Decentralised Multi-Agent Path Planning based on Self-Organised Time Division Multiple Access

By

Runze Yuan

MSc Robotics Dissertation

Department of Engineering Mathematics
UNIVERSITY OF BRISTOL
&
Department of Engineering Design and Mathematics
UNIVERSITY OF THE WEST OF ENGLAND

A MSc dissertation submitted to the University of Bristol and the University of the West of England in accordance with the requirements of the degree of MASTER OF SCIENCE IN ROBOTICS in the Faculty of Engineering.

August 9, 2023

Declaration of own work

I declare that the work in this MSc dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

Name and Date

Acknowledgement

I would like to thank ...

Abstract

Abstract should give a short summary of the motivation, the approach and important insights and results.

Number of words in the dissertation: .... words.

Contents

List of Tables

1 Introduction

1.1 Brief Introduction

The main idea of this project is to attempt to utilise the principles of a decentralised channel sharing communication protocol (*Self-organised Time Division Multiple Access*, STDMA[1]) to achieve collision-free movement among multiple decentralised agents on a 2D plane.

1.1.1 Principles of STDMA

In STDMA, a 1D channel is represented with repeating frames that are consisted of discrete time slots.

STDMA enables agents to achieve **S**elf-organised **T**ime-**D**ivided **M**ulti-**A**ccess within the channel. That is, each agent independently finds a slot which is uniquely its own and uses this slot to broadcast its message.

The key idea of STDMA is to determine empty slots and apply for them.

This idea also works for collision-free moving on 2D plane —— to determine free space and apply for usage.

For detailed explanation about STDMA, please see **PLACEHOLDER**.

1.1.2 The 2D Plane

The 2D plane used in this project is represented with discrete pixels / grids, just like the STDMA protocol represents continuous time with discrete slots.

For such a representation of a 2D plane, there is a term called **grid world**[2].

For detailed assumptions, please see **PLACEHOLDER**.

1.1.3 Agents

To meet the requirement of decentralisation, the agents are assumed to be identical.

The agents can:

- Broadcast and receive messages in the given channel, but cannot do both at the same time (i.e., cannot be listening while speaking or speaking while listening).
- Move only one step / one grid in the map in one time step.

For detailed assumptions, please see **PLACEHOLDER**.

1.2 Aims

- Based on the idea of STDMA, design an algorithm for decentralised agents to achieve colission-free movement and space sharing on 2D plane.
- Evaluate the advantages and drawbacks of the designed algorithm, and therefore get a better understanding on the problem that the alrogithm aims to solve.

1.3 Objectives

1. Implement original STDMA communication protocol with ROS2, use nodes in ROS2 as agents in the channel, achieve self-organised channel sharing and communicating among agents.
2. Design the specific algorithmic content for agents to achieve collision-free movement.
3. Implement the designed algorithm with ROS2, use ROS2 nodes as agents moving on the 2D plane.
4. Build proper test scene and performance evaluator that could extract metrics (makespan, average finish time, etc.) from simulations.
5. Examine the advantages, disadvantages and limitations of the designed algorithm, summarize the results from observations.

1.4 Motivation

The motivation of this project is to answer a question which is inspired by [3], which is:

What would happen if use STDMA for dencentralised 2D resource sharing and path planning?

And this quesiton could be seperated to the following two parts:

1.4.1 Why STDMA?

The reason for using STDMA is its **characteristics**[4]:

1. **Deterministic**: Agents arrange their data transmission based on a determined timetable.
2. **Decentralised**: Agents listen to the channel first, then independently seek and allocate free slots for themselves to use.

These characteristics are useful for multiple agents to achieve collision-free (use free slots only), slf-organised (find slots on its own) moving and resource sharing.

There are also **reasons that making this challenging**:

1. **Dimensional Difference**: STDMA is designed for sharing discrete time slots (1D), and cannot be directly applied for resource sharing on a 2D plane. Modification is needed for 2D application.
2. **Moving by Grids**: In the communication scenario, agents don't have destiniations in the timetable, i.e., don't need to move to specific slot in the timetable. But that's different for 2D space moving, where agents have their destinations and need to move grid by grid to reach their goals. Agents could easily be trapped in situations of inefficiency or even deadlock situations [5], [6].

These characteristics and challenges are making this topic interesting and worthy for investigation.

