an otherwise equal environment structure. The agents start out in a cluster in the center of the environment. We expect to observe the performance to increase up to a certain point, at which additional agents would block each other, instead of actually accelerating the logistics processes.

In order to receive the relevant figures for the comparison, we tracked the number of completed parts that were delivered to the despawn points. Via linear regression on the time series of accumulated delivered products, we calculated an average production rate accross several runs for each configuration. In Figure 3 this value is shown in relation to the number of available transportation agents. For this experiment, we repeated each configuration fifty times in order to obtain meaningful results. Error bars indicate the spread of the distribution of the results. We ran this experiment on a workstation computer, processing ten runs in parallel. Overall this took around 8 hours to compute.

As can be seen, it seems to be beneficial to have roughly twice the number of transportation agents to provide supplies to the assembly stations. In this configuration, from the relative durations of transportation times vs. duration of the

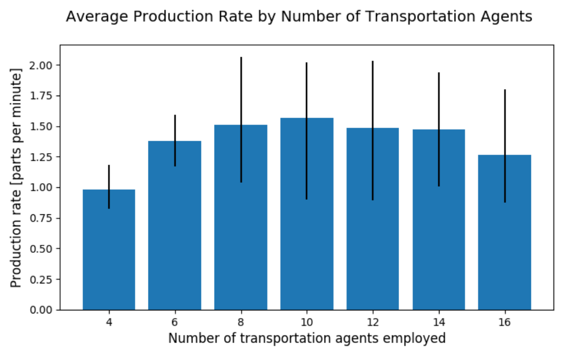

Figure 3: Production rate in relation to the number of available transportation agents. The error bars indicate the minimum and maximum value of the distribution accross all experiment runs.

assembly process, providing new parts as quickly as possible translates directly to an increase in the production rate. As expected, we observe that above a certain number of agents the effective production rate decrease again, probably due to the increasing amount of traffic.

From the size of the error bars, we conclude that randomness also has a large influence on the outcome of the experiment. We suspect the employed reactive, local agent coordination scheme is the main contributor to this effect. In this scheme, it is often a matter of chance, whether an agent will prefer one path over another. This demonstrates the importance of a reliable scheme for multi-agent path finding and navigation. We plan to integrate and evaluate such in the future.

Conclusion

With this paper, the core of a simulation testbed is presented, for the exploration and evaluation of different approaches to the problems of Job-Shop Scheduling, Multi-Agent Task Planning and Multi-Agent Path Finding. So far, we have implemented baseline approaches to those problems, that were shown to be able to coordinate multiple agents to manufacture several parts in a parallel production environment.

We have presented a system that is able to coordinate a fleet of robotic agents to take care of a plant's intralogistics in order to enable a continuous production of assembled goods. The modular design makes it possible to switch out single parts, and evaluate other software components in its place. This includes the possibility to replace the multi-agent simulation with actual robotic systems to carry out the transportation tasks in a real facility.

We presented an experiment, to present the general viability of the software setup as a testbed to address research questions. Within the experiment we compared the performance achieved by different numbers of agents executing a production plant's intralogistics.

We think that the application domain that lies at the basis of this testbed, the equipping or the assembly of airplane wings, is an interesting use case for planning research. Expensive parts and relatively small production rates have so far successfully deterred from actually implementing large scale automation of production processes.

Undisputedly impressive results have been shown for the case of warehouse automation already. These have not been transferred to manufacturing plants yet, not only because of robotic systems lacking manipulation or navigation skills, but also due to underdeveloped fleet coordination capabilities of commercially available robotic systems. Which is why we think, that our work can contribute to bridge this gap, and to focus attention on this problem domain.

Future Research

We plan to integrate more advanced planning approaches to address the problems of multi-agent scheduling and task planning. We have already started to develop the PDDL modelling for the planning domain description and the environment representation, and to integrate external planner instances, namely a classical planning approach using the Fast-Downward planner (Helmert 2006) and a temporal planning approach using the POPF planner (Coles et al. 2010) via the ROSPlan planning system. (Cashmore et al. 2015). We are looking forward towards comparing their performance to the above described approach.

Another very interesting application would be the integration of planners that use learning algorithms. This framework could provide a basis for comparing traditional planning algorithms with modern reinforcement learning approaches. Especially Neural Network based approaches are highly interesting from a scientific point of view. These methods could be integrated by adapting parts to the OpenAI Gym (Brockman et al. 2016) framework, to create a specialized learning environment for testing in industrial settings.

Also modern strategy games like DOTA or StarCraft require complex multi-agent planning and awareness of their environment. Being able to transfer the recent progress of OpenAI Five (Raiman, Zhang, and Wolski 2019) and DeepMind AlphaStar (Vinyals et al. 2019) to the setting of industrial production seems to be an interesting angle for further research. Also our simulated testbed system could serve as a good starting point for activities in this direction, as it seems infeasible to let an AI agent train for years worth of experience directly with a physical production plant.

Acknowledgments

This work was funded by the German Federal Ministry for Economic Affairs and Energy, grant number 20X1724C.

