Fynn Boyer, Sefika Efeoglu, Ricardo Gabriel Herdt, David Mezey *(Group 4)*

Multi-Robot Planning and Free Navigation in a Warehouse Environment

Abstract— **The vast demand of online purchase and same-day delivery as well as of cost minimization makes autonomous warehouses a widespread reality over the globe. The currently used strict floor plans used for robot navigation, however, can not cope with unforeseen situations and make the delivery process shifted from purchase due to the duration of planning in the meanwhile (offline planning). In the following document we present a warehouse system with delivery robots that are now not bound to strict floor plans and can navigate freely in the environment on paths calculated by the Push and Swap algorithm. Unforeseen events can be handled by the agents using custom obstacle avoidance behavior and the replanning of the delivery happens on-the-fly, scoring the suitability of robots for a given task (online planning). Other usual or unforeseen battery-scenarios are handled by a local Markov Decision Process increasing the overall stability of our system. Hereby we show that our system is able to accomplish delivery tasks in a warehouse environment with freely moving robots. We furthermore demonstrate how the throughput of the system changes over the number of robot-agents used in the system due to planning scenarios or implementational bottle-necks.**

A. Introduction

The expansion of internet based trading in general imposes several challenges to guarantee an efficient distribution of products. As mobile robot technology becomes more affordable warehouses around the world utilize the possibility of automation using groups of robot-agents to catch up with the high demand. The fact that numerous providers (Walmart [1], DHL [2], Amazon [3]) have already started to use autonomous robot systems (ARS) in their warehouses is paving the way to a widespread and general [4] usage of applied robotics.

Currently developed fully automated autonomous warehouse solutions, however, are mostly using a *grid-like floorplan* to operate a large number of robots to find ordered articles and move them to the packaging area efficiently. Furthermore, the exact plan of which task belongs to which robot-agent is *planned offline* meaning that the overall task plan might have been planned several hours or even a day before and thus, unexpected changes in the environment (such as robot-failures, accidents, package-failures, etc.) may lead to a full shutdown of the automated system until the source of the failure has been removed (usually by human manpower). Such warehouse outages may lead to many hours of delay in package-delivery and a complete recomputing of the offline task plans.

To optimize efficiency and increase robustness against such scenarios our approach was to operate robots online (i.e. the tasks are dynamically assigned as they arrive to the facility) and remove the constraint of a grid-like space in which the robot-agents can move.

To accomplish such a system, we assign tasks to the most suitable robots on-the-fly according to the MURDOCH method. Scores are calculated according to the estimated path length and duration of journey for the robot given a specific task as well as to the battery status of the robot. To provide the continuity of the system we implemented an energy management solution. In this work, timing of the charging process and the selection of the charging station are addressed by MDP [15] and A* search [14], respectively. During the fulfillment of a task the robots are moving freely in the warehouse environment on dedicated paths calculated by a central path planning unit using the Push and Swap algorithm [5]. The movement of the robots along the paths is achieved by applying a path smoothing algorithm and then calculating the cross-track error between the robot and the path to finally feed it into a PID-controlled trajectory following algorithm. To handle unforeseen scenarios during the task fulfillment process the robot-agents are using a custom obstacle avoidance behavior using the on-board laser scanner to avoid collisions and execute evasion maneuvers. We designed a possible Traffic Management system as well, which even though has not been integrated to the system, due to its modularity can be easily integrated to systems with different path planning methods.

As a result we were able to implement a solution which operates robot-agents in a free floor-plan warehouse environment to fulfill dynamically arriving tasks planned online. To measure the throughput of our system we measured 3 key metrics, namely the used energy, the travelled distance and the time needed (on average) for a robot to fulfill a single task/delivery. We furthermore compared the throughput in different scenarios where different numbers of robot-agents were available. In this work we show that according to different conditions different numbers of robot-agents are optimal in the system and we give further advice for future improvements.

B. Related Works

Modern, automated warehouses are operating with a vast volume of goods at any given time which leads to a problem, namely the *multi robot task assignment problem* (MRTA). Even though many robot-agents might be available for a given task one should choose the most suitable robot to optimize the overall behavior. According to the taxonomy of task planning problems proposed by [6], our case consists of a "single-task robots, single-robot tasks" (ST-SR) problem i.e. robots can fulfill only a single task at once, and tasks need a single robot to be accomplished. Regarding assignments, the authors distinguish "instantaneous assignment (IA)" from "time-extended assignment (TA)" problems. In the former case, there is no information regarding future tasks (which is our case), while in the latter case either all tasks are known in advance, or at least there is a model of how tasks are created. Our system qualifies as an "online assignment" variant of the ST-SR-IA problem [6] in which we don't allow reassignments, as that would cause extra efforts for task and path planning. For this particular case, the MURDOCH assignment algorithm is shown to have the best performance possible [6] [7]. As our system is rather small (with 8 robots at maximum) we chose to implement a central Task Planner and Assigner complex scoring the robots suitability for a given task and according to the MURDOCH method assigning the task to the robot with the highest score. To make our system more modular we decided to separate the assignment from the task scoring in such a way that the estimation measures used to create the mentioned scores are calculated by a separate Estimator node.

Another crucial problem in designing autonomous multi-robot agents with free navigation is the *multi-robot path planning* problem (MRPP). A solution to the problem can be described as *complete* (i.e. a solution is always found when it exists), *optimal* regarding time/resource related criteria and *efficient* [8]. Finding an optimal solution is known to be an intractable problem [9], therefore, any solution has to trade off some desired properties. Existing solutions to MRPP can be categorized as centralized or decentralized methods. In the former case the paths are computed by a central unit which has access to global information regarding the map, robot configurations and task assignment information. In contrast, in decentralized methods the paths of robots are not calculated in advance. Instead, the robots calculate their initial route independently of other robots. As the robot paths are based on incomplete information, conflicts between robots will arise. Centralized approaches can be further categorized as coupled or decoupled methods. In the former case the method searches for an optimal solution considering all agents simultaneously. Such an approach is complete and optimal, but intractable making it unsuitable for problems with more than 5 robot-agents [10]. To deal with a larger number of robot-agents, decoupled approaches typically trade off solution quality (such as completeness) for efficiency by solving some aspects of the problem independently from others [11].

As our system design was chosen to be centralized and we have to deal with more than 5 agents simultaneously in some scenarios we chose to have a centralized decoupled path planner which computes the paths with the Push and Swap algorithm due to its simplicity and efficiency [5]. This algorithm is shown to be scalable to relatively large scenarios with more than 100 robot agents. It follows a simple strategy of moving agents towards their goals ("push" operation) and of switching positions of agents when two agents are in conflict, i.e. one blocks the other towards its goal ("swap" operation). As pointed out by [11] "Push and Swap" is not a complete algorithm but can be further improved to accomplish completeness.

To make the robots smoothly and reliably follow the calculated paths as well as to define their behavior during loading and unloading packages we implemented a local motion planner. [12] This work follows the trajectory of the car in a new way, taking into account the direction of the front wheels, not the body of the vehicle. Depending on the cross-track error between robot and the trajectory, the front wheels will be steered to eliminate any deviation from the track. To control the velocity of the vehicle, the brake and throttle are operated by a proportional integral (PI) controller. [13] This approach calculates a smooth motion trajectory that supplies the kinematic of the vehicle using B-spline parameterization [22]. The reason to use B-splines is the convex hull property which states that the spline is always contained in the convex hull of its control points. [24] Here arc-based and spline-based path creation from a base path were compared. The spline-based method created more traversable paths while requiring more time to calculate. [23] Stanley is the robot that won the DARPA challenge in 2005. In the challenge a database that contains only an approximated path along the challenge route was given. Before the race the path is smoothed to get a base trajectory along the path. They adjust this path online using a local path planner. We use the same methodology to implement the motion planning in this work.

