MASTR-BOT: Multi Axis Simultaneous Transportation RoBot

Julien Moreno
EPFL + Isochronic
SCIPER : 300840
julien.moreno@epfl.ch

Thesis Supervisor & Expert:
Dr. Albrecht LINDNER

Thesis Advisor:
Prof. Dr. Mohamed BOURI

March 14, 2025

Abstract

Delta robots are widely regarded as the industry standard for pick-and-place automation due to their high dynamic properties and efficiency. However, they also present some limitations. This thesis presents the development and evaluation of a proof-of-concept pick-and-place machine, developed by Isochronic, for small part pick-and-place task automation, as an alternative to other solutions.
As part of this study, a complete motion control algorithm, user interface and simulation tool were developed specifically for this machine. The feasibility of this system is assessed by evaluating the impact of key parameters, including robot scheduling strategies and kinematic properties. This work aims is to evaluate the machine's performance and highlight areas for improvement.

I would like to express my deepest gratitude to the entire Isochronic team for welcoming me, assisting me, and providing valuable feedback throughout both my internship and my Master's thesis. I am incredibly grateful for this unique opportunity.
I am also deeply grateful to my supervisor, Albrecht Lindner, for his continuous support and guidance. He welcomed me into Isochronic, entrusted me with the Excenter project, and gave me the opportunity to contribute meaningfully to this work.
Finally, I'm extremely grateful to my family, Sébastien, Valérie and Evaristo, and my beautiful girlfriend, Ludivine, for their unwavering support throughout my entire journey at the EPFL.

Contents

1 Introduction

Pick-and-place systems are widely used in industrial automation task such as assembly, sorting and packaging. Due to their high speed and precision, delta robots are commonly employed for these applications. However, despite their popularity, they aren't without flaws. The most notable drawback is their constrained workspace, making it challenging to operate multiple delta robots within the same area. A common work around has been to modify the tool head. However, these modifications aren't perfect either. Additionally, a delta robot still requires a large vertical space to operate.

To overcome these limitations, Isochronic has developed a novel pick-and-place machine, the Excenter, designed to rival the performance of delta robots. The mechanism of this robot features a beam, with a rail on each side, each capable of hosting multiple sliders, which are equipped with an arm capable of pick and placing objects. The key advantage of this system lies in its expandability and compact vertical footprint, effectively addressing the workspace constraint of the delta robot. Instead of relying on a lightweight design and high-speed motion like a delta robot, the Excenter leverages collaborative motion between multiple sliders to achieve high efficiency. However, having multiple robots (the sliders) working in the same workspace and operating along the same rail, introduces a new challenge. This requires a new control approach to ensure coordinated motion and to prevent collisions, differing from the methods used for delta robots.

The primary objective of this thesis is to develop a control algorithm along a software to maneuver the robot, and to evaluate the performance of Isochronic's new machine. This study will serve as a proof of concept for this new mechanism, assessing its feasibility. Furthermore, this research aims to identify potential improvements to enhance the system's performance and explore areas for further development.

This thesis presents several key contributions. A user interface (UI) and a simulation tool have been fully developed to visually inspect the robot and assess its performance. The entire software system and algorithm have been developed with flexibility in mind, enabling future experimentation with different layouts. Since a machine's design is never final and will always be subject to modifications, this approach ensures that the machine can be modified without requiring a major rework of the software.

The remainder of this thesis is structured as follows: First, a literature review will be presented, examining relevant research made on similar topics to guide the approach of this project. Then, the entire system configuration will be described, including the physical layout of the machine used in the thesis, the software's architecture, and an overview of how everything is generated and moves, and how the components interacts. This will provide a better hindsight and understanding of the project. This section will also introduce the user interface and the simulation tool developed specifically for this thesis. Next, the 'Method' section will provide a detailed explanation of the algorithm and the behaviors driving the system's operation. The following section, the 'Analysis', will delve into the results from thousands of simulation, aiming to assess the impact of various methods and parameter's modification on the performance while identifying patterns in the system's behavior. Finally, the conclusion will summarize the findings and reflect on the work accomplished throughout this thesis.

2 Literature review

The field of multi-robot pick-and-place systems is continuously evolving. Robots are getting increasingly sophisticated, with more of degrees of freedom (DoF), smaller and lighter design that enables higher speeds, or larger configurations that expand their range of applications. However, the mechanical design of these system is not the only factor influencing their performance. The strategy used to control these robot can range from relatively simple approaches to highly complex ones. With the recent advance in artificial intelligence, new methods for controlling multi-robots system have emerged. Recent works such as Lan, Qiao, and Lee [1]Lan, Qiao, and Lee [2]Lan, Qiao, and Lee [3]Lan, Qiao, and Lee [4]Jermann et al. [5] have explored the use of reinforcement learning in multi robot systems, and Chen et al. [6]Mandi, Jain, and Song [7]Kannan, Venkatesh, and Min [8]Yu et al. [9] have put their attention toward the use of LLM (Large Language Model) instead.
However, the robot used in this study is fairly simple. It consists of multiple beams, that are fixed. On both sides of the beams, there is a rail, and on each rail, there are two tool heads that can move. Hence, there will always be a parallelism of movement along a rail, and the motion will be relatively simple, and won't require complex solutions such as the ones explored in the research papers above.

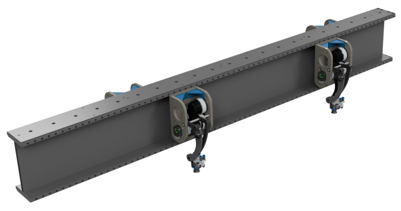
Humbert et al. [10] provide an interesting approach to study a system. In this research, they compare the performance of an over sized system (more robots than necessary) versus a properly sized one, using 4 different collaborative strategies, and a once more with no strategies. They then compare their performances by evaluating the number of boxes unfilled, lost products, balanced workload and total workload. These last two are particularly interesting: Multiple strategies and scheduling could be used to accomplish 100% filled boxes, and 0 lost product, but an unevenly distributed workload would also mean uneven wear among the different robots, which isn't ideal. Additionally, the total workload is also studied, which is how much does a robot work. Once these are done, more simulations are ran, to compare the different individual scheduling rule (FIFO, LIFO and STP) per robot, and the same four metrics are studied. Finally, these results are then studied with varying conveyor speed, for both the input and output feed, separately.
While their robotic system isn't the same as ours, it remains a very interesting approach that we will reuse in this study. This will provide a comparison point.

Bozma and Kalalıoğlu [11] take a different approach, where the target's assignment is done through an objective function. The overall workspace is divided into subdivision, one for each robot, and subsequently, each of these robot's workspace is divided into small sub regions. The target's assignment strategy is done through an objective function, for each robot, that takes into account multiple factors from itself, as well as its neighboring robots. The objective function is composed of 3 sub function, each aiming at a particular objective.
The goal of the first sub function is to reduce the travel time of the robot, while disregarding its neighbor. However, if no targets are present in that subregion, the cost assigned to it is very high.
Then, the second sub functions aims at picking a subregion that would minimize the distance for its neighbors.
Finally, the third and final sub function of the objective function aims at avoiding redundancy between the different sub workspace. If one robot works in a specific sub region, it should be maximally distant to the one of its neighbors. Another way to see it is, as the conveyor is moving, the parts from a sub region of a robot, will move to a different one. Having two robot work on the same side of the conveyor belt is therefore redundant and should be avoided. The ideal scenario would be to have the different robot be placed evenly across the product's flow.
This approach is very different from the one in Humbert et al. [10], but could very well be applicable to our system too. The Isochronic system also has robots that won't share the same workspace, if we count all the slider's from the same rail as a single robot. However, this also renders the target's assignment more challenging. Splitting the range of a slider makes the workspace of the robot smaller, and our sliders are constrained by the movement of the slider sharing the same rail. Therefore, target's assignment using this method is more complex and may requires asynchronous movement of the two sliders. Still, an objective function's approach is very interesting, as it would help balance the overall workload of the system, thus avoiding uneven wear. Additionally, their approach was made to be flexible. Changing the number of robots doesn't require any reprogramming, which is perfectly in line with this project's goal.

Huang et al. [12] published a research paper on multi-robot coordination using part-dispatching rules. When trying to use an algorithm to appropriately assign the different targets to each robot and ensure even distribution, it becomes hard to have an efficient code capable of real-time control. In their study, they used a greedy randomized adaptive search procedure (GRASP) integrating the Monte Carlo Strategy (MCS). This approach allows to explore multiple scenario, without exploring every single one of them, making the process much more efficient. Finally, they propose a method for testing different part dispatching rules.

3 Configuration

This project focuses on Isochronic's small part pick-and-place machine, the Excenter, using their new slider. One module of this machine consists of a beam (fig. 3.1a), with one rail on each side. Depending on the configuration, each rail can accommodate two or four sliders. For the context of this project, we use a single input conveyor, which results in two sliders per rail. The sliders employed are the latest version, the mini Delta 2D (fig. 3.1b).


(a) Excenter's beam with 2 rails and 4 sliders

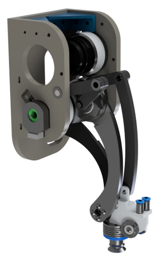

(b) Mini delta 2D (the tool head / slider)

Fig. 3.1: Render of the Excenter's two main elements

Throughout this thesis, the items that needs to be picked up will be referred as **picks** or **targets** and the position where they will need to be placed will be referred as **place**, **placing position**, **drops**, **drop/dropping location** or **targets**.
Furthermore, the layout employed in this thesis is based on a slightly modified version of a prior customer's design.

3.1 System's physical configuration and property

The workspace is defined by multiple variables, as shown in Fig. 3.2. The software allows the user to modify all the variables, and can work with most workspace configurations. The one depicted in the Fig. 3.2 is the workspace that has been used throughout the entirety of this entire master's thesis.
The system consists of two conveyors, going in opposite direction. This is known as a counter-current flow configuration and has a lot of benefits over co-current flow configuration. In a co-current flow configuration, both input and output conveyor move in the same direction. However, the flow rate of the output feed is determined by the packaging layout. As such, it is almost guaranteed that the input feed will have a different flow rate, thus also a different velocity and this represents one of the main challenges of using

co-current flow configuration. Additionally, co-current flow configuration requires a perfect strategy and part dispatching rule. Otherwise, this would lead to a lot of scenarios where there are pick targets which cannot be assigned to a placing position before they leave the workspace.

During the simulation runtime, picks and drops are generated at evenly spaced intervals, based on the defined parameter. These items follow the direction and speed of their respective conveyor. To simulate imperfections in the system for incoming items, noise is introduced to the position of the picks when they are generated, and their position can vary of $\pm 5mm$ on both x or y. This noise affects the picking behavior, as two picks may be too close to be picked simultaneously. Furthermore, a portion of the picks will be randomly marked as "bad", according to the `badProductRatio` parameter, meaning they should be ignored by the pick heads.

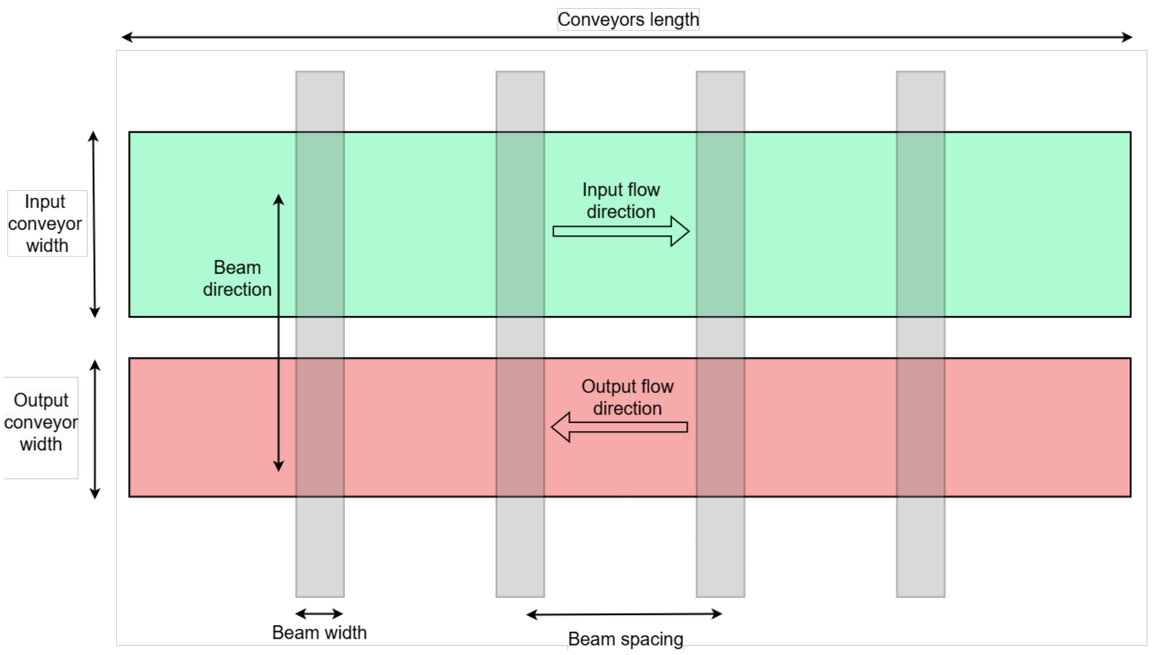


Fig. 3.2: The different variables that define the workspace.
The beams are all equally spaced.

On the other hand, the drop generation provides perfect positions, without any added noise. The placement position is defined by the packages, and a compartment inside a package, for the desired item is never a zero tolerance fit. Therefore, the placement position represents the ideal location where a pick should be placed and adding noise to it is unnecessary.
Fig. 3.3 illustrates some of the variables involved in the drop generation. First, a pattern is generated, based on the number of rows in a packages, and the number of drops per row.

The software supports two packages in parallel, with the uses of `packagesRowSplitting`.

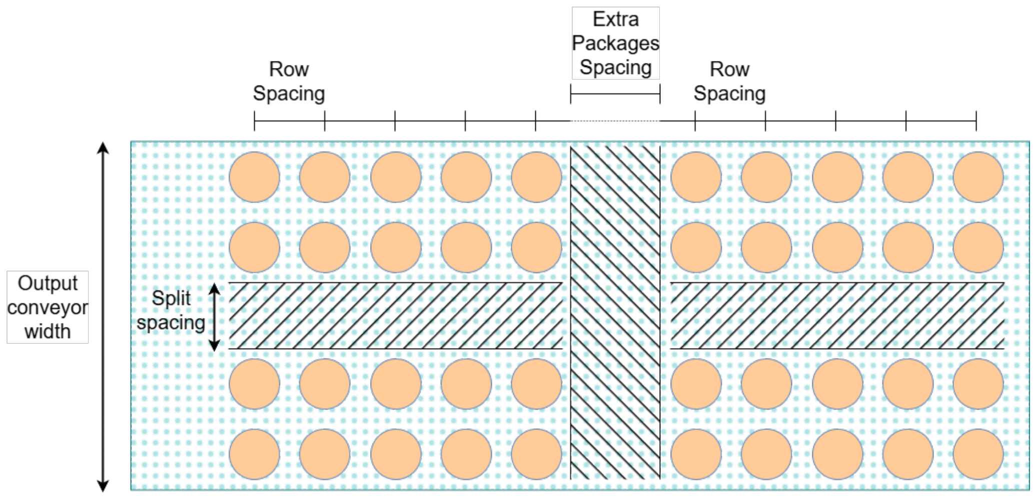


Fig. 3.3: The drop generation is defined by different parameters, shown here.

The drops are evenly distributed across the conveyor's width, and if the latter is enabled, an additional spacing is added in the middle, thus forming two packages instead one. The software generates picks and drops at a given interval; For the picks, it is defined by `inRowSpacing`, whereas for the drops, it is defined by the number of rows in a package `nPackagesRow`, their spacing `packagesRowSpacing` and an additional spacing between the packages with `packagesExtraSpacing`.

Once a pick leaves the workspace, its entire row is deleted, and the information is stored in a specified file for performance tracking. The process or the drops is slightly different. For the outfeed conveyor, the notion of package is introduced. If a drop or an entire row is deleted, it becomes more challenging to track the system's performance per package. Instead, when the last drop location of a package exits the workspace, the entire package is deleted, and the information is again stored. This workflow minimizes the number of instance of picks and drops that exist in the simulation at each time frame and significantly reduces the file size when a simulation is saved for rendering.

As explained earlier, in our simulations, two sliders are used per rail, and there is one rail per beam's side. A slider is defined by its `width` along the beam's length, its `reachOffset`, which is the offset from the rail to the start of its pick up area, and it's `reach`, which is the width of its pick up area, as illustrated in Fig. 3.4. In other words, `reachOffset` and `reach` define the boundaries of the slider's workspace. In the user interface, which we will discuss in Sec. 3.3, the axis of the sliders are defined as shown in 3.4. The x-axis is the rail's axis, the y-axis is its perpendicular, and the z-axis is always vertical. Usually, when defining a workspace, the flow of item defines the x-axis. This is also the case **inside** the algorithm. However, the program has been made to be flexible, which means the Excenter's beams aren't necessarily perpendicular to the item's flow. Therefore, to preserve this modularity, when defining the slider's parameters, we express them in this referential, and they are then translated accordingly inside the program.

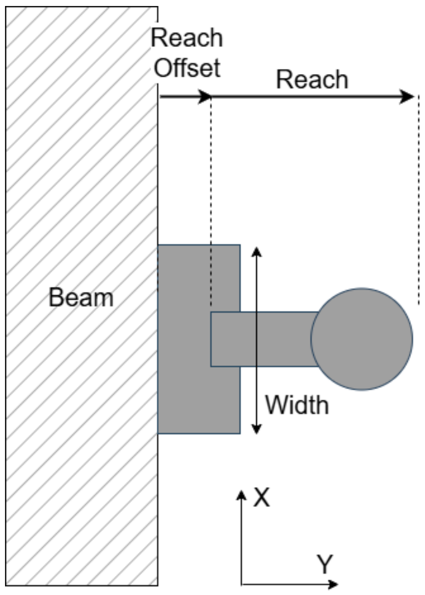


Fig. 3.4: A slider is defined by different parameters shown here. The diameter of the pick head is assumed to be smaller than the width of the slider. Therefore, its value is irrelevant for the software.

Finally, the geometrical properties of the picking targets are ignored. The design of the mini delta tool head includes a suction cup, and it is assumed the product it will work with will fit inside this cup. Therefore, the picking product should never be sufficiently large to consider collision avoidance with other objects.

3.2 Code infrastructure

The simulation is implemented in Python, while the UI is developed using Typescript. The UI and its code will be discussed in section 3.3.1. In this section, we will discuss the Python's code infrastructure. The environment is structured using several classes: One for the conveyors (`Conveyor`), one for the sliders (`Slider`) and one for the beams (`Beam`).

