technische universität dortmund

Process Dynamics and Operations

TU Dortmund University
Department of Biochemical and Chemical Engineering
Process Dynamics and Operations Group
Prof. Dr.-Ing. Sebastian Engell

Group Project

Optimization and Scheduling of a Multi-robot Production Plant

by

Anay Ghatpande (214787)
Khaled Elewa (215176)
Kiran Nivrutti Borse (214816)
Monika Rajagopal Balamurugan (215172)
Sowmiya Angamuthu(214799)

Supervision: Marina Rantanen Modéer, Egidio Leo, Taher Ebrahim
Examiner: Prof. Dr.-Ing. Sebastian Engell
Place, Date
Dortmund, 14/04/2020

ABSTRACT

Pipeless batch production plants are used as an alternative to traditional batch production plants. In contrast to the traditional batch production plants with fixed piping, pipeless plants perform processing steps at fixed stations and materials are moved around in mobile vessels. The key benefits of pipeless plants are having specialized cleaning stations for localized cleaning, which help in minimizing the downtime at all other stations and having flexibility with material transfers. The DYN experimental pipeless batch plant setup consists of 3 processing stations, 1 storage unit, control and positioning elements and 4 AGVs, which aid in transporting vessels between the stations according to the recipes.

The objective of this project is to plan efficient use of the resources, i.e., the AGVs and the processing stations, by using a scheduler to obtain an optimal sequence of actions to be executed in order to complete the requested recipes. While following the schedule, an optimal trajectory should be planned for the AGVs using dynamic optimization such that they avoid collisions with the static and dynamic obstacles. To verify and visualize the obtained schedule and the planned trajectories, a simulation is performed.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Abbreviation	**Meaning**
AGV	Automated Guided Vehicle
DFO	Derivative - Free Optimization
DYN	Process Dynamics and Operations
MILP	Mixed - Integer Linear Programming
PC	Personal Computer
PLC	Programmable Logic Controller
ROS	Robot Operating System

CHAPTER ONE

INTRODUCTION

A batch process is one in which a series of operations are carried out over a period of time on a separate, identifiable item or parcel of material. It is different from a continuous process, during which all operations occur at the same time and the material being processed is not divided into identifiable portions [20]. In traditional batch processing plants, the materials are dispensed through fixed-piping which leads to a considerable amount of time spent in cleaning and sterilizing the containers and piping when a recipe change is needed. Pipeless plants, on the other side, offer a high degree of flexibility. They allow change of priorities of the orders of the recipes and bypassing of a blocked station.

Figure 1.1 illustrates the experimental pipeless production plant at the DYN group. The plant consists of 3 processing stations, namely - 2 filling stations (A and B) and 1 mixing station, 1 storage unit, 4 Automated Guided Vehicles (AGVs) and, control and processing elements. The plant units are controlled by a PLC, and the AGVs are controlled by an on-board Raspberry Pi and a central PC. The software framework is based on ROS, programmed in python. The purpose of the plant is to demonstrate a versatile production of a wide range of products using mobile robots moving vessels, rather than having fixed pipes dispensing materials. The role of an AGV is to download a recipe, pick up the required vessel from the storage unit and then fulfil the recipe by travelling between the required stations. To plan the efficient use of resources, i.e., the AGVs and the processing stations, an effective scheduler is needed. The scheduler's task is to find the optimal order of actions to be executed in order to complete the requested recipes. The trajectories of the AGVs are planned in such a way that they follow the schedule provided by the scheduler thereby avoiding collisions with the static and dynamic obstacles. The correctness of the schedule and trajectories are then verified by simulation.

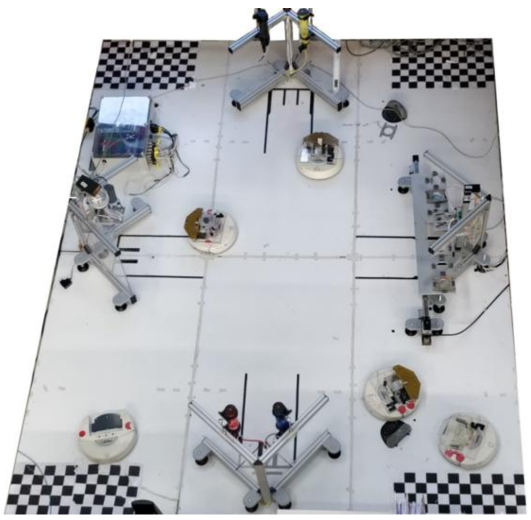
Figure 1.1: DYN Pipeless Plant Setup

1.1 Interaction between modules

In order to fulfil the objectives mentioned above, the project is divided into 3 modules, namely - Scheduling, Trajectory Planning and Simulation as shown in Figure 1.2. The scheduling module takes the sequence of the tasks to be performed (recipes) and the available resources as inputs. The AGVs and the processing stations are considered as resources. The schedule obtained from the scheduler provides the information about the starting time and the completion time of each task. When an AGV is travelling, the time at which the AGV leaves from one station and the time at which it reaches another station is fixed by the scheduler. This time and position information is then fetched for trajectory planning. The trajectory planning module utilizes this information from the scheduling module along with the kinematic model of the AGVs to obtain a feasible trajectory for each AGV. The trajectories are obtained in such a way that the AGVs avoid collisions with the static (processing stations) and dynamic obstacles (other AGVs). The control inputs of the obtained trajectories, i.e., angular velocity of the left and right wheel of the AGVs and the schedule are given to the simulation module which then verifies the correctness of the schedule and the trajectories.

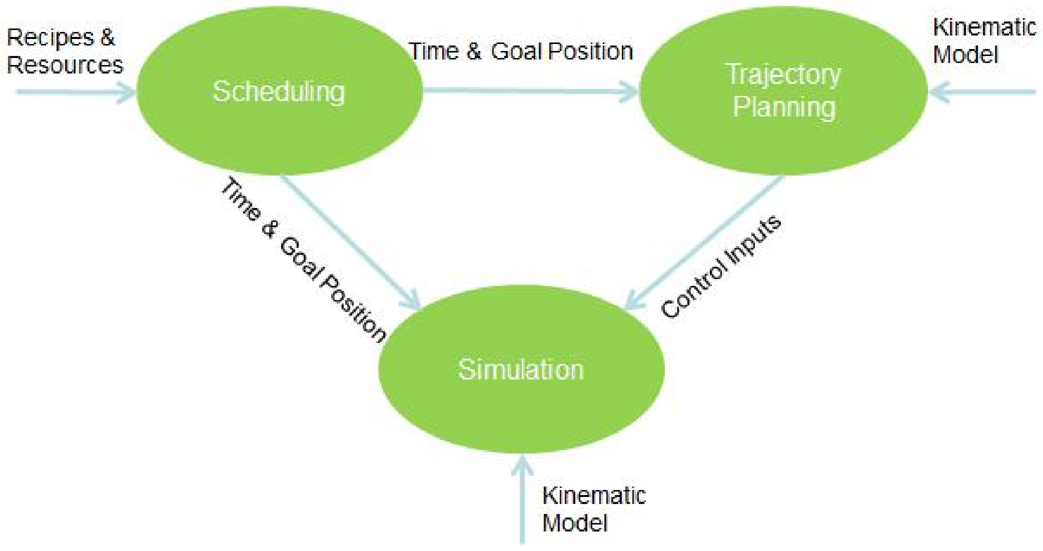

Figure 1.2: Interaction between Modules

1.2 Literature Review

Research done in the area of batch plant scheduling in the past 25 years have led to general modelling frameworks such as STN (State - Task Network) and RTN (Resource - Task Network). Both STN and RTN models represent the plant model with high level of detail such as capacity constraints on units and products, sequence - dependent changeovers, timing constraints between tasks, complex recipes, etc. In both the frameworks, representation of time highly influence the performance of the solvers. An approach in the form of a different MILP formulation describing the plant at a lower level of detail, reducing the model complexity was proposed [19]. It helps in improving the performance of the solver, but the accuracy of the model is compromised. Though STN and RTN models affect the performance of the solver, they are effective in terms of plant model accuracy and provide more generalised model. In the short-term scheduling of pipeless batch plants [9], STN approach was used for the mathematical formulation of the scheduling problem. This multipurpose pipeless plant operation is primarily driven by a number of orders for different products, each characterized by a nominal deadline, which is similar to our considered problem. Another scheduling problem of the melt shop of a steel plant with energy constraints was modelled using the RTN approach [18]. Three alternative models with an increasing level of detail were proposed, in order to find the most computationally efficient but still representative process model. The RTN approach was preferred over STN due to the unified treatment of states, units, and utilities as resources, which leads to a more elegant model with a single rather than multiple sets of resource balance constraints [18]. This work was used as a reference to formulate the mathematical scheduling model for our project.

Dynamic programming is a powerful method to solve an optimisation problem which calls for a recursive solution where both mathematical modelling of the numerical optimisation problem as well as computer programming is employed. A number of direct and indirect methods are available to solve the dynamic optimisation problem in a batch or a semi-batch process, like the Pontryagin's minimum principle (PMP) [21], but the difficulty to solve inequality path constraints is inherent in these methods. "Dynamic optimisation of batch processes I. Characterisation of the nominal solution" [21] provides a unified view of the methods available to solve dynamic optimisation. A simultaneous approach of the numerical solution methods was discussed, where both the inputs and the system equations were parameterized using a finite number of decision variables which in turn leads to a large NLP(NonLinear Program). This approach calls for a trade off between the approximation and optimisation. The improvement in Successive Quadratic programming allows the improvement in integration accuracy, where the accuracy can be increased by simply increasing the number of collocation points. "Dynamic free range routing for automated guided vehicles" [11] by M. B. Duinkerken presents an algorithm for the free range route planning for AGVs that are optimised regarding the arrival time, however interaction with machines was not considered.

