Potential Field Path Planning for Swarm Production Robotics

Anders Christoffersen
agch20@student.aau.dk

***Abstract*—In moving towards the Swarm Production Paradigm, the need for a planner capable of developing paths for multiple robots of different capabilities is required. Such a planner needs to generate paths for robots while remaining effective when topological or environmental reconfiguations occur. This project investigates the development of a model predictive controller for multiple robots using potential fields as the primary path generation method for short intervals into the future for multiple robots. This controller is implemented in ROS2, and the supporting topics and data handling created to allow dissemination of the desired control outputs to multiple robots. The controller and method are then validated in both a Python based occupancy grid simulation, as well as in a ROS2 physics simulator for multiple robots.**

I. Introduction

Swarm Production, as proposed in [2] is the basis for a new paradigm of manufacturing, allowing high flexibility not only in tool configurations and workflows, but also in the layout of machinery in the physical environment. The limitations in current manufacturing paradigms include rigid linearity, fixed machinery locations, or lack of flexibility of tool selection at each stage of the manufacturing production. By establishing a variety of mobile manufacturing platforms for each potential operation in the production process, as well as a set of carrier robots to move materials between stages of the process, swarm production provides flexibility in physical layout, step order of the production flow, and multitasking between different sets of machines. Such a flexible environment requires adequate management of the robots, including path finding for each carrier and production robot, establishment of priorities in both moving and job scheduling, as decisions being made by each robot without coordination may result in collisions, deadlock, or inefficient planning if each robot lacks an awareness of the others.

This paper presents a centralized method of controlling multiple robots in a manufacturing context through the use of a model predictive potential field pathing algorithm combined with A* navigation. The aim of this project is to allow a variety of robots to move around each other to accomplish a goal such as material delivery, or to move around an environment with the goal of robot topology reconfiguration. The novelty of this paper's solution compared to previous work is the incorporation of both model predictive potential fields and A* pathfinding in the context of discrete task completion. This novelty is combined with simultaneously allowing for reconfiguration of the manufacturing environment according to a higher level job scheduler and topology manager. Unlike several approaches to multi-robot navigation, this paper does not adopt a leader-follower approach, and assumes a known environment in which all operations occur, as opposed to an unknown or exploratory environment.The jobs undertaken by carrier robots are allocated to available robots based on the highest available job in a list, while jobs for process robots are set via jobs marked for specific roles or robot IDs. It is assumed that the priorities of these jobs are set by a different system. The proof of concept for this solution is achieved by a occupancy grid simulation run in Python, which provides a simple and fast test ground for parameter tuning and validation of the method, as well as simulations in the MAES simulator for ROS2 develope by students at Aalborg University [3]. The MAES simulator provides ROS2 RVIS and Unity visualization output, as well as the ability to capture data in a rosbag for further playback and analysis. The result of this paper is a control system for multiple robots in a swarm production environment allowing for pathfinding of robots for both delivery and topology reorganization according to the needs of the overall manufacturing flow.

II. Related Work

While swarm production itself relatively new and only proposed in 2021 by [2], much work has previously been done on managing swarms of robots, and finding effective ways of planning paths for one or multiple robots in an enclosed environment. Potential field based robot planning has been the subject of academic research since the 1990s, including early implementations on the CARMEL platform in 1991 as described in [9]. Koren and Borenstein describe an effective method to calculating a navigation vector based on an cell based occupancy grid, in which the robot receives influence from the goal it should navigate to as well as any obstacles nearby. This approach is mathematically straightforward, relying on the calculation and combination of the different vectors, but has shortcomings when navigating through close walls or tunnels. Limitations of the potential field approach are described in [9], and potential solutions to some of the shortcomings in [1]. In the expected environment for a swarm production system, several of the identified shortcomings expressed in [9] are not expected to be common, specifically the oscillations in narrow corridors, or the oscillation when between obstacles. [9] only considers the potential fields in the context of one robot, while swarm production requires planning for multiple robots and types of robots. In considering multiple robots, the work of Elkilany et al. in [1] offers