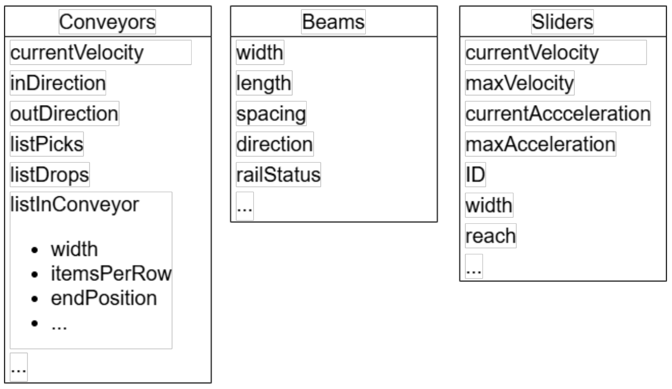


Fig. 3.5: The classes have the structure shown here. Not all the parameters are displayed here.

3.2.1 Conveyors and the picks and places

Most of the parameters of the conveyors are shared. As a result, it was decided that a single instance of the conveyor class would be created. Within this instance, among other things, two lists are maintained: one for the inflow conveyors and one for the outflow conveyor. These lists contain the unique information for each conveyor, such as their position, width and items per row.

The generation of picks and drops is dependent on the parameters of the inflow and the

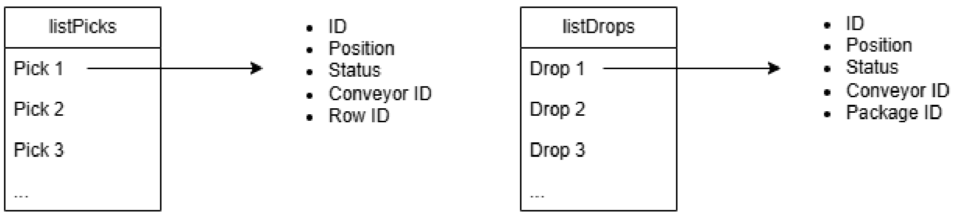


Fig. 3.6: Structure of the list of picks and the list of drops

outflow conveyors respectively. Consequently, they are also stored in the conveyor class inside a list of picks (`listPicks`) and a list of drops (`listDrops`). When the program needs to generate new targets, it iterates through the list of conveyors, creates the new targets and stores these newly created targets in their respective list. These lists are in fact python dictionaries, and are as shown in Fig. 3.6. Their possible statuses are :

- `FREE` : The pick/drop is available.
- `ASSIGNED` : The pick/drop has been assigned to a slider, and will be ignored by the others.
- `SKIPPED` : The pick/drop was a candidate at the given time frame, but didn't pass the tests (i.e. would cause a collision).
- `GONE` : The pick/drop has been handled. If it's a pick, it is deleted, if it's a drop, it will be shown as such.
- `BAD` : The pick is considered invalid (i.e. broken biscuit) and should be ignored.

This approach ensures modularity. The number of conveyors and their positions can change, and the generation would still happen the same way. This also enables modularity for the beams and sliders. The sliders only have to look through a single dictionary to find all the picks, and can decide which conveyors to pick from, if required, using the conveyor ID.

For simplicity, it was decided that parameters such as the direction would be shared for all inflow conveyors, and another direction would be shared for all outflow conveyors. The same principle applies to the row spacing, which is shared among all the inflow conveyors,

while the package's row spacing and the package's extra spacing are shared among all the outflow conveyors. All other parameters are shared across all conveyors.
From the code's perspective, the conveyors themselves do not move; instead, it is the picks and packages that move. The speeds of the conveyors are set as constants, and at each time frame, we update the position of all the picks and places using the $\Delta Position$. This is another reason a single instance is created. The picks and places are the only elements updated at each iteration, managed by the conveyor's class.
The values used for the conveyors are as follow :

- **Length** : 2100 mm
- **Inflow conveyor's width** : 387 mm
- **Outflow conveyor's width** : 290 mm
- **Inflow item per row** : 6
- **Outflow item per row** : 4
- **Number of rows per packages** : 5
- **Inflow row spacing** : 60 mm
- **Outflow row spacing** : 60 mm
- **Packages split spacing** : 40 mm
- **Inflow reject ratio** : 1%

Most of these parameters have been set based on a prior client's request. The parameters that are displayed here are parameters that will be modified and tested during the different simulation runs, such as the conveyor's velocity or the extra spacing between packages.

3.2.2 Beams

All the beams share the same properties. As a result, a single class of beam has also been created. This class contains attributes such as the width, the length, the spacing between beams and their direction. In this case, the sign of the direction is irrelevant; it only serves to determine whether the beams as vertical or horizontal. This class is also responsible for managing the rails. Each rail has a status, which dictates the next action for the sliders on that rail. The possible states of the rails are as follows:

- `CAN_PICK` : The rail is ready to pick.
- `PICKING` : The sliders of this rail are picking.
- `CAN_PLACE` : The rail is ready to place.
- `PLACING` : The sliders of this rail are placing.

The program has been made such that two sliders are always working together and the rail's state will only change if **both** sliders are ready to move on to the next phase.
The values used to define the beams are as follow :

- **Width** : 100 mm
- **Spacing** : 420 mm
- **Number of beams** : 4

3.2.3 Sliders

The slider is one of object that changes the most throughout the code, and each slider changes differently. For this reasons, each slider gets its own class's instance. The axis [x,y,z] are set as described in Fig. 3.4 The main attributes of this class are :

- `maxJerk` : The maximum jerk of the slider in [x,y,z]. This is the same value for every instance.
- `maxAcceleration` : The maximum acceleration of the slider in [x,y,z]. This is the same value for every instance.
- `maxVelocity` : The maximum velocity of the slider in [x,y,z]. This is the same value for every instance.
- `currentAcceleration` : The current acceleration of the slider in [x,y,z].
- `currentVelocity` : The current velocity of the slider in [x,y,z].
- `currentPosition` : The current position of the slider in [x,y,z].
- `ID` : The ID designating the slider.
- `trackedTargetID` : The ID of the target the slider has been assigned to.
- `status` : The status of the slider.
- `inScheduling` : The scheduling used when assigning picks.
- `outScheduling` : The scheduling used when assigning drops.

The values for the maximum kinematic properties have been set based on prior testing and the fixed values defining a slider are as follow :

- **Maximum velocity** : [2'500, 5'000, 5'000] mm/s
- **Maximum acceleration** : [20'000, 50'000, 50'000] mm/s^2
- **Maximum jerk** : [250'000, 500'000, 500'000] mm/s^3
- **Width of the slider** : 120 mm
- **Reach** : 100 mm

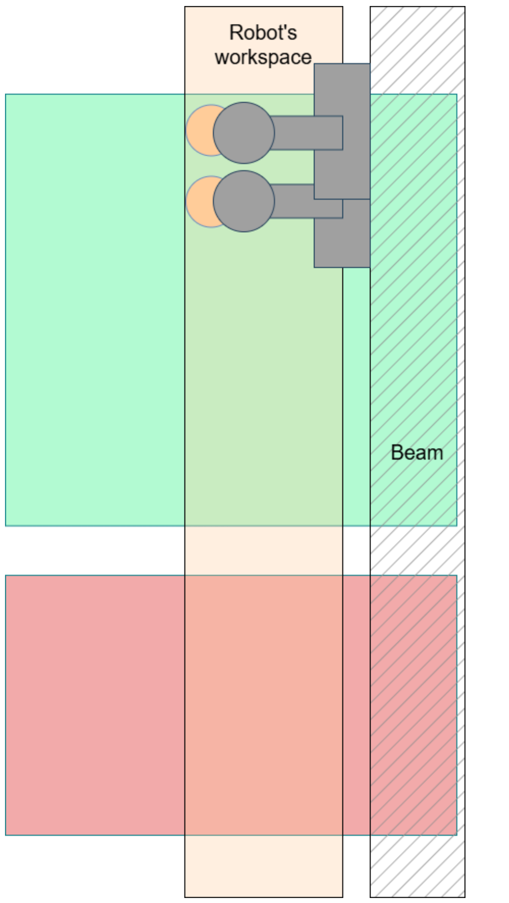
The slider states : This program uses a hierarchical state machine, where the slider states serve as the sub-state machine. The possible states are as follows:

- `IDLE` : The slider is doing nothing.
- `SKIP_N_WAIT` : The slider couldn't pick/place. It will wait for the other slider with which it collaborates to be done and then try again.
- `SKIPPED` : The slider couldn't pick, even after waiting (`SKIP_N_WAIT`). This is only set while the rail is in `CAN_PICK` and means the slider will have nothing to drop and should be ignored when the rail switches to `CAN_PLACE`.
- `PICKING` : The slider has been assigned a target and will proceed to picking.
- `PLACING` : The slider has been assigned a target and will proceed to placing.
- `Z_MVMT` : The slider (which was in `PICKING` or `PLACING` state previously) has reached its target and will proceed to the vertical movement.
- `WAITING` : The slider is done picking/placing operation and will wait for the other slider, which was previously in `SKIP_N_WAIT`.

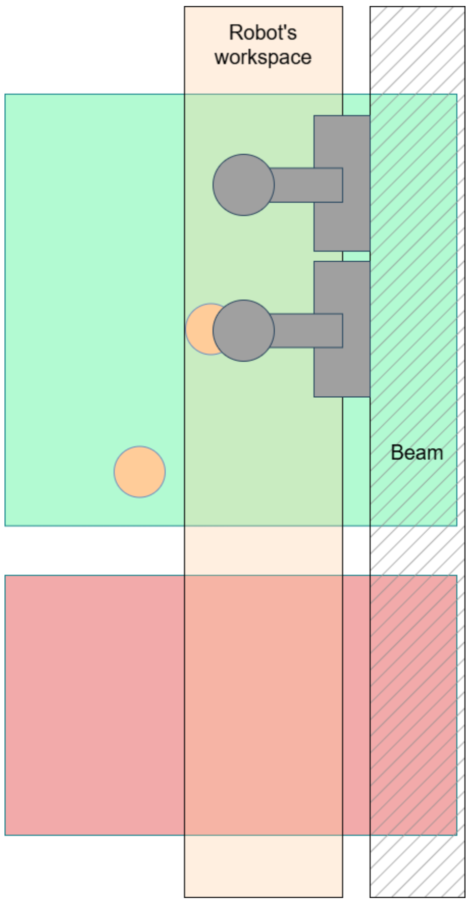
Once the slider is done with the vertical movement (`Z_MVMT`), the target is changed to the status `GONE`.

Why the `SKIP_N_WAIT` state ? The `SKIP_N_WAIT` and `WAITING` states are important elements that contributes to the efficiency of the system. In many scenarios, both sliders cannot proceed with picking or placing simultaneously. For instance, available targets may be too close and would lead to a collision of the sliders, or there may be only one target available, as shown in Fig. 3.7. In such cases, it is better to proceed sequentially. First, one of the slider (and its rail too), goes into the `PICKING` state, while the other slider enters to the `SKIP_N_WAIT` state. The first slider then performs the picking operation. Once completed, it moves to the `WAITING` state (`PICKING -> Z_MVMT -> WAITING`), while the rail returns to the `CAN_PICK` state to check for a new target for the other slider. If a target is available, the program assigns it to the second slider and proceeds with its operation. Finally, once both sliders have completed their tasks, the rail transition to the `CAN_PLACE` and both sliders returns to the `IDLE` state.
In both the `SKIP_N_WAIT` and `WAITING` state, the slider assigned to one of these state follows the other slider. Its target position is set relative to the target position of the active slider, with a margin. This ensures that no matter where the new target is, there isn't any collision.

This sequence of operation applies for both the picking and placing phase, with one key difference.



(a) Two target candidates are too close and would cause a collision of the sliders



(b) Only one target candidate is in range, while another is about to be.

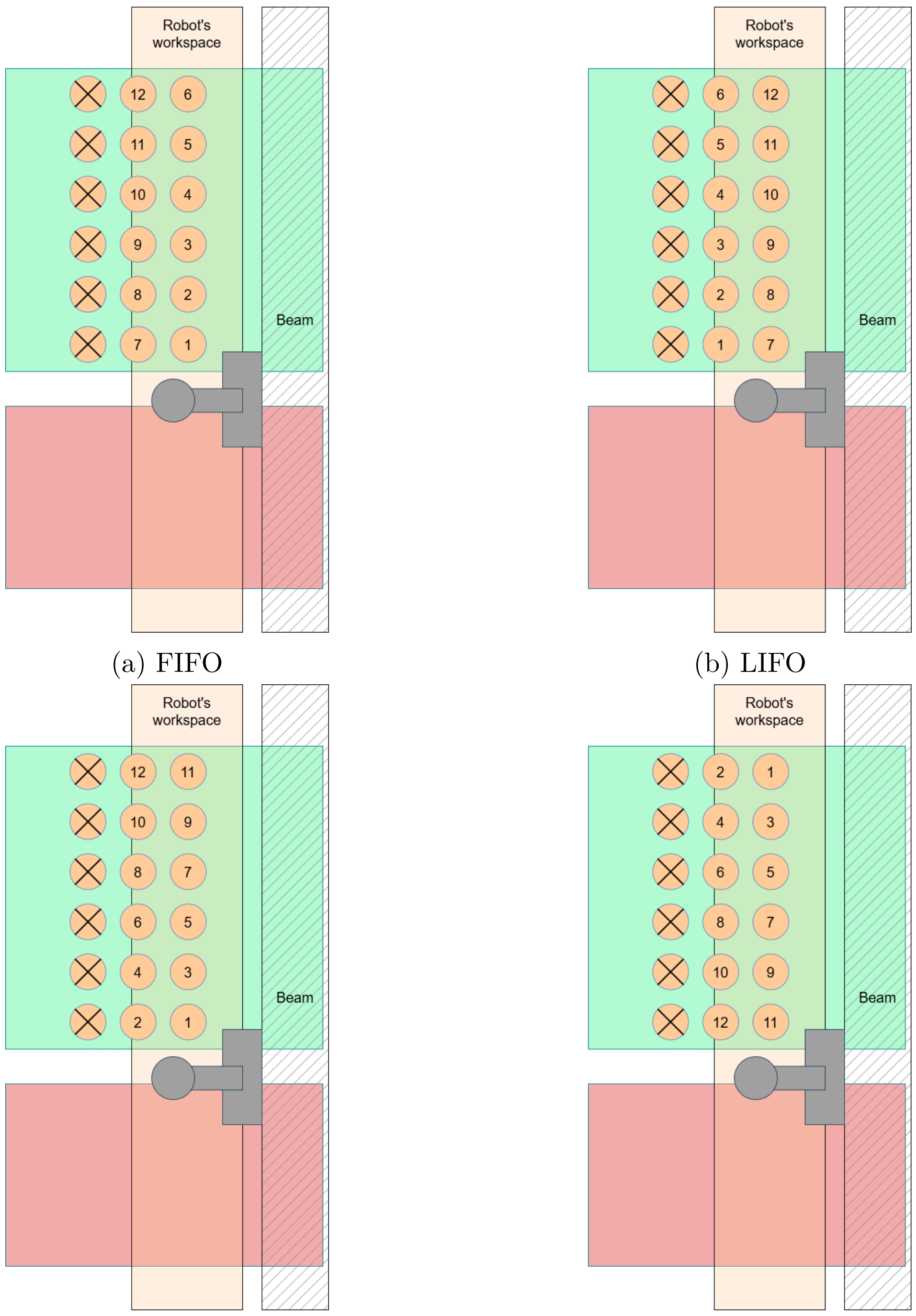
Fig. 3.7: Two scenarios that make use of the `SKIP_N_WAIT` state.

- When attempting to assign a pick to a slider, the program may need to give a slider a second chance. However, if it fails again, the slider transitions to the `SKIPPED` state, and the rail moves to the `CAN_PLACE`.
- When attempting to assign a drop location to a slider, the program may need to give a slider a second chance. However, if it still fails, the slider cannot go back to picking, because it is already holding a target. The program has to find a new drop position before moving forward. Therefore, the slider can never be given the `SKIPPED` state while trying to place.

Schedulings : The program uses schedulings to determine how to assign the targets. There are currently four types of scheduling used:

- `FIFO` : First in, first out. This sets the priority to the candidates that first entered the robot's workspace.
- `LIFO` : Last in, first out. This is the opposite of the `FIFO`, and sets the priority to the last target that entered the robot's workspace.

- `SPT` : Shortest Path Time. The closest target to the opposite conveyor has the priority. This isn't the closest target to the slider! It has been defined this way to reduce the overall travel distance, and not the individual travel distance. When looking for picks, it looks at the distance to the outflow conveyors, and vice-versa.
- `LPT` : Longest path time. This is the opposite to `SPT`. It gives the priority to the furthest candidates, from the opposite conveyor.



(a) FIFO (b) LIFO (c) SPT (d) LPT

Fig. 3.8: Representation of the list of target candidate with their position in the list, per scheduling. The list's order defines the priority queue.

In this project, scheduling has been assigned per beam rather than per rail. Furthermore, since we have opted for a counter-current flow configuration, the scheduling distribution is mirrored for the outflow conveyor. This significantly reduces the number of combinations to test. With four beams and four different scheduling options, mirroring the distribution results in $4^4 = 256$ possible combinations. In contrast, assigning scheduling per rail would yield $8^4 = 4096$ combinations, and without mirroring, the total would increase to $16^4 = 65536$.

The decision to mirror the scheduling distribution was made based on the assumption that both conveyors should exhibit relatively similar behavior, as the code for the picking and placing phases are largely identical. While some differences exist, such as variations in flow distribution, this assumption helps reduce the number of parameters to test, making the process more manageable.

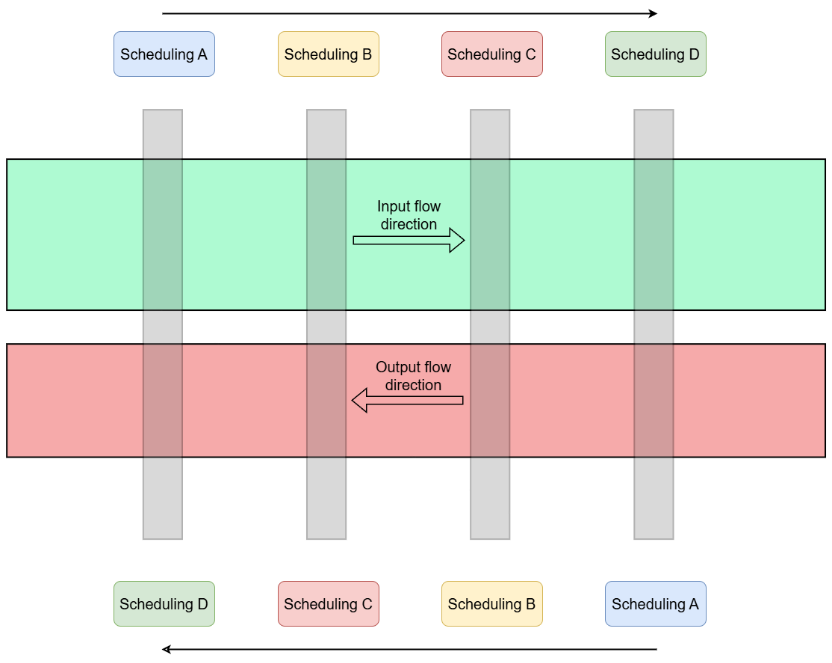


Fig. 3.9: The scheduling distribution is inverted when using counter-current configuration.