The operation of AGVs in a plant needs safe automation from the initial to the final point making sure that the interaction with machines is considered. "Trajectory planning for AGVs in automated container terminals using avoidance constraints: a case study" [14] presents a similar case where automated guided vehicles (AGVs) are scheduled to interact with machines making sure that the collision constraints are obeyed. Both dynamic and static collision avoidance constraints were taken into account, where the trajectory planning problem is formulated as an MILP (using binary decision variables) and was solved using a SCIP(Solving Constraint Integer Programs) solver. Another similar problem "Path Planning of a Simple Car like robot" [4] gives the realisation of path planning of a robot where Pyomo.DAE [6] package was used to formulate the problem and the problem was solved with the help of the IPOPT solver [2]. The above references served as a base for setting up the mathematical model and to formulate the optimisation problem.

Since AGV is playing an important role in current manufacturing industries, many manufacturing process companies introduced the use of AGVs such as Amazon, Volkswagen, BMW, Deutz and Denso. Simulation is a very important step before implementation to the real plant. Most of the simulations are computer-based such as the simulation software Promodel which is used to plan and design manufacturing processes. It is a software that allows building complex logic in a simple and visual way using animations [15]. Another software is eM-Plant which is used for simulating and optimizing production lines. EM-Plant consists of modeling, simulation, and optimization statistics [24]. The software Uppaal is a tool used for modeling and verification of real-time systems modeled as timed automata [10] [13]. Another software Gazebo is used for modelling and simulation of the AGV movement [23].

CHAPTER TWO

SCHEDULING

2.1 Resource - Task Network (RTN)

A scheduling process is required to decide when and how a certain set of tasks should be performed, constrained by some rules to improve some performance criteria. In this work, we obtain the process schedule by solving an optimization problem. The optimality of the schedule depends on the objective function of the optimization. The objective function varies with the plant needs such as minimum energy consumption, minimum idle time of resources, maximum productivity, maximum utilization of resources, minimum tardiness, minimum cost, etc. To integrate the process plant into an optimization problem, we have used Resource - Task Network representation.

The RTN diagram regards all processes as bipartite graphs comprising of two entities: Resources and Tasks. The concept of resource is entirely general and includes all entities that are involved in the process steps, such as materials (raw-materials, intermediates and products), processing and storage equipment (vessels, tanks, reactors, etc.), utilities, cleaning states, material location. A task transforms a certain set of resources into another set. Fill, Transfer, Heat, Mix are some examples of task which consume a set of resources and produce another set.

For the pipeless plant described in chapter 1, given the processing time of the tasks and initial availability of the resources, the goal is to obtain an optimal schedule to process all the recipes with minimum total completion time by satisfying all the process constraints.

For the proposed plant, we have considered the following resources and tasks.
Resources : A (AGV), S (Storage station), FA (Filling station A), FB (Filling station B), M (Mixing station), VE (Empty Vessel), A1 - A10, B1 - B8, C1 - C8, D1 - D8, E1 – E9 (Intermediate AGV resources), VA0, VA1, VB0, VB1 (Intermediate vessel resources), VP1, VP2, VP3 (Final Products).
Tasks : Take (task of taking the required vessel from storage station), T_S_FA, T_S_FB, T_FA_M, T_FA_FB, T_FB_FA, T_FB_M, T_0_S, T_S_0, T_M_S, Fill_A1, Fill_B1, Fill_A2, Fill_B2, Mix (task of mixing the materials in the vessel), Store (task of storing the vessel back in the storage station), Wait (task of waiting for hardening of the intermediate product), where T_A_B represents transfer task from station A to station B and Fill_A1 represents filling task at station A with material 1.

2.2 RTN Model

A recipe is a sequence of tasks that should be followed to produce the final product. In this project, we have considered 3 Recipes. The recipes are,
Recipe 1: T_0_S – Take – T_S_FA – Fill_A1 – Fill_A2 – T_FA_M – Mix – T_M_S – Store – T_S_0.
Recipe 2: T_0_S – Take – T_S_FB – Fill_B2 – T_FB_M – Mix – T_M_S – Store Wait – T_S_FA – Fill_A2 – T_FA_M – Mix – T_M_S – Store – T_S_0.
Recipe 3: T_0_S – Take – T_S_FA – Fill_A1 – T_FA_FB – Fill_B2 – T_FB_M – Mix – T_M_S – Store & Wait – T_S_FB – Fill_B1 – T_FB_M – Mix – T_M_S – Store – T_S_0.

In the above recipes, "Store & Wait" represents the combination of tasks "Store" and "Wait", where the vessel with the intermediate product is stored back at the storage station and waits until the product is ready for the next task of the recipe. During this task, the AGV is released after the vessel is stored and is sent back to the initial position. This released AGV will now be available for the next recipe. Meanwhile, the vessel with the intermediate product hardens at the storage station. Once the intermediate product is ready, the recipe can resume with the available AGV. The selected AGV then moves to the storage station and takes the vessel with the intermediate product and proceeds with the remaining recipe.

2.2.1 RTN Diagram

The RTN diagram describes all processes as bipartite graphs comprising of two entities, i.e., resources and tasks. The RTN diagram is obtained in two steps:
1. Identify resources and tasks
2. Relate resources with tasks

We have already enlisted the tasks and resources for our plant. The RTN diagram for a recipe can be formulated by relating the required resources and tasks. Resources are represented by circles while tasks are represented by rectangles. Arrows define the direction of resource flow. In Figure 2.1, the task "T_0_S", represented with a rectangle, consumes resources "A", "S" and "VE", and produces the resource "E1" (represented by circles), once the task is finished. Figure 2.2 depicts the complete RTN diagram for the Recipe 1.

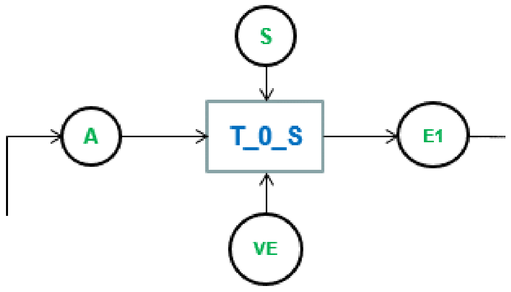

Figure 2.1: Resource-Task Network example

2.2.2 Mathematical Formulation

To convert an RTN diagram into an RTN model, the next step is to form a mathematical formulation featuring variables with resource r, task i, and time t, indices.

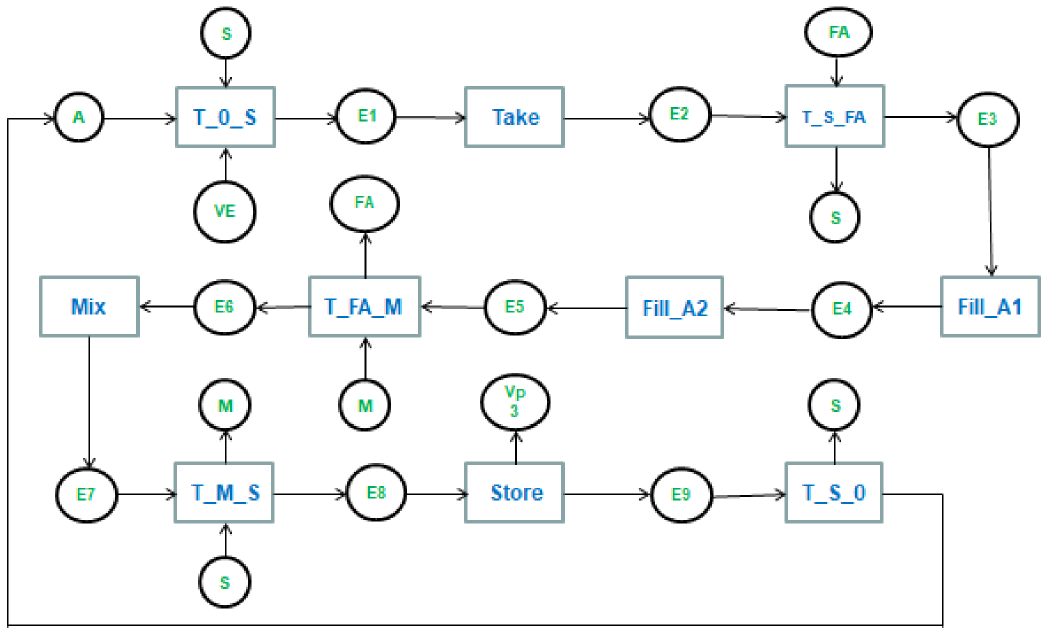

Figure 2.2: RTN Diagram For Recipe 1

2.2.2.1 Discrete-Time Representation

The RTN discrete time formulation proposed by Pantelides [17] formulates the simple, elegant, and very tight MILP models. This general approach can handle problems of industrial relevance, such as recipe optimization of a chemical process. The main advantage of a discrete-time representation is that the exact location of every time point t in the grid is known, leading to a straightforward modeling of intermediate events [18]. The discrete-time formulation is linked to a uniform time grid having $t \in \mathrm{T}$ slots of width δ that span 180 [s] (see Figure 2.3). The parameter δ is chosen by the modeler to set the approximation level of the problem data effectively. Though an exact model can be obtained with δ equal to the greatest common factor of the duration of all tasks (5 sec for our problem), this option is rarely used in practice. Selecting smaller δ value will result into bigger optimization problem and hence increasing overall computational time taken by the solver. A drawback of the discrete time representation is that it cannot accurately model events occurring at time instants that are not multiples of δ.