insight to the existing literature for refining potential field implementations. Additionally, [1] discusses tuning methods for the forces and gains experienced by a robot when in the presence of obstacles while pursuing a moving target. While swarm production does not require the tracking of a moving target to be successful, the possibilities of continuing to deliver materials to process robots while the manufacturing environment reconfigures is of potential interest. Several non potential field based approaches are present and integrated into the second version of the Robot Operating System, or ROS2. ROS2 provides a base for standard development of robots, as well as a set of integrated planning and communication frameworks and tools [7]. Included in the toolbox of ROS2 is a set of navigation tools, which plan pathing for individual robots to calculate paths based on the A* algorithm or Dijkstra's algorithm [6]. While widely used in the field of path planning, both of these algorithms create paths only for one robot at a time. Model predictive control provides a set of control tools for a system, as opposed to a single component, and assumes that short term predictions eventually approach optimal solutions over longer time intervals [10]. As model predictive control allows simulation and control of a complicated system, such as multiple mobile robots, with near optimal results over the long term while planning for the search term, this approach is of interest to this project due to it's ability to control complicated systems without planning a complete solution in one step.

III. Methods

The goal of this paper was to create a ROS2 centralized model predictive potential field based path planner for a variety of robots in the context of swarm production. The controller should be capable of handling robots with different capabilities, plan pathing for each robot to a receding horizon with assumed knowledge of the robot positions and environment obstacles, and react to changes in the environment or job order flow. A central ROS2 node will provide the path information to each robot, and receive information from each robot to update the controller's knowledge of each robot after each step is taken, and adjust the predictions based on any deviation from the intended path. The information for each robot will be sent to every robot, allowing for robustness in communication between robots in the case of communication failures, the information could be passed between individual robots instead of directly from the controller. In designing and implementing the controller, it is assumed that effective localization is present in the environment, and the location of robots and obstacles can be known and given to the controller for the calculations.

Simulations and calculations were performed with ROS2 Galactic, a Lenovo Thinkpad P53 20QN, with an Intel i9-9850H CPU, 16GB RAM, an Nvidia T2000 GPU, and Ubuntu 20.04 LTS with ROS2 Galactic Geochelone and Python 3.8.10. Simulations were performed in the MAES simulator. All information between the controller and simulated robots took place over the ROS2 topic system.

A. System Architecture

The overall goal of the system architecture design is to create a centralized control structure in which pathing can be generated, and regenerated, to best fit the needs of the production flow. The controller therefore is designed with integration to job scheduler and topology manager not included in this project, which would dictate the organization of the environment, and the order in which jobs are presented to the controller for execution. The controller therefore only needs an understanding of the environment, primarily the robot and obstacle locations, and have access to a list of jobs to be done. Information on the locations of all relevant objects is assumed to be reliable, and not part of the system architecture beyond the ability for robots to send their location info back to the central control node. The central control node performs all job allocation to robots, as well as all calculations for the paths using potential fields, or a handover to local robot navigation such as A* or Dijkstra, as described in [6]. Depending on a variety of navigation properties, such as remaining distance, time elapsed, and potential future additions, the A* or Dijkstra algorithm built into the ROS2 navigation stack serves as a "last-mile" method of achieving the navigation and orientation while close to the goal. As discussed in [9], the usage of potential fields as all or part of a navigation system is a known method of solving multi robot navigation problems. The architecture presented in this paper improves on the usage by allowing a hybrid control solution for multiple robots in a model predictive context, and flexibility with different jobs and robot types for the swarm production paradigm, in which large scale movements and orchestration is handled by a central controller, and control handed off to a local controller for final approach of the robot to it's goal.

B. Communication Architecture

Communication of all data is handled by the ROS2 topic system. There are two central communications topics: one from the controller to the robots, containing all of the intended navigation information, and one available to the robots to update the control node with their positions. The implementation in this paper assumes a static environment devoid of obstacles that would be unknown to the controller, but it is acknowledged that an additional obstacle detection topic could be easily added to inform the controller of unknown obstacles encountered by the robots.