3.2.4 Running the python script

There are two ways of running the python script. The first, and easiest way, is through the user interface, which we will discuss in Sec. 3.3.1. The other, and more complete way is through the terminal. To run the program, the user needs to write the following command:

```
python script/main.py
```

By running this command the program will run with all the default running parameter. However, the user can modify the running parameter by adding them to the command line. The running parameters are :

- `-plot [show/no]` : Decides whether to plot live with matplot live (`-plot show`), or not (`-plot no`). **Default: No**
- `-save_hist` : Saves the history of the simulation in a `.json`. This command is necessary if the user wants to visualize the simulation in the typescript simulation tool. **Default : False**
- `-seed [XXX]` : Defines the seed to fix the random generation. The randomness of the program determines which target will be considered bad ($\neq$ bad product ratio) and the noise on the pick's position. This basically allow the user to repeat an experiment with the exact same conditions.
- `-single_run` : The program can run multiple set of parameters to test things. Using this will run a single simulation instead.
- `-no_csv` : The python script will store the data of the simulations, such as the number of picks missed, in a `.csv` file. This is for analyzing the results in a jupyter notebook. Using this will disable this feature.
- `-name [XXX]` : Name under which to save the `.csv` or the `.json` files. **Default: current date/time**
- `-state_bouncing` : Enables the state bouncing feature (will be discussed in Sec. 4.2.2).
- `-pre_move` : Enables the pre-move feature (will be discussed in Sec. 4.2.1).

Additionally, the user can also run the following command, and the terminal will guide him through the different parameters.

```
python script/main.py -buildHelper
```

When the program is run with `-save_hist`, at every time step, the position and status of every object are stored, and are saved into a `.json` file at the end of the runtime. This `.json` is necessary if the user wants to use the simulation tool.

Finally, if the user wants to run multiple simulations with different parameters, this is the only scenario that will require him to modify the code.

3.3 Tools used / created

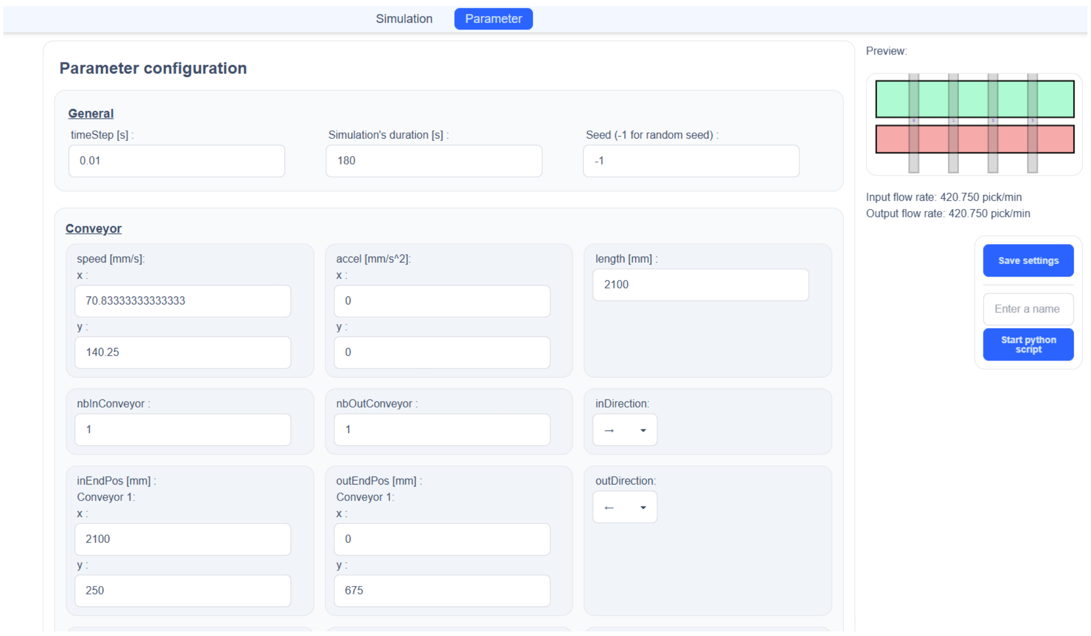
3.3.1 User interface using Typescript

When developing a program for a robot, feedback is essential. There will always be bugs or behavior that weren't expect, and the feedback should be able to analyze the performances, but also highlight those issues. After all fixing bugs and improving a code is an ever lasting dance between the program and the programmer. Usually, when working a robot, it is available and is the feedback tool the developer need. However, no Excenter machine were available for this purpose. Additionally, the entire code of the robot has been developed throughout the duration of this master's thesis. One of the goal for this master's project was to make a proof of concept of this machine, to showcase its capabilities, and that it works. Another goal was to make the solution flexible and modular. It has to be able to handle different configuration. For these reasons, a user interface and a simulation tool were created. Thanks to this interface, the user can modify the parameters and the layout of the simulation, without touching the code, and receive instantaneous visual feedback to verify his modifications. This tool was developed using TypeScript (with React and Next.JS).
For the parameter page, it will read a `.json` file, which contains all the parameters, and display them on the screen. Therefore, the user can add and remove parameter without being required to modify the typescript code.
All of the parameters are separated by which object they are related too, in the same manners as the classes described in Sec. 3.2. This page also allows the user to launch a simulation and save it under the desired name. Running the simulation through the typescript tool will run

```
python script/main.py -save_hist -single_run -no_csv -pre_move
-state_bouncing [-seed XXX] [-name XXX]
```

However, the most important feature of this web app is the simulation tab. As briefly explained earlier, this webpage provides a new simulation tool that will render an entire simulation. This simulation is not live. It requires to run the python script first, with the `-save_hist` argument. Then, this webpage allows the user to select a `.json` file containing a history, and it will display it. Previous to this, the python library `matplotlib` was used. This wasn't efficient at all, this library isn't made to display animation. Even with low `dt` value (the time between two frames), the simulation was running extremely low, and the more objects had to be rendered, the slower it got. Additionally, it didn't provide a lot of freedom. It couldn't pause or play, or go back to previous frames. All off these issues are fixed with the new simulation tool. The different features currently available are

- Load any saved file without rerunning the python script.



Simulation Parameter

Preview:

Parameter configuration

General

timeStep [s] : 0.01

Simulation's duration [s] : 180

Seed (-1 for random seed) : -1

Input flow rate: 420.750 pick/min
Output flow rate: 420.750 pick/min

Save settings

Enter a name

Start python script

Conveyor

speed [mm/s]:
x : 70.83333333333333
y : 140.25

accel [mm/s^2]:
x : 0
y : 0

length [mm] : 2100

nbInConveyor : 1

nbOutConveyor : 1

inDirection: →

inEndPos [mm] :
Conveyor 1:
x : 2100
y : 250

outEndPos [mm] :
Conveyor 1:
x : 0
y : 675

outDirection: ←

Fig. 3.10: Screenshot of the parameter page from the web app.

- Play/pause.
- Go forward **and** backward in the simulation.
- Skip to any given time frame.
- Slow down the simulation.
- Provide visual feedback on the different statuses.

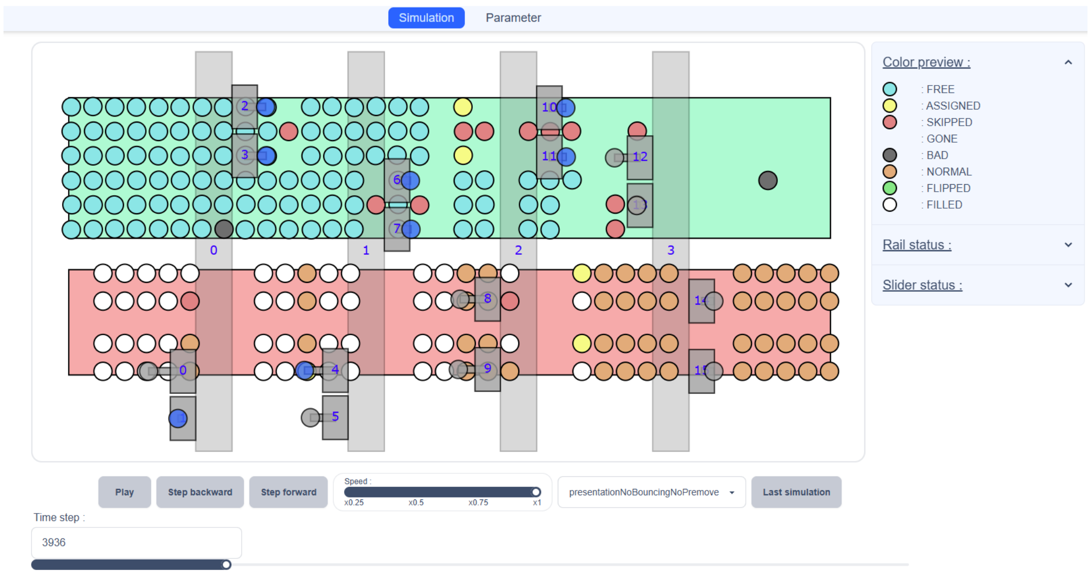


Fig. 3.11: Screenshot of the simulation tool from the web app.

Thanks to this tool, we can try to understand behavior of diverse performance, and things that could be improved or not.

3.3.2 Ruckig's library

Ruckig is a highly efficient online trajectory generation library that enables jerk-limited motion planning in real time [13]. Given an initial state—comprising position, velocity, and acceleration—along with constraints on maximum velocity and acceleration, as well as a target state, Ruckig computes smooth motion trajectories at any given time steps, with the given `dt`. Furthermore, it allows dynamic updates to target parameters.

In the program, Ruckig is used to control the motion of sliders by continuously updating their kinematic states. When a pick-or-place position is identified, its predicted future location is set as the slider's target position, and Ruckig recalculates the trajectory. This process ensures a highly realistic simulation, closely mirroring real-world behavior and enabling precise performance estimation.

The library is available in two versions: a free version for general use and a pro version for enterprise applications, offering enhanced precision and computational efficiency. In the context of this project, we used the free version, which, while effective, introduced certain limitations, especially with precisions. The challenges encountered and the solutions implemented to mitigate these limitations will be discussed in Sec. 4.1.2.

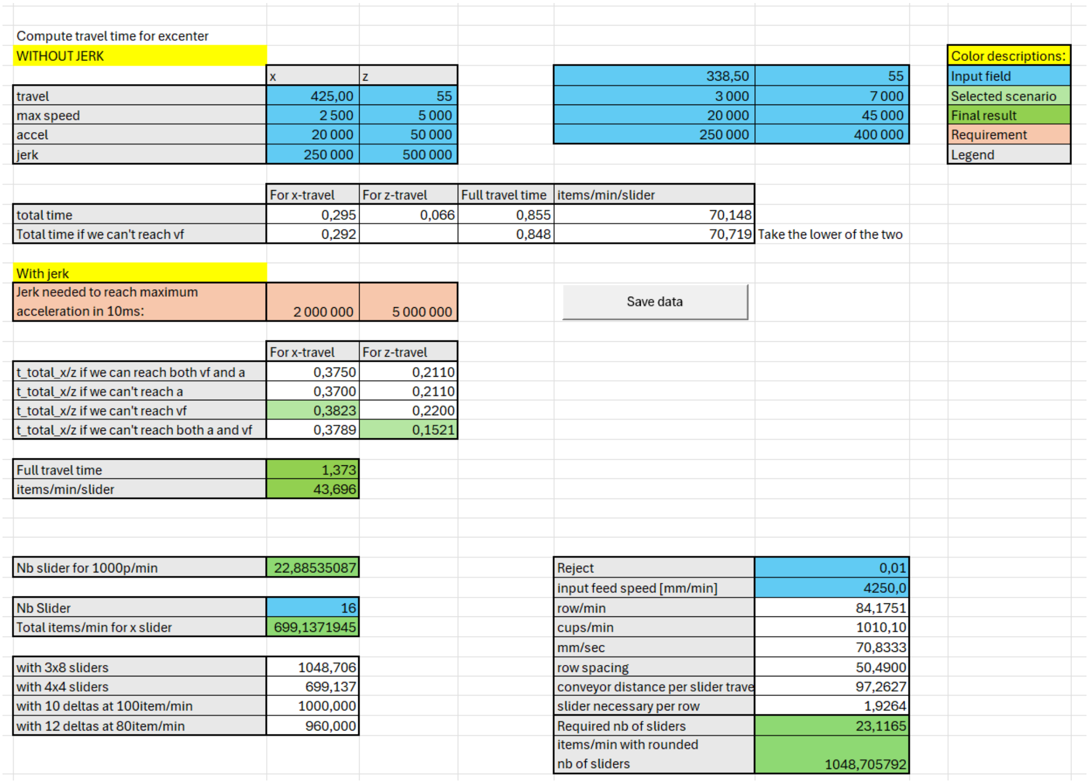
3.3.3 Excel sheet for dimensioning the machine

Finally, prior to using Ruckig's library, an excel sheet was made. This sheet estimated the different aspect of the kinematic of the sliders, such as the estimated travel time for a given movement, and the maximum average picks/min. To estimate this, the average travel distance was taken, as the distance between the center of the inflow conveyor and the center of the outflow conveyor. This excel sheet still provides insight on the performance we could reach with the configuration.

4 Method

The goal of this chapter is to describe the algorithm of this program and the approach implemented to assess the performances of the system with a given set of parameters. Additionally, during the evaluation process, some improvement were brought to the algorithm and will also be described here.

The evaluation process consists of two methods: Visual inspection and data analysis. When developing the algorithm for a robot's pick-and-place system from the ground up, there will be unexpected behavior, wether they be bugs or edge scenarios. Visual inspection provides qualitative assessment of the robot's behavior and ensures everything is working as intended. Furthermore, it helps to identify the difference source of down time that could be negated to improve the performances of the system. On the other hand, data analysis provides insight on the overall system's performance for a given set



Compute travel time for excenter

WITHOUT JERK

	x	z			
travel	425,00	55		338,50	55
max speed	2 500	5 000		3 000	7 000
accel	20 000	50 000		20 000	45 000
jerk	250 000	500 000		250 000	400 000

Color descriptions:
🟦 Input field
🟩 Selected scenario
🟩 Final result
🟧 Requirement
⬜ Legend

	For x-travel	For z-travel	Full travel time	items/min/slider	
total time	0,295	0,066	0,855	70,148	
Total time if we can't reach vf	0,292		0,848	70,719	Take the lower of the two

With jerk

Jerk needed to reach maximum acceleration in 10ms:	2 000 000	5 000 000

	For x-travel	For z-travel
t_total_x/z if we can reach both vf and a	0,3750	0,2110
t_total_x/z if we can't reach a	0,3700	0,2110
t_total_x/z if we can't reach vf	0,3823	0,2200
t_total_x/z if we can't reach both a and vf	0,3789	0,1521

Full travel time	1,373
items/min/slider	43,696

Nb slider for 1000p/min	22,88535087

Nb Slider	16
Total items/min for x slider	699,1371945

with 3x8 sliders	1048,706
with 4x4 sliders	699,137
with 10 deltas at 100item/min	1000,000
with 12 deltas at 80item/min	960,000

Reject	0,01
input feed speed [mm/min]	4250,0
row/min	84,1751
cups/min	1010,10
mm/sec	70,8333
row spacing	50,4900
conveyor distance per slider trave	97,2627
slider necessary per row	1,9264
Required nb of sliders	23,1165
items/min with rounded nb of sliders	1048,705792

Fig. 3.12: Inside the excel sheet, the user can enters different values, inside the blue field. The final results are shown in bright green, and intermediary results are also given.

of parameter, but also on specific metrics such as the speed, workload, pick/min and so on. Data analysis is a more objective performance evaluation method.

The first step, and probably the most important one, is to confirm the system's behavior and fix any potential bugs. To ensure that the system is behaving as intended, the evaluation is mainly done through visual inspection, thanks to the developed simulation tool presented in Sec. 3.3.1. We need to ensure that the different schedulings are implemented correctly, the collision avoidance is working as intended, and that the hierarchical state machine is cycling properly and isn't skipping any state, or getting stuck in edge cases.

Once the system's algorithm is confirmed to work as intended, we can then move on to the performance analysis, which is mainly, but not only, done through data analysis. To assess the performance through data analysis, multiple set of parameters were tested with a single parameter varying, with each being run multiple times to eliminate possible outliers.

4.1 Algorithm

Although the algorithm was made to be very modular, one assumption that was made that removes such flexibility is that there will always only be two sliders working together on a single conveyor. As such, some simplifications or special behaviors were programmed.

Additionally, the program doesn't loop through each slider, but rather each rail, and both slider on the same rail are assigned a target together. The slider's behavior for both picking and placing is near identical, but there are still some small differences. These are mostly through state handling, allowing the same code to be used for both.

The first and the second slider Throughout the entire program, when looking for targets to pick or place, it often uses the notion of first and second slider. This is one of the simplification made due to always having exactly two sliders working together per conveyor. In some way, the first slider has the priority over the second slider. The program will always attempt to assign a target to the first slider. However, for the second slider, it will try to assign it, and when it does, only then does the program checks for collisions. If this leads to one, it will attempt to assign a new target to the second slider. The first slider's assignment is *permanent*. As such, the slider defined as the first will always be the one on the same side as the highest priority from the scheduling.