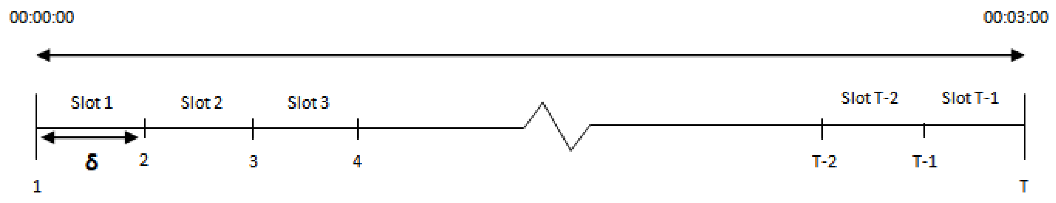

Figure 2.3: Uniform time grid used by discrete-time formulation

2.2.2.2 Model Variables

To describe our plant with an RTN formulation, we consider as decision variables the binary variables $N_{i,t} \in \{0,1\}$, which assigns the start of task i to time point t and the non-negative continuous variable $R_{r,t}$, which gives the amount of resource r available at time t.

2.2.2.3 Structural Parameters

The RTN graph is converted into a mathematical formulation by the structural parameters. The structural parameter $\mu_{\mathrm{r,i},\theta}$ gives the discrete interaction of resource r with task i at a time θ relative to the start of the task. The relative time index $\theta = \{0,\ 1,\ ...,\ \tau_{\mathrm{i}}\}$ is related to the start of task i, where τ_{i} is the processing time of the task i in terms of δ. In the RTN-based mathematical model, structural parameters are used in the resource balances, involving the $N_{i,t}$ and $R_{r,t}$ variables as mentioned earlier.

2.2.2.4 Designing of the structural parameters $\mu_{r,i,\theta}$

In order to implement the RTN model, the mi ($\mu_{\ r,i,\theta}$) matrix that gives the interaction of resource r with task i at time t has to be designed. The Table 2.1 represents $\mu_{r,i,\theta}$ for task "T_0_S". The processing time for task "T_0_S" is 10 seconds. The time discretization δ is considered to be 5 seconds as explained before in section 2.2.2.1. Therefore, the processing time within the discertized grid (τ) for task "T_0_S" is 2. The consumption and production of resources was explained earlier in 2.2.1. At $\theta = 0$, the consumed resources are marked as -1. Resource E1 is produced at $\theta = 2$ (marked as +1). Table 2.2 gives an example of $\mu_{r,i,\theta}$ formulation to avoid the assignment of stations when the

Table 2.1: $\mu_{r,\,i,\,\theta}$ for task "T__0__S"

r/θ	0	1	2	3	4
A	-1	0	0	0	0
S	-1	0	0	0	0
VE	-1	0	0	0	0
E1	0	0	+1	0	0

stations are not available. For example, even though the task at a particular station completes, the AGV might still be present at that station waiting for the next station to become available. In Table 2.2, the storage station (S) which was consumed before the starting of the task "T__0__S" is released only before the starting of the task "T__S__FA" to ensure that the storage station (S) is not released until the AGV moves from the station. Similarly, $\mu_{r,\,i,\,\theta}$ for

Table 2.2: $\mu_{r,\,i,\,\theta}$ for task "T__S__FA"

r/θ	0	1	2	3	4
E2	-1	0	0	0	0
S	+1	0	0	0	0
FA	-1	0	0	0	0
E3	0	0	+1	0	0

all the tasks are defined in CSV files according to their processing times.

2.2.2.5 Model Constraints

For our problem, three sets of constraints are considered. Resource availability over the time grid is managed by the resource balances, in which the amount of available resource r at event point t is equal to the amount of resource available at the previous event $(t-1)$ adjusted by the amounts produced/consumed by all tasks ending or starting at t and also by the external inputs/outputs. Note that the initial resource availability term R_r^0 - only appears for $t = 1$. Eq. (2.1) represents the resource availability constraint, where K is set of resources and I represents set of all the tasks.

$$R_{r,\,t} = \mathrm{R}^0_{r,t=1} + R_{\mathrm{r,t\text{-}1}} + \sum\nolimits_i^I \sum\nolimits_{\theta=0}^{\tau_\mathrm{i}} \mu_{r,\,i,\,\theta} N_{i,\,t\,-\,\theta} \qquad \forall t \in [1,T], r \in K \tag{2.1}$$

There might be more than one task that can be executed on a particular resource (unit). For example, the tasks "Take" and "Store" are executed on the resource "S" (storage station). The second constraint is unit allocation constraint which ensures that a unit can be used only by one task at a time. Eq. (2.2) gives the mathematical formulation of unit allocation constraint. J is the sets of tasks performed by each resource.

$$\textstyle\sum_{i \in J} N_{i,t} \leq 1 \qquad \forall t \in [1, T] \tag{2.2}$$

The third constraint ensures that all tasks are completed exactly once and it is given by Eq. (2.3). I represents the set of all tasks.

$$\textstyle\sum_{t \in T} N_{i,t} = 1 \qquad \forall i \in I \tag{2.3}$$

2.2.3 Objective

To solve this scheduling problem, different objective functions can be considered like makespan minimization, total cost minimization, energy cost minimization, deadline related objectives, etc. In this project, minimization of total completion time is considered as an objective. Total completion time is obtained by the summation of the completion times of all the tasks. The completion time of each task is given by adding the starting time of the task, i.e., $N_{\mathrm{i,t}}(t-1)$, with the processing time of the task $p(i)$. The objective function is represented mathematically as in Eq. (2.4).

$$TCT \geq \textstyle\sum_{i \in I} \sum_{t}^{T} N_{i,t}(t - 1 + p(i)) \tag{2.4}$$

2.2.4 Implementation

The RTN formulation is implemented using the modeling environment Pyomo. The MILP problem is solved adopting the solver Gurobi. Pyomo is a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating, solving, and analyzing optimization models. Pyomo's modeling objects are embedded within Python, a full-featured, high-level programming language that contains a rich set of supporting libraries

[6]. The Gurobi Optimizer is a commercial optimization solver for multiple programming, such as linear, quadratic, quadratically constrained, mixed integer linear programming, mixed-integer quadratic programming, and mixed-integer quadratically constrained programming [1].

2.2.4.1 Implementation in Pyomo

The RTN model is implemented in pyomo with the help of data files which gives the relation between tasks and resources. The data set include files with information about resources, tasks, initial resource amount, resources consumed by each task, resources produced by each task, processing time of each task, tasks performed by each resource and $\mu_{r,i,\theta}$ for each task. The implemented plant model can be used for any different type of batch processes, by changing the information in the data files.

2.2.5 Results

The optimal schedules obtained with the MILP solver Gurobi for processing 3 recipes with different number of AGVs, namely - 4, 3 and 2 are shown in the Figure 2.4, Figure 2.5 and Figure 2.6 respectively. The Figure 2.4, Figure 2.5 and Figure 2.6 depict a Gantt chart representing the station utilization by the AGVs as they execute the tasks. The Recipes 2, 3 and 1 are represented by blue, red and green colored blocks respectively in the Gantt charts.

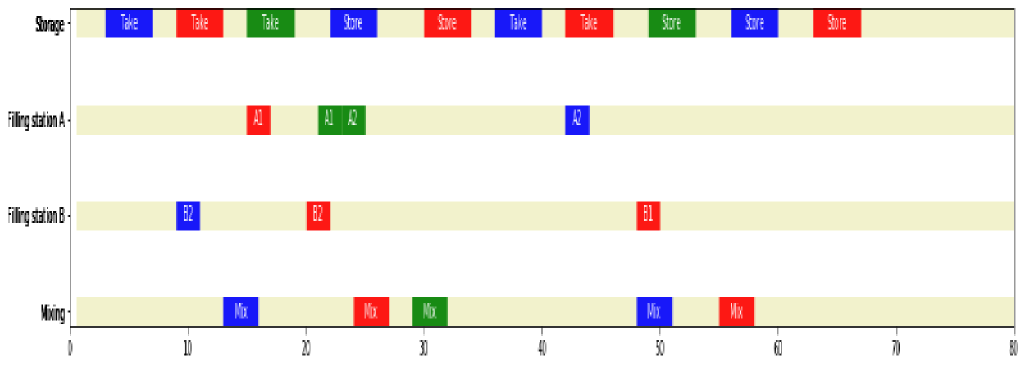

Figure 2.4: Schedule obtained for processing 3 recipes with 4 AGVs

In Figure 2.6, we can see that the Recipes 2 and 3 are picked up first by the 2 available AGVs. After storing the vessel with the intermediate product, the released AGV then picks up the next recipe to execute. In this way, the waiting time for the hardening of an intermediate product is utilized to perform other tasks. Thus, the idle time is avoided and the resources are used efficiently. In Figure 2.5

From the Figure 2.4 and Figure 2.6, though we can see that the total completion time is reduced when more AGVs are available, it might not be the same always. For example, the total completion times for processing 3 recipes with 3 AGVs (Figure 2.5) and 3 recipes with 4 AGVs (Figure 2.4) are same, as there were not enough tasks for the extra AGV to perform.