To provide the continuity of any autonomous system energy management is one of the most significant issues that needs to be solved. This issue is managed in different ways depending on Centralized and Decentralized architecture in the literature. [18] The energy management issue is handled with three key components: the timing of the charging process, the selection of a charging station and duration of charging. [18] A flexible scheduling approach is used different methodologies on these key components: threshold for timing of charging, heuristic algorithm for the selection of the charging station and threshold/swap time for the duration of the charging. Another approach [19] in the literature to find the charging station is Online Connected Dominating Sets (OCDS), which are used in undirected graphs, when the battery level of the robot agent drops below a critical level. In this approach, any two charging stations shall not be farther from each other than a specified distance. In [20], the authors manage the energy of autonomous systems via the docking and charging system. This approach is needed to let the robot itself handle autonomous docking and charging problems. On the other hand, [17] to learn the individual energy consumption, Reinforcement Learning (Q learning and MDP) is used with daily household data. Its reward mechanism which is learnt by utility function provides a learning approach with negative and positive rewards. The results of the work show the reduction in daily energy consumption. [15] To decide the charging time, a MDP model which has 2 battery states is proposed. According to the reward mechanism of the model, an agent decides to go to a charging station. We propose an energy management approach that takes into account the energy consumption for a charging station and a task using Markov Decision Process which considers two battery states to decide the timing of the charging depending on the negative reward signal which calculates using linear regression model with the training data. This decision mechanism of the battery threshold aims the working of the high number of robots for a long time in the system. In addition to this, to find the nearest charging station, we use A * algorithm [14] which considers the static obstacles of the warehouse.

C. Methodology

A. General Framework

In our model warehouse packages are being transported from designated pick-up areas to assigned package end-tra by multiple robot-agents. Generating the orders/packages and planning which socket they must be transported to is out of the scope of our project and is already implemented. The floorplan and the exact position of environmental elements such as pick-up areas, package sockets, charging stations are given. Our goal is to implement a solution to assign continuously arriving orders/tasks to agents and to coordinate the motion of these robot-agents to fulfil the assigned tasks. Upon a new package has been generated the Task Planning and Task Assignment Nodes are scoring each robot how suitable they are for the given task, namely to deliver a package from start to end-tray. To score the robot-agents the aforementioned nodes are using the estimated time of journey given an A* path by the Estimator Node of the system. As soon as a task is assigned to a specific robot-agent, the robot requests a new path from the central path planner. The path is calculated taking all current robot tasks into consideration by the Push and Swap algorithm. The robot then follows the assigned path smoothly and reliably using Cross-Track Error calculation of the motion planner unit as well as the PID controllers of the robot. The motion planner unit is also responsible for proper driving behavior before and after loading and unloading packages. Upon any unforeseen obstacles the local obstacle avoidance behavior of the robots are triggered to avoid possible collisions. In case a robot “needs” to charge due to low energy level (further described in “Energy Management and Charging Management”) the robot will refuse to take any new tasks and will navigate to the closest charging station. To handle these scenarios a local Markov Decision Process as well as a central Charging Management unit is used. For further (future) improvements we implemented a separate Traffic Management node as well which is able to estimate all robots journey given the planned paths and is able to code the estimated traffic information into a Dynamic Map. Even though this node has not been used due to our choice of path planning algorithm, it is easy to integrate into any other system.

B. Task Planning and Assignment (TP, TA)

Upon a new package is generated by the Package Generator node of the overall architecture a new task is created and broadcasted by the Task Planner Unit (TP). This message includes information such as where is the package currently and where should it be delivered (using the ID of the start- and end-trays). A task includes multiple possible start and end trays for a given task. The Task Assignment Node (TA) serves as a general score calculation node which calculates suitability scores for all possible robot-task pairs upon a new task broadcast. After the calculation, according to the MURDOCH method TA acts on behalf of the robot agents and announces the best score for each robot towards TP which will then assign the task to the robot of highest score. This proxy solution is necessary between Task Planner and Task Assignment as the Package generation relies on the TP unit. This restricts the solution to use TP to assign tasks to the robots. Our system defines a score for task i and robot j such as:

$$S_{i,j} = \frac{\eta}{T^{EST}(P^{A^*}(C_{R_i}, C_{S_j}, C_{E_j}))} + B_i$$

where $\eta = 8$ is the emphasis factor of how much more important the estimated duration of journey is than the battery level when scoring robots. T^{EST} is the custom journey duration estimator function of the estimator node, $P^{A^*}(C_{R_i}, C_{S_j}, C_{E_j})$ is the estimated path from the current robot coordinate through the starting point of the task to the end/goal point of the task, and B_i is the battery level of the given robot i as a float number between 0 and 1. In the TA node we calculate score S_{ij} for all robot i and possible start-end-tray combination j. I.e. if there are N_S possible start- and N_E possible end-trays for a given task and there are N_R robot-agents, we can define a score matrix **S** with shape $N_R \times (N_S N_E)$ which includes now all possible scores calculated in the system for a new task. For each robot we can then choose the most fitting task index (which denotes a start and end tray combination together according to our **S** score matrix) as

$$I_i^{Max} = argmax_j(S_{ij}) \mid 1 < j < (N_s N_E)$$

For each robot the task given by this index as well as it’s score to the given task is sent back to the Task PLanner which is going to assign the task for the robot with the maximal score.

C. Path and Journey Estimation (EST)

Estimating the suitability for a given task of a robot must include some information about the physical distance between the robot and the area-of-interest of a given task. An easy-to-implement and fast solution is to take some distance between the robot and the start position of the task and estimate the suitability score accordingly. This, however, does not take obstacles of the environment into consideration and therefore, can lead to a distorted solution quickly. To avoid such biases in our score estimation we rather generate a path for a given task and estimate how long does it take for the robot to travel along the path. Respectively, this node has 2 functions. On one hand it uses the A* algorithm to give an estimation of a robot's path given the start and end positions [14]. This takes only static obstacles of the environment into consideration. On the other hand it also estimates the duration of journey along an estimated path using a custom heuristic method. It is calculated using the following formula:

$$t_{est} = \frac{1}{\alpha}\sum_i |\vec{v}_i| + \frac{1}{\beta}\sum_i |angle(\vec{v}_i, \vec{v}_{i+1})|$$

where v describes the vector between two points in the path and α and β are learned values from the motion model of the robot. In our overall architecture the estimated path function is used by the Charging Management node to find the closest charging station for a robot. TA furthermore uses both the estimated path, and journey duration functionalities of the node as described before. The implemented Traffic Management node uses the mentioned time estimation method customized to estimate ROS timestamps along a planned path.

D. Path Planning (PP)

The Path Planner is a Central Path Planner unit. It collects through heartbeat-messages current information of agents: their status (idle or not), current position and current goals. Upon receiving from an agent a request for a new path, the Path Planner computes new paths for all agents, answers the request with the corresponding agent's path, and broadcasts the updated paths to the remaining agents.

The central nature from the Path Planner demands an efficient (low-latency) path planning algorithm, since in our approach the path plan is recomputed at each request. For this reason we chose the decoupled centralized Push and Swap algorithm. As described by its authors [9] it constrains the search for a plan to a subset of the search space by focusing on two primitive "action": Push and Swap. It starts by computing the shortest paths of agents towards the corresponding goals (in our case A* was used). Then each agent is moved towards its goal as follows. First the algorithm tries to "push" agents towards their goals. A "push" action moves the agent in the conflict-free case through its shortest-path, and forces agents "on the way" out of the way. In case a simple sequence of "push" action doesn't find a solution, the algorithm tries to "swap" conflict agents. For that, the algorithm searches for possible positions where such a swap is possible, as illustrated on Fig. 1. After swapping agents, the algorithm moves them back to the original conflicting place, now with swapped positions. In case "swap" also fails, the algorithm returns that no solution is found.

As pointed out by [11], this doesn't necessarily mean that there is no solution, since a different ordering of the agents could possibly find a solution. This means "Push and Swap" isn't complete, as originally thought by its authors.

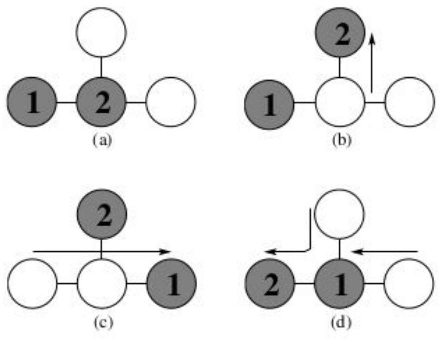


Fig. 1. Possible swap operations [9].

Due to its general good efficiency and quality it proved to be a good solution to our requirements, especially considering that in a real setting the path planning cannot perfectly avoid collisions, and so the system has to take other measures to achieve collision-freeness. This has several factors. First the physics of agent movements combined with collision detection behavior lead to varying velocities of robots, whereas the Push and Swap plan assumes a steady and perfect execution of the planned path. Secondly the path planner has an approximated view of the real scenario through its representation as a roadmap graph. In particular our robots are allowed to move in all directions, leading to situations where a conflict-free solution without support from collision detection and other strategies becomes a challenge. Consider for instance the example illustrated at Fig. 2: the roadmap graph's resolution consists solely of the black nodes, our robots are allowed though to take diagonals through nodes. As such the algorithm doesn't detect a collision and generates conflicting paths. Combined with collision detection and random waiting times the system can be designed to robustly deal with such situations, which was not done in the current implementation due to time constraints.