Figure 1.1: Kiva system operating in Amazon warehouse.[2]

1.4.2 Why decentralised 2D resource sharing and path planning?

There are many algorithms aim to solve this problem, and their shared aspect is a **focus on the movement of multiple decentralised agents within a grid world**.

To easily get a better understanding of this problem, please refer to this website[1]. This website presents the problem scenarios and their solutions through a simple and engaging animation (estimated time required: 1∼2 min). For detailed explanation, please see **PLACEHOLDER**.

For such situations, **there are myriad corresponding real-life problems and applications**.

- **Warehouse Automation**: Automated pick-pack-and-ship system in warehouse (like Fig 1.1.) [7]. Delivering items in sorting station[8], [9].
- **Intersection Management**: Coordinate autonomous vehicle movement through intersections [10].
- **Robot Fleet**: Automating fleets of autonomous robots like forklift fleets [11], [12].
- **Agents in video games and CGIs**: Flock simulating and animating[13], [14].
- **Swarm Robots**: Controlling self-organised robot swarms[15].

[1]https://primalgrid.netlify.app/primal

[2]https://spectrum.ieee.org/interview-brad-porter-vp-of-robotics-at-amazon

In general, the application of this algorithm is very broad. Furthermore, in the context of Industry 4.0 and flexible manufacturing, it has even better prospects for application[16]–[18], as centralized control is gradually becoming inadequate to meet new production needs.

2 Literature Review

This section will provide the details of the STDMA[1] protocol and a review of the related *Distributed Model Predictive Control* (DMPC) algorithms.

2.1 STDMA Explained

As previously mentioned in Section 1.1, STDMA stands for ***Self-organised Time-Divided Multiple Access***, and it allows multiple agents to share the same channel for communication without centralised control. The main assumtion of this protocol is that all agents have synchronised clocks. In practice, this is achieved through GPS[19].

The core idea of STDMA could be summarised as follows: Represent continuous time with repeating frames that are consisted of discrete time slots. While agents are always listening to messages from the channel, they look for free slots to occupy, and therefore use the occupied slots to broadcast their own data.

Agents using STDMA have **four phases**, which are arranged in chronological order as follows:

1. **Initialisation**: Agents in this phase have not yet joined the network. The device listens to an entire frame and determines current allocation of each slot.

2. **Network Entry**: Randomly choose an unallocated slot to broadcast their existance and reserve one slot for the next phase. If the message sent didn't collide with others (i.e., only one agent which is myself choose to use this slot for entering), then the entering is successful. If the entering failed, reverse to the previous phase.

3. **First Frame**: Use the slot reserved in the previous phase to reserve more slots for themselves. The number of reserved slots depends on the size of the data packet that the agent needs to send in each frame.

4. **Continuous Operation**: Use the previously reserved slots to work normally. If some slots are released or more slots are needed, reapply for slots.

Although the description above omitted some details (such as slot choosing strategy, calculation of required number of slots, etc.), it is clear that **the core idea of STDMA is the strategy of finding and reserving unallocated slots**.

This protocol also has several limitations, such as: **(1)** Collision in entering: In the Network Entry phase, multiple agents may accidentally choose the same unallocated slot for entering and broadcast their existance. **(2)** Capacity: When slots are not enough, conflicts would inevitably occur. There are some studies [20], [21] that proposed improvements for its limitations, but improving STDMA is not the focus of this project.

For detailed implementation, please see **PLACEHOLDER**.

2.2 Distributed Model Predictive Control

This section would cover:

- An brief introduction to DMPC.
- Why is DMPC related to this project.
- A summary of related latest research developments.

2.2.1 Brief Introduction

What is Model Predictive Control[22]?

Model Predictive Control or MPC is a form of control that based on the model and the online prediction of the controlled system.

MPC achieves its control objective by optimising the predicted outcome of the model of the controlled system[22]. **On every sampling instant**, a typical MPC controller would **use the current state of the plant as initial state**, and **solve a finite-horizon open-loop optimal control problem in real-time speed**, then **apply the first step of the generated control sequence** as the next control action.

The major challenge confronting the MPC algorithm is the stability issue, with several factors contributing to this problem, such as inadequate terminal conditions and limited prediction horizons.

MPC has been extensively studied and is a widely used control algorithm[23]. It's primarily employed in fields such as process industry, power electronics, building climate and energy management, and manufacturing[24], [25].