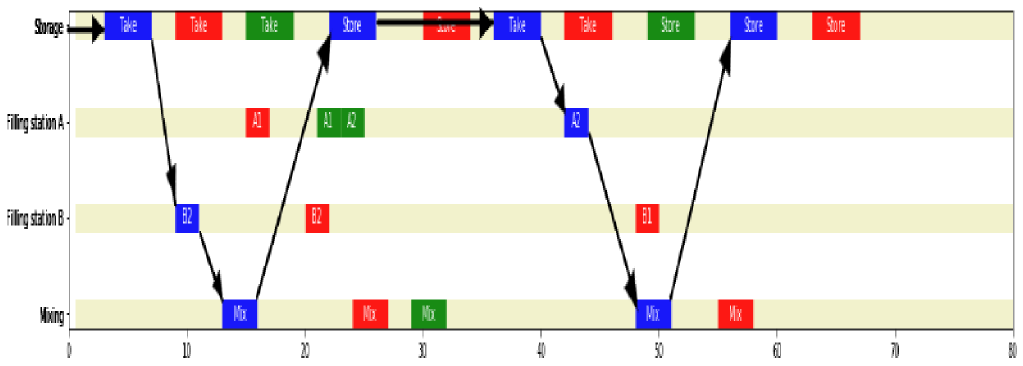

Figure 2.5: Schedule obtained for processing 3 recipes with 3 AGVs

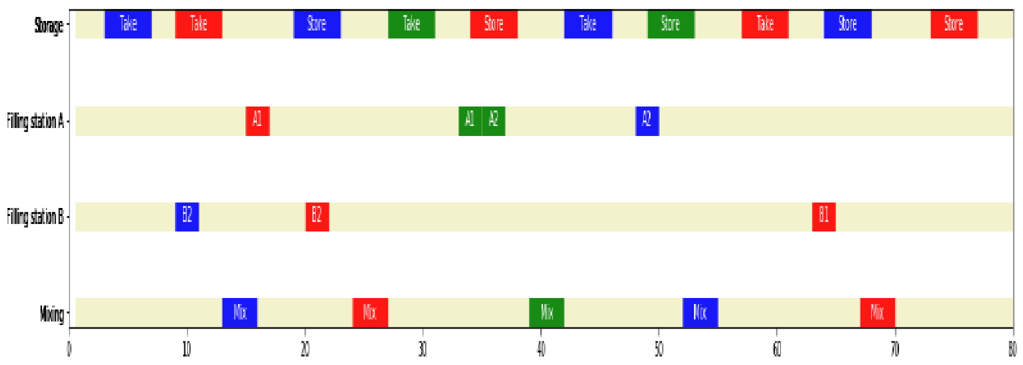

Figure 2.6: Schedule obtained for processing 3 recipes with 2 AGVs

Table 2.3 contains the computation time taken by the solver to obtain an optimal schedule for different number of AGVs and Recipes. It can be observed from the table that when the number of recipes, i.e., number of tasks increases, the computation time taken by the solver also increases. It is also evident from the Table 2.3 that for a given number of recipes, the computation time decreases when the number of resources (AGVs in this case) increases, though the number of constraints and binary variables remain constant.

Table 2.3: Computation time for different number of AGVs and Recipes

AGVs	Recipes	Constraints	Binary variables	Computation Time [s]
2	3	8771	4000	5.751034
2	6	19671	9000	1687.700731
3	3	8771	4000	3.213582
3	6	19671	9000	73.690957
4	3	8771	4000	1.434493
4	6	19671	9000	66.053579

The time and position information of the AGVs are fetched from the obtained schedule and written into a text file, which is then used for the trajectory planning of the AGVs.

CHAPTER

THREE

TRAJECTORY PLANNING

Trajectory planning involves the definition of a feasible path from the initial position to the final goal position. This can be done offline or online depending upon the requirements of the system. Online optimisation involves feedback control where the feedback allows us to correct the control inputs online and react to unknown disturbances on the state, thus aiding the performance of optimal control. The decision set is bounded and subject to all the necessary restrictions (constraints) and an online player iteratively makes decisions, which requires high performance solvers. The pose of a robot at a given time can be described by its position, usually described with respect to the x and y coordinate axes and its orientation which can be described by the heading angle θ if the system is analysed in 2-D.

Obstacle avoidance is a basic requirement as it ensures a collision free trajectory and making sure that no damage is perceived by the robot. Static obstacles are the boundaries of the workspace and other fixed objects within the working area whose position is stationary and dynamic obstacles are those whose position varies with respect to time e.g., other AGVs navigating in the same workspace. In this project, the four stations serve as a static obstacles. For each AGV, the other AGVs that are also performing their tasks serve as the dynamic obstacles. So it is important that each of the AGV must not collide with the stations and as well they must not collide with one another.

3.1 Dynamic optimisation

Dynamic optimisation problems are problems where an integration of dynamic fitness function or the objective function is maximised or minimised over a time period and a solution is interpreted at each time interval within that given time period. They can therefore be defined as problems where a set of optimal decisions have to be made with respect to time in order to maximise a certain objective which is a function of all these decisions made w.r.t time.

Dynamic optimisation involves a dynamic system described by ordinary differential equations in continuous time on a time interval $[t_{init}, t_{fin}]$. The goal of dynamic optimisation is to perform optimal control, i.e., deciding on the optimal control with respect to a certain perfromance index while respecting the process model. With the knowledge of the initial state x_{init} and a control input trajectory u(t) for all t, thus allowing us to determine the whole state trajectory $x(t)$ for $t \in [t_{init}, t_{fin}]$.

The problem is to find the values of decision variables, i.e. control variables, that minimize the objective function as well as satisfy the constraints[14].

$$\dot{x} = f(x(t), u(t))$$

Where,
$x(t)$ - represents the state variables
$u(t)$ - represents the inputs It consists of three elements:

1. The objective function
2. Dynamic model
3. Constraints

To find the control inputs u that result in states x which optimizes the objective function while respecting the constraints.

1. Trajectories of AGVs need to be scheduled with interacting machines.
2. Constraints are added to avoid collisions.
3. Collision-free trajectories of individual AGVs must be determined with optimal control problem.

3.2 Modelling of a problem

The iRobot Roomba 500 [3] is the automated guided vehicle implemented in the plant. It is a two wheeled robot and the linear velocity v and the angular velocity ω are the control inputs given for mobilising the robot. The position of the AGV is represented by x and y , and θ represents the orientation. In order to have accurate mobilisation of the robot, the kinematic model has been interpreted with respect to the radius of the wheel 'R' and the length of the axel 'L'. Thus the same model can be applied for different dimensions of AGVs without much change in the kinematic equations.

3.2.1 Kinematic model of AGV

The generalized Kinematic model of the two wheeled robot is considered. As shown in the figure 3.1, localization is done at the center of the AGV. The mathematical model [4] of a two wheeled robot can be given by:

$$\dot{x} = v \cdot \cos\theta$$
$$\dot{y} = v \cdot \cos\theta$$
$$\dot{\theta} = \omega$$

Where,
x, y - represent the position of the AGV
θ - represents the orientation of the AGV
v - represents the linear velocity of the AGV
ω - represents the angular velocity of the AGV

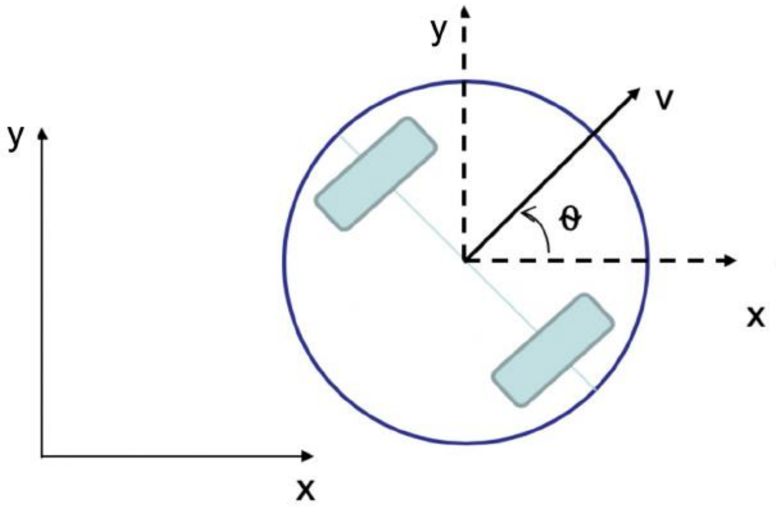

Figure 3.1: Model of an AGV

The model of the robot in terms of radius of the wheel R and the length of the axel L can be given by the relation:

$$\begin{aligned}
\omega &= \frac{v}{R} \\
v_r &= R \cdot \omega_r \\
v_l &= R \cdot \omega_l \\
\Rightarrow \omega &= \frac{R \cdot \omega_r - R \cdot \omega_l}{L} = \frac{v_r - v_l}{L}
\end{aligned}$$

Where,
ω_r, ω_l - angular Velocity of right wheel and left wheel
v_r, v_l - linear Velocity of right wheel and left wheel
R - radius of Wheel
L - length of the axle

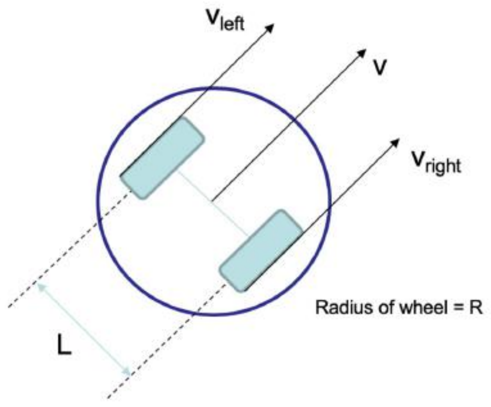

Figure 3.2: AGV Model representing linear velocity

Thus the state equations considered will be:

$$\dot{x} = (\frac{R \cdot \omega_r + R \cdot \omega_l}{2}) \cdot \cos\theta$$
$$\dot{y} = (\frac{R \cdot \omega_r + R \cdot \omega_l}{2}) \cdot \sin\theta$$
$$\dot{\theta} = (\frac{R \cdot \omega_r - R \cdot \omega_l}{L})$$

With respect to these relations, the control inputs in this problem will be ω_r and ω_l , the angular Velocity of right and left wheel.