Another approach to achieve a better collision-free performance would be to use a dynamic map as an underlying graph for computing the path plan (see subsection regarding "Traffic Management") below. This would have serious implications to our design though; for instance in order to keep using the Push and Swap approach we would have to transform the 3D dynamic map into a graph suitable for it, probably by collapsing the time-based traffic information in a flat domain, which would affect its usefulness. Also the extra dimension would possibly increase the latency considerably, violating the low-latency requirement explained above.

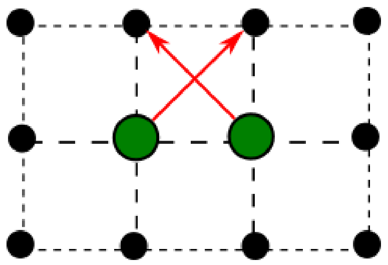


Fig. 2. Collisions due to diagonal graph traversal.

E. *Motion Planning (MP)*

Once a path has been determined and provided to the robot by the path planner the motion planning will calculate the appropriate commands for the actuators to traverse the trajectory. The objective is to drive the given path in a smooth, efficient manner while minimizing any deviations from the trajectory both in time and space as this can lead to collisions. In order to achieve this goal several techniques have been implemented which will be described in the following paragraphs:

1) *Path smoothing*

The path provided by the path planner is constructed using the connectivity graph of the warehouse. Due to performance reasons this graph has a limited resolution and thus the calculated paths have many sharp turns and are not well suited for driving. To improve the given trajectory a B-Spline interpolation [22] was used to smooth the path. As the B-Spline is not well-defined for the start and end of the trajectory, a line was constructed to connect the smoothed path to the start and end point of the trajectory. An illustration of this can be found in Fig. 3.

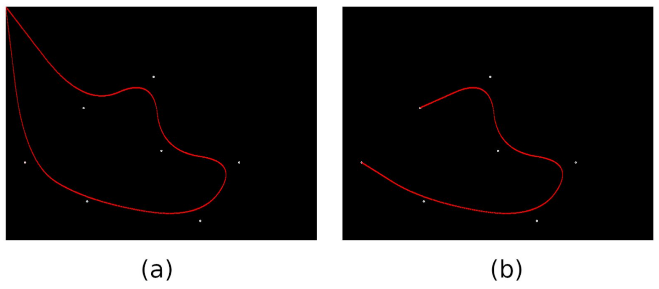


Fig. 3. Use of B-Spline interpolation in basic form (a) and with additional connecting line to create a drivable trajectory (b).

2) *Cross-track error elimination*

In order to follow the smoothed path an algorithm is necessary that evaluates the current pose of the robot relative to the expected pose on the trajectory and that adjusts for any error that is detected. Optimal behavior is achieved if the center of the robot is always on top of the trajectory with its orientation parallel to the line tangent to the current point in the trajectory. To achieve this behavior the lateral distance of the robot to the trajectory is calculated (cross-track error, CTE). To determine the error, the line between the current and next point in the trajectory is constructed and then the distance of the robot to that line is calculated. The error will have a different sign depending on if the robot is to the left or to the right of the trajectory. If the error is zero the robot will align its orientation with that of the trajectory, otherwise it will add an offset to its desired orientation in order to merge onto the trajectory.

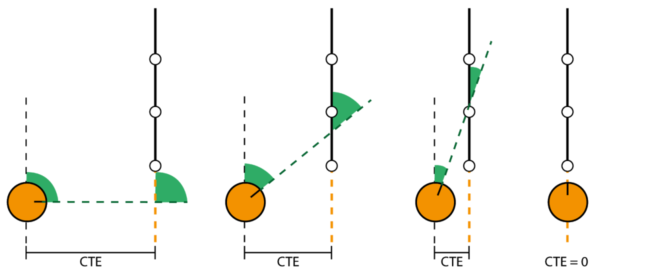


Fig. 4. Offset from parallel alignment of the robot to the trajectory based on the CTE to achieve a dynamic merging behavior

The desired value for the angle between the robot and the trajectory (marked green in Fig. 4.) is then fed into a PID controller to calculate the commands that will be sent to the actuators using the following control law

$$PID(\underbrace{0 + f(CTE)}_{\text{target}}, \underbrace{angle}_{\text{state}})$$

where $f(CTE)$ denotes a transformation from the cross-track error to an angular offset which is calculated in the following way:

$$f(CTE) = \begin{cases} max(-\frac{\pi}{2}, -\frac{\pi}{2} \cdot \sqrt{\frac{CTE}{CTE_0}}), & CTE < 0 \\ 0, & CTE = 0 \\ min(\frac{\pi}{2}, \frac{\pi}{2} \cdot \sqrt{\frac{CTE}{CTE_0}}), & CTE > 0 \end{cases}$$

where CTE_0 is a parameter that determines the CTE value at which the robot will have the maximum offset of ±90°. A positive CTE indicates that the robot is to the right of the trajectory and vice versa for the negative CTE. It can be seen that the angular offset will converge to zero as the CTE is decreased. It would also be thinkable to use a simple linear relation between CTE and offset but the more aggressive steering based on the square-root function showed better results.

3) *Target point update*

The trajectory consists of a list of points that the robot has to traverse. The robot needs to decide when it reached a certain point in order to start following the next one. The technique used for this also influences the way in which the robot will merge onto paths where the start position of the path is not equal to that of the robot. The solution used in this system determines the line from the current target point to the next one and then calculates the line perpendicular to that line. This perpendicular line serves as an imaginary, infinite target line that the robot has to cross in order to reach the point (Fig. 5.).

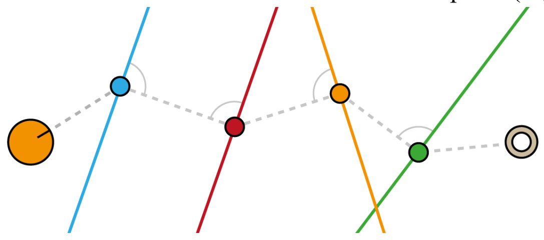


Fig. 5. Virtual target line the robot has to cross in order to reach a point.

Note that the first target line is calculated differently so that it has a similar orientation as the second line. This is done to allow a smooth merging behavior independent of the position of the robot. As illustrated in Fig. 6. the robot could be in a position that is close to a more advanced part in the trajectory than the starting point. It will notice that it has reached multiple of the points in the trajectory already and will merge onto the trajectory where it's fitting.

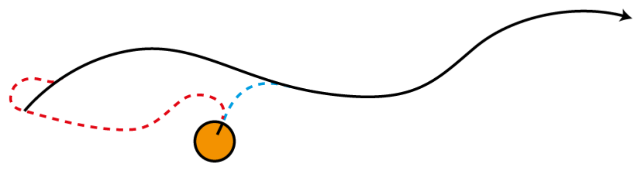


Fig. 6. Merging behavior of the robot with (blue) and without (red) adjustment of the orientation of the first target line in the trajectory

The last point in the trajectory is checked differently using a circular threshold area as it is important to hit the last point exactly.

4) Speed Adjustments & Delay Compensation

If the robot has to perform a sharp turn, it can't do so with full speed. At the start of the turn the robot will overshoot the curve slightly which will increase the CTE. To enhance the steering behavior, the linear speed of the robot is coupled to the CTE. As the CTE increases, the speed of the robot will be decreased to gain better steering performance.

The path planner assumes the same duration for every unit length in the path that is based on the motion model of the robot. It is important to stay close to the expected time for a position in the trajectory as the time-dimension is important to prevent collisions. Due to curves, inefficient driving, obstacles or shortcuts the robot might be delayed or too fast. In order to compensate for any deviations in the time-dimension, the robot always compares expected and actual position and adjusts its speed accordingly.

F. Obstacle Detection and Collision Avoidance

Even though our path planner can calculate collision-free solutions, practical tests showed that collisions between robots can still occur due to imperfections in execution. To avoid these collisions, an obstacle detection algorithm has been implemented that is based on the on-board laser.