References

Bellingham, J.; Tillerson, M.; Richards, A.; and How, J. P. 2003. Multi-task allocation and path planning for cooperating uavs. In *Cooperative control: models, applications and algorithms*. Springer. 23–41.

Biswas, S.; Anavatti, S. G.; and Garratt, M. A. 2017. Nearest neighbour based task allocation with multi-agent path planning in dynamic environments. In *2017 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*, 181–186. IEEE.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *CoRR* abs/1606.01540.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carreraa, A.; Palomeras, N.; Hurtós, N.; and Carrerasa, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'15, 333–341. AAAI Press.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'10, 42–49. AAAI Press.

Enright, J. J., and Wurman, P. R. 2011. Optimization and coordinated autonomy in mobile fulfillment systems. 33–38.

Fiorini, P., and Shiller, Z. 1998. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research* 17(7):760–772.

Fujita, K. 2002. Product variety optimization under modular architecture. *Computer-Aided Design* 34(12):953–965.

Godoy, J.; Karamouzas, I.; Guy, S. J.; and Gini, M. 2016. Implicit coordination in crowded multi-agent navigation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, 2487–2493. AAAI Press.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hvilshøj, M.; Bøgh, S.; Nielsen, O. S.; and Madsen, O. 2012. Autonomous industrial mobile manipulation (aimm): past, present and future. *Industrial Robot: An International Journal*.

Janovský, P.; Cáp, M.; and Vokrínek, J. 2014. Finding coordinated paths for multiple holonomic agents in 2-d polygonal environment. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '14, 1117–1124. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Jefferson, T. G.; Crossley, R.; Smith, T.; and Ratchev, S. 2013. Review of reconfigurable assembly systems technologies for cost effective wing structure assembly. Technical report, SAE Technical Paper.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12(4):566–580.

Kulturel-Konak, S. 2007. Approaches to uncertainties in facility layout problems: Perspectives at the beginning of the 21 st century. *Journal of Intelligent Manufacturing* 18(2):273–284.

Le, D., and Plaku, E. 2017. Cooperative multi-robot sampling-based motion planning with dynamics. In *ICAPS*, 513–521.

Ma, H.; Hönig, W.; Cohen, L.; Uras, T.; Xu, H.; Kumar, T. S.; Ayanian, N.; and Koenig, S. 2017. Overview: A hierarchical framework for plan generation and execution in multirobot systems. *IEEE Intelligent Systems* 32(6):6–12.

Mohan, J.; Lanka, K.; and Rao, A. N. 2019. A review of dynamic job shop scheduling techniques. *Procedia Manufacturing* 30:34–39.

Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The robocup logistics league as a benchmark for planning in robotics. *Planning and Robotics (PlanRob-15)* 63.

Pecora, F.; Andreasson, H.; Mansouri, M.; and Petkov, V. 2018. A loosely-coupled approach for multi-robot coordination, motion planning and control. In *ICAPS*, 485–493.

Raiman, J.; Zhang, S.; and Wolski, F. 2019. Long-term planning and situational awareness in openai five. *arXiv preprint arXiv:1912.06721*.

Schillinger, P.; Bürger, M.; and Dimarogonas, D. V. 2018. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The international journal of robotics research* 37(7):818–838.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Snape, J.; van den Berg, J.; Guy, S. J.; and Manocha, D. 2010. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS*, 4584–4589. IEEE.

van den Berg, J.; Guy, S. J.; Lin, M.; and Manocha, D. 2011. Reciprocal n-body collision avoidance. In *Robotics Research: The 14th International Symposium ISRR*, volume 70 of *Springer Tracts in Advanced Robotics*. Springer-Verlag. 3–19.

van den Berg, J.; Guy, S. J.; Snape, J.; Lin, M.; and Manocha, D. 2016. Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulations. http://gamma.cs.unc.edu/RVO2/. Accessed: 2020-02-18.

Vinyals, O.; Babuschkin, I.; Chung, J.; Mathieu, M.; Jaderberg, M.; Czarnecki, W.; Dudzik, A.; Huang, A.; Georgiev, P.; Powell, R.; Ewalds, T.; Horgan, D.; Kroiss, M.; Danihelka, I.; Agapiou, J.; Oh, J.; Dalibard, V.; Choi, D.; Sifre, L.; Sulsky, Y.; Vezhnevets, S.; Molloy, J.; Cai, T.; Budden, D.; Paine, T.; Gulcehre, C.; Wang, Z.; Pfaff, T.; Pohlen, T.; Yogatama, D.; Cohen, J.; McKinney, K.; Smith, O.; Schaul, T.; Lillicrap, T.; Apps, C.; Kavukcuoglu, K.; Hassabis, D.; and Silver, D. 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

Wang, K.-H. C., and Botea, A. 2011. Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research* 42:55–90.

Yang, F., and Chakraborty, N. 2019. Multirobot simultaneous path planning and task assignment on graphs with stochastic costs. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 86–88. IEEE.

Yao, F.; Keller, A.; Ahmad, M.; Ahmad, B.; Harrison, R.; and Colombo, A. W. 2018. Optimizing the scheduling of autonomous guided vehicle in a manufacturing process. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, 264–269. IEEE.