3.2.2 Area Constraints

The actual area of the plant is 4000 x 3000 mm. For convenience the scaled value of the area is considered and the area is taken as a 2000 x 1000mm. The static obstacles greatly decide the maximum allowable area.

In order to avoid collision with the walls of the four stations, a minimum distance of d has to be maintained by each AGV from any of these stations. This minimum distance is measured from the center of the AGV as the localization of an AGV is done at its center.

Based on these conditions, the maximum allowable area was computed to be the largest allowable rectangle was. For the AGV to remain only in this workspace, the following constraints were implemented:

$$x_{low}(t) \leq x_i(t) \leq x_{high}(t)$$

$$y_{low}(t) \leq y_i(t) \leq y_{high}(t)$$

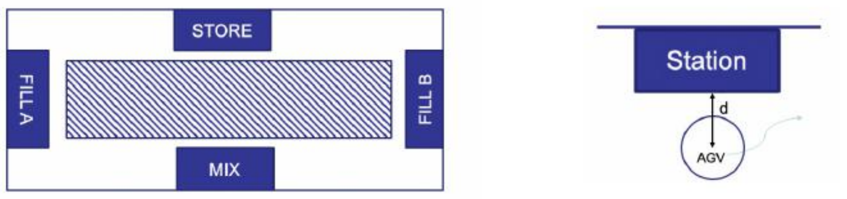

Figure 3.3: Constraints with Area bounds

Where,
x_{low}, y_{low} - represent the lower bound of the allowable area
x_{high}, y_{high} - represent the upper bound of the allowable area
x_i, y_i - represent the position coordinates of i^{th} AGV

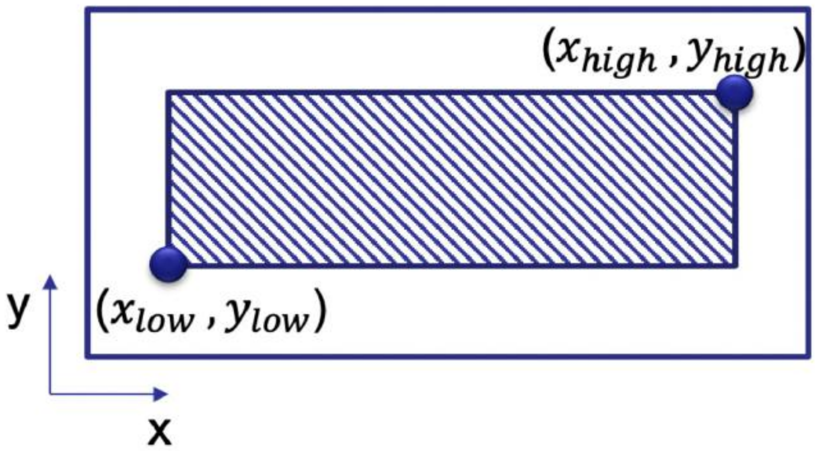

Figure 3.4: Constraints with Area bounds[14]

3.2.3 Dynamic Obstacle Constraints

For each AGV, all the other AGVs serve as dynamic obstacles. Therefore, it has to be made sure that during the run there is no collision between one another. This can be avoided if a minimum safe distance SD is maintained between each AGV at all times. This safe distance is measured from the center of one AGV to another and represented as:

$$SD = 2 \cdot Radius_{AGV} + D_{collision-free}$$

$$SD > Radius_{AGV}$$

Thus, the constraints can be defined as:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} > SD$$

Where,

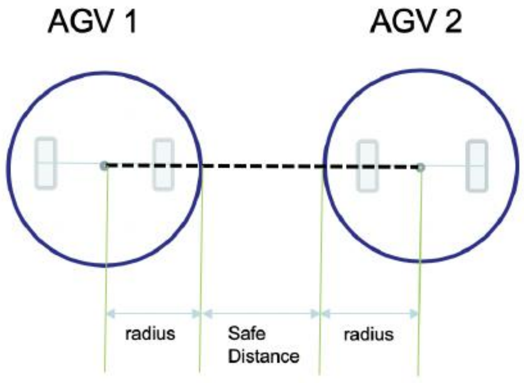

Figure 3.5: Distance between AGV and dynamic obstacle

x_1, y_1 - position coordinates of AGV1
x_2, y_2 - position coordinates of AGV2

3.2.4 Bounds on state variables

The bounds on the state variables $x(t)$ and $y(t)$ were already defined with the help of the area constraints. The bounds of the state variable θ can be defined by :

$$-\pi \leq \theta \leq \pi$$

3.2.5 Bounds on control variables

From the specification sheet the bounds on the linear velocity of the right wheel v_r and the linear velocity of the left wheel v_l were obtained. From this the bounds on the control inputs were deduced as follow:

$$v_{min} \leq v_r \leq v_{max,r}$$

$$v_{min} \leq v_l \leq v_{max,l}$$

$$\frac{v_{min,r}}{R} \leq \omega_r \leq \frac{v_{max,r}}{R}$$

$$\frac{v_{min,l}}{R} \leq \omega_l \leq \frac{v_{max,l}}{R}$$

3.3 Objective function

Minimizing the error between the current pose of an AGV and its goal pose is considered as the objective. Based on this the cost function was modelled.

$$J = min \int_{t=0}^{t_f} \sum_{i=1}^{n} (x_{i,goal}(t) - x_i(t))^2 + (y_{i,goal}(t) - y_i(t))^2$$

$$t \in \Re^+, \quad i = 1, 2, ..., n$$

Where,
t_f - Total time of execution of the schedule
$x_{i,goal}(t), y_{i,goal}(t)$ - Goal pose of 'i'- th AGV
$x_i(t), y_i(t)$ - Current pose of 'i'-th AGV

This $x_{i,goal}(t), y_{i,goal}(t)$ is a user defined continuous function that contains the

set of goal and intermediate positions for an AGV defined in the program as a constraint.
Consider the following schedule:

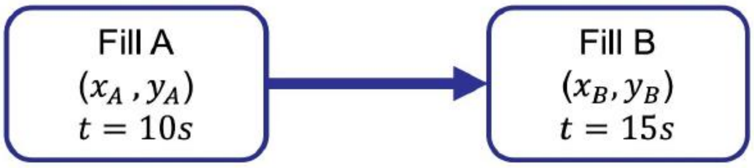

Figure 3.6: Example of schedule

The schedule is to reach Filling station A at $t = 10s$ and then Filling station B at $t = 15s$ then the goal function will be defined as:

$$x_{goal,1} = x_A \; and \; y_{goal,1} = y_A, 0 \leq t \leq 10$$

$$x_{goal,1} = x_B \; and \; y_{goal,1} = y_B, 10 < t \leq 15$$

3.4 Softwares used for Programming

3.4.1 Pyomo.DAE [6]

The pyomo.DAE is a modeling extension that allows users to incorporate differential algebraic equations DAEs. The pyomo open source package for python is utilized in this project. For our non-linear optimization the IPOPT (Interior Point Optimizer) solver is used to solve the underlying nonlinear programs NLPs [2]. For solving a dynamic problem in Ipopt the following steps are followed :

• Declare the differential equations, here the state equations as the constraints.
• Formulate the optimization problem including the constraints and bounds.
• Declare the objective function.

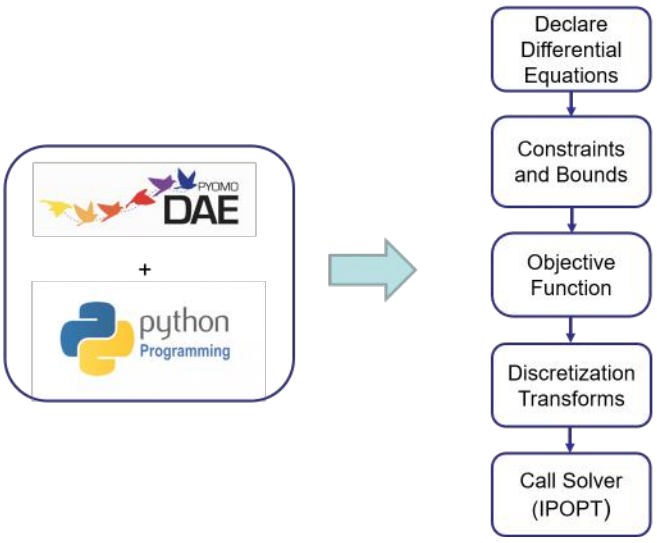

Figure 3.7: Steps of programming

- Discretize the differential equations.
- Call the solver to solve the resulting NLP optimization problem.