The searching algorithm If a rail is in `CAN_PICK` or `CAN_PLACE` state, the search algorithm will trigger, and is depicted by Fig. 4.1. This algorithm will first try to find if there are any targets in range of the sliders. If there is, it will continue the algorithm, otherwise, it will stop and try again in the next time step. The `findInRange` function will return a list of the targets that are in range, which will then be sorted using the scheduling. The order of the list defines the priority of the targets. Then, we ask ourselves the first question: Are we trying to assign a target to the first slider, or the second slider. This question can arise thanks to the states `WAITING` and `SKIPPED`. These states are described in Sec. 3.2.3. If a slider is in the `WAITING` state, it means that it has already gone through this process, and waits for the other slider to be done. On the other hand, testing if a slider is in the `SKIPPED` state is the first difference between the picking and placing phase. If a slider is in the `SKIPPED` state, it means it couldn't **pick** a target and should be ignored. This state verification is only relevant for the dropping phase and can't occur in the picking phase, because a slider can't be given the `SKIPPED` state during the dropping phase. Following this verification, the algorithm will then attempt to assign a target to the current slider. If we are trying to assign a target to the first slider, and we do not succeed, the search algorithm will stop, and try again in the next time step. However, if it succeed, the rail's state will change to the `PICKING` or `PLACING` state at the end of the iteration. Next, it will repeat the process for the second slider.
One special scenario occurs when we only attempt to assign a target to the second slider, and fail to do so (This is the ① in the Fig. 4.1). If this scenario were to happen, it can only mean one thing: In the last attempt, only the first slider managed to find a suitable target, while the second slider went into the state `SKIP_N_WAIT`. Therefore, it was given another opportunity for the next attempt, and the first slider went into the

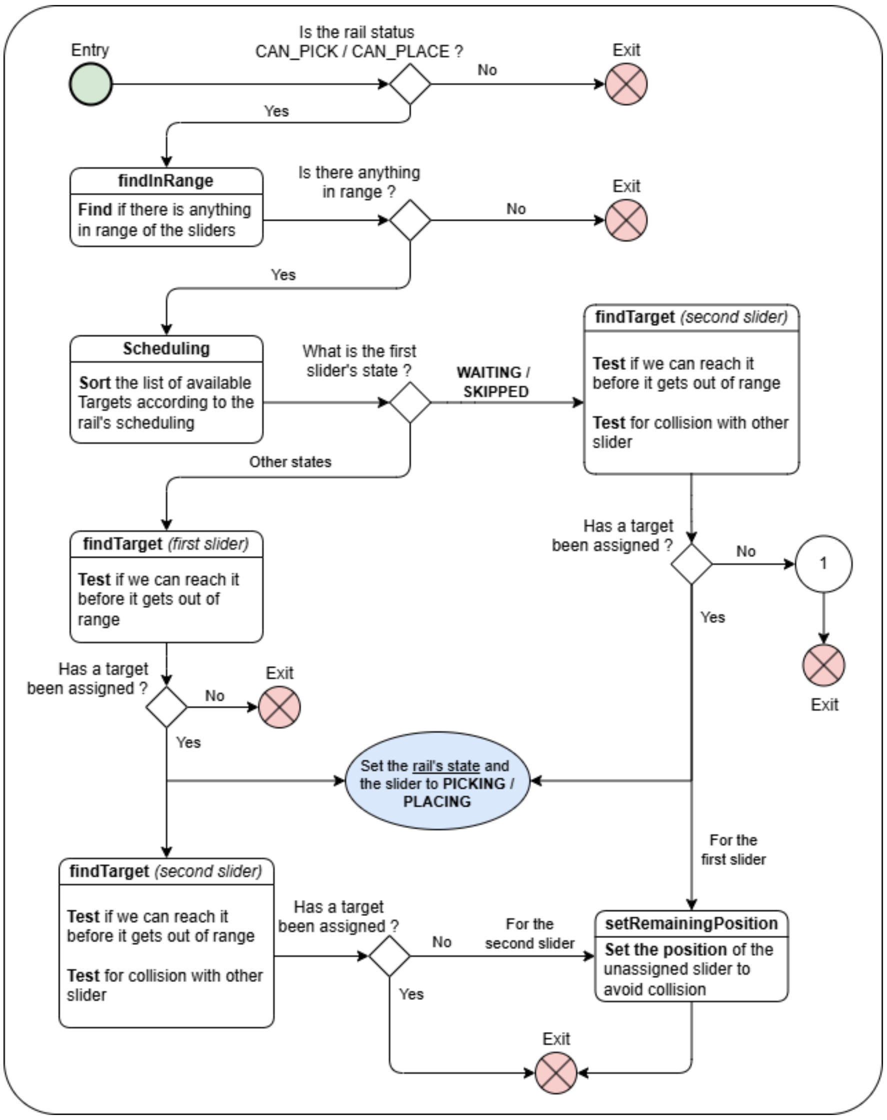

Entry
Is the rail status CAN_PICK / CAN_PLACE ?
No
Exit
Yes
findInRange
Find if there is anything in range of the sliders
Is there anything in range ?
No
Exit
Yes
Scheduling
Sort the list of available Targets according to the rail's scheduling
What is the first slider's state ?
WAITING / SKIPPED
findTarget (second slider)
Test if we can reach it before it gets out of range
Test for collision with other slider
Other states
findTarget (first slider)
Test if we can reach it before it gets out of range
Has a target been assigned ?
No
1
Yes
Exit
Has a target been assigned ?
No
Exit
Yes
Set the rail's state and the slider to PICKING / PLACING
For the first slider
findTarget (second slider)
Test if we can reach it before it gets out of range
Test for collision with other slider
Has a target been assigned ?
No
For the second slider
setRemainingPosition
Set the position of the unassigned slider to avoid collision
Yes
Exit

Fig. 4.1: This algorithm is responsible for assigning targets to the slider, when a rail is in the `CAN_PICK` / `CAN_PLACE`. If it is sucessful, the rail's state will be changed to `PICKING` / `PLACING`.

state `WAITING`. Whenever this happens, there are two possibilities.

1. We are currently trying to pick a target, and there aren't any suitable candidate for the second slider, despite giving it another chance. In that case, we set the second slider's state to be `SKIPPED`, and it will be ignored for the placing phase.

2. We are currently trying to place a target, and there aren't any spot available. However, we **need** to wait for one to be available. We can't go back to picking, because the slider isn't available. Therefore, nothing happens, and we will try again in the next few iterations, until a placing spot has been found.

Finally, in any given scenario, if only a single slider has been assigned a target, the function `setRemainingPosition` will be called for the other one. This ensure that regardless of the given scenario, there will be no collisions.

4.1.1 The `findInRange` function

The very first step of the searching algorithm is to find what is available for each rail. This step is crucial to ensuring the proper workflow of the system. There are two version of this function, one for the pick, and one for the place position. There are two reasons behind this difference. First, the pick position have noise added to them. Albeit very minor, it still required a change in the way we look for targets. Second, it is again for flexibility. This program can work with multiple conveyors, and the picks have the ID of the conveyors stored within them. This function can make use of this information to choose from which conveyor to pick from. For our testing, this scenario hasn't been used, but the code is still implemented and ready. From now on, until specified otherwise, both these functions (`findPicksInRange` and `findDropsInRange`) will be mentioned as a single function, ready for both scenarios, named `findInRange`.

This function first takes the list of targets (picks or drops, depending on the scenario), and will extract the following information: their ID, status and position along a single axis, which the slider's arm axis (the Y axis in the Fig. 3.4). This coordinate is the only relevant information for any given target, to determine whether it is in range of the slider or not, and will be named x_i^n where n refers to the target's ID, and i the axis we've decided to keep. Additionally, for the picks, we retrieve their conveyorID, should this information be used, as well as their rowID. By comparing x_i^n, we remove any target that is not inside the range of the rail.

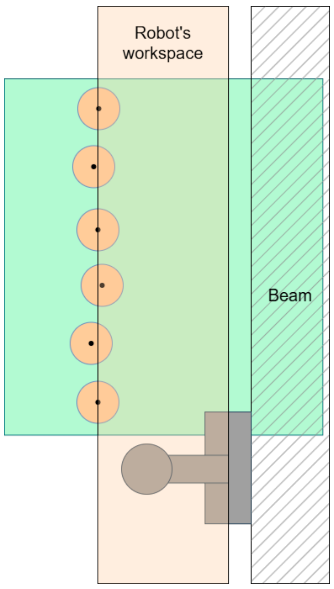


Fig. 4.2: Because of the noise added on the picks, some of the picks in a row may be inside the range of a slider at a given time frame, while others aren't. Here, the second and the fifth picks, aren't in the range of the slider.

Fig. 4.2 depicts the issue that arise with the noise on the position. With such algorithm, 2 of the 6 targets would be considered as out of range. At first glance, this appear to be a minor inconvenience. However, it can have a terrible impact on the system's performance. If a single target is in range while the check occurs, then only this target will be assigned, despite all the others probably laying only 1 or 2mm too far. This is especially impactful when that target is on further end of the rail, and will be assigned to the first slider, therefore blocking the second slider from picking anything. To remedy to this issue, when a target has been found to be in range, we use the rowsID to find all of its neighbors.

Finally, we use the status of the targets to remove all of the ones that aren't available to be assigned. This process provides a reliable target extraction, but also has one major flaw. Most of the times, when assigning a target from one conveyor, the sliders are above the other conveyor and will move only when assigned a new target. By the time they reach the other conveyor, the targets will have moved. In some scenarios, the desired target may now be out of range of the sliders, and a lot more may now be available for the given slider. Furthermore, the workspace of the sliders is very narrow, with a reach of 100mm. Thus, the faster the conveyor, the more problematic it gets. After realizing

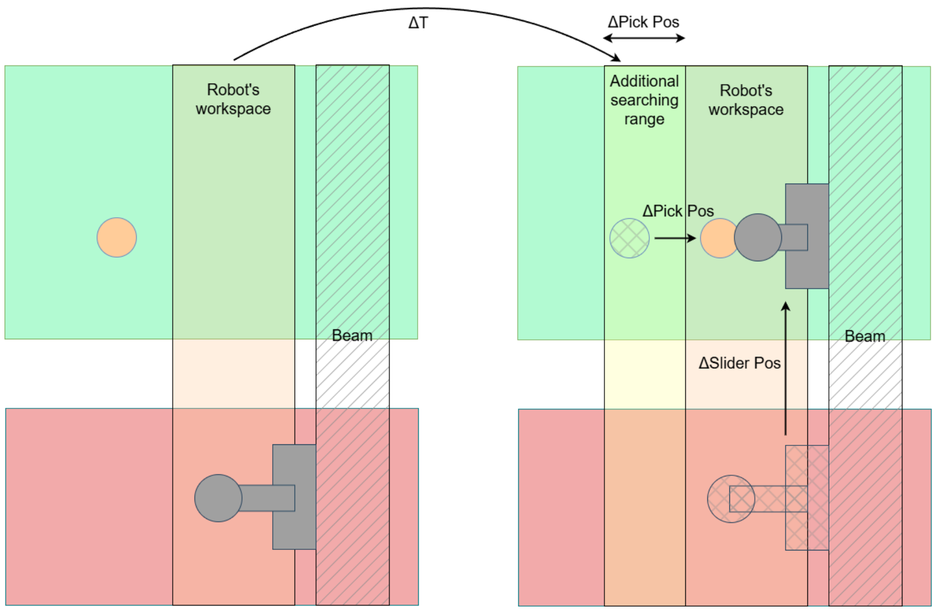


Fig. 4.3: An extra distance is added to the looking range when assigning a target to a slider to circumvent the travel of a target, during the slider's travel.

this, the function was improved, by including an additional searching range. On average, the travel distance of all sliders, should be roughly the distance between the center of one conveyor, to the distance of the center of the other conveyor. While traveling this average distance, given a conveyor speed, the targets will have traveled a different distance, and it is this ΔTarget Position that is added to the **start** of the searching range. The end of the searching range isn't modified and doesn't need to be. With this revision, the `findInRange` functions now return an estimated list of candidates instead of an exact list. The list now provides the list of targets that are currently in range, as well as the one that **should** be in range. This approach has a very low computation cost. Computing the traveled distance for each target, with any given slider at any given time would be significantly more computationally heavy.

4.1.2 The `findTarget` function

The `findTarget` function is the most computationally heavy function of the program and is the reason the `findInRange` was designed to provide an estimated list of candidates. In the earliest version of the software, it would assign the `targetID` to a slider, and at each time step, the program would exploit Ruckig to update the trajectory in order to follow the target. This approach was very computationally heavy, but most importantly, was faced with two major problems. First, the slider would always lag behind the target. When using Ruckig, we provide the Ruckig's instance with a target speed, acceleration and position. The goal is always to arrive to the destination with a velocity and an acceleration of 0. However, the target isn't static.

The other issue was the precision problem of Ruckig's free version. By calling the program this many time, and with a target position getting ever so close to the actual slider's position, Ruckig would end up failing most of the time. Therefore, `findTarget` was redesigned to be the way it is today.
The current version of the `findTarget` now tries to find the meeting point of the slider to its target, instead of making it follow it. This way, we can ensure multiple things.

1. When testing a candidate target, we can ensure it will still be in range when we reach it

2. If it is, we give the slider the meeting point as a target position, and ensure it isn't between frames, or at an improper resolution.

3. We don't update Ruckig's trajectory at every time step anymore.

4. Because the sliders aren't following the target, but rather joining them to a meeting point, the traveled distances are shorter.

The findTarget algorithm is described by Fig. 4.4. First, the program enters this function with a list of available targets. This list has been sorted, using the scheduling. Therefore, the first element of the list will always be the higher priority target to test. This algorithm uses a double loop. The outer loop iterates over every candidate targets, while the inner one iterates over the possible meeting points. The inner loop is a dance between the slider's position and the target's position. First, it will estimate the travel

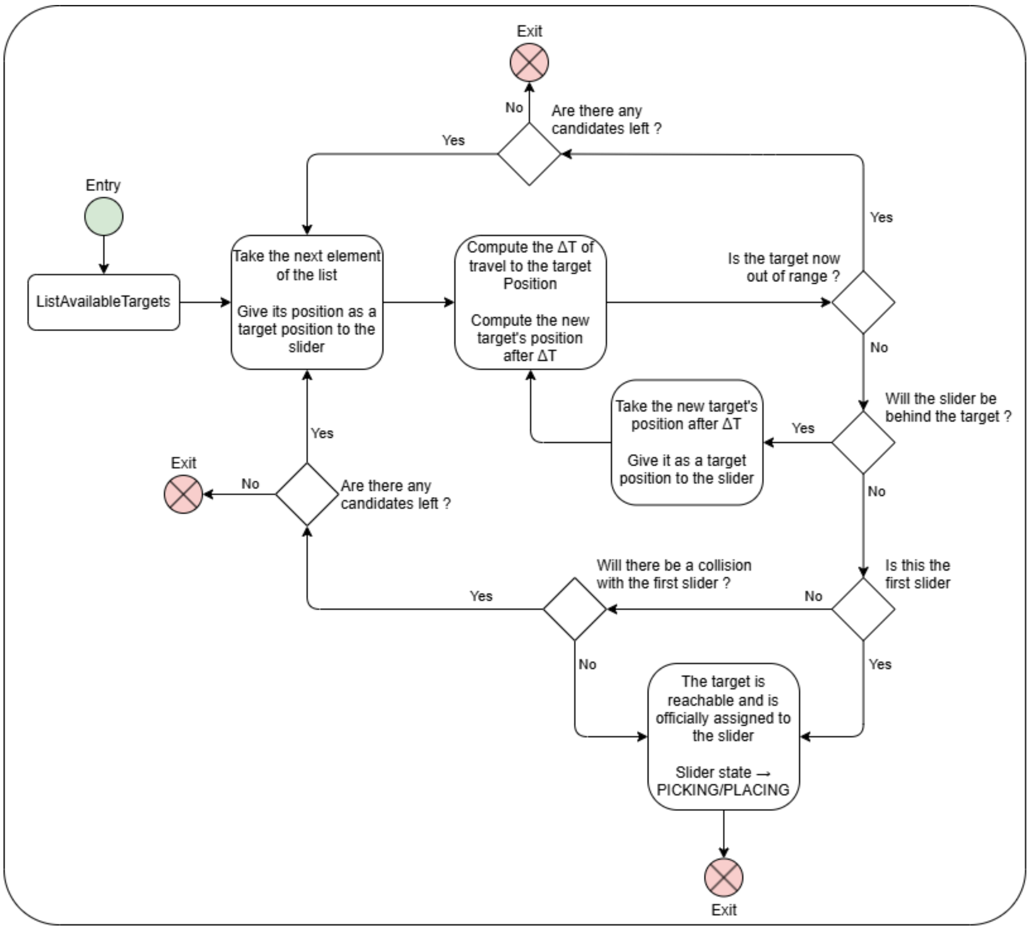

Exit
No
Are there any candidates left ?
Yes
Entry
Yes
ListAvailableTargets
Take the next element of the list
Give its position as a target position to the slider
Compute the ΔT of travel to the target Position
Compute the new target's position after ΔT
Is the target now out of range ?
No
Will the slider be behind the target ?
Take the new target's position after ΔT
Give it as a target position to the slider
Yes
Yes
Exit
No
Are there any candidates left ?
No
Will there be a collision with the first slider ?
Is this the first slider
Yes
No
No
Yes
The target is reachable and is officially assigned to the slider
Slider state → PICKING/PLACING
Exit

Fig. 4.4: The algorithm responsible for testing and assigning a target to a slider follows this logic. It is entered with a given list of available targets.

time required to move the slider from its current position, to the other conveyor. Then, it will estimate the new target's position after this duration. If the slider is behind the target, the loop will repeat, but this time by trying to reach the predicted target position from the last attempt. If the predicted position is outside of the slider's boundaries, the target is rejected, and we exit the inner loop and we repeat the process for the next target in the list. However, if the slider is ahead of the target's predicted position, then it is considered to be a valid meeting point and is given to the slider. If we are trying to assign a target to the second slider, we need to ensure this target assignment doesn't lead to a collision. If it does, then it is also rejected. Finally, if we have reached the end of the list and no target has been found to be suitable, the algorithm will exit, and the slider will be either set to `SKIPPED` or `SKIP_N_WAIT`.
This approach is also the reason we can afford to provide an estimated list of candidates. It doesn't need to be exact, but it has to include the strict minimum. We could also avoid making a list of candidate altogether, but as `findTarget` is the most computationally heavy function of the software, we can restrict its usage to only what we know is possible.

4.2 Improvement based on the visual inspection

After implementing these algorithms, a visual inspection was conducted to verify that the system was behaving properly. Despite being correctly implemented, some unexpected behaviors were observed, which could be mitigated along with minor optimizations to enhance the overall system performance.
Their impact will be analyzed in Sec. 5.2.

4.2.1 The addition of "pre-move"

In the initial implementation of the algorithm, a slider would only move when given a target. As a result, a slider would remain idle above the outflow conveyor while looking for picks, or conversely, above the inflow conveyor while searching for a placing target. This behavior was not a problem for some rails, where target were continuously available. But for others, it led to inefficiencies.
To address this issue, a "pre-move" mechanism was introduced. If no immediate target is available, the sliders of this rail proactively move above the opposite conveyor. Thanks to this adjustement, the travel time of the sliders, once assigned a target, are effectively reduced, due to being above the correct conveyor. This improves the reaction time and reduces the waiting time. Importantly, the pre-move isn't defined by a state. When this happens, the sliders are still in the `IDLE` state, allowing the program to still look for possible candidates while they are moving. If one is found, the trajectory is updated accordingly.