1) JSON Architecture: While the controller assumes contact with all robots to continuously send the most recently calculated path information, the communications in a real world environment may not allow optimal distribution of commands. A JSON file of all robot IDs and each robot's next navigation steps is therefore sent to all robots, instead of sending individual commands to each robot. As each robot is aware of every other robots next set of commands, it becomes possible for the commands to be distributed locally from robot to robot if one or more are unable to reach the central control node. The format of the JSON file is presented below with an example JSON structure for 3 robots with IDs 1, 2, and 3, and

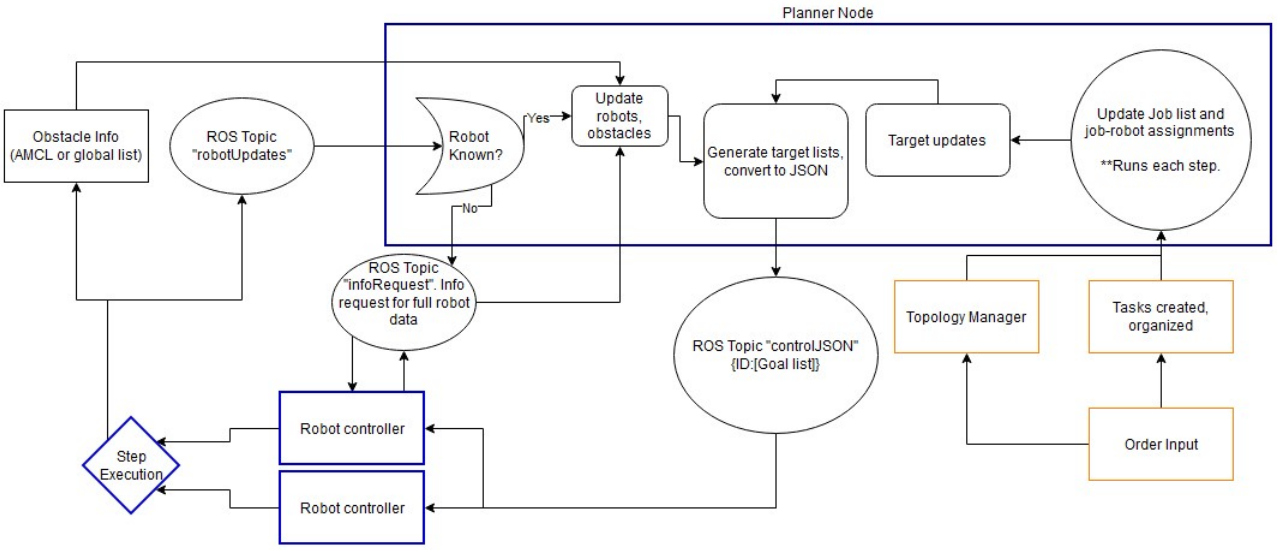

Fig. 1. Information flow and high level ROS2 topic diagram of the controller, including assumed interface to production and topology management.

4 steps of calculation complete for each. The structure of the JSON is of each entry being in the format of

$$"robotID" : [[x_1, y_1], [x_2, y_2]...[x_n, y_n]] \quad (1)$$

Therefore a set of three robots with 4 calculated next points would be constructed as:

```
{
    "1" : [[0,0][0.1,0.2][0.2,0.3]],
    "2" : [[1,2][0.6,1.8][0.2,1.6]],
    "3" : [[5,6][4.5,6.3][4.2,6.6]]
}
```

When calculated and converted to a JSON in the main control node, the JSON is further converted to a string object using the Python $json$ library. The ROS2 topic system does not include support for JSON topics, so a conversion from the JSON format to a supported format of String is required.

2) Unknown Robot Inclusion: The topic for the robots to update their position in the controller additionally allows the controller to become aware of new robots entering the arena, and potentially request further information about their capabilities if needed. The simulations presented in this paper assume identical robot capabilities for all courier robots, and the specifics of the capabilities of the process robots are not considered relevant to the design of the path planning.

C. Job allocation

This paper assumes that the order in which jobs are to be processed is determined by a higher level authority than the path planner, as while swarm production allows a high degree of flexibility in the organization and order of jobs, this paper focuses primarily on how to move the robots transporting materials, or the manufacturing robots, around the manufacturing space effectively. In this paper a set of jobs is presented to the controller, emulating pickup and dropoff of materials from one location to another. The structure of the model predictive calculations, as well as the centralized knowledge of each robot and each robots intended next steps, allows for additional flexibility in routing materials, as the goals of a material delivery robot can be linked to the location and orientation of a manufacturing robot, regardless of whether or not it moves.