Discretization Transforms: The discretisation transforms are used to transform continuous functions into discrete functions, thus allowing us to solve both ordinary and partial differential equations. Some of the most widely used methods include finite difference transforms and collocation transforms. These methods ensure simplification of the complex problem and also higher order of accuracy.

Finite Difference Transformation: [5]
The ordinary differential equations are solved with the help of this method found by L. Euler. For any scheme of equations, the approximation is done by the following method. For an ODE,

$$\dot{y} = f(x,t), \quad x_0 = x(t_0)$$
$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

Where,

h - the step size between the discretization points.

Three different types of schemes are offered:
• Forward Finite Difference
• Backward Finite Difference
• Central Finite Difference

Augments for finite difference transformations:

'nfe' – Represents the number of finite elements in discretization.

'scheme'- Represents the scheme of Finite difference used.It takes up values as 'CENTRAL', 'FORWARD', 'BACKWARD' depending upon the scheme used.

Collocation Transformation:

The idea of collocation transform is to discretize the entire domain into a number of intermediate points called as collocation points and to choose parameters and variables such that the objective is satisfied at these collocation points. Dividing the domain into n equal segments results in n+1 points within each segment. Orthogonal Discretization is used to discretize the differential equations. The existing scheme in pyomo.dae employs Lagrange polynomials with either Gauss-Radau roots or Gauss-Legendre roots[5].

Augments for Collocation transformations:

• **'ncp'** - Represents the number of collocation points within each finite element.
• **'scheme'** - Takes up values as 'LAGRANGE - RADAU' or 'LAGRANGE - LEGENDRE'.

3.5 Results Obtained

The optimization problem was applied to the assumed plant with dimensions 2000mm x 1000mm and for two AGVs. The optimization problem can be formulated as follows:

$$J = min \int_{t=0}^{t_f=250} \sum_{i=1}^{2} (x_{i,goal}(t) - x_i(t))^2 + (y_{i,goal}(t) - y_i(t))^2$$

$$t \in \Re^+$$

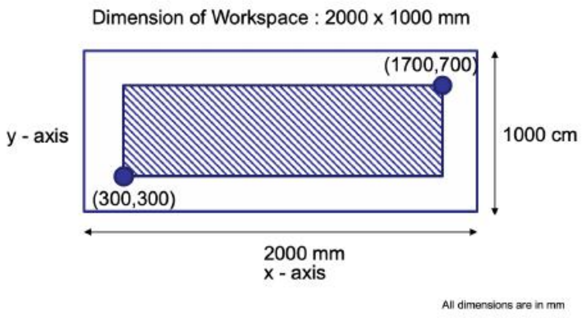

Figure 3.8: Area Bounds

subject to:

$$\dot{x} = (\frac{R \cdot \omega_r + R \cdot \omega_l}{2}) \cdot \cos\theta$$

$$\dot{y} = (\frac{R \cdot \omega_r + R \cdot \omega_l}{2}) \cdot \sin\theta$$

$$\dot{\theta} = (\frac{R \cdot \omega_r - R \cdot \omega_l}{L})$$

Dynamic Constraint :

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} > 250mm$$

Bounds on area :

$$300mm \leq x_1 \leq 1700mm$$

$$300mm \leq y_1 \leq 700mm$$

$$300mm \leq x_2 \leq 1700mm$$

$$300mm \leq y_2 \leq 700mm$$

Bounds on state variable :

$$-\pi \leq \theta \leq \pi$$

Initial values of the states:

$$x_0 = x(t_0)$$

$$y_0 = y(t_0)$$

Bounds on control input :

$$-0.5m/s \leq v_r \leq 0.5m/s$$

$$-0.5m/s \leq v_l \leq 0.5m/s$$

Therefore,

$$-50s^{-1} \leq \omega_r \leq 5s^{-1}$$

$$-50s^{-1} \leq \omega_l \leq 5s^{-1}$$

The results of the optimization will be the optimal control inputs ω_r and ω_l, along with the values of the optimal state variables x, y and θ.

Figure 3.9 and 3.10 shows the individual trajectory traced by AGV 1 and AGV 2 and figure 3.11 shows the the resultant trajectory traced by both the AGVs. The AGVs are made to follow the schedule obtained from the result of scheduling. Both the AGVs start from their initial position (fixed here) shown by the red marker and their final pose will also be their initial rest pose. Thus, the stations that each of these AGVs have to visit will become their intermediate goal poses. Collision between the AGVs at any point of time is determined by computing the distance between them as shown in figure

3.12 where it is observed that there is no collision between these AGVs as the distance between them is always greater than or equal to safe distance.

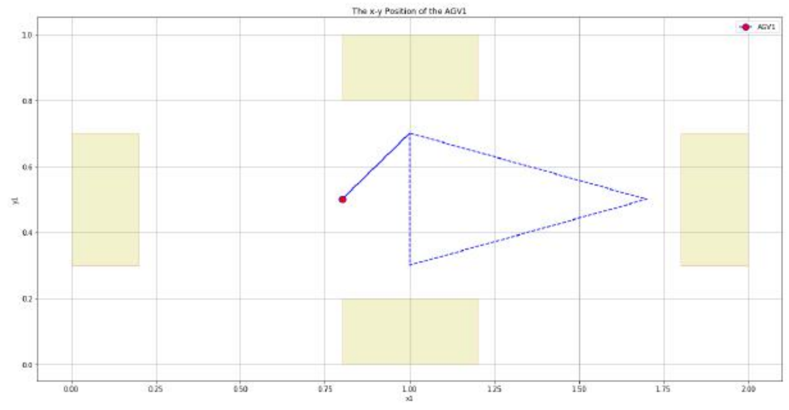

Figure 3.9: Result of AGV 1

The Figure 3.9 shows the trajectory traced by AGV 1 with respect to the x and y axis. The red marker depicts the initial and the final position of the AGV.

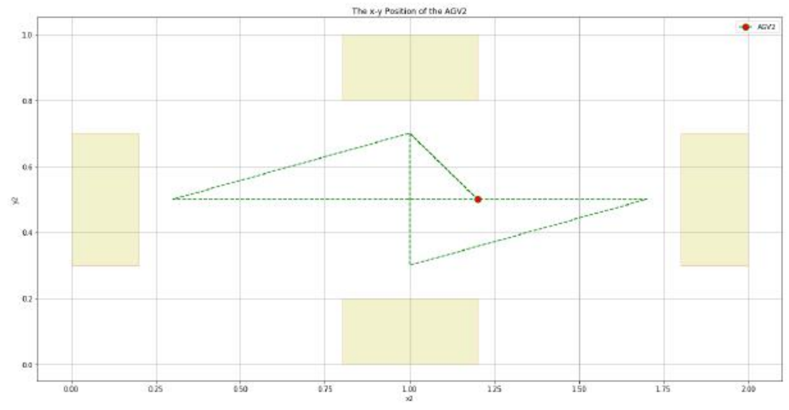

Figure 3.10: Result of AGV 2

The Figure 3.10 shows the trajectory traced by AGV 2 with respect to the x and y axis. The red marker depicts the initial and the final position of the AGV.

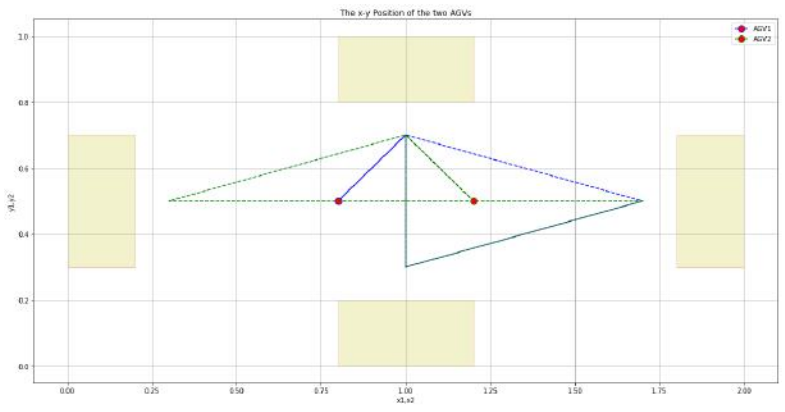

Figure 3.11: Result of AGV 1 and AGV 2

The Figure 3.11 shows the entire trajectory traced by the two AGVs (AGV 1 in blue and AGV 2 in green) with respect to the$x$ and y axis.