1) Static & Dynamic Obstacles

The path planner will generate paths that will not collide with any static obstacles as it has the occupance map of the warehouse. The obstacle detection is used to detect dynamic obstacles and static obstacles shall be ignored. To achieve this, the values generated by the laser scanner are combined with the current position of the robot to gain information about the position of the scan in the occupancy map of the warehouse. If the space at that location is occupied, the laser scanner has detected a static obstacle and this point will be ignored in the consecutive laser scan analysis.

2) Laser scan analysis

After removing static obstacles from the laser scan the remaining values are investigated for dynamic obstacles. First any scans that exceed a distance that is considered risky will be discarded. Then, the information of the laser scan is compressed by categorization. The closest obstacle to the left and right of the robot is searched with the corresponding angle. In addition to that the closest distance to obstacles in front of the robot is determined and if that frontal obstacle is to the left or right of the robot. This information is then used to adjust the motion of the robot as is explained in the next paragraph.

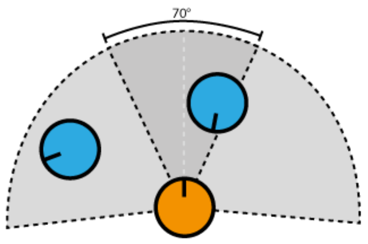


Fig. 7. Different zones used for laser analysis. For each zone the distance and position of the closest obstacle is determined. Note that the left & right zone include parts of the frontal zone.

3) Evasive maneuvers

Using the data acquired by analysing the laser data the robot performs evasive maneuvers to avoid other robots. The linear velocity of the robot is adjusted with a factor depending on the proximity of the frontal obstacle. Initially the robot will slow down and at some point come to a stop. If the obstacle then comes even closer, the robot will drive backwards to avoid a collision. This is especially important to avoid robots that drive backwards and are unable to detect obstacles as the laser covers only a radius of 210°.

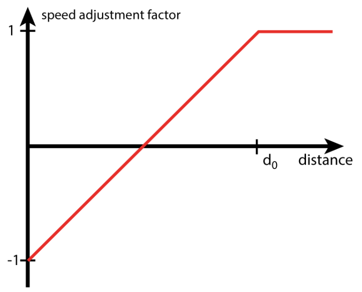


Fig. 8. Adjustment of the linear velocity of the robot based on the distance to obstacles in front of the robot.

The angular velocity of the robot is changed based on obstacles to the left & right of the robot. If an obstacle is detected to the left of the robot it will steer right and vice versa. Depending on the angle and proximity of the obstacle, the robot will adjust its steering. As the distance to the obstacle increases and becomes positioned more laterally, the robot will steer less aggressively as the obstacle becomes less of a threat. This behavior allows robots to avoid each other and find solutions at crossroads or frontal collisions as illustrated in Fig. 9.

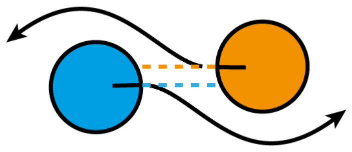


Fig. 9. Example of evasive maneuvers that allow robots to avoid each other and continue on their path. Both robots will detect an obstacle to their left and steer right which will allow them to pass each other. They will also slow down as they detect a frontal obstacle which will also help to prevent a collision.

G. Charging Management and MDP Approach

In this work, 2 energy management problems which are finding the nearest charging station and deciding the timing of the charging process are solved by A* search algorithm and Markov Decision Process, respectively.

1.) Finding the nearest charging station

To tackle the selection of the charging station, A* algorithm is used via the estimated path service. The most important reason why this algorithm is used for this problem is that it considers the static obstacles in the warehouse. By this way, the nearest charging station to a robot is recommended by Charging Management when it needs to charge.

2.) Deciding the timing of the charging process

Another most important issue of energy management is the timing of the charging process. This issue is solved by dynamic battery threshold approach via Markov Decision Process using an estimated linear regression model for dynamic rewards. This approach pays attention to the current battery level of a robot and an estimated battery consumption to the nearest charging station and/or to fulfill the upcoming task. Each finite MDP uses different reward values since the robot has a different distance to reach the charging station in each time interval. This leads to different battery consumption rates so as to reach the charging station. The MDP model in this work is an example of Recycling Robot MDP [15]. The most significant difference of MDP used in this work from the Recycling Robot is that it has dynamic reward values. The dynamic reward values provide different battery threshold values depending on the robot position, robot configuration and the current battery level of the robot.

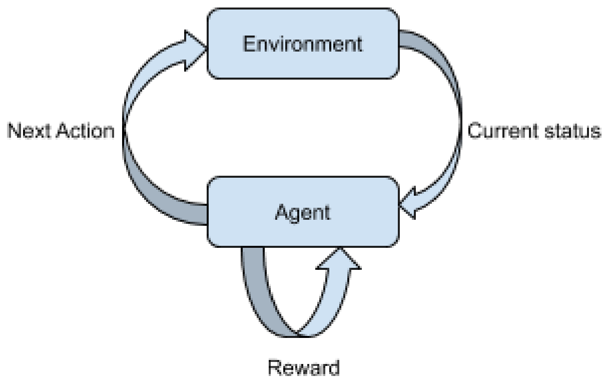


Fig.10. Reward signal comes from the inside of an agent, and the structure of reinforcement learning approach.

I. *Dynamic Reward Values*

Dynamic reward values are obtained while using the estimated linear regression model which uses data consisting different threshold values with their reward values. The velocity and the discharging rate of a robot is taken into account while calculating the battery consumption for a task and/or a charging station. If it makes a decision for a task, estimated battery consumption will be considered for this task and a charging station which is near to the next position of the robot when this task is completed. On the other hand, estimated distance for a task or a charging station also affects this reward signal, and the environment also changes this reward signal. Matlab stats fitlm [16] is used to implement the estimated linear regression model, which provides the prediction of a new reward depending on the estimated battery consumption, for the data below in Fig. X.

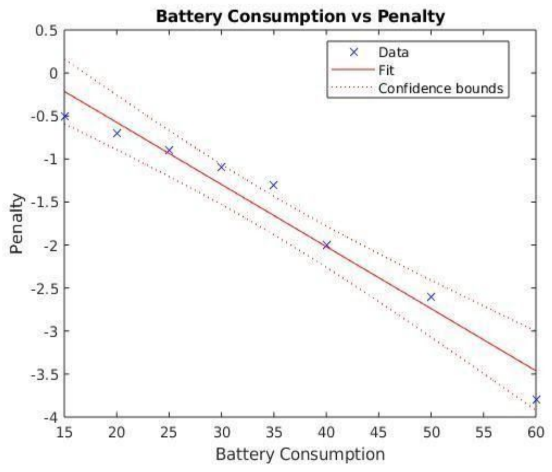


Fig.11. The relation between battery consumption and penalty value in estimated linear regression model. Penalty refers to a negative reward signal.

Fig. 11 handles a battery consumption rate between 15 and 60. Therefore, this regression model can predict a new reward signal for this battery consumption interval. It might return wrong reward signals for less than *15.0* or more than *60.0* because out of the training data above in Fig. 11 is unknown, for instance, it might predict positive reward signal instead of negative reward due to the upper bound of confidence interval for 15.0 battery consumption in this linear regression model. The reason why this data interval is selected for the prediction of the reward signal is that the environment has uncertain collision avoidance behavior and gripper problem on unloading or/and loading of a package. These problems lead to a longer journey than the distance for a task or/and a charging station which is estimated for the reward signal of the next action. To prevent this issue, an extra battery consumption as *15.0* is added to estimated overall battery consumption while predicting the reward signal of the next action. For robust results, average spending time for collision avoidance and system problems has to be taken into account on the prediction of the reward signal. The behavior of collision avoidance and system problems are out of the existing energy management problem which is handled in this part. However, this system problem is solved by merely adding an extra battery consumption to overall consumption.

As a result, the negative reward signal is measured depending on velocity, discharging rate of a robot as well as distance to a goal by this estimated linear regression model.

II. *Markov Decision Process*

The battery of a robot has two states which are high and low. High state of the battery is defined as more than a half of the full battery level, and low state is also defined less than 50 % of the full battery.

$State\ Space\ S = \{Low,\ High\}$
$Action\ Space\ A = \{Idle,\ Search, Recharge\}$
$Probabilities\ Pr(S_0,\ S_1) = Pr(S_1 | S_0,\ a_0)$
$Rewards = \{R_{search},\ R_{Idle},\ R_{recharge}\}$
$Action(Low) = \{Idle,\ Search, Recharge\}$
$Action(High) = \{Idle,\ Search\}$

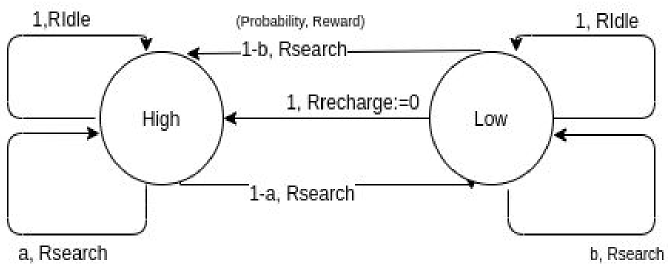


Fig. 12. MDP model.