What is Distributed Model Predictive Control? [26]

In many scenarios, centralised controllers fail to perform control tasks due to reasons such as the computational load increasing with the expansion of system scale, resulting in the controller's inability to complete real-time computations, or the inability to dispatch control information to all components of the system within a specified time. In summary, **the limitations of centralised controllers made them incapable of meeting the needs of many control scenarios**, and that is why we need distributed control.

For *Distributed Model Predictive Control* or DMPC, a large system is decomposed to numerous subsystems, and each of them:

- Has its own dynamic.
- Has an MPC controller to control its actions (predict finite future states and choose the optimal current move).
- Could interact with and be influenced by other subsystems (usually described as *variables* and *constraints*).

DMPC algorithms have the following **key characteristics**, which can help readers to understand the algorithms and distinguish one algorithm from another:

- **Architecture**: Different relationships between local MPC controllers. **Hierarchical**: The entire system is divided into layers, each containing several subsystems with MPC controllers, and there exists a hierarchical relationship between these layers. **Decentralised**: Each local MPC controller controls the local subsystem and don't communicate with each other. **Distributed**: Local MPC controllers cooperate to find better solution for the overall system.

- **Local - Global**: How much do the local controllers know about the overall system information.
- **Selfish - Cooperative**: Whether the local controller only optimizes its own cost.
- **Iterative - Non-Iterative**: Whether the control strategy is generated through exchange and iteration among multiple local controllers.
- **Serial - Parallel**: Whether the local controllers transmit their messages in a specific order or multiple controllers could communicate with each other at the same time.
- **Synchronised - Asynchronised**: Whether the local controllers follow a shared schedule for computation and communication.

The advantages and disadvantages of DMPC both stem from its decentralised nature. Like many other decentralised algorithms, **DMPC has better scalability and robustness**, can bypass communication bottlenecks, and to some extent, allows for parallel processing. But **coordination among multiple controllers can be quite complex**, e.g., all local controllers tend to achieve Nash equilibrium for themselves, but such a tendency may not contribute to reaching Pareto optimality and might even be counterproductive for the entire system[27].

Why is this project related to DMPC?

The designed algorithm (detailed explanation please see Section **PLACEHOLDER**) in this project allows each agent to:

- Create and join a self-organised serial communication channel.
- Determine the time window for planning and plan broadcasting.
- Use the shared channel to broadcast its own moving plan.
- Receive plans of other agents and use the received information to its own movement.

Which means, the presented algorithm could be classified as a DMPC algorithm, and the problem it solves could be treated as a DMPC problem.

2.2.2 Latest Researches of DMPC

3 Research Methodology

This section would cover:

- Algorithm detail and implementation.
- Experiment designaiton.

Environment

Hardware

All the content mentioned in this project was carried out on a ROG Zephyrus M16 laptop.

The CPU in the laptop is 11th Gen Intel Core i7-11800H @ 2.30GHz.

All experiments and simulations in this study were conducted without GPU acceleration.

Software

System: Ubuntu 22.04 LTS installed on WSL2 of Windows 11 23H2 22631.2129

ROS2: ROS2 Humble

All programs involved in this experiment were written in Python within this software environment.

3.1 Algorithm Detail and Implementation

The algorithm can be described as:

1. Agents establish self-organised communication among themselves using STDMA.
2. Use the information transmitted in the channel for planning, and subsequently send their plans at the designated times.

In this section, we will sequentially detail the principles and methods of implementation for these two components.

3.1.1 STDMA Implementation

Like mentioned before in Section 2.1, in STDMA, agents have synchronised clock, share one channel, determine empty slots in frames, and apply for the slots needed, then use their own occupied slot for broadcasting.

Assumptions

Before delving into the details of the algorithm, it's necessary to first mention the assumptions of this algorithm.

- **Synchronised Clock**: Agents have synchronised clock.
- **Slots and Frames**: Continuous time is represented with frames of equal length, and each frame includes an equal number of discrete slots, with each slot having the same duration.
- **Uniform Frame Length**: Frame length is predetermined for all agents and the value is the same.

Synchronised Clock and Shared Channel

These features are implemented with topics in ROS2[1].

- **Clock**

  A ROS2 node is created as clock signal generator and publisher.

  At specified intervals, this node toggles an internal Boolean value and sends this Boolean value to the clock topic. This results in a square wave with a 50% duty cycle in the clock topic, where the period is adjustable.

  Agent nodes subscribe the clock topic to get the synchronised clock.

[1]http://docs.ros.org/en/humble/Concepts/Basic/About-Topics.html

- **Division of Slots and Frames**

  A pair of True and False signal in the clock topic is considered as one slot by the agents.