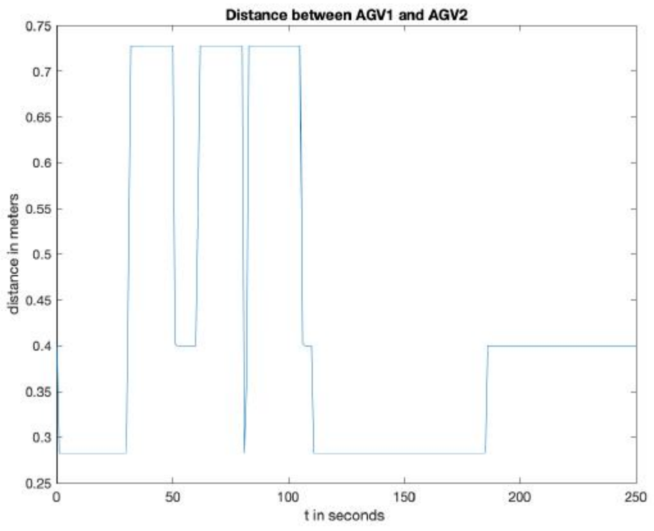

Figure 3.12: Distance between AGV 1 and AGV 2

The Figure 3.12 shows the distance between AGV 1 and AGV 2 with respect to time. Here the minimum distance between them is 0.25 meters, showing that there are no collisions between the two AGVs.

Test Case 1:

Collision avoidance was tested by making the AGVs move opposite to one another at the same time, such that the initial point of one AGV will be the final destination of the other and vice versa.The following trajectory was observed Which shows that there were no collisions.

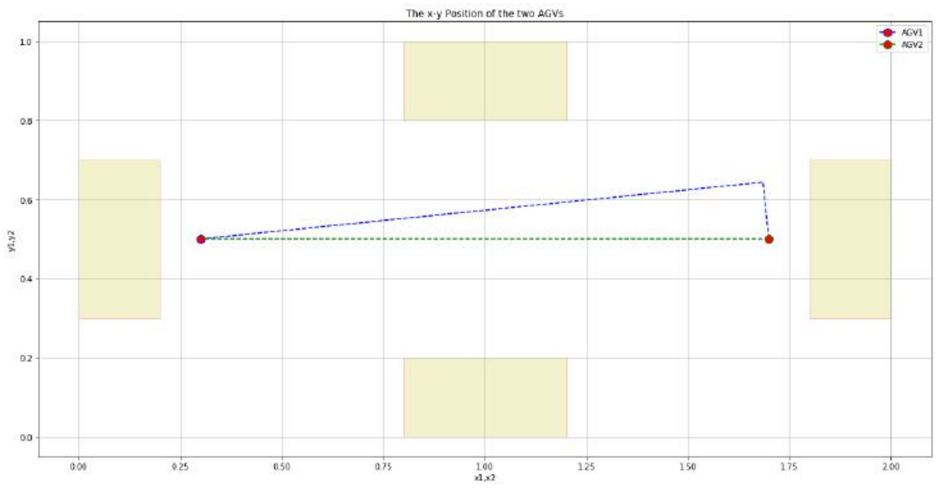

Figure 3.13: Test case with 2 AGV

Test Case 2:

Collision avoidance was further tested this time with the same conditions, with an addition of a third AGV that was static (still) as shown in Figure 3.14. The initial position of AGV 1 and AGV 2 is represented by the red marker and AGV 3 is at the centre.

It can be seen that here the AGV 3 acts more like a static obstacle and the other two AGVs have successfully passed around AGV 3 and the collision constrains can be further verified by determining the distance between any two AGVs at a given time as shown in Figure 3.15, Figure 3.16 and Figure 3.17. A much smoother trajectory can be obtained in any of the above cases by increasing the number of the discretization points.

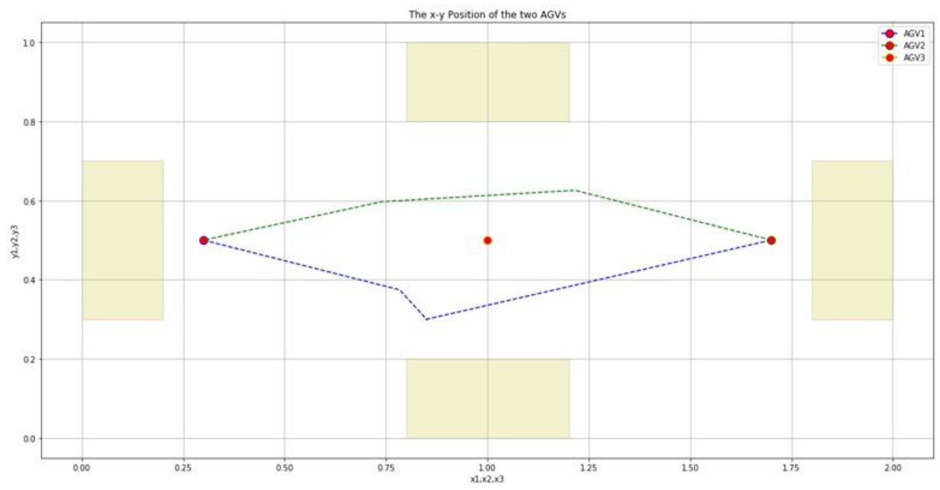

Figure 3.14: Trajectory of 3 AGVs

Comparison of Distances among all three AGVs.

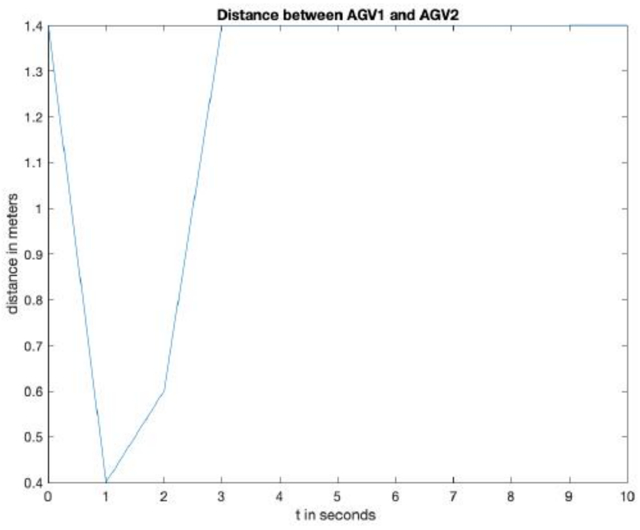

Figure 3.15: Distance between AGV 1 and AGV 2

The Figure 3.15 shows the distance between AGV1 and AGV2 with respect to time and the minimum distance between them is is greater than or equal to 0.25 meters, showing that there are no collisions between the two AGVs.

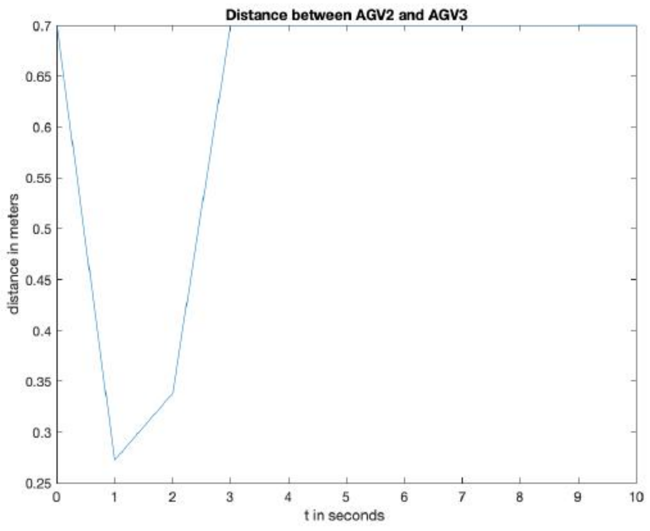

Figure 3.16: Distance between AGV 2 and AGV 3

The Figure 3.16 shows the distance between AGV2 and AGV3 with respect to time and the minimum distance between them is is greater than or equal to 0.25 meters, showing that there are no collisions between the two AGVs.

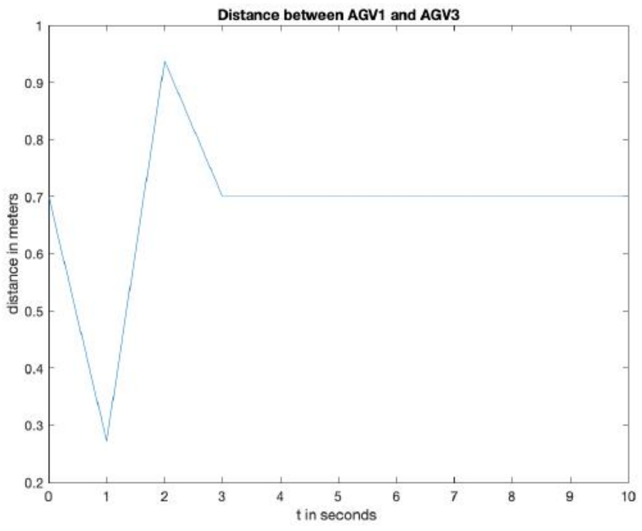

Figure 3.17: Distance between AGV 3 and AGV 1

The Figure 3.17 shows the distance between AGV3 and AGV1 with respect to time and the minimum distance between them is greater than or equal to 0.25 meters, showing that there are no collisions between the two AGVs.