4.2.2 The addition of the "state-bouncing"

Relying only on the scheduling assigned per beam results in an imbalanced configuration (see Sec. 5), where the two outer beams remains idle most of the time. The first beam in the input feed direction predominantly waits on an available placing position, while the first beam in the output feed primarily waits for a pick.
A problem arise when a single pick/place location become available: one of the slider proceeds with the operation, while the other goes into the `SKIP_N_WAIT` state, and then wait for its turn. Two possibles scenarios follow:

1. **Ideal case :** Another single target comes into the range of the sliders, allowing to proceed without issue.
2. **Problematic case :** Two targets come into the range of the sliders. Since only one of the two sliders is available, a target will be missed.

To address this issue, the state-bouncing mechanism was introduced. This method's name come from the method's approach of dynamically switching between the `CAN_PICK` and `CAN_PLACE` state until a suitable target has been found. The key idea behind this process is to determine whether the slider has enough time to complete a placing routine and come back before the next picking target becomes available or not, if its initial task was to pick, and vice-versa when initially trying to place a target. If it has, then the slider will proceed with this action, otherwise it will remains in its current-state.

The state bouncing can only occur if :

1. It is one of the outer beams,
2. One slider has successfully picked or placed an object, while the other has failed again—meaning that after entering the `SKIP_N_WAIT` state, no new target has immediately become available.

By enforcing these conditions, state-bouncing is only applied when necessary, ensuring that critical targets are not missed in resource-limited areas. Specifically:

- The first beam in the input feed direction typically waits for a placement opportunity while having an abundance of pick targets.
- The last beam in the input feed direction experiences the opposite situation, primarily waiting for a pick while placement locations are readily available.

To determine whether a slider can complete a placement and return before the next pick becomes reachable, the system leverages the additional looking range introduced in Sec. 4.1.1, which is determined by the travel time of a slider from one conveyor to the other.
When the state bouncing is enabled, and we are initially trying to pick (`CAN_PICK` state), the program will attempt to find a pick target, by using $2 \cdot additionalLookingRange$ instead. If it failed, it switches the rail's state to `CAN_PLACE`, and look for another target, but by using only $1 \cdot additionalLookingRange$. The reasoning is the same when initially trying to place instead.

5 Analysis

Now that the entire software has been validated, we must assess the system's performance under different operating conditions. In an industrial setting, customers prioritize both the speed and the efficiency while minimizing the losses. The performance of the system is affected by several factors, including kinematic properties, the scheduling's distribution and the product's arrangement on the conveyors.

First, we will study the impact of the conveyor's speed. The goal of this section is to define a speed that will be used for the remainder of this section.
Then, we will analyze the impact of the modifications introduced in Sec. 4.2.1 and 4.2.2.

Based on the visual inspection, these changes should improve the performance. However, a thorough evaluation is necessary to confirm their effectiveness.
Next, we will assess the performance with different throughput. The throughput depends on both the kinematics and the product's arrangement. The slider's kinematic properties have been defined based on past testing and will remain unchanged due to the unavailability of a physical robot. Their impact will need to be analyzed once a robot becomes available. Fortunately, we can still modify the conveyor's speed and the product's arrangement on the conveyor.

Finally, the scheduling plays a critical role in the performance of the system. Therefore, we will evaluate different scheduling combinations and attempt to identify patterns that optimize efficiency.

For all experiments in this study, each parameter combination are tested 10 times using different seeds and a simulation time of 180 seconds. Additionally, an extra packages distance of 140mm is applied. The results shown are the average of these 10 runs. Unless specified otherwise, these settings remain consistent across all test.

5.1 Conveyor's velocity

The conveyor's velocity is one of the key properties affecting throughput. While this sub section's goal is not to evaluate the different throughput levels, we must first determine an appropriate range of conveyor's speed at which to test our system. Since the slider's kinematic properties are fixed, selecting a conveyor speed that is too low will results in near-perfect performance, regardless of parameters and schedulings. On the other hand, having conveyor's move too fast will yield terrible results, regardless of the algorithm and schedulings.

The feed rate of a conveyor is defined by the its velocity, the number of targets per row, the spacing between rows, and the reject rate of the inflow conveyor. To minimize the losses, we need to ensure that both conveyor have the same flow rate. Calculating the flow rate of the input conveyor is straightforward, as the rows are always evenly spaced. It is given by :

$$\text{inFlowRate} = \frac{\text{inConveyorSpeed} \cdot \text{itemsPerRow}}{\text{rowSpacing}} \cdot (1 \text{ - badProductRate}) \tag{1}$$

However, the flow rate equation for the outflow conveyor is slightly different. Unlike the input conveyor, there are two different spacings in place here, along the conveyor's length. The equation uses both the row spacing and the extra packages spacing into account. The split spacing (the distance separating two parallel packages, See Fig. 3.3) is ignored, as it does not affect the number of targets per row. The flow rate for the output conveyor is given by :

$$\text{outFlowRate} = \frac{\text{outConveyorSpeed}}{\text{packagesSpacing}} \cdot \text{itemsPerRow} \cdot \text{nPackageRows} \tag{2}$$

$$\text{packagesSpacing} = \text{nPackageRows} \cdot \text{packagesRowSpacing} + \text{extraPackagesSpacing} \tag{3}$$

To ensure both conveyors maintain the same feed rate, only the inflow's conveyor speed is set. The outflow conveyor's speed is computed accordingly.

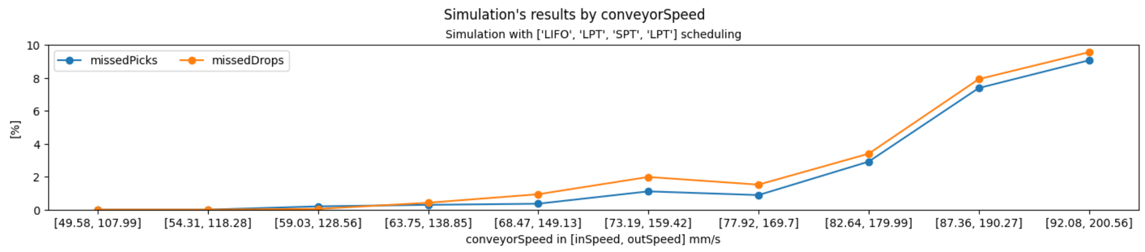


Fig. 5.1: A range of conveyor speed have been tested. The x-axis represents to set of speeds, while the y-axis represents the percentage of missed drops or missed picks. Lower is better.

Fig. 5.1 presents the results of testing a range of speeds using a fixed set of schedulings, no pre-move nor state bouncing. As expected, lower conveyor speed yield near-perfect. However, this is not desirable for our analysis. We want to identify a speed at which where system's flaw becomes apparent. This will allow us to assess the impact of different factors on performance. Based on these results, and unless specified otherwise, the chosen inflow speed for the remainder of the analysis is **77.92 mm/s**, as it is the first speed at which a noticeable decline in performance is observed.

Additionally, when evaluating the performance of the system, it is crucial to consider the workload.

Workload, balanced workload and total workload. The workload of a slider defines how much the slider is working, over the duration of the simulation.

$$\text{Workload} = \frac{\text{Traveling Time} + \text{Picking/Placing Time}}{\text{Total Time}} \tag{4}$$

We want this value to be as high as possible. The higher this value is, the less we are wasting time by waiting for a task. However, we also want the system to be balanced. A balanced system is a system where most of the robots have approximately the same workload. If a system is unbalanced, it means the workload is not evenly distributed across the robots, and it would cause uneven wear of the system. For customers, this may be an important aspect to consider. The balanced workload is given by :

$$\text{Balanced Workload (BW)} = \sum_{k=1}^{nSliders-1} |W_k - W_{k-1}| \tag{5}$$

Finally, we look at the total workload, which is how much the entire system works.

$$\text{Total Workload (TW)} = \sum_{k=0}^{nSliders-1} W_k \tag{6}$$

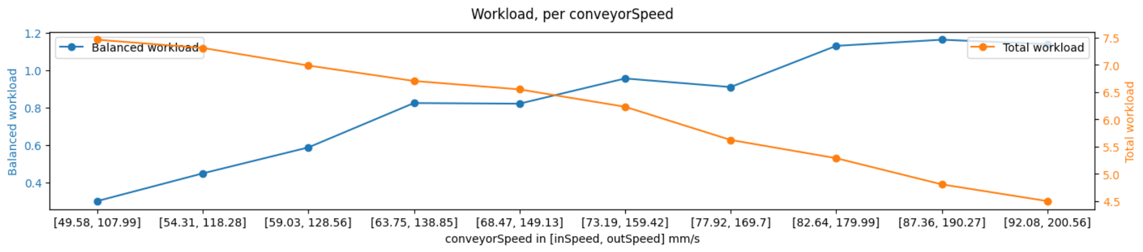


Fig. 5.2: The total workload and the balanced workloads, for each conveyor speed. Note that there are two different scales. One for each plot line.

Fig. 5.2 clearly demonstrates that the higher the conveyor's speed, the worse the system gets. The system becomes increasingly unbalanced while also decreasing the total workload. This can be attributed to the fact that higher speeds make pick-and-place tasks more challenging. The system has a narrow workspace for each beam, and it becomes a constraint at those higher speeds. Due to the narrow workspace the beams have, this issue becomes more pronounced at higher speeds.

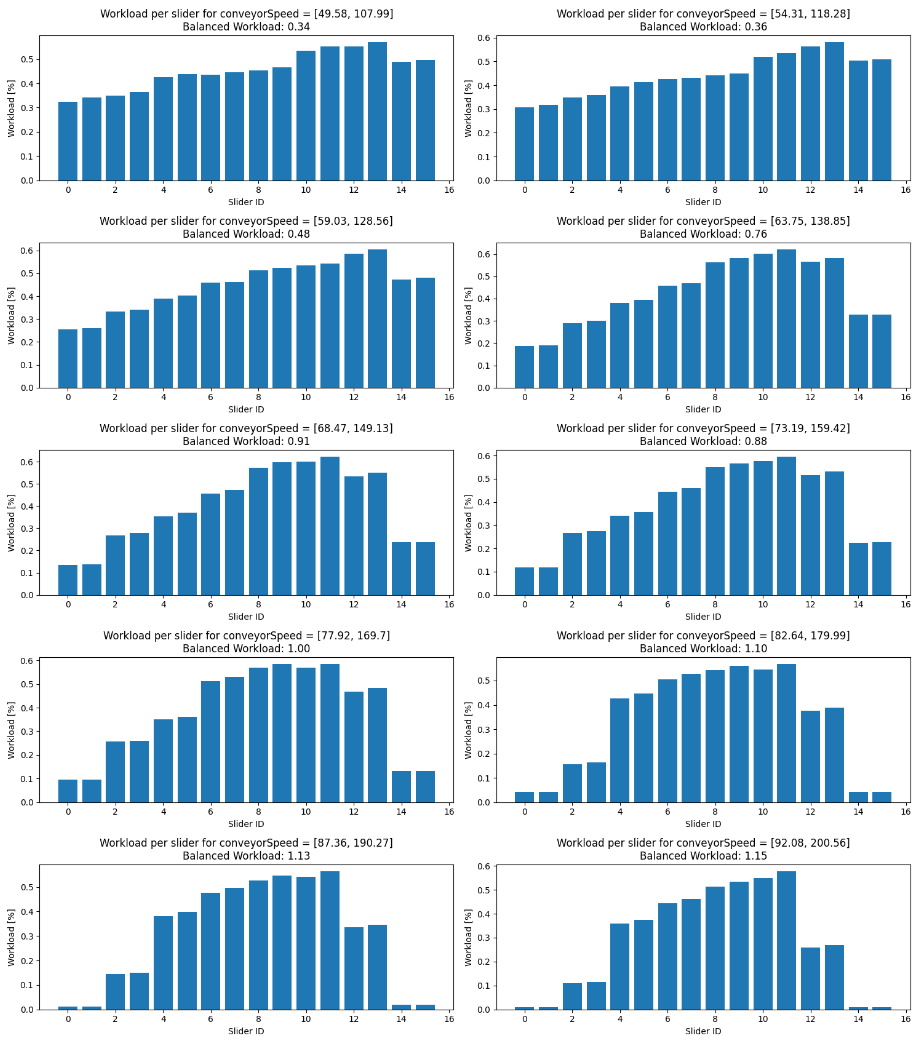


Fig. 5.3: This showcases the workload per slider, per conveyor speed.

Examining the individual workload of each slider in Fig. 5.3, we observe that the outer beams handle fewer tasks at higher speeds. This is probably because they are left with the most difficult pick-and-place tasks while having a minimal reaction time. Furthermore, if a batch of 3+ targets are available to the last rail, it is highly probable that at least one target will be missed. This issue is further expanded in Sec. 5.4. In contrast, the inner two beams have a larger selection of picks-and-place position available. As such, they handle most of the workload.

Finally, we can look at the picks/min for each slider. These plots are extremely similar to the workload's one, which makes sense. A travel time represent a pick or a place. In this metric, a pick is considered as the full pick and place task for a target. This is also called as items/min. Fig. 5.4 displays the different values of picks/min per slider for each conveyor speed. It also displays the average picks/min of this configuration, as well as the theoretical maximum average pick/min, computed with the Excel sheet (see Sec. 3.3.3).

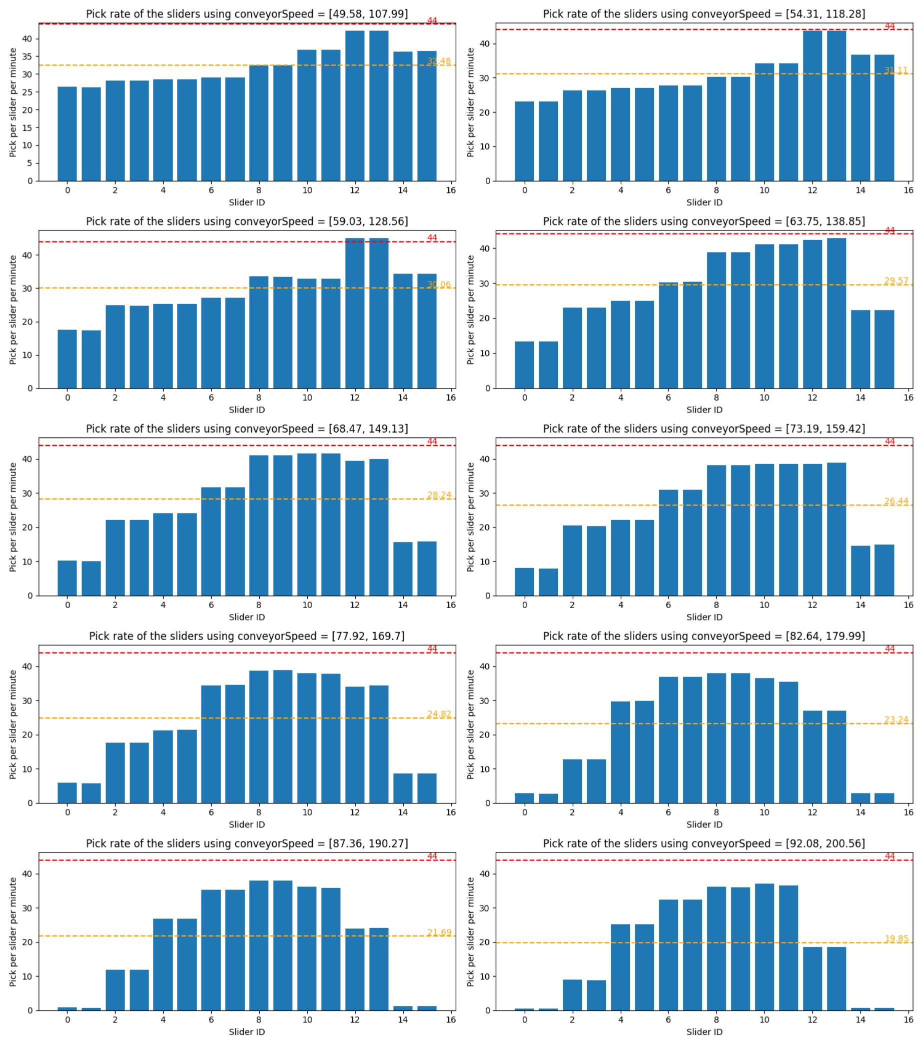


Fig. 5.4: Picks/min per slider, for each of the conveyor speed tested.

Once more, we can observe that with higher conveyor's speed, the picks/min of the system deteriorates.

These plots highlight a major flaw. The inner two beams handle most of the workload, while the outer two beams do significantly less. When this was first noticed, it has been decided to add the fourth scheduling strategy: LPT. The idea behind this decision, was to make the work harder for the inner two beams. These beams have plenty of choice to pick/place from, and therefore should prioritize the worst case scenario. In contrast, this would mean the easiest pick/places will be left for the outer beams, thus increasing their workload. In Sec. 5.3, we will analyze the different scheduling combinations, and see whether the introduction of this fourth scheduling strategy did help or not, and if different scheduling combination helps in regards of the balanced workload.

5.2 State bouncing and pre-move's impact

Following the visual inspection from Sec. 4.2, it was determined that modifications to the algorithm were necessary to improve the system's performance and eliminate avoidable idle time. However, this inspection was conducted on a limited number of seeds. Each evaluation required watching the entire simulation runtime (typically 3 minutes), making it time-consuming, and potentially unrepresentative of the general use case. Moreover, a visual inspection focuses solely on identifying imperfections or incorrect behaviors, while disregarding factual results.
To objectively assess the impact of the proposed modifications, a performance analysis is required. In most scenarios and most conveyor speed, the inner two beams handle most of the workload, with minimal downtime. Therefore, these modifications shouldn't have a significant impact when applied to those beams. To confirm this assumption, these methods have been tested both on all the beams, and only the outer beams. For each configurations, these methods have been tested across 10 runs at different conveyor speed, in the same manner as in Sec. 5.1. This will provide hindsight on the stability of the system, and the independence of the conveyor speed and these methods.
Finally, the best-performing configurations of both the pre-move method and state bouncing were tested together to assess their combined impact on system performance.

5.2.1 Pre-move's impact

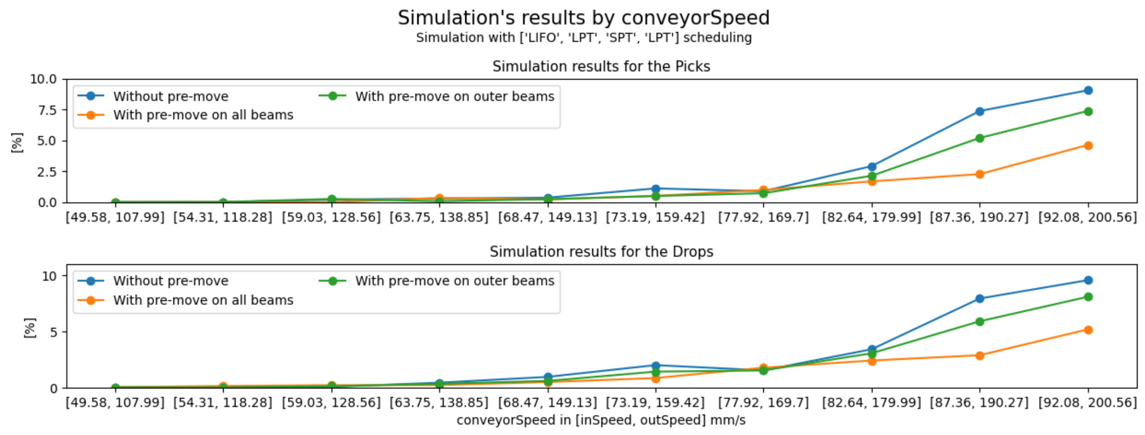


Fig. 5.5: Percentage of missed picks and missed drops, for all three configurations of pre-move. Lower is better.