As it can be seen above in Fig. 12., "High" state has 2 actions; *Search* and *Idle*, whereas "Low" state has 3 actions; *Search*, *Idle* and *Recharge*. Recharge action does not have any reward signal. In addition to this, if "Search" action goes to "High" state from the "Low" state with a low battery percentage, it will take a negative reward signal. In other words, it takes a penalty. In this case, this mdp model returns a "Recharge*"* action to the robot as a best policy in low current state, and the robot goes to the nearest charging station using A* algorithm manually. This learning method is defined as threshold based learning [21].

$$V^{\pi*}(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi*}(s')]$$

Equation 1: Bellman Equation for optimal policy selection of MDP.

Regarding the best policy selection of the model, Bellman equation in Equation 1 is applied using transition probability matrix which includes the current battery probabilities for battery level rate. This MDP is called when task assignment (TA) module wants to assign a task, the system is run, and while sending robot heartbeats if a robot is idle. This next action which is decided by MDP is sent by robot heartbeat to the system. This approach provides a dynamic decision mechanism to a robot while the system is running.

As a result of this work, both lines in Fig.13. illustrate the battery level of two different robots. Both robots have almost the same battery level (approximately 20 percent) when the system is run, but they have different positions in the warehouse which means that each robot has a different distance to reach a charging station. It can be seen that both robots have different the lowest battery levels at different times which means that they have these battery levels when they reach a charging station. In addition to this, in Fig.19, robots having a battery status between 50% and 20% would go to the charging station if we used classic fix battery threshold because classic charging management approach makes a decision according to critical battery level and battery threshold level which are defined as 50% and 20%, respectively in existing codebase and literature. Finally, the number of robots (8) has been stable for almost 7 mins, otherwise we would observe a fluctuation in Fig. 19.

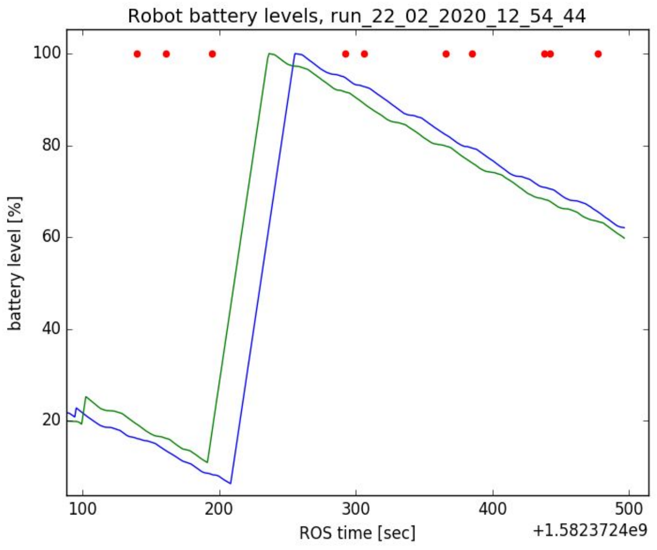


Fig. 13. Battery level of 2 robots throughout a simulation (lines) and the time of package deliveries (red dots). The 2 robots went to charging at 2 different battery threshold values. In our warehouse robots have the same configuration (discharging rate and velocity), which means that these threshold values are affected by merely the distance between a robot and a charging station.

H. Traffic Management (TM)

As in every fixed floor plan warehouse, in our system as well some routes to deliver packages are of higher traffic than others due to the fixed position of start and end trays of the package delivery. A central path planner node which does not take future traffic estimations into considerations, therefore, leads to "risky" scenarios, i.e. peak traffic areas where the chance of collision between robots is higher. Unwanted traffic scenarios can also cause delays in the system or an overall decrease in the delivery throughput. For this reason, a traffic monitoring/management node is generally preferred which can estimate the future traffic given the robots planned path and position and can serve the path planning unit with this information.

Here we implemented a simple traffic management node that according to the planned paths of the robots as well as according to their position can dynamically estimate the traffic throughout the whole warehouse. After starting the simulation upon a new planned path has been generated and assigned to a robot by the path planner unit the TM node will use the journey duration estimation method mentioned in the section "Estimator Node" to assign an estimated timestamp for each node of the path graph. In other words, if given the planned path of robot R as $P_R = (x_i, y_i)_R,\ 0 \le i \le N$, we estimate the estimated path of robot i (using the time estimation method mentioned in "Estimator node, time estimation" part) to get $E_R^k = (x_i, y_i, t_i)_R^k,\ 0 \le i \le N$, namely the k^{th} estimated path for robot R. Please note, that k in the upper index stands for the version of the estimation.

Furthermore, in each simulation timestep $(\Delta k = 1s)$, we check the current position of the robot and compare it with the position where the robot shall be according to the previous estimation. To accomplish that we use linear interpolation (in time) between the estimated path nodes of the previous estimation, so that the estimation error is

$$\varepsilon_R^k = norm[(x_R, y_R, t_E) - (\hat{x}, \hat{y}, t_E)_R^{k-1}]$$

where (x_R, y_R, t_E) is the robot coordinate and $(\hat{x}, \hat{y}, t_E)_R^{k-1}$ is the estimated coordinate of the robot at the time of the error calculation t_E using simple linear interpolation between $(x_i, y_i, t_i)_R^{k-1}$ and $(x_{i+1}, y_{i+1}, t_{i+1})_R^{k-1}$ nodes of the previous estimation E_R^{k-1} for which $t_i \le t_E \le t_{i+1}$.

If the error between the estimation and the current robot position is larger than a predefined threshold the estimation is carried out again, given the current robot position and the planned path as before, but now from the closest node of the path graph. If the error is smaller than the threshold we do not update the estimation, so $E_R^k = E_R^{k-1}$. If there is no such i that $t_i \le t_E \le t_{i+1}$ holds, the estimation error is set above threshold so that the re-estimation happens irrespectively of the estimation error.

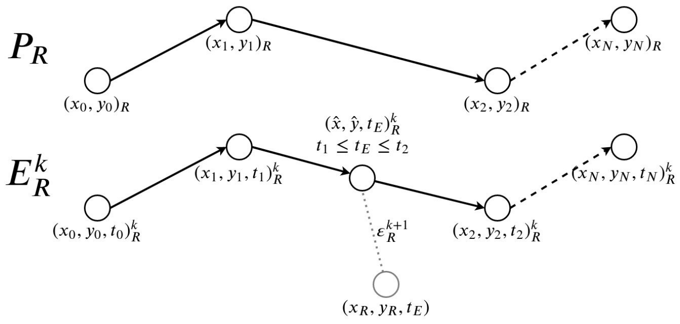


Fig. 14. Planned (above) and Estimated (below) path for robot with index R. On the estimated path an example of the estimation is denoted between path node 1 and 2 at time t_E. The robot position is denoted with a grey circle, the estimation error ε_R^{k+1} with a dotted grey line. As the path can include more nodes the link between node 2 and the final node is depicted with a dashed line.

As the clock of the simulation environment (MORSE) and ROS is not in sync it can happen that the heuristic parameters of the estimation process differ from system to system (according to the system's performance). To solve this problem we also tune these parameters on-the-fly according to the previously mentioned estimation errors.

Upon request the node serves the traffic estimation information of a preferred time window with a given temporal resolution as a Dynamic map. Each slice of this map has an equivalent structure as a Roadmap graph (initially implemented in the codebase) and codes the traffic information in the weight of the graph edges, i.e. the nodes of a higher traffic area will be connected with larger weight edges. Edge weights are increased in each slice of the Dynamic Map if the edge is in some radius of the estimated position of the robot at the corresponding time of the slice. The weight increase is linear and additive in case multiple robots are close to each other. The final Dynamic map is easy to integrate to any other system where the planned paths and the robot positions are broadcasted throughout the system.