D. Path Generation

Generation of the paths each robot should take is done by the central control node and distributed to each robot in the schema described in section III-A and III-B1. The calculation of the paths is described by two main components: the computation algorithm to generate a "step" for a robot to take in a given time interval, and the model predictive process of the usage of the computation algorithm.

1) Computing "Steps": A "step" for a robot is defined in this context as the movement that a robot is directed to take in a given time interval. Based on the maximum usable speed of the robot and the frequency that the model is run at, each step is calculated such that the steps at a given time t are calculated for all robots before any step at time $t+1$ is calculated. The calculation of a step is based on several factors: The location of the robot relative to its goal, the location of the obstacles, and the location of other robots. Each of the objects that are in those lists are considered on an individual basis relative to the robot who's step is being calculated, and a vector created to represent the influence of that object.

2) Goal Contribution: The influence of the goal is calculated first, as it is always providing some influence to the robot. The goal is considered in three scenarios, dependent on the distance d, the radius r of the goal, θ angle to the goal relative to the global frame, and the spread s of the

goal. If the robot is within the radius of the goal then the navigation directions from the central controller are considered complete, and control can be handed off to the robot itself for any fine tuning of it's position, for example docking with the manufacturing robot. The vector of the angle to the goal from the robot is established in the global frame as

$$\vec{\theta_g} = [cos(\theta), sin(\theta)] \tag{2}$$

The equations that govern the vector generated by the goal are as follows:

$$\vec{v_g} = \begin{cases} d < r + s : 30 * (d - r) * \vec{\theta_g} \\ d > r + s : 50 * (d - r) * \vec{\theta_g} \\ d < r : 0 * \vec{\theta_g} \end{cases} \tag{3}$$

3) Obstacle Contribution: In considering each other robot in the environment, while the controller is aware of the difference between robots and obstacles, the other robots are considered something to avoid for each robot. They are therefore considered obstacles, albeit obstacles that move and have their influence recalculated on each iteration of the model predictive calculations. The awareness of the difference between robots and obstacles allows different multipliers on the vector contribution to the final result, as well as the ability to change the influence of a robot on others based on job or robot type. The influence of robots, v_r, and obstacles, v_o is calculated as follows, where θ is the angle in the global frame between the robot who's vectors are being calculated and the entity creating the influence.

$$\vec{\theta_r} = [cos(\theta), sin(\theta)] \tag{4}$$

As each other robot is considered an obstacle at each point in time, the calculations for robot and obstacle influence are the same. The equations that govern the vector are generated by are as follows:

$$\vec{v_r} = \begin{cases} d < r + s : -10 * (d - r) * \vec{\theta_o} \\ d > r + s : 20 * (d - r) * \vec{\theta_o} \\ d < r : 50 * \vec{\theta_o} \end{cases} \tag{5}$$

4) Combination and Iteration: After all components of the main vector have been calculated, the different contributions are added together, normalized to a unit vector, and then scaled by the speed of the robot multiplied by the time step used in the model. This normalization and scaling is done to create a step that matches the maximum distance the robot is able to move in the simulation step, and the intended orientation is set to match the orientation of the intended movement vector.

$$\vec{v_r} = \vec{v_g} + \Sigma\vec{v_o} \tag{6}$$

$$\vec{v_{step}} = Speed_{robot} * \frac{\vec{v_r}}{\|v_r\|} \tag{7}$$

With the vector and orientation established, the robot's next intended position is defined by the end of the $\vec{v_{step}}$ vector defined in equation 7 when the vector is positioned to start at the robot position used to calculate the vector. This intended position after the next step, as well as the intended orientation of the robot, are added to internal lists in the controller's representation of the robot. This approach allows the calculations to be run using intended positions of all robots, as the controller is aware not only of the current positions of the robots, but also the intended positions at every calculated time. When calculating to the horizon of the predictive model, each robot includes the position of the other robots at $t + n$ when calculating their individual step for $t + n + 1$. These lists held in the controller representations form the basis of the JSON control message sent to all robots, as described in III-B1

E. Parameter Selection

Four main parameters were identified as key parameters for effective simulation and control. The multipliers on the vector components for distance to goal, both in and out of the spread region, and the multipliers of the obstacle vector contributions, also both in and out of the influence region, and the multiplier used when a robot is within the radius. These parameters were chosen largely by a rough parameter sweep, where simulations were run for multipliers ranging from 10 to 100 in increments of 10, and all collisions and successful navigation to goal were recorded. These parameters were then slightly refined by hand for further tuning.