In Fig. 5.6, we can observe that this method had a positive impact on the performance. This improvement is significantly felt at higher speeds. However, surprisingly, it performs best when the pre-move is applied on all the beams. This can be explained by the fact that the output conveyor doesn't have a constant flow. Because there are two spacings, the one between rows, and the one between packages, the beams sometimes have to wait for the next packages to arrive, leading to some downtime. The impact of varying packages spacing will be tested in Sec. 5.4. Additionally, by looking at the density of missed picks/drops across different speed with Fig. 5.6, we can again confirm that the pre-move indeed has a positive impact on the system.

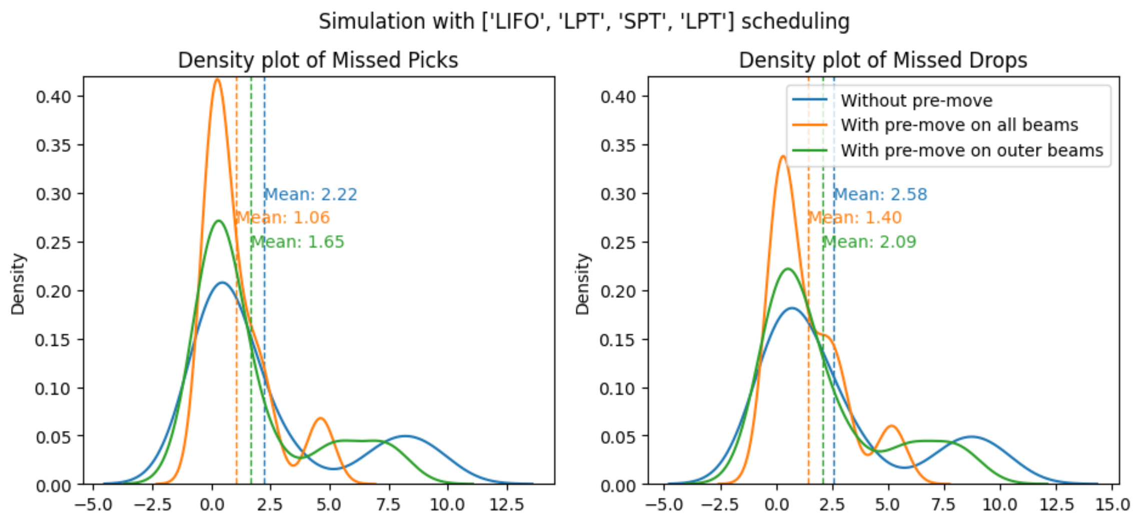


Fig. 5.6: Density of missed picks and missed drops's percentage.

Although the balanced workload has been defined as one of the parameter to optimize, it shouldn't be compared across these three configurations. Because the pre-move makes a slider move from one conveyor to another, and not necessarily directly to the next target, it induces more movement on the sliders. The outer beams are the one benefiting the most from this method, therefore, the system automatically becomes slightly more balanced than without the pre-move.

5.2.2 State bouncing's impact

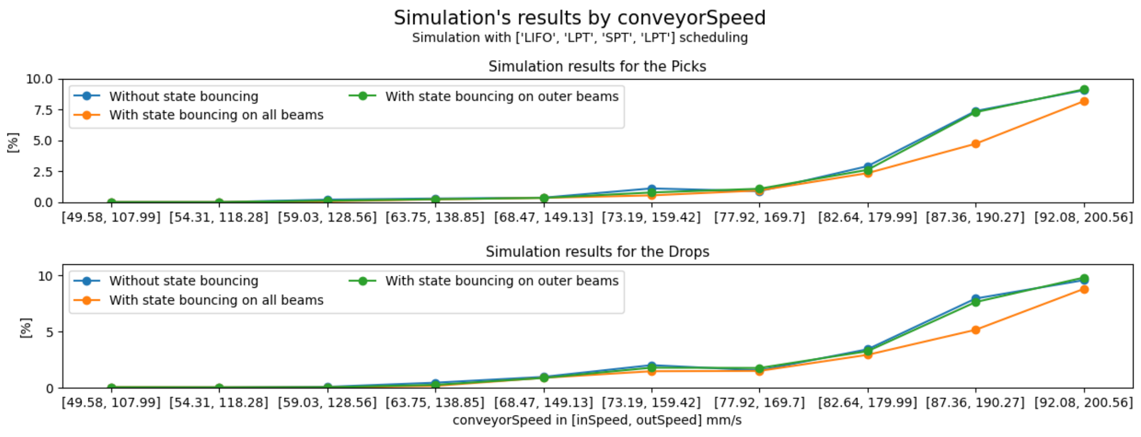


Fig. 5.7: Percentage of missed picks and missed drops, for all three configurations of state bouncing. Lower is better.

Similar to the pre-move method, an improvement in performance is noticeable, particularly when applied to all beams, which is again unexpected. The state-bouncing was initially introduced to assist the outer beam to avoid missing a target by waiting for an opportunity for only one of the two sliders. However, according to these graph, the primary beneficiaries of this method are the inner two beams. Again, this effect is likely due to the packages spacing, which creates idle time for all sliders, regardless of their beam.
Based on this observation, we can assume that both the pre-move and state-bouncing will provide diminishing returns on the performance improvement as the package's spacing decreases. This assumption is further examined in Sec. 5.4.
For the same reasons outlined in the analysis of the pre-move's method, we do not compare workload metrics for these configurations. While the pre-move method has minimal impact on the workload, the state-bouncing can, in some case, double or even triple the travel time of a single pick-and-place task.

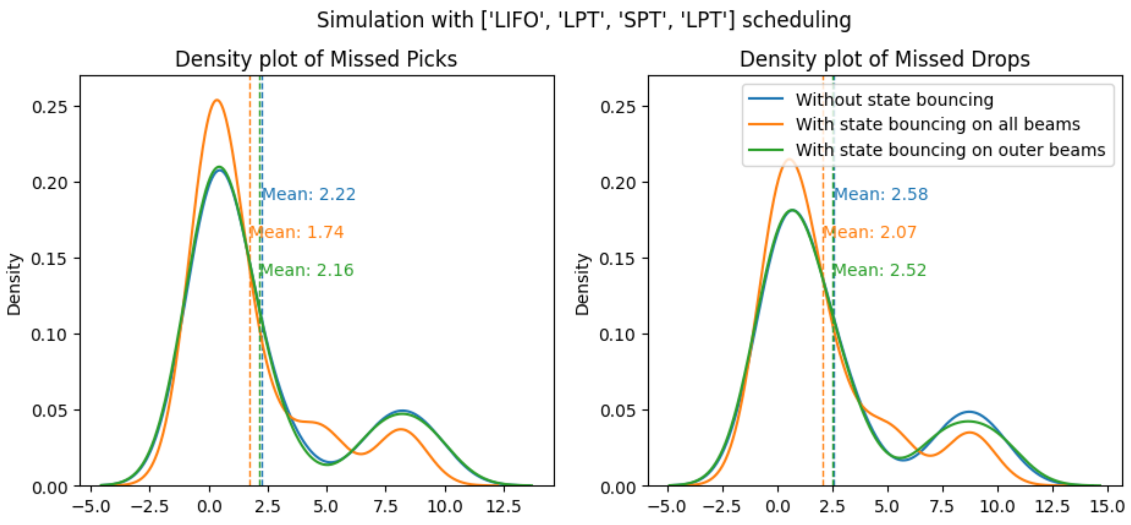


Fig. 5.8: Density of missed picks and missed drop's percentage.

5.2.3 Pre-move and state-bouncing combined impact

Finally, we can compare the results when both are applied. Based on the previous results, this final comparison is with these methods applied on all the beams.

Fig. 5.9 highlights the benefit of combining the state-bouncing with the pre-move function on all beams. By using both, we manage to combine both benefits to make an even better system.

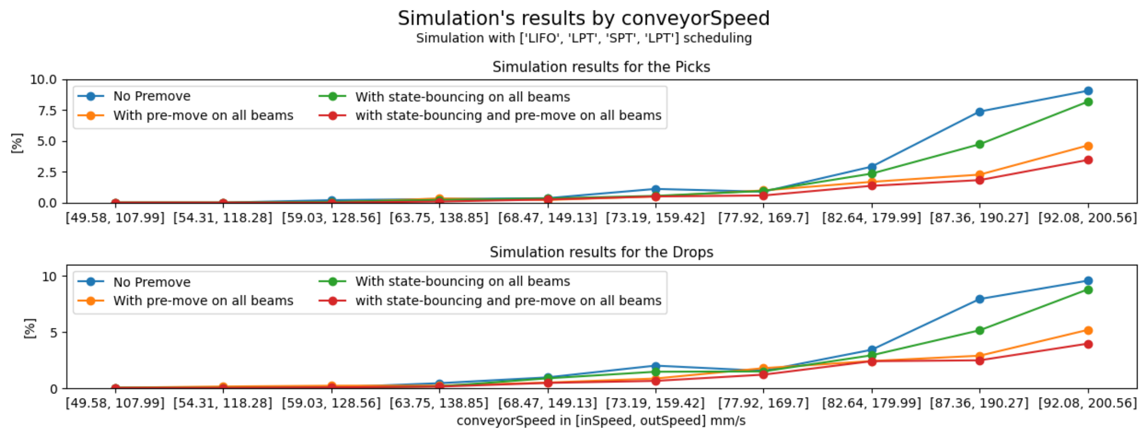


Fig. 5.9: Percentage of missed picks and missed drops, with state-bouncing, with pre-move, with both and with neither. Lower is better.

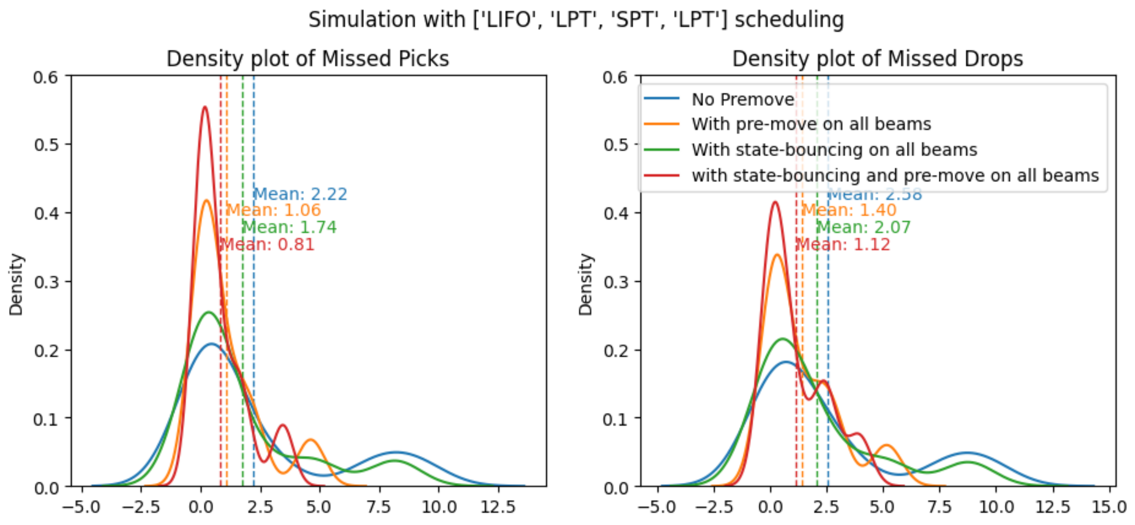


Fig. 5.10: Density of missed picks and missed drop's percentage.

5.3 Schedulings

Each scheduling has a different behavior, and impacts the system differently. Furthermore, the beam which we assign a scheduling also matters. For instance, Humbert et al. [10] have shown that assigning the last robot with LIFO should be avoided, as it yielded their worst results, and it make sense. With LIFO, we give the priority to the newcomers, meaning that if a target was already present but not assigned yet, it is likely to be missed by the machine. Therefore, the LIFO is removed from the possible scheduling strategies for the last beam. With this decision, we are left with $4^3 \cdot 3 = 192$ possible combinations. Each set of parameters is once more tested 10 times, resulting in 1920 simulations.

With this many simulations, we have a lot of numbers and data to analyze. We need an efficient way to evaluate these. The first step is to standardize the data, which is data centering and data normalization. It is done through :

$$Z = \frac{X - \mu}{\sigma} \tag{7}$$

Where X are the data points, μ is the mean of the data, and σ is the standard deviation of the data. This step is necessary, because not all the data have the same scale, such as the workload, or the percentage of missed drop.
Then, we can sum the values that are of interest for us, namely the percentage of missed drops, the percentage of missed picks, and the BW, with given weights. This will give a score to the given set of parameters. Our goal is to analyze what scheduling, on what beam has the biggest impact. This is best carried out by using a heat map, linking the scheduling strategies to the beams.

Using a score based on the missed drops and missed picks only. In this first case scenario, we want to observes the raw performance of the system, meaning the score will only include the normalized missed drops and missed picks, and is given by :

$$score = 0.9 \cdot missedPicks + 1.2 \cdot missedDrops \tag{8}$$

A missed drops means an unfilled packages. For customers, it is crucial that every packages are filled. An unfilled package represents a higher loss than a missed pick, because the entire package cannot be distributed if it isn't full. Therefore, the missed drops is the most important metric and should be given the higher weight. The weights have been given arbitrarily.

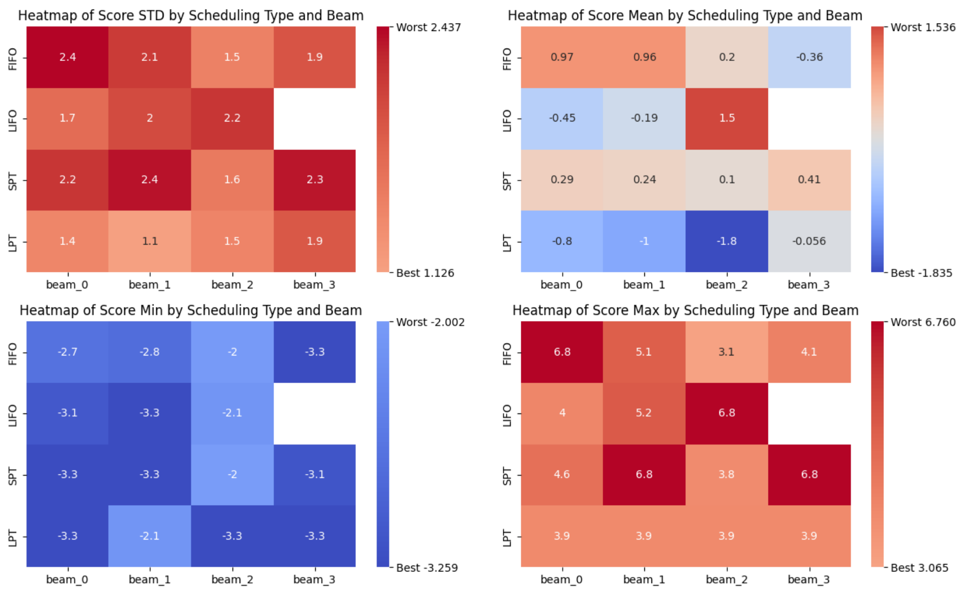


Fig. 5.11: Heatmaps of the standard deviation of the scores, their mean, their minimum value and maximum value.

In Fig. 5.11, we can observe different heatmaps of the scores. The heatmap of standard deviation describes how stable the results are with the given scheduling on the given beam. For instance, using FIFO on the first beam or SPT on the second beam results in the biggest variation of results, where as using LPT on the second beam yields the most stable results. Ideally, we want to keep the more stable results. The second heatmap shows the mean of the scores for a given scheduling on a given beam. The lowest value is the best, and using LPT on the third beam yields by far the best results, and using LIFO on the third beam returns the worst results. We can already assume that using LIFO on the third beam should by avoided. We can also observe that the system doesn't perform very well when FIFO is applied to the first or the second beam and should also be avoided. Finally, we can observe the addition of the LPT scheduling is a great addition to the system. It has the best average score on the first three beams and provides the most stable results on all beams.

Table 5.1 shows the results from the 20 best parameter combinations with the score formula given in equation (8). The first noticeable thing is all of those results have LPT assigned to the third beam (Beam2). Furthermore, the second beam never has LPT. In general, it has been observed that when LPT is assigned to both the second and the third beam, the system doesn't perform very well. Finally, FIFO has the lowest presence of all scheduling, in this top 20, for the first and the second beam.

MP	MD	Score	BW	Beam0	Beam1	Beam2	Beam3
-1.573	-1.536	-3.259	-0.453	LPT	LIFO	LPT	FIFO
-1.317	-1.726	-3.257	-0.245	SPT	SPT	LPT	LPT
-1.381	-1.574	-3.131	-0.376	LPT	SPT	LPT	SPT
-1.369	-1.548	-3.090	1.188	LPT	SPT	LPT	FIFO
-1.318	-1.574	-3.074	-0.365	LIFO	SPT	LPT	LPT
-1.240	-1.574	-3.004	1.062	SPT	SPT	LPT	FIFO
-1.547	-1.320	-2.976	-0.326	SPT	LIFO	LPT	FIFO
-1.381	-1.345	-2.857	-0.353	LIFO	LIFO	LPT	FIFO
-1.137	-1.497	-2.820	-1.640	SPT	LIFO	LPT	LPT
-1.313	-1.360	-2.813	0.947	LIFO	SPT	LPT	FIFO
-1.213	-1.383	-2.751	0.713	LPT	FIFO	LPT	FIFO
-1.158	-1.418	-2.744	-0.189	LPT	SPT	LPT	LPT
-1.149	-1.396	-2.709	-1.625	LPT	LIFO	LPT	SPT
-1.137	-1.383	-2.683	-0.869	FIFO	SPT	LPT	LPT
-1.276	-1.269	-2.671	0.232	FIFO	SPT	LPT	FIFO
-0.997	-1.472	-2.663	-1.673	LIFO	LIFO	LPT	LPT
-1.292	-1.231	-2.640	0.666	LIFO	FIFO	LPT	FIFO
-1.315	-1.142	-2.554	-1.613	LIFO	LIFO	LPT	SPT
-1.276	-1.155	-2.534	-1.467	SPT	LIFO	LPT	SPT
-1.302	-1.075	-2.462	0.051	FIFO	LIFO	LPT	FIFO

Table 5.1: List of the 20 best results from the 192 parameter combinations, according to the score. MP stands for missed picks, MD stands for missed drops. Lower is better. The best value of each column is shown in green.