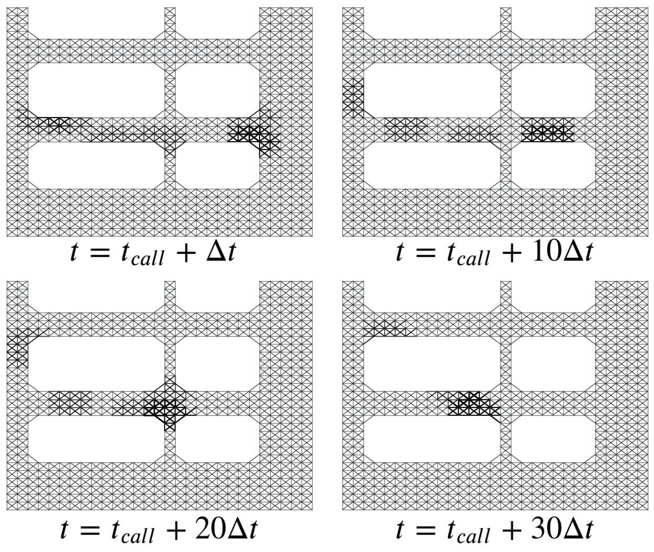

$t = t_{call} + \Delta t$
$t = t_{call} + 10\Delta t$
$t = t_{call} + 20\Delta t$
$t = t_{call} + 30\Delta t$

Fig. 15. Example of four slices in a Dynamic map for a scenario with 6 robot-agents. The estimation time is given as a difference from the time of the request call and $\Delta t = 0.25\,sec$ in the example. The edges between the graph nodes are darker if their weight is larger (around the estimated location of the robot at the time corresponding to the slice of the Dynamic map).

D. Results

Experiments and Descriptions

To measure the throughput of our system we measured the used energy, the travelled distance and the time needed (on average) for a robot to fulfill a single task/delivery. To do so we ran the system four times for each scenario (number of robots). The simulations were stopped when there were no new packages generated. This behavior of the system is due to the fact that the task queuing of the robots is not implemented. Runs are not the same length, therefore we used normalization w.r.t the deliveries. Some bias in the results may be due to the initialization of the simulation when robots are moving to the package area from a more distant initial position. 2 unwanted runs have been removed from the data as in these cases either a robot has been stuck at a tray or a package has dropped off the end tray upon delivering, in this way blocking the path of other robots. Simulations have been carried out on a Dell G5 15 laptop (CPU: 2.2GHz Intel Core i7-8750H, GPU: Nvidia GeForce RTX 2060 6GB, RAM: 24GB). All simulations are using the simple full warehouse layout of the codebase. Data has been saved by a custom made Delivery Evaluator Node and has been processed later with a custom python script (energy_evaluator.py). During the runs we saved the following data:

- battery level of the agent over time
- amount of time needed for a delivery
- overall distance travelled during a delivery

The data is available as individual json files for each run and the data can be summarized in a single json file with the aforementioned script.

To measure the average amount of time needed for a delivery in a given scenario we first took all runs' delivery data. This includes the mentioned metric for each package delivered during the simulation. We then took the average and standard deviation. Therefore, the standard deviation here is from the number of deliveries. The same applies to the average distance travelled during the deliveries. As in the codebase the battery consumption for the delivery data was not correct we needed to measure it differently. We first took the battery level data of all robots in a run and differentiated it to get the energy consumption over time. We truncated this data so that only negative slopes are taken into consideration (excluding the charging process). We then further calculated the overall (sum) energy consumption and normed it with the number of packages delivered during that specific run. This means that we now had a score of normed energy consumption for each run and not for each package. To summarize the results we calculated the average and STD of this energy consumption score across runs. Note that here we take the average across runs and therefore the standard deviation is not comparable to the previous 2 metrics where the average is across deliveries in all runs of a scenario. This method has known biases, such as the method includes energy used outside of delivery. For example it does not exclude the energy needed to go to a charging station (which would not belong to the energy needed for a delivery, but rather energy for sustaining the robot). As the energy used for sustaining the robots is changing in different scenarios this can be also reflected in the results.

Graphical analysis

According to our results the system needs a different number of robots to be optimal w.r.t. different aspects of the system. Taking only the average distance traveled for a delivery [Fig. 16] the system was the most efficient (on average the less distance needed to be traveled by the robot to successfully deliver the package) when the simulation was initialized with 6 idle robots. The results, however show little difference between scenarios and therefore we suspect that this metric is rather related to the warehouse layout than the optimality of the path planning algorithm we used. This fact is clearly visible from the standard deviation which is rather high and comparable for each scenario due to the layout of the warehouse. (Namely the packages were to be delivered from the start trays to a relay tray with short distance, or from the start trays to the end trays with large distance.)

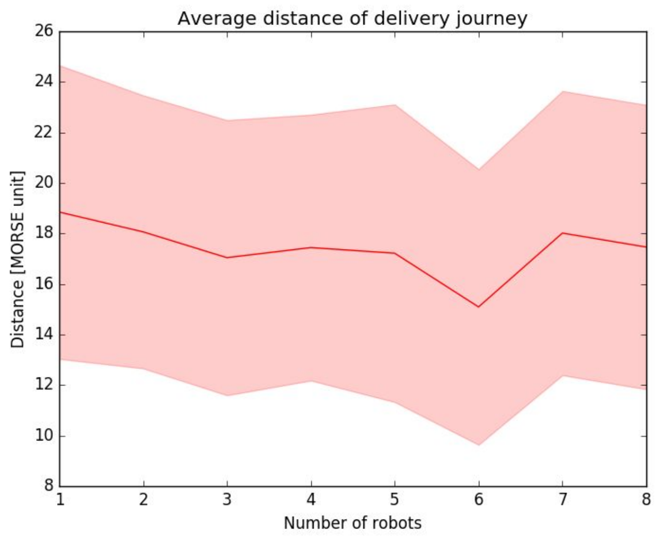


Fig. 16. Average distance (line) needed for a package delivery in 8 different scenarios (number of robots) with the standard deviation depicted with lighter red area.

Taking only the average time needed for a successful delivery it can be shown that in general the more robot-agents we used the less optimal the result was. This is due to unforeseen traffic scenarios when the agents needed to use local obstacle avoidance to prevent collisions. This maneuvering is in general slower than moving on a planned path.

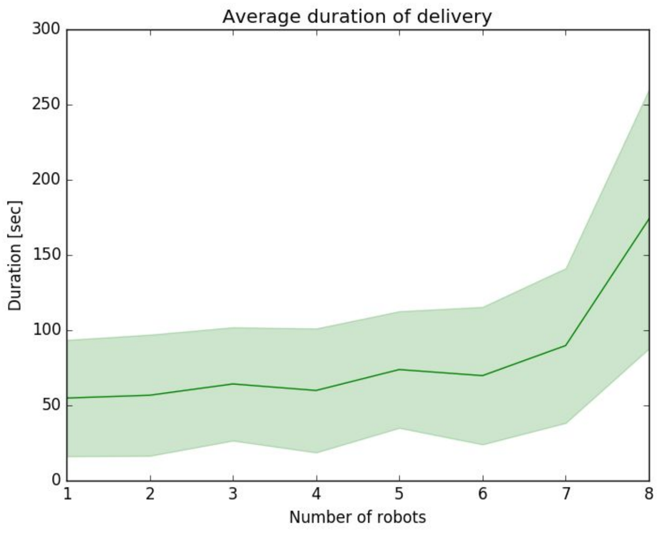
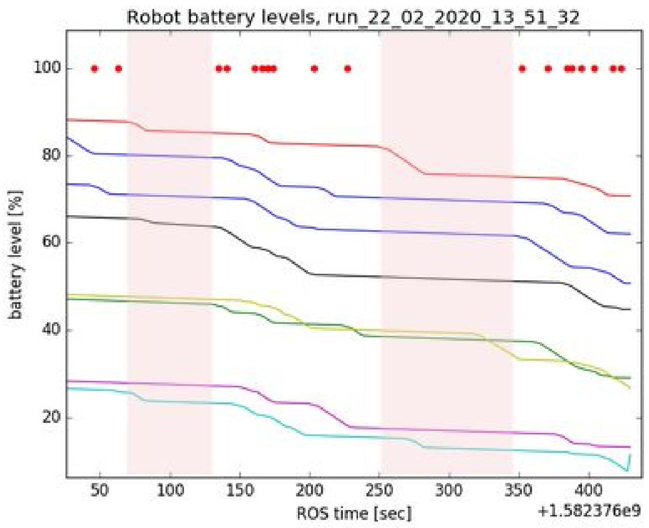


Fig. 17. Average duration (line) needed for a package delivery in 8 different scenarios (number of robots) with the standard deviation depicted with lighter green area.