IV. Simulation

The model's effectiveness was validated using two simulation methods. First, while developing the core model predictive control functionality, an occupancy grid type environment was created in Python, where multiple robots, obstacles, and tasks can be run. This environment allows easily visualization of the environment and intended pathing as each time step is calculated and executed, however does not include ROS messaging support as shown in III-A Once the intended behaviour of the system was designed in the occupancy grid environment, the MAES simulator was used to provide ROS compatible physics simulator testing [3].

A. Occupancy Simulation Design

The main components of the Python simulation are the model predictive loop, the calculation set described in section III-D1 and III-D4, a collision detection method, and the visualization capture when running the simulation. These components are designed to be independent of one another, so that changes to each component can be implemented without adverse effect, or so that each piece can be removed, replaced, or reused in similar contexts.

1) Control Loop and Calculation: As discussed in III-D4, the controller runs multiple steps of calculations as described in III-D1. The calculation of each next intended step is done independently for each step, and for each robot. The controller additionally recognizes when to hand off control to the local A* planner on the robot, such as when the robot is within the intended radius of its goal.

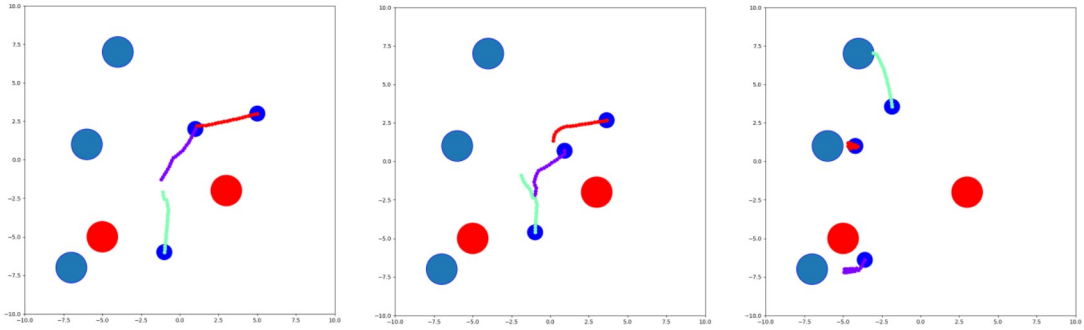

Fig. 2. Captures of the occupancy grid simulation at start (left), at the closest approach of two robots at the 7th step (center), and as robots are closing in on the goals at the end of the simulation (right). Robots are in dark blue, the goal regions in lighter blue, and obstacles in red. Each robot has a trail of colored dots extending from it, representing the set of intended next positions for the robot. Parameters for the control loop are 20 iterations at .1 second each.

2) Collision Detection: Collision detection is split into two main categories, an actual collision, and a warning of close proximity. Collisions are detected at each step of the simulation by iterating over the list of robots, and performing two sets of checks. First, against the list of static or environmental obstacles, where a collision is registered when the radii of the robot and obstacle overlap, and a proximity warning determined by an overlap of the robot radius with the influence radius of the obstacle, determined by

$$distance = \sqrt{x_{robot} - x^2_{obstacle} + y_{robot} - y_{obstacle}} \quad (8)$$

$$\begin{cases} distance <= radius_{robot} + radius_{obstacle} : Collision \\ distance <= radius_{robot} + radius_{obstacle} \\ + radius_{obstacleinfluence} : Warning \end{cases}$$

Close proximity is not in itself to be completely avoided, as there may be cases in which a small robot is able to squeeze through a gap of two larger robots, and serves only as an indicator that something in the environment has pushed a robot into an area typically to be avoided.

B. MAES Simulation

The MAES simulator [3] was developed to provide easy simulation of multiple robots with ROS2 support and minimal setup and configuration. The simulator requires that an update function be created for the robots that are simulated inside of it, where the base setup requires that all robots share the same controller. Since all robots are assumed to be the same, and the simulator does not allow for the addition or removal of robots from the simulation environment without restarting the simulator, a controller was written to listen to the ROS2 topic described in III-B to obtain the control JSON, however the functionality needed for reporting new robot information to the controller was omitted. Several robots were simulated in MAES, and a list of tasks given to them to complete. The simulator was warned to be less stable and performant running more than 3 robots, however a test with 5 robots was performed.