  The frame length is predetermined within the agents, and all agents have a consistent frame length parameter.

  At the end of each slot, the agent increments its internal slot counter. If the slot count reaches one frame, it clears the slot counter and increments the frame counter. The algorithm does not require the frames to be synchronised between agents, i.e., the starting slot of the frame can differ for various agents.

  In this implementation, all agents require only one slot.

  ADD PICTURE HERE

- **Shared Channel**

  In the original STDMA[1], only one channel is used for communicating and broadcasting. But in the implemention here, **the channel is seperated in two, one for slot occupation messages, and one for other common messages**.

  In the STDMA part, only the channel for slot occupation message is used.

  Both of the channels here are implemented with ROS2 topic: All agent nodes subscribes to channel topics and could publish message to the channel topics.

  Please note that this modification (splitting the channel into two) is purely for convenience and **does not impact its functionality in any way**. This modification in the current implementation is equivalent to using just one channel.

Node Actions within One Slot

Before understanding how agents allocate slots, it's essential to know how agent acts within a slot.

As mentioned above, a slot is divided into two equal parts: one half with the clock signal being True, and the other half with the clock signal being False.

Given the characteristics of ROS2, the node operates in the following manner: each time it receives a new clock signal (when the clock node's boolean value changes and broadcasts the new boolean value), it executes a corresponding callback function.

- **True** Received from the Clock Topic:

  The clock signal's **rising edge**, treated as the **start of a slot**.

  Agents **send their message** if this slot belongs to them.

  To be specific, agent nodes publish their message to the topic, and whenever an agent node receives a new message from the topic, it pushs the new message into a buffer.

- **False** Received from the Clock Topic:

  The clock signal's **falling edge**, treated as the **end of a slot**.

  Agents use the received messages to **update the current slot allocation and its state machine**.

  To be specific, agent nodes pop everything from the message buffer on slot's end, then use the messages received in the past slot to take corresponding actions.

ADD PICTURE HERE

Slot Allocation Scheme and the State Machine

Agents **always** listen to the channel and updates the local slots' allocation record.
There are two situations of the slot occupation:

- **Occupied**: Only one agent sent message during this slot.
- **Free**: No agent sent message or multiple agents collided within one slot.

ADD PICTURE HERE

The agent controls its behaviour through a finite state machine. The various states of this machine represent the agent's status within the network, reflecting the four phases in the STDMA protocol. **Through the gradual transition of states within the state machine, the agent could join the network.**

There are four states in the state machine:

1. **Listening**

   Agents in this state **have not joined the network yet**, and they only *listen* to the messages in the channel.

   Transition Conditions:

   The agent **always records the allocation of past slots**, no matter which state it is in.

   - **Empty**: If no agent sends a message within a slot, or if messages from multiple agents collide in a slot, then that slot is recorded as empty. The reason for recording a slot as empty when a collision occurs is that, whenever an agent detects a collision, it will abandon the collided slot and seek a new slot for itself.
   - **Occupied**: When only one agent attempts to send a message within a slot, that slot is recorded as occupied.

   Everytime finished listening to an entire frame, the agent will attempt to leave the Listening state.

   - $\rightarrow$ Entering: When the number of free slots ≥ 1, transit to 'Entering'. Agent would randomly choose a free slot for later use, note that this doesn't mean that this chosen slot is already secured by this agent.
   - $\circlearrowleft$ Listening: When no free slots left, stay in 'Listening'. This means the channel has already reached its maximum capacity and cannot take any other agents in.

2. **Entering**

   Agent in this state would **wait for the free slot it selected** during the Listening state to arrive and then **attempt to claim that slot**, i.e., agent **sends its ID to the slot occupation message channel in the middle of the selected slot**.

   Transition Conditions:

   - $\rightarrow$ Checking: Agent transit to the 'Checking' state after the occupation attempt message send.

3. **Checking**

   Everytime an agent sends a message in the middle of its selected slot or its occupied slot, it enters this state. At the end of the slot, based on the number of messages received in that slot, it transitions to another state.

   In a word, agents in this state is 'Checking' its right to use the slot.

   Transition Conditions:

   - → In: If only itself sent a message (receiving only one message and the content is its own ID) in the selected or occupied slot, it means it has successfully occupied that slot or maintain occupation to the slot, i.e., 'In' the network.
   - → Listening: If the agent doesn't successfully transition to the 'In' state after message broadcasting, it reverts to the 'Listening' state.