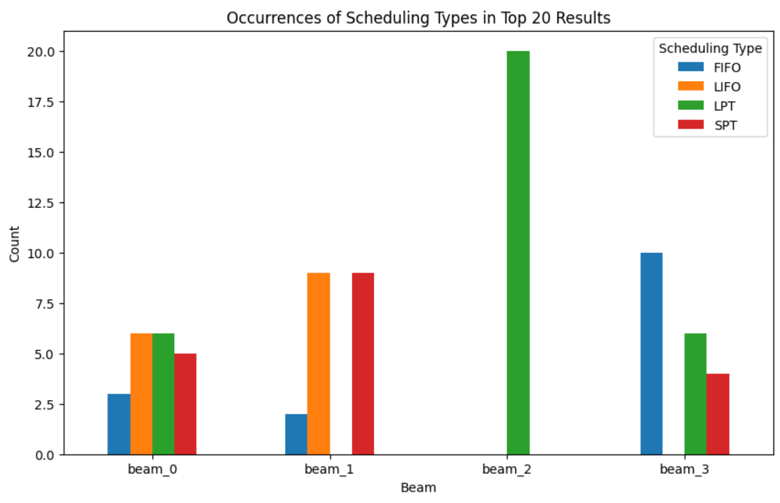


Fig. 5.12: Bar plot of the presence of each scheduling, per beam, for the best 20 results. This supports the Table. 5.1

Using a score that now includes the BW If the customer wants to consider even wear of the system, the BW should be taken into account. The formula becomes :

$$score = 0.9 \cdot missedPicks + 1.2 \cdot missedDrops + 0.4 \cdot BW \tag{9}$$

With this new equation, the LPT remains the best performing scheduling for the third beam, and LIFO the worst. This again confirms LPT is a great addition to the list of schedulings.

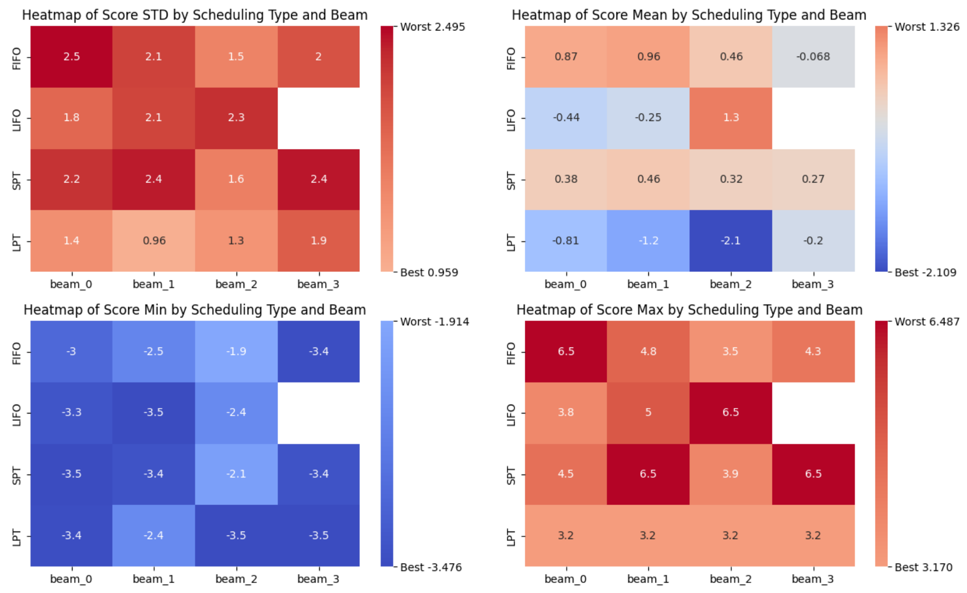


Fig. 5.13: Heatmaps of the standard deviation of the scores, their mean, their minimum value and maximum value.

We can conclude that including the BW into the score's formula didn't drastically change the best 20 parameter combinations. Some swapped places, such as the first and the second best combination.

A visual inspection has also been conducted, to determine what combinations makes the system better, but it is a difficult task. With the current configuration, we have a circular problem. Each beams impacts the next, but also impacts the other conveyor's feed. Because we are using counter-current flow, the impact of each beam loops around the system rendering such analysis extremely difficult.

Furthermore, the issue with this approach, is that it analyses the individual impact of each scheduling on each beam, but there is probably a collaborative behavior. For instance, scheduling X could work better when paired with scheduling Y following it, rather than scheduling Z.

MP	MD	Score	BW	Beam0	Beam1	Beam2	Beam3
-1.137	-1.497	-3.476	-1.640	SPT	LIFO	LPT	LPT
-1.573	-1.536	-3.440	-0.453	LPT	LIFO	LPT	FIFO
-1.149	-1.396	-3.360	-1.625	LPT	LIFO	LPT	SPT
-1.317	-1.726	-3.355	-0.245	SPT	SPT	LPT	LPT
-0.997	-1.472	-3.333	-1.673	LIFO	LIFO	LPT	LPT
-1.381	-1.574	-3.281	-0.376	LPT	SPT	LPT	SPT
-1.318	-1.574	-3.220	-0.365	LIFO	SPT	LPT	LPT
-1.315	-1.142	-3.199	-1.613	LIFO	LIFO	LPT	SPT
-1.276	-1.155	-3.121	-1.467	SPT	LIFO	LPT	SPT
-1.547	-1.320	-3.107	-0.326	SPT	LIFO	LPT	FIFO
-0.891	-1.256	-3.036	-1.817	LPT	LIFO	LPT	LPT
-1.137	-1.383	-3.031	-0.869	FIFO	SPT	LPT	LPT
-1.381	-1.345	-2.999	-0.353	LIFO	LIFO	LPT	FIFO
-1.186	-1.045	-2.856	-1.337	FIFO	LIFO	LPT	SPT
-1.158	-1.418	-2.820	-0.189	LPT	SPT	LPT	LPT
-1.369	-1.548	-2.615	1.188	LPT	SPT	LPT	FIFO
-1.084	-1.205	-2.601	-0.447	SPT	SPT	LPT	SPT
-1.240	-1.574	-2.579	1.062	SPT	SPT	LPT	FIFO
-1.276	-1.269	-2.579	0.232	FIFO	SPT	LPT	FIFO
-0.803	-1.078	-2.561	-1.361	FIFO	LIFO	LPT	LPT

Table 5.2: List of the 20 best results from the 192 parameter combinations, according to the score, when using the BW. MP stands for missed picks, MD stands for missed drops. Lower is better. The best value of each column is shown in green.

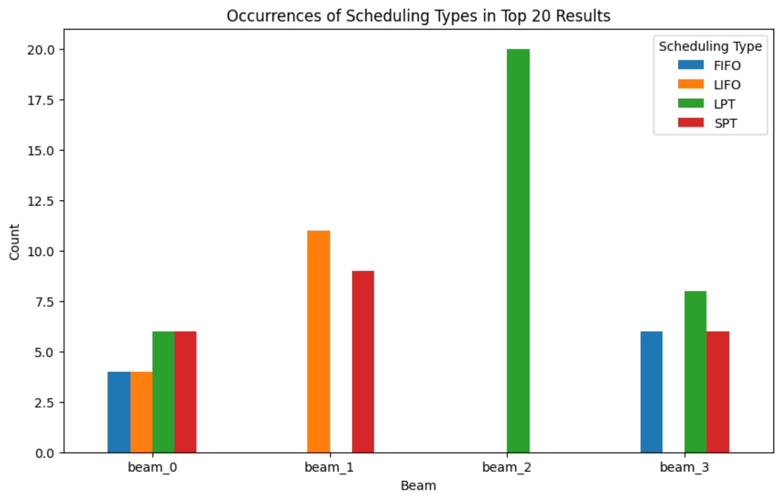


Fig. 5.14: Bar plot of the presence of each scheduling, per beam, for the best 20 results. This supports the Table. 5.2

5.4 Changes of throughput

The issue of the packaging spacing inducing idle time in the process has already been mentioned Sec. 5.1, Sec, 5.2.1 and Sec. 5.2.2. In theory, the outflow conveyor's speed has been computed to have the same flow rate as the inflow conveyor. But because of this package spacing, the outflow rate isn't continuous.

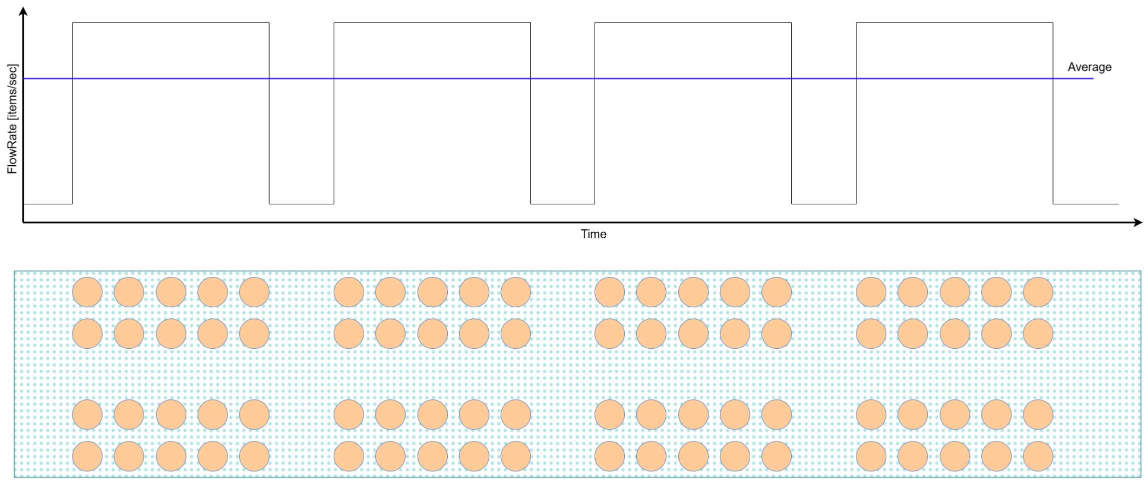


Fig. 5.15: Depiction of the outflow rate with the packages spacing

According to the excel sheet (see Sec. 3.3.3), the system should be able to handle the average flow rate set in the program. However, with the extra packages spacing, the flow rate is actually a two step flow rate. Either we have a high flow rate (the peak value), or there is no targets available. This induces idle time which the state bouncing and the pre-move aims to fix. But because the peak flow rate is higher than what it is supposed to be, when a targets slips through the beams, it is actually often a batch of targets, instead of a single one. And as we've discussed before, if multiple targets present themselves to the last rail, it won't be able to handle all of them.

To confirm this theory, multiple simulations have been made using different extra packages distance. For each, the outflow conveyor speed was computed to ensure the average flow rate was the same for both conveyor. These simulation have been ran without the pre-move and the state-bouncing.

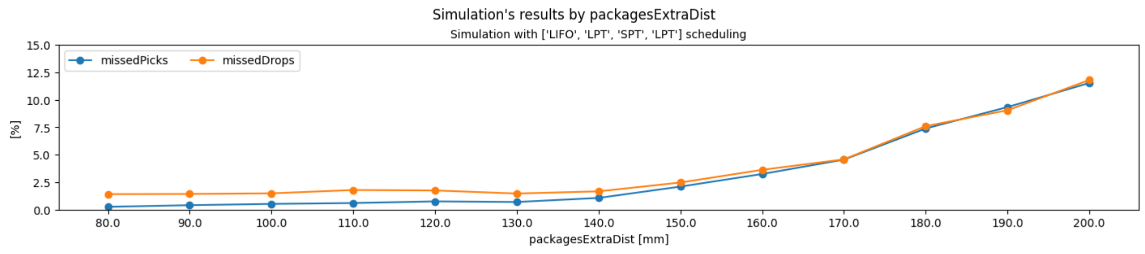


Fig. 5.16: Percentage of missed drops and missed picks, for varying extra packages distance.

Fig. 5.16 clearly captures the impact of the extra package's distance. With greater spacing, the peak flow rate becomes increasingly difficult to follow for the system, and it can't keep up. The last previous simulations have been made with an extra package's spacing of 140mm.

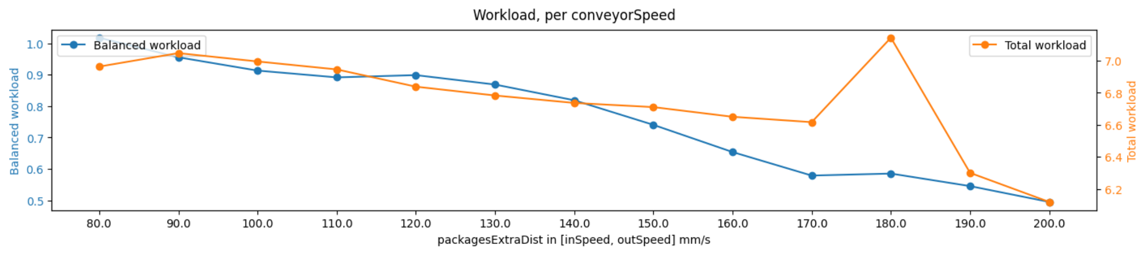


Fig. 5.17: The total workload and the balanced workload of the system under different packages spacing.

Finally, Fig. 5.17 showcases the total workload and the balanced workload under these different spacing values. For both, their value decreases with higher packages spacing. With higher spacing, there are more idle time, and this results in lower total workload value. But, interestingly, it also makes the system more balanced. This is probably caused by the fact that having such high package spacing makes the entire system waits, and then work all at once.

From these findings, we know the packages spacing does have an impact on the performance of the system. However, this is not always a variable that can be played with. It is up to the customer's need and constraints.
Despite knowing this, it could be interesting to see whether the scheduling's impact are the same with different packages spacing, or if they are dependent. For this reasons, the same experiment as shown in Sec. 5.3 is conducted, but with a spacing of 100mm. We limit the testing to this value only and not a range of packages spacing, because this experiment is already very time consuming and contains a lot of data. Having more parameters would make it even harder to extract information than it already is. The score value used is without the balanced workload. First, we can see the LPT is still a really good option, if not the best, for the third beam, while the LIFO is remains the worst. FIFO isn't as bad as previously for the first beam, but remains a poor choice for the second beam. SPT has also become a much better choice for the first three beams. Finally, FIFO is looking like a great choice for the last beam, just as Humbert et al. [10] have shown. If we look at the table 5.3 of values, we can observe the LPT is still an extremely dominant choice for the third beam, but not as dominant. Out of these 20 solutions, 3 have SPT for the third beam. This time, FIFO is looking like a much better options for the last beam, as the 7 best solutions uses it. FIFO also has a higher appearance for the first beam, with a presence of 6/20, versus 3/20 in Table. 5.1, at the expense of the LPT, which now has a presence of 1/20 for the first beam, versus 6/20.

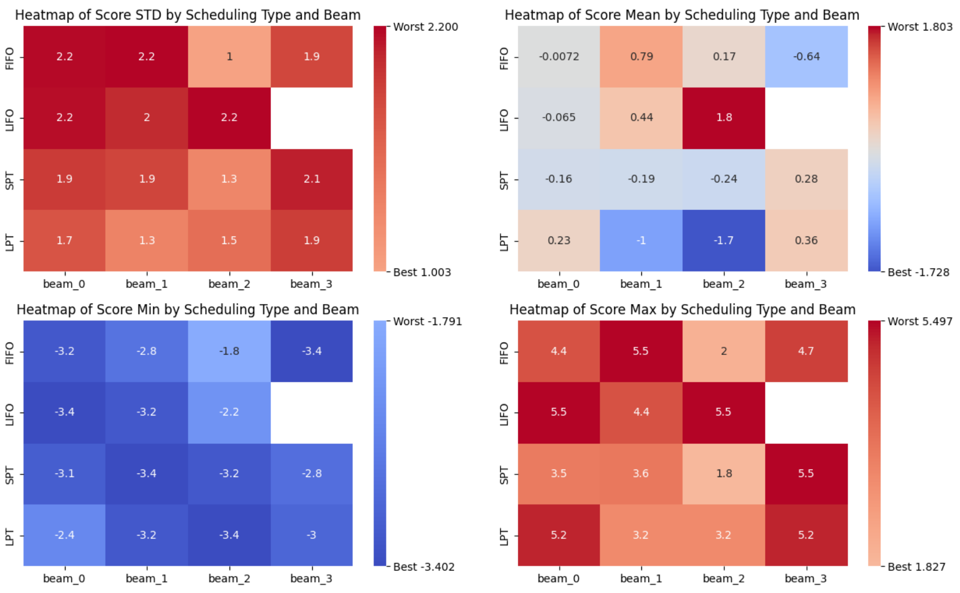


Fig. 5.18: Heatmap of the scores using a shorter packages's spacing.

MP	MD	Score	BW	Beam0	Beam1	Beam2	Beam3
-1.445	-1.751	-3.402	-0.003	LIFO	SPT	LPT	FIFO
-1.454	-1.585	-3.211	0.166	FIFO	LPT	SPT	FIFO
-1.378	-1.637	-3.205	-0.556	FIFO	LIFO	LPT	FIFO
-1.346	-1.599	-3.130	-0.092	SPT	LIFO	LPT	FIFO
-1.280	-1.599	-3.071	0.026	LIFO	LIFO	LPT	FIFO
-1.379	-1.523	-3.069	1.033	SPT	SPT	LPT	FIFO
-1.445	-1.447	-3.038	-0.613	FIFO	SPT	LPT	FIFO
-1.082	-1.675	-2.984	-0.844	LIFO	SPT	LPT	LPT
-1.180	-1.485	-2.845	-1.790	FIFO	SPT	LPT	SPT
-1.147	-1.485	-2.815	-1.016	SPT	SPT	LPT	LPT
-1.313	-1.333	-2.781	0.842	LIFO	FIFO	LPT	FIFO
-1.115	-1.409	-2.695	-1.318	LIFO	LIFO	LPT	SPT
-1.350	-1.126	-2.566	0.704	SPT	FIFO	LPT	FIFO
-0.949	-1.409	-2.545	-1.523	FIFO	SPT	LPT	LPT
-0.982	-1.371	-2.529	-1.264	LIFO	SPT	LPT	SPT
-1.015	-1.333	-2.513	1.003	LIFO	LPT	SPT	FIFO
-1.245	-1.143	-2.492	-0.026	SPT	FIFO	LPT	SPT
-1.378	-0.991	-2.429	0.182	LPT	SPT	LPT	FIFO
-1.114	-1.187	-2.427	-1.183	SPT	SPT	LPT	SPT
-0.884	-1.333	-2.395	-0.083	FIFO	LPT	SPT	SPT

Table 5.3: List of the 20 best results from the 192 parameter combinations, according to the score. This time, it is by using a package's spacing of 100mm instead of 140mm. MP stands for missed picks, MD stands for missed drops. Lower is better. The best value of each column is shown in green.