Here we would like to also emphasize an implementational bottleneck in the current codebase, namely the mutex solution used in the simulation environment for robot gripper requests. As we increased the number of robots we observed a long inactive behavior upon loading or unloading packages when there were more than one robot trying to grip/ungrip a package. Without further quantification we also observed that the length of this inactive duration is not linearly dependent on the number of robots waiting to grip. As an example visualized on [Fig. 18] this results in long waiting periods with typically very low energy consumption in the overall system. This of course will lead to time periods when there are no packages delivered, even if the robot is already at the goal destination. This biases the “average duration of delivery” in a very visible way, and therefore this issue is of high priority to solve in the overall solution first before further optimization.



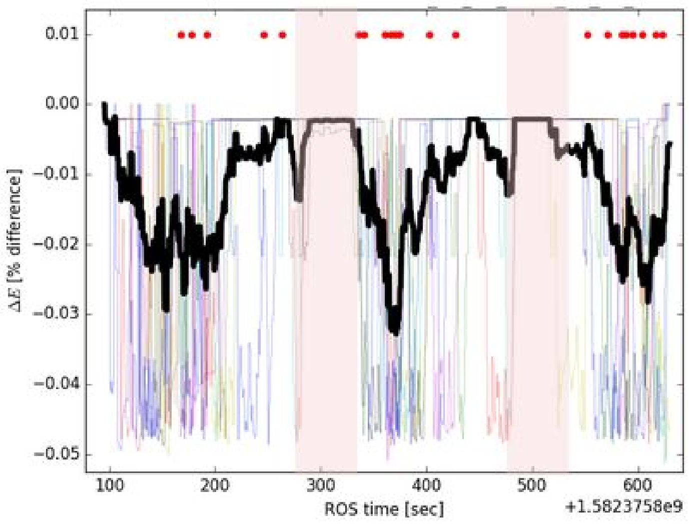


Fig. 18. Demonstration of inactive periods throughout a simulation run with 8 robots initialized. **Top:** Change of battery level of all robots in the system over one part of a simulation run (lines). The time points of a successful delivery is denoted with a red dot on the top of the figure. **Bottom:** The energy consumption of individual robots (faded lines) and the average energy consumption of the system (black line) in the same simulation run for the whole time domain. Inactive periods are emphasized with light red background in both subplots.

Taking only the average energy consumption for a delivery into account we found that the only scenario improving upon a single-robot case occurs when initializing the system with 3 idle robots. What we observed in this case is that there is an emerging rotation between the robots w.r.t. fulfilling long and short distance tasks, which might explain the overall better energy consumption of the system. This is not visible for the scenarios when there are 2 robots only. Even if this behavior is emerging in scenarios with more than 3 robots, we suspect that merely due to the avoidance behavior used to avoid traffic scenarios that rise from the increased number of robots, the average energy consumption per package is increased.

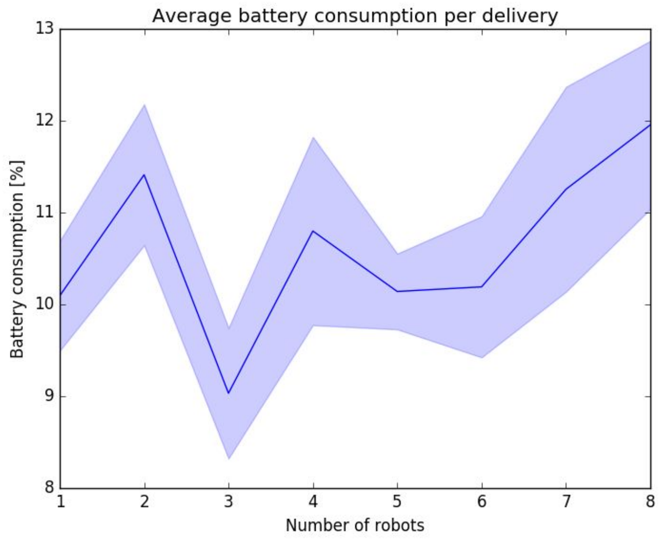


Fig. 19. Average energy (line) used for a package delivery in 8 different scenarios (number of robots) with the standard deviation depicted with a lighter blue area.

Discussion

According to the presented results we propose that initializing the system with 6 robots provides the optimal trade-off between throughput and energy consumption. Even though scenarios with more than 6 robots offer a higher throughput potential, they will also introduce unwanted traffic scenarios as well as inactive waiting periods due to the mentioned implementational bottleneck. Our solution could be further improved by integrating the traffic management node fully. The current path planning method introduces a high spatial overlap between paths as it is calculating the optimal path for each robot using the same roadmap graph without taking current or future traffic scenarios into consideration. Therefore we suggest that using traffic data for path planning would improve the system's overall behavior by achieving a better trade-off between controlled traffic density and path optimality. Furthermore, as the path planner assumes constant velocities the system relies on speed adjustments at the motion planning side to traverse the planned path nodes at the desired ROS timestamps. This could also be improved by adjusting these heuristics or using traffic information. To accomplish this integrational step first the path planning algorithm shall be replaced or customized in such a way that temporal change of the roadmap graph can be introduced and used for path planning. This would lead to a decreased number of traffic scenarios that are handled by the obstacle detection system. We suspect that as a result both the delivery duration and the energy consumption during the delivery process would decrease.

The collision avoidance behavior implemented in the system works to prevent most collisions and allows evasive maneuvers for multiple scenarios, but since a central architecture with a global path planner was pursued in this work, the development of a sophisticated avoidance mechanism was not focussed and improvement is possible. As static obstacles are currently ignored by the laser scanner, robots can be forced into walls as they try to avoid dynamic obstacles. There are also scenarios where robots try to switch lanes and get into oscillating behaviors between merging and avoiding the robot on the other lane. Next to simple strategies such as a grid search to improve the parameter tuning, new approaches could be taken. A more advanced analysis of the laser points (e.g. Hough transform, shape fitting, convolutional neural networks) could be undertaken or an inter-agent negotiation protocol could be implemented to improve performance. One idea would be to use a dynamic potential field of static and dynamic obstacles in combination with a negotiation protocol to avoid local minima and deadlocks. Using the laser scanner to find out the position of other robots is also a suboptimal and unreliable solution (e.g. as the laser does not cover all directions). Instead the globally available poses of all robots should be distributed to the robots so they can incorporate this much more reliable data in their avoidance algorithm. The laser would serve only to detect broken robots and other unexpected, non-communicating objects. As the throughput potential of a higher number of robots is mainly limited by inefficient collision avoidance behavior and the occurrence of potential collisions in the first place, an efficient evasion protocol in conjunction with global information networks and an anticipatory traffic management to guide the path planning seem like promising future steps to push boundaries and overcome bottlenecks.

To enable the system to be used in the real world it needs to take into account the noise that is associated with actual sensors. To account for issues such as wheel slippage, GPS uncertainty, and IMU noise an Extended Kalman filter can be used to improve the pose prediction by using sensor fusion. In a real world scenario the (un)loading process would also be more complicated. To allow reliable operation of the gripper, visual aids such as marker systems or depth perception can be useful tools. Another thing to consider is the need for maintenance and the occurrence of unexpected events (such as a power outage or robot malfunction). The system needs to be flexible enough to react to such situations without breaking down and provide means to reset in a smooth and fast manner.

E. Conclusion

The currently strict floor plans used for robot navigation in autonomous warehouse systems cannot cope with unforeseen situations and make the delivery process shifted from purchase due to the duration of planning in the meanwhile (offline planning). Here we presented a warehouse system with delivery robots that can navigate freely in the environment to fulfill dynamically assigned online-planned tasks. Unforeseen events can be handled by the agents locally using custom obstacle avoidance behavior. To increase the overall stability of the system and to handle energy management scenarios we implemented a Markov Decision Process. Hereby we have shown that our system has different throughput and energy consumption for different numbers of robots in the environment and we presented an implementational bottleneck which avoids the system to be efficient with more than six robots. Furthermore, we have proposed possible improvements which would decrease both the time and energy needed for a given delivery on average, such as the integration of an already implemented traffic management node or the improvement of the obstacle avoidance system. Finally, we have assessed problems that can arise when adapting the system to the real world and proposed possible solutions.