     This condition encompasses multiple scenarios, including:

     – Collision: Multiple agents sent their messages in one slot. In this situation all senders reverse to 'Listening', which means that those attempting to claim the slot have failed, and those who previously occupied the slot relinquish it.

     – Sending Failed: No message wasn't sent to the channel in this slot's duration.

   Reminder about Collision:

   Within the framework of this algorithm, the only potential collision between agents occurs when multiple agents happen to select the same free slot for occupation while 'Listening' and attempted to occupy that slot while 'Entering'.

   In this situation, agent recognises the collision and reverse to 'Listening'.

4. **In**

   Agents in this state are 'In' the network and not expected to drop off unless the agents decides to drop off (which could be simply done by stop sending regular message on their occupied slots).

   Transition Conditions:

- $\rightarrow$ Checking: Like mentioned above, everytime an agent sent their message in the middle of their slot, it transits to 'Checking'.

ADD PICTURE HERE

Agents in this state are expected to immediately transit back to 'In' after the checking at the end of the slot, because in theory no agent should try to send message in occupied slots. At times, collisions might occur between agents that haven't joined the network and those that have. However, such situations typically arise when one or more clock pulses are missed, resulting in a loss of synchronisation. Such situations should not happen or should be avoided as much as possible.

The consequence of this situation is that the agent which has already joined the network is unexpectedly ejected from it. Depending on the timing, various outcomes might ensue. For instance, if an operational agent is suddenly kicked out of the network, it may be unable to broadcast its information for at least one frame, potentially leading to accidents.

Summary

Assumptions: Synchronised clock, predetermined frame length.

Functions: Self-organised single serial discrete channel sharing, allows agents to jumpin and leave.

3.1.2 Path Planning Scheme and Implementation

Now, we have a self-organised method of serial communication for multiple agents. Let's discuss how to implement decentralised collision-free path planning based on this basis.

3.2 Experiment Design

4 Results

The Results section should provide

- An overview of all obtained results
- An in detail discussion/explanation of all results
- A scientific interpretation of the results

4.1 Common attributes to pay attention to are:

- When comparing plots, keep the scale of axes consistent. To do otherwise is misleading for the reader.
- If you are going to compare separate plots, consider if they can be better evaluated when combined into a single plot.
- When plotting data, particularly the *mean*, ensure that you also plot error bars (or other method) of indicating the distribution.
- If a figure or plot is included, ensure it is referenced explicitly in the body text discussion.
- When a large table of data is included, consider whether it would be better communicated as a box-plot or something similar.
- All axes should be labelled and include units of measurement where applicable.
- All captions and figures should have captions with enough information to be understood at a glance. Do not use captions to provide information that is better placed in the body text.
- Remember to identify result outliers and anomalous data and to attempt an explanation or justification.

5 Discussion and Conclusion

The conclusion needs to provide

- A short summary (What has been done and what are the main results)
- Limitations of your work, where applicable.
- Discussion of your work in the bigger picture (How does this contribute to the research field?)
- Future work (What could be next steps in this work?). Remember to keep future work realistic. A good approach is to discuss what the next progression of this project would be, and to justify why this would be interesting.

You will find it easier to write your conclusion if you copy-and-paste your *Aims, Objectives*, and any research questions or hypotheses you stated. You can then discuss each of these explicitly in turn, and how you were able to answer them or complete them successfully. When things have not gone as well as you would have hoped, demonstrate your critical thinking and reasoning to analyse the short-comings of your project - to demonstrate that you understand the underlying causes and that you could conduct good futurework from this learning experience.

A Appendix

This is optional. Not every report needs an appendix If you have additional information like code pieces, long tables, etc. that would break the flow of the text in the report, you can put it here.

References

[1] T. Gaugel, J. Mittag, H. Hartenstein, S. Papanastasiou, and E. G. Ström, *In-depth analysis and evaluation of self-organizing tdma*, 2013. DOI: `10.1109/VNC.2013.6737593`.

[2] B. Guillaume, World representation in artificial intelligence, in *Encyclopedia of Computer Graphics and Games*, N. Lee, Ed. Cham: Springer International Publishing, 2020, pp. 1–10, ISBN: 978-3-319-08234-9. DOI: `10.1007/978-3-319-08234-9_308-1`. available from: `https://doi.org/10.1007/978-3-319-08234-9_308-1`.