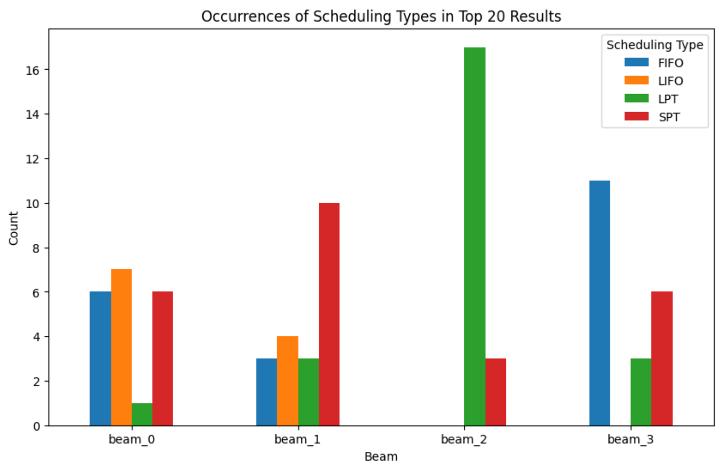


Fig. 5.19: Bar plot of the presence of each scheduling, per beam, for the best 20 results. This supports the Table. 5.3

From this, we can conclude the schedulings and the packages spacing are somewhat dependent. LPT remains a good choice for the third beam. For the rest, the scheduling assignment should be done on a per-case basis.

5.5 Final configuration and final note

With all of these experiment, we can conclude the robot behaves best with the state-bouncing and the pre-move enabled. Furthermore, the scheduling combination SPT-LIFO-LPT-LPT should be preferred if we want to consider the BW, and LPT-LIFO-LPT-FIFO should be preferred if we choose to ignore the BW.

Fig. 5.20 compares our initial simulation across different speed (from Sec. 5.1), to the best configuration (ignoring the BW). It uses both the state-bouncing and pre-move method, and the LPT-LIFO-LPT-FIFO scheduling assignment. Both simulations used a package spacing of 140mm.

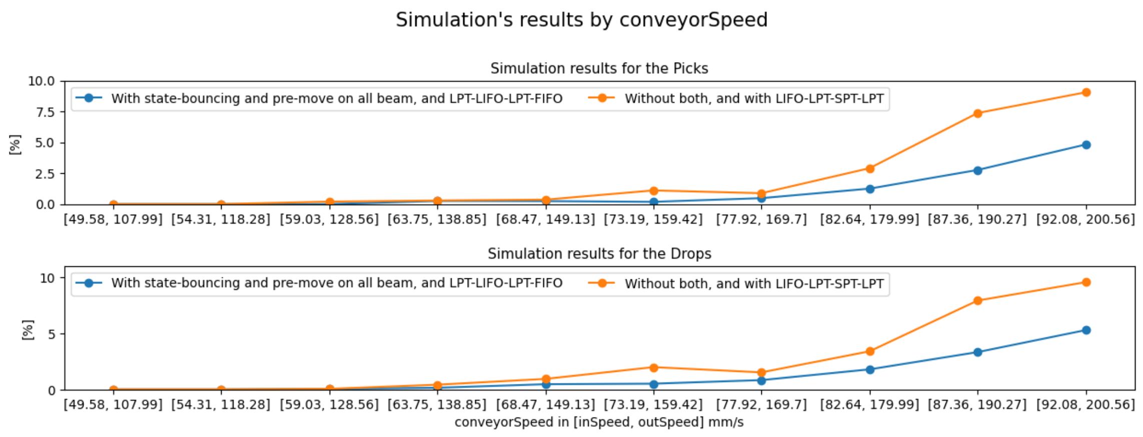


Fig. 5.20: Percentage of missed drops and missed picks, across different conveyor speed. Lower is better. The values are compared to the values found in Sec. 5.1

We can observe a great improvement at all speeds, compared to the initial test. It unfortunately doesn't reach a 0-loss at 73.19 mm/s for the inflow conveyor. At said speed, the percentage of missed picks is 0.18% (versus 1.11% on the initial simulation), and 0.5% of missed drops (versus 1.98% for the initial simulation).

Looking at the workload per slider in Fig. 5.21, we realize that the system is still largely imbalanced at higher speed. This is due to, among other things, the state-bouncing and pre-move method, as well as the issue highlighted about the package spacing. Additionally, we can notice the workload never exceeds 70%. This is another effect of the package spacing. it induces idle time which can be felt in the workload of each slider.

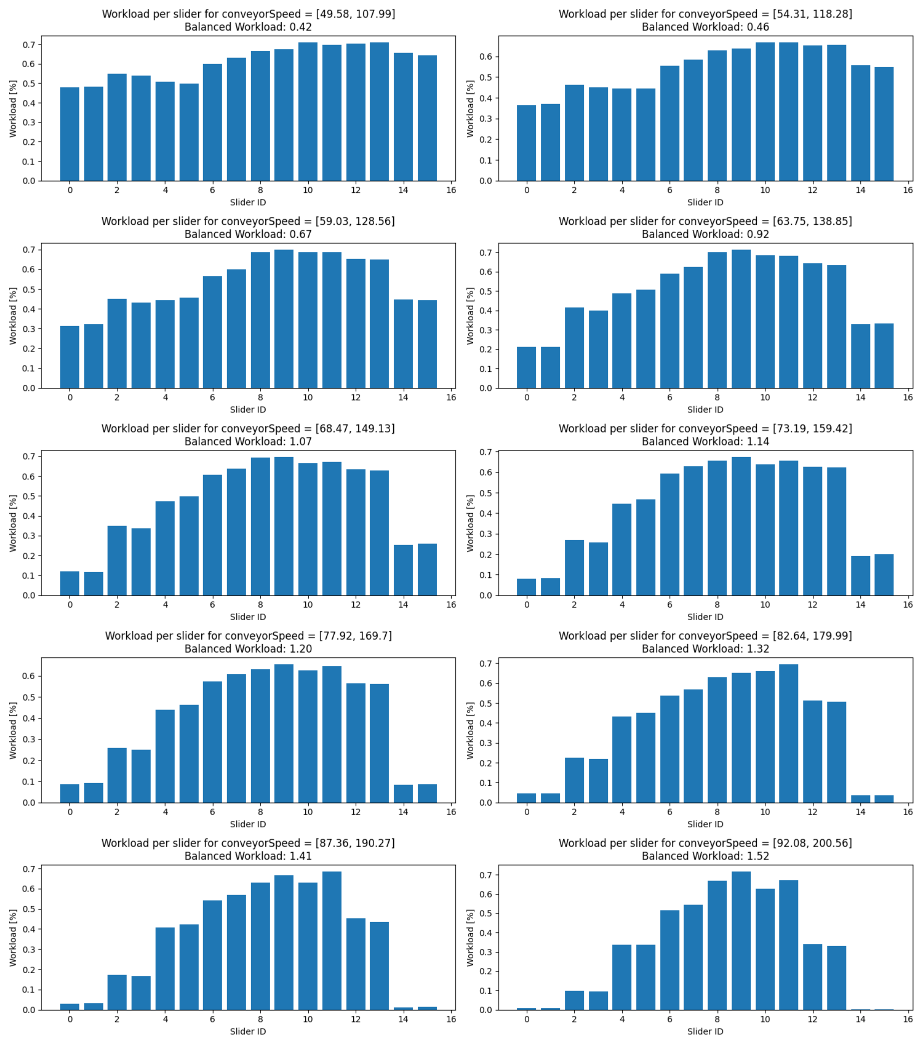


Fig. 5.21: Workload per slider per conveyor speed, for the best configuration at 140mm of spacing

Another issue which should be highlighted is the start routine. In the current implementation, both conveyors start a different times to ensure that the first row of picks meets the first row of drops at the center of the conveyors. The delay is computed based on the conveyors' respective flow rates. However, visual inspection has revealed that a significant amount of loss occurs on startup before performance stabilizes. This observation is confirmed by Fig. 5.22 which present a histogram of the losses recorded every 10 seconds.

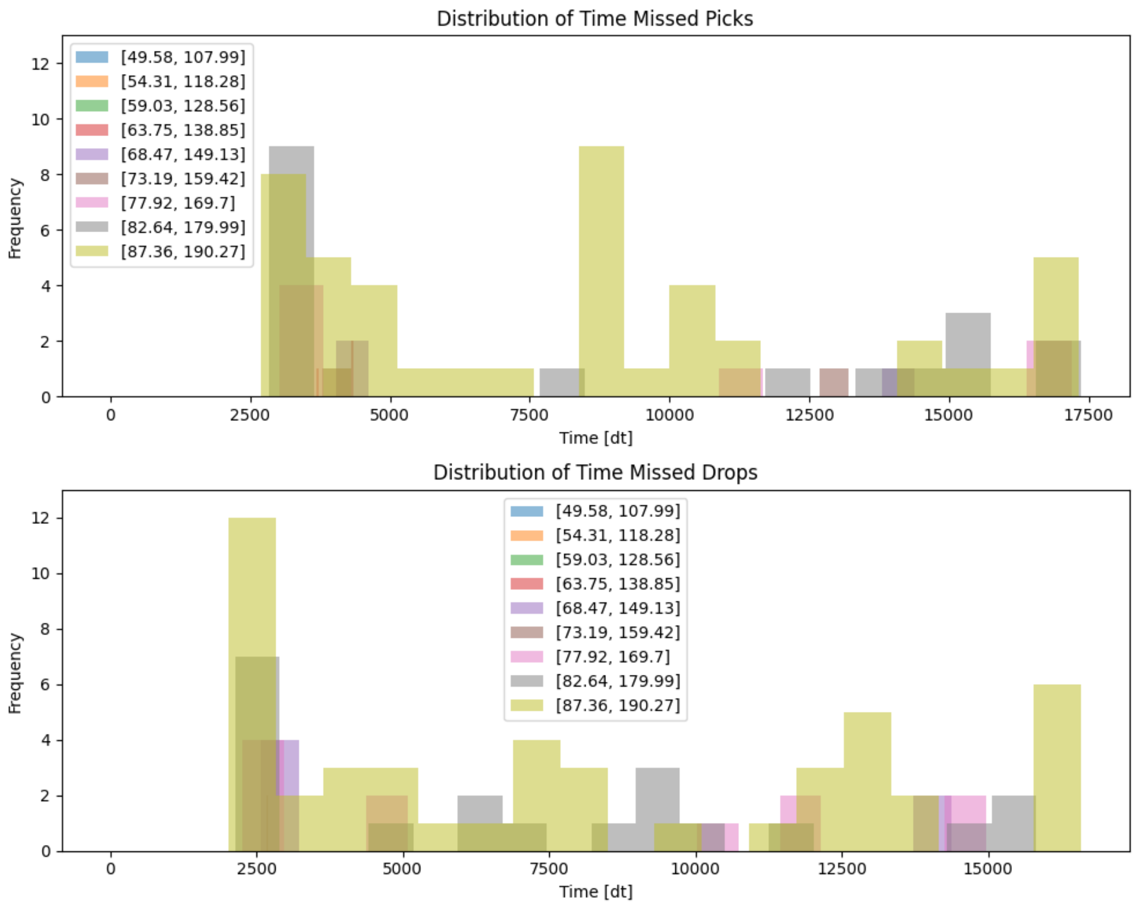


Fig. 5.22: Histogram of the loss for each 10 secs (1dt = 10ms). Each color represents a different set of speeds. The delay between 0 and the first missed picks/drops is the time it took for the first missed pick/drop to leave the workspace.

An improved startup routine would allow the conveyors to start and pause their movement on demand. This approach could enhance overall system performance by preventing losses, albeit at the cost of a increase in processing time.
However, such an approach is not compatible with the `findTarget` function. This function assumes a constant speed for the conveyor, and relies on this information to find a meeting point between a target and a slider. If the conveyor can pause on demand, these meeting point could become incorrect. Since this function is unable to predict when the conveyor is going to pause, and adjust accordingly, implementing this method would require substantial modifications. Moreover, the new implementation would need to be highly robust, as multiple scenarios could potentially block the entire system.

6 Conclusion

This thesis presented the development, validation, and performance analysis of a novel pick-and-place system operating under various scheduling strategies. The system's behavior was evaluated through multiple approaches to assess the impact of scheduling assignments, conveyor speed, and algorithmic modifications.

Initially, a visual inspection was conducted to reveal potential inefficiencies of the system, particularly avoidable idle time and workload imbalance among the beams. To address these issues, the LPT scheduling strategy was added to the pool of scheduling and the state-bouncing and pre-move's method were introduced in the algorithm. Data analysis confirmed these decisions were more effective than expected, with the LPT proving to be a strong candidate, particularly for the third beam. Unfortunately, no clear pattern emerged for scheduling assignment, other than the LPT being an excellent scheduling candidate, especially for the third beam, regardless of the context. This reinforces the idea that each case is unique.

Further experiments revealed the significance of package spacing. The larger this parameter is, the more idle time are imposed and are disrupting the system's throughput. Although the pre-move and state-bouncing method have been introduced to mitigate the idle time issue, the non-continuous flow rate remained a challenge, with a peak flow rates difficult for the system to handle. Additionally, the system remains unbalanced, with the outer beams often idle. Therefore, the distance between packages should be set as low as possible.

Another critical issue, which has been discussed in Sec. 5.5, is the high percentage of losses occurring at the system's startup. The current implementation struggles to efficiently manage the initial phase of operation, leading to higher losses, before stabilization.

Most of these issue could potentially be solved by using another layer of strategy. In Humbert et al. [10], a strategy was used to assign the targets to each robot as soon as they appeared in the system. With such strategy, the machine automatically becomes more balanced. Such implementation could also improve the performance of the system and could help it achieve its theoretical maximum average pick/min, which was found using the excel sheet (see Sec. 3.3.3). However, implementing such a strategy might increase the computational complexity and require greater processing power.

Another potential area for future work could be to explore an improved start routine, either by making one that is compatible with the `findTarget` algorithm, or by modifying it, to mitigate the early losses of the system. Further exploration of scheduling combinations could also be beneficial, particularly through other machine learning approaches. Given the number of parameters influencing system performance, deep learning techniques could be a viable option for identifying optimal configurations.

Finally, it is important to note that all work in this thesis was conducted in a simulation environment. While the simulation was designed to be as realistic as possible, real-world testing is necessary to validate the software and apprach.

A large amount of work, during this thesis, has been toward developing the algorithm, the control strategies and the visualization tool. The software was designed for flexibility, allowing for easy configuration modifications and testing of different functions. This lays the foundation for future work. Since no general rule was identified and system performance depends on many parameters, for now, the software has to be configured on a per-case basis. These tools should be taken advantage of to efficiently find the most appropriate configuration with a given set of parameters, by running a simulation which iterates through multiple different parameters combination.

References

[1] Xi Lan, Yuansong Qiao, and Brian Lee. "Towards Pick and Place Multi Robot Coordination Using Multi-agent Deep Reinforcement Learning". In: *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*. Prague, Czech Republic: IEEE, Feb. 2021, pp. 85–89. ISBN: 978-1-66540-469-3. DOI: 10.1109/ICARA51699.2021.9376433.

[2] Xi Lan, Yuansong Qiao, and Brian Lee. "Coordination of a Multi Robot System for Pick and Place Using Reinforcement Learning". In: *2022 2nd International Conference on Computers and Automation (CompAuto)*. Paris, France: IEEE, Aug. 2022, pp. 87–92. ISBN: 978-1-66548-194-6. DOI: 10.1109/CompAuto55930.2022.00024.

[3] Xi Lan, Yuansong Qiao, and Brian Lee. "Multiagent Hierarchical Reinforcement Learning With Asynchronous Termination Applied to Robotic Pick and Place". In: *IEEE Access* 12 (2024), pp. 78988–79002. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3409076.

[4] Xi Lan, Yuansong Qiao, and Brian Lee. "Application of Multi Agent Reinforcement Learning to Robotic Pick and Place System". In: *2024 10th International Conference on Automation, Robotics and Applications (ICARA)*. Athens, Greece: IEEE, Feb. 2024, pp. 1–6. ISBN: 9798350394245. DOI: 10.1109/ICARA60736.2024.10552987.

[5] Tizian Jermann et al. *An Efficient Multi-Robot Arm Coordination Strategy for Pick-and-Place Tasks using Reinforcement Learning*. arXiv:2409.13511 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2409.13511.

[6] Yongchao Chen et al. "Scalable Multi-Robot Collaboration with Large Language Models: Centralized or Decentralized Systems?" In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, May 2024, pp. 4311–4317. ISBN: 9798350384574. DOI: 10.1109/ICRA57147.2024.10610676.

[7] Zhao Mandi, Shreeya Jain, and Shuran Song. "RoCo: Dialectic Multi-Robot Collaboration with Large Language Models". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, May 2024, pp. 286–299. ISBN: 9798350384574. DOI: 10.1109/ICRA57147.2024.10610855.

[8] Shyam Sundar Kannan, Vishnunandan L. N. Venkatesh, and Byung-Cheol Min. *SMART-LLM: Smart Multi-Agent Robot Task Planning using Large Language Models*. arXiv:2309.10062 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2309.10062.

[9] Wenhao Yu et al. *MHRC: Closed-loop Decentralized Multi-Heterogeneous Robot Collaboration with Large Language Models*. arXiv:2409.16030 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2409.16030.

[10] G. Humbert et al. “Comparative analysis of pick & place strategies for a multi-robot application”. In: *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. Luxembourg, Luxembourg: IEEE, Sept. 2015, pp. 1–8. ISBN: 978-1-4673-7929-8. DOI: 10.1109/ETFA.2015.7301450.

[11] H. Işıl Bozma and M.E. Kalalıoğlu. “Multirobot coordination in pick-and-place tasks on a moving conveyor”. en. In: *Robotics and Computer-Integrated Manufacturing* 28.4 (Aug. 2012), pp. 530–538. ISSN: 07365845. DOI: 10.1016/j.rcim.2011.12.001.

[12] Yanjiang Huang et al. “Robust multi-robot coordination in pick-and-place tasks based on part-dispatching rules”. en. In: *Robotics and Autonomous Systems* 64 (Feb. 2015), pp. 70–83. ISSN: 09218890. DOI: 10.1016/j.robot.2014.10.018.

[13] Lars Berscheid and Torsten Kroeger. “Jerk-limited Real-time Trajectory Generation with Arbitrary Target States”. In: *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, July 2021. ISBN: 978-0-9923747-7-8. DOI: 10.15607/RSS.2021.XVII.015.