Appendix

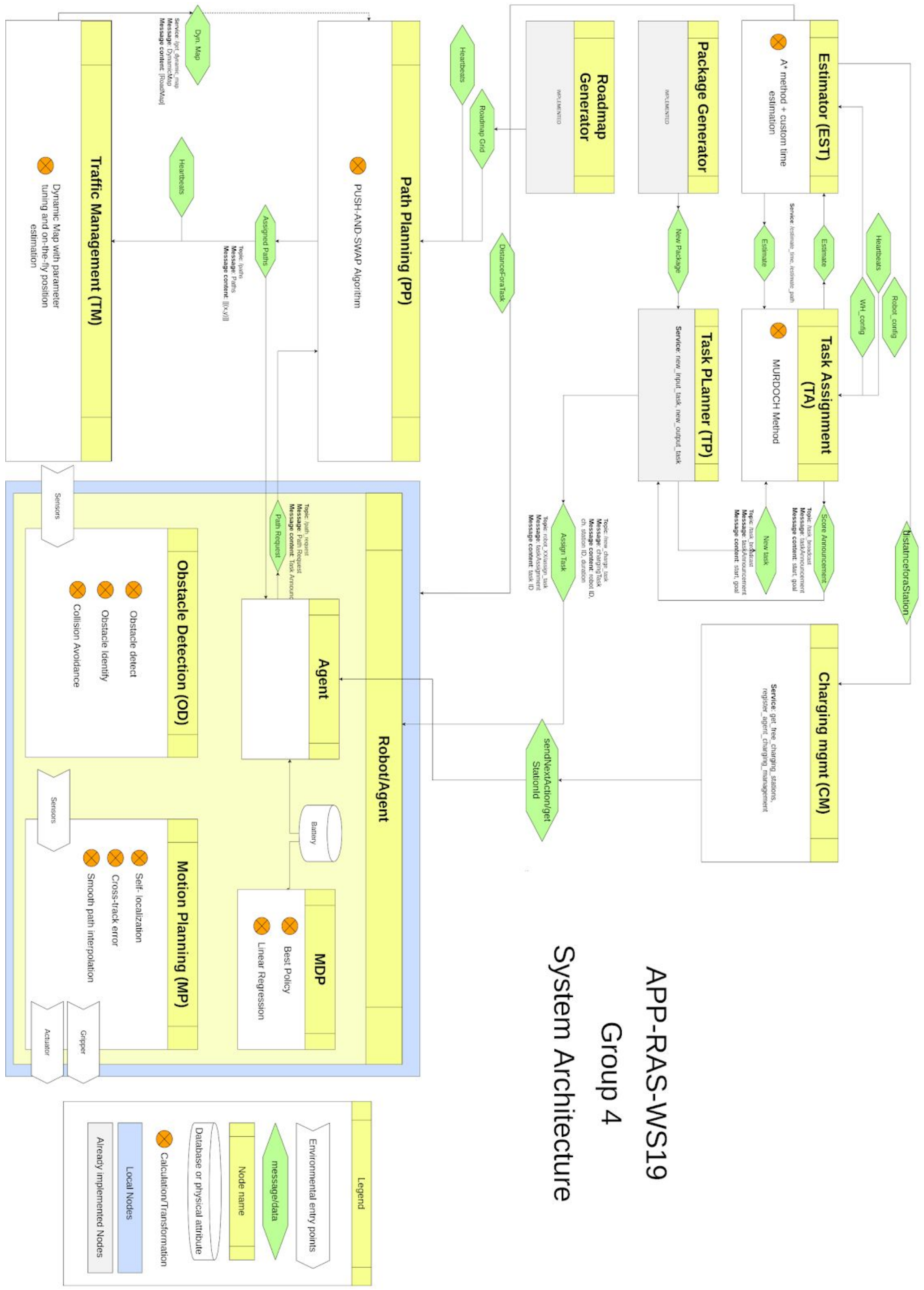


Fig.20. System Architecture

References

[1] S. Crets, “Are the robots taking over?,” Digital Commerce 360, 04-Feb-2019.[Online]. Available:https://www.digitalcommerce360.com/2019/01/31/are-the-robots-taking-over/. [Accessed: 23-Feb-2020].

[2] M. O'Brien, “DHL Investing $300M in Warehouse Robotics, Automation,” Multichannel Merchant, 05-Dec-2018. [Online]. Available: https://multichannelmerchant.com/operations/dhl-investing-300m-warehouse-robotics-automation. [Accessed: 23-Feb-2020].

[3] M. Simon, “Your First Look Inside Amazon's Robot Warehouse of Tomorrow,” Wired, 05-Jun-2019. [Online]. Available: https://www.wired.com/story/amazon-warehouse-robots/. [Accessed: 23-Feb-2020].

[4] B. Oyster, “50,000 Warehouses to Use Robots by 2025 as Barriers to Entry Fall and AI Innovation Accelerates,” ABI Research: for visionaries. [Online]. Available: https://www.abiresearch.com/press/50000-warehouses-use-robots-2025-barriers-entry-fall-and-ai-innovation-accelerates/. [Accessed: 23-Feb-2020].

[5] R. J. Luna, and K. E. Bekris. "Push and swap: Fast cooperative path-finding with completeness guarantees." Twenty-Second International Joint Conference on Artificial Intelligence. 2011.

[6] B. P. Gerkey and M. J. Matarić, “A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems,” The International Journal of Robotics Research, vol. 23, no. 9, pp. 939–954, 2004.

[7] B. Kalyanasundaram, and K. Pruhs,"Online Weighted Matching " ,Journal of Algorithms on Elsevier, pp. 478-488, 1993.

[8] L. E. Parker,"Path planning and motion coordination in multiple mobile robot teams.", Encyclopedia of complexity and system science, 5783-5800, 2009.

[9] J. E.Hopcroft, J. T. Schwartz, and M. Sharir, "On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the" Warehouseman's Problem", The International Journal of Robotics Research 3.4, pp. 76-88, 1984.

[10] P. Švestka and M. H. Overmars,"Coordinated path planning for multiple robots." Robotics and autonomous systems 23.3, pp. 125-152, 1998

[11] B. de Wilde, A. W. ter Mors, and C. Witteveen,"Push and rotate: cooperative multi-agent path planning." Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[12] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, “Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing,” 2007 American Control Conference, 2007.

[13] T. Mercy, R. V. Parys, and G. Pipeleers, “Spline-Based Motion Planning for Autonomous Guided Vehicles in a Dynamic Environment,” IEEE Transactions on Control Systems Technology, vol. 26, no. 6, pp. 2182–2189, 2018.

[14] P. Sanders and D. Schultes, “Engineering Fast Route Planning Algorithms,” Experimental Algorithms Lecture Notes in Computer Science, pp. 23–36.

[15] R. S. Sutton and A. G. Barto, Reinforcement learning: an introduction. Cambridge, MA: The MIT Press, 2018.

[16]Fit linear regression model - MATLAB. [Online]. Available: https://www.mathworks.com/help/stats/fitlm.html. [Accessed: 23-Feb-2020].

[17] K. Valogianni, W. Ketter and J.Collins,“Smart Charging of Electric Vehicles Using Reinforcement Learning”, Trading Agent Design and Analysis: Papers from the AAAI, 2013 .

[18] M. Selmair, S. Hauers and L. Gustafsson-Ende, “ Scheduling Charging Operations of Autonomous AGVs in Automotive In-house Logistics”, 2019.

[19] H. Hamann,, C. Markarian,& F. Heide,and M.Wahby, "Pick, Pack, & Survive: Charging Robots in a Modern Warehouse based on Online Connected Dominating Sets.", 9th International Conference on Fun with Algorithms (FUN 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[20] R. Cassini, F. Tampalini, P. Bartolini and R. Fedrigotti, "Docking and charging system for autonomous mobile robots." Department of Electronics for Automation, University of Brescia, 2005.

[21] K. L. Lopez, C. Gagne, and M.-A. Gardner, “Demand-Side Management Using Deep Learning for Smart Charging of Electric Vehicles,” IEEE Transactions on Smart Grid, vol. 10, no. 3, pp. 2683–2691, 2019.

[22] C. de Boor, “B (asic)-Spline Basics”, Wisconsin Univ-Madison Mathematics Research Center: Madison, WI, USA, 1986.

[23] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, and P. Stang “The 2005 DARPA Grand Challenge – The G. R. Race.” Industrial Robot: An International Journal, vol. 35, no. 6, 2008.

[24] J. Connor and G.Elkaim, “Trajectory Generation and Control Methodology for an Autonomous Ground Vehicle”, AIAA Guidance, Navigation and Control Conference and Exhibit, 2008.