[3] E. F. Asadi and A. Richards, Scalable distributed model predictive control for constrained systems, *Automatica*, vol. 93 2018, pp. 407–414, 2018.

[4] J.-H. Lim, Understanding stdma via computer simulation: Feasibility to vehicular safety applications, configurations, and time synchronization errors, *EURASIP Journal on Wireless Communications and Networking*, vol. 2016 2016, pp. 1–14, 2016.

[5] R. Stern, Multi-agent path finding–an overview, *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures* 2019, pp. 96–115, 2019.

[6] R. Stern, N. Sturtevant, A. Felner, *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, 2019, pp. 151–158.

[7] P. R. Wurman, R. D'Andrea, and M. Mountz, Coordinating hundreds of cooperative, autonomous vehicles in warehouses, *AI magazine*, vol. 29, no. 1 2008, pp. 9–9, 2008.

[8] N. M. Kou, C. Peng, H. Ma, T. S. Kumar, and S. Koenig, "Idle time optimization for target assignment and path finding in sortation centers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 9925–9932.

[9] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, Persistent and robust execution of mapf schedules in warehouses, *IEEE Robotics and Automation Letters*, vol. 4, no. 2 2019, pp. 1125–1131, 2019.

[10] K. Dresner and P. Stone, A multiagent approach to autonomous intersection management, *Journal of artificial intelligence research*, vol. 31 2008, pp. 591–656, 2008.

[11] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 2018, pp. 485–493.

[12] J. Salvado, R. Krug, M. Mansouri, and F. Pecora, "Motion planning and goal assignment for robot fleets using trajectory optimization," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7939–7946. DOI: `10.1109/IROS.2018.8594118`.

[13] R. Olfati-Saber, Flocking for multi-agent dynamic systems: Algorithms and theory, *IEEE Transactions on automatic control*, vol. 51, no. 3 2006, pp. 401–420, 2006.

[14] H. Ma, J. Yang, L. Cohen, T. Kumar, and S. Koenig, "Feasibility study: Moving non-homogeneous teams in congested video game environments," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 13, 2017, pp. 270–272.

[15] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, Trajectory planning for quadrotor swarms, *IEEE Transactions on Robotics*, vol. 34, no. 4 2018, pp. 856–869, 2018.

[16] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, Industry 4.0, *Business & information systems engineering*, vol. 6 2014, pp. 239–242, 2014.

[17] D. A. Rossit, F. Tohmé, and M. Frutos, Industry 4.0: Smart scheduling, *International Journal of Production Research*, vol. 57, no. 12 2019, pp. 3802–3813, 2019.

[18] M. De Ryck, M. Versteyhe, and F. Debrouwere, Automated guided vehicle systems, state-of-the-art control algorithms and techniques, *Journal of Manufacturing Systems*, vol. 54 2020, pp. 152–173, 2020.

[19] Z. Fernández, I. Val, M. Mendicute, and E. Uhlemann, "Analysis and evaluation of self-organizing tdma for industrial applications," in *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2019, pp. 1–8. DOI: `10.1109/WFCS.2019.8758039`.

[20] H. Mosavat-Jahromi, Y. Li, Y. Ni, and L. Cai, Distributed and adaptive reservation mac protocol for beaconing in vehicular networks, *IEEE Transactions on Mobile Computing*, vol. 20, no. 10 2020, pp. 2936–2948, 2020.

[21] T. Terauchi, K. Suto, and M. Wakaiki, "Harvest-then-transmit-based tdma protocol with statistical channel state information for wireless powered sensor networks," in *2021 IEEE 93rd vehicular technology conference (VTC2021-Spring)*, IEEE, 2021, pp. 1–5.

[22] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, Constrained model predictive control: Stability and optimality, *Automatica*, vol. 36, no. 6 2000, pp. 789–814, 2000.

[23] R. Scattolini, Architectures for distributed and hierarchical model predictive control–a review, *Journal of process control*, vol. 19, no. 5 2009, pp. 723–731, 2009.

[24] 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 2021, pp. 1327–1349, 2021.

[25] J. H. Lee, Model predictive control: Review of the three decades of development, *International Journal of Control, Automation and Systems*, vol. 9 2011, pp. 415–424, 2011.

[26] J. M. Maestre, R. R. Negenborn, *et al.*, *Distributed model predictive control made easy*. Springer, 2014, vol. 69.

[27] T. Başar and G. J. Olsder, *Dynamic noncooperative game theory*. SIAM, 1998.