Fig. 3. Example of multiple robots in the same arena being calculated in the MAES simulator. 5 robots are visible in the left of the arena, and topic information including the last received messages from the controller are visble at the top of the image.

V. Results

The initial stages of validation for the project were performed in the Python occupancy grid based simulator, a visualization of which visible in 2. In these trials, several robots and obstacles were placed in an an environment, and 50 or more time steps calculated at a .1 second time resolution for the control loop. The robots exhibit pathing towards their goals while avoiding each other preemptively through recognition of the intended paths, and all approach the goals to a degree that would allow reasonable handover to the local controller for the final approach. Once validated in the Python environment, a more robust simulation was attempted in MAES. An interface was designed for the MAES simulator, and trials attempted. The initial phase of the MAES simulator with display of an incoming message from the controller is seen in 3. While the interface is able to receive incoming messages from the controller, the simulator has difficulty in implementing them due to suspected timing problems between the two, as the controller iterates through the positions of the robots and begins sending position commands of NaN.

VI. Discussion

This paper set out to create a controller for multiple robots in the swarm production context. While potential fields have some limitations in navigation, the usage of a model predictive loop to generate constant short horizon sets of commands for the pathfinding, as well as the expectation of a known environment demonstrate in initial simulations that the method has viability in guiding robots close enough to their goal that the local planner on the robot can complete the navigation and alignment. Further tuning of the potential field coefficients could further the usefulness of this method, as well as developing a form of hybrid control between the potential field planner and the best times to hand off to the local controller or another algorithm, for example to reduce the likelihood of being trapped in a local minima. Job handling and organization in this project is sufficient for the needs of the controller, however further integration with a more sophisticated job scheduler is an avenue of future work. The Python based simulations provide a proof of concept model in two dimensional space for multiple robots, and those same calculations are integrated into a ROS2 controller for use with an existing simulator. Finally, fully integrating the controller built in this paper to a controller with inclusion of bad message rejection and handling and proper frequency is proposed future work that would allow better data collection for large job sets.

VII. Conclusion

This paper provides a centralized,flexible ROS2 compatible model predictive hybrid control architecture and proof-of-concept implementation using potential fields and A* navigation in the context of swarm production robotics. This solution allows for additional methods of path planning to be easily integrated, as well as support for changing environments and job priority, both key components in the swarm production paradigm.

References

[1] B. G. Elkilany, A. A. Abouelsoud, A. M. Fathelbab, and H. Ishii, "Potential field method parameters tuning using fuzzy inference system for adaptive formation control of Multi-Mobile Robots," Robotics, vol. 9, no. 1, p. 10, 2020.

[2] C. Schou, A. Avhad, S. Bøgh, and O. Madsen, "Towards the swarm production paradigm," Towards Sustainable Customization: Bridging Smart Products and Manufacturing Systems, pp. 105–112, 2021.

[3] M. Andreasen, P. Holler, M. Jensen, and M. Albano, "Comparison of online exploration and coverage algorithms in continuous space," Proceedings of the 14th International Conference on Agents and Artificial Intelligence, 2022.

[4] N. Boysen, P. Schulze, and A. Scholl, "Assembly line balancing: What happened in the last fifteen years?," European Journal of Operational Research, vol. 301, no. 3, pp. 797–814, 2022.

[5] N. Schmidtke, A. Rettmann, and F. Behrendt, "Matrix production systems - requirements and influences on logistics planning for decentralized production structures," Proceedings of the Annual Hawaii International Conference on System Sciences, 2021.

[6] S. Macenski, F. Martin, R. White, and J. G. Clavero, "The marathon 2: A navigation system," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, 2022.

[8] Y. K. Ever, "Using simplified swarm optimization on PATH planning for Intelligent Mobile Robot," Procedia Computer Science, vol. 120, pp. 83–90, 2017.

[9] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," Proceedings. 1991 IEEE International Conference on Robotics and Automation.

[10] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on Model predictive control: An engineering perspective," The International Journal of Advanced Manufacturing Technology, vol. 117, no. 5-6, pp. 1327–1349, 2021.