3.6 Issues faced while obtaining results

Local Minima Problem:
One of the major issues faced while designing the static obstacle was the attainment of a local minima. Initially area constraints were set up by defining points on the boundary of the static obstacle, that would not allow the AGV to pass through.
For the highlighted point (red) shown in the Figure 3.18, the constraints for an AGV to avoid this point can be given as follows:

$$\sqrt{(x_i - x_{obstacle})^2 + (y_i - y_{obstacle})^2} > d$$

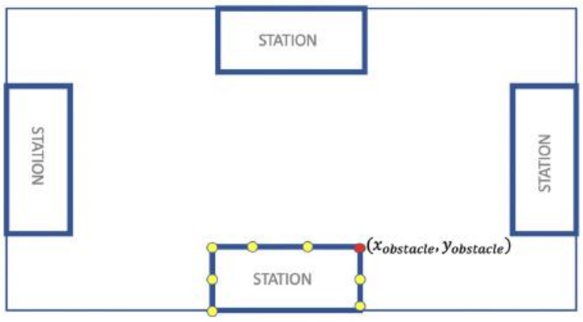

Figure 3.18: Defining points on the station boundaries

Where,
x_i, y_i – represents the position of the 'i'-th AGV with respect to the x and y axes
$x_{obstacle}, y_{obstacle}$ – represents the Position of the obstacle with respect to the x and y axes
d - represents the minimum distance from the center of the AGV to the Static obstacle.
A series of points were set up for all the four plants and the problem was solved

by using only one AGV. But due to the problem of attainment of local minima, the performance of the solver was greatly affected and also the AGV failed to reach the goal pose as shown in Figure 3.19. Considering these difficulties, the area was scaled and finally the maximum allowable area was decided on this criterion.

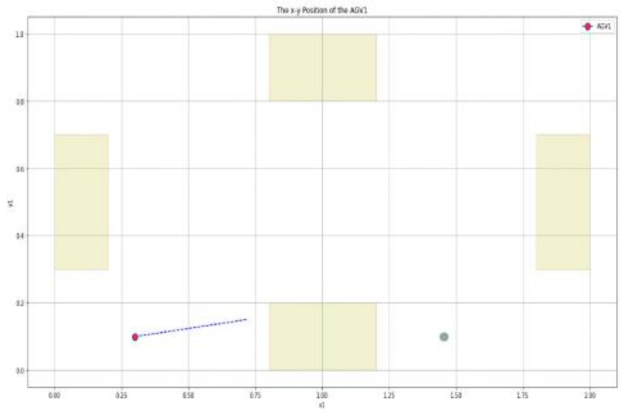

Figure 3.19: Attainment of Local Minima

3.7 Advantages

• The results obtained are giving optimal trajectory with respect to the performance index used.
• Depending upon the requirements of the plant model, the Objective and constraints can be remodelled (e.g. to minimize the energy consumption, maximize the velocity, minimize the time).
• There is a scope of online optimization which can be done with the help of the nonlinear model predictive controller (NMPC).

CHAPTER

FOUR

TESTING AND SIMULATION

4.1 Introduction to simulation

The purpose behind the simulation is to give a visualization of the schedule sequence and the position of each AGV during operation. The simulation model is also used as a test for a correct sequence of the schedule against some common mistakes such as visiting one station by more than one AGV at the same time or having an incorrect sequence of operations like storing a vessel before mixing the recipe in it. Moreover, the model is used to detect any collision between the AGVs while moving between stations. Note that the simulation model is not made for getting a correct sequence for the schedule or a collision-free path for the AGVs, it is just for visualization and schedule sequence testing. The Simulation does not contain a dynamic model of the plant but it shows the sequence of operations done on the plant using timed Automota.

The simulation consists of two parts: schedule sequence simulation and AGVs trajectory visualization. The software used for the both parts is MATLAB/Simulink. MATLAB was preferred over UPPAAL as it has the toolboxes for both timed automota and AGV kinematic model in one software. UPPAAL has extra checks for the timed automota that are missing in MATLAB such as a check for reachability and possible deadlocks. Both checks were solved in this MATLAB model by adding states and conditions to do them manually.

Gazebo also is better in AGV simulation and visualization but it lacks the plant simulation part. Further improvement can be done to this model by adding the real plant dynamic model along with the AGV existing model for a full real time simulation.

For the schedule part, the simulation is done based on timed Automota where there are states for the system for every station and the simulation moves from one state to another according to a timed schedule. The Simulink toolbox Stateflow was used to implement this timed Automota and that will be stated in section 4.2.1. The AGVs trajectory simulation is done using Simulink to create the AGV kinematic model from the Simulink library Robotics System Toolbox, and MATLAB to plot the AGV trajectory.

4.2 Schedule simulation

The first part of the simulation is the schedule simulation. Since the schedule is a sequence of events happening sequentially with a time stamp, timed-Automota will be one way to represent it. Construction of the timed-Automota is done by considering each operation (e.g.filling, mixing, storing) as a state and the transitions between the states are timed according to the timing sequence from the schedule. Timed Automota is here executed in Simulink using Stateflow toolbox.

4.2.1 Introduction to Stateflow[7]

Stateflow toolbox in Simulink is a logic control tool used to model systems using state machines. The two blocks in the toolbox used here are Chart block and Sequence Viewer block.

Chart block is used to create states and transitions between them. The transition condition is written on the transition arrow and it can be a condition (e.g. $Input == 1$) or time (e.g. $after(10, sec)$) or both. The data types inside the chart block are input, output, constant and local parameter. The inputs and constants can not be changed inside the chart itself while the outputs and local parameters can be changed. These data types can be changed on a transition arrow, on entry of a state, during a state or on exit of a state.

Figure 4.1 shows an example of a state machine inside a chart block. As an example, the state machine has three states. In $State1$, the parameter value can be changed on entry of the state, in $State2$ it changed during the state and in $State3$ it changed on exit of the state. Transition condition from $State1$ to $State2$ is on the transition arrow and that is $Input == 1$. Also on that transition arrow, the output value is set to one before $State2$ is entered. The output value can be also changed during the state itself as show in $State3$. The transition from $State2$ to $State3$ is done after 10 seconds.

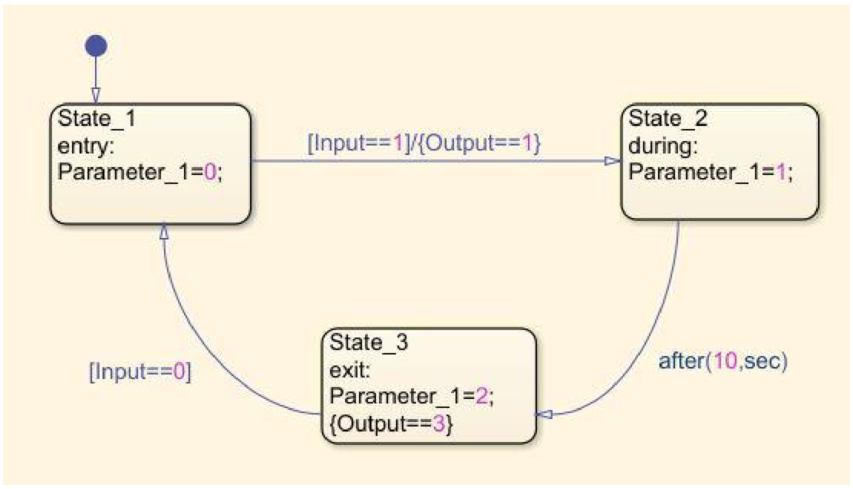

Figure 4.1: Example of chart block

Sequence View toolbox is used to visualize the sequence of transitions between the states and the conditions that caused them. For example in Figure 4.1, for an $Input = 1$, the Sequence View is shown in Figure 4.2. The system is initially in $state1$, then the condition $Input = 1$ is true and it goes to $State2$ then after 10 seconds it goes to $State3$. The Sequence View block makes it easier to visualize the sequence instead of slowing down the simulation and tracking it manually.

4.2.2 Simulation of AGV schedule

The first part of the simulation is to simulate a schedule sequence only for one AGV and then to generalize the model to include multiple AGVs. As illustrated in Figure 4.3, the model consists of a chart for the states of the

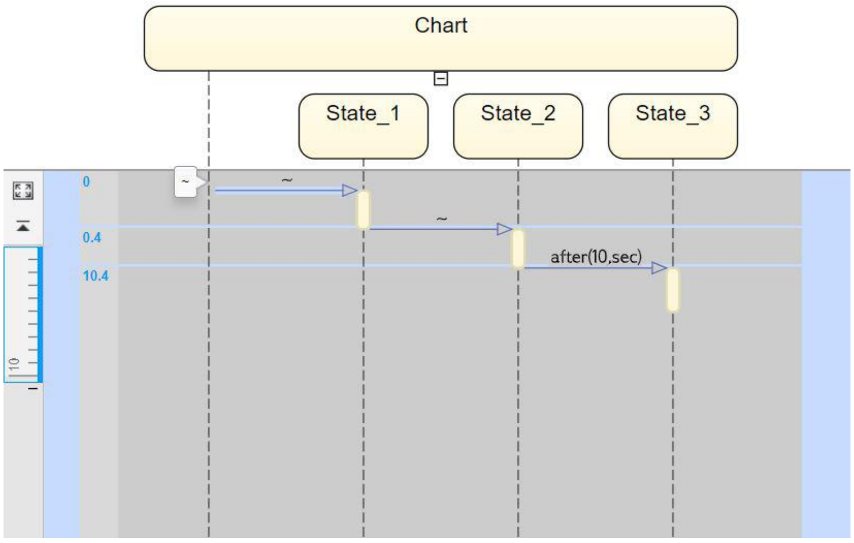

Figure 4.2: Example of Sequence View

AGV, a chart for the states of yellow and black filling station, a chart for the state of red and blue filling station and a chart for the mixing station. Each chart has blocks that represent the states of the subsystem that will be stated later in details. The AGV chart sends output messages to control the start or the stop of each station. Each station has a chart that is waiting for a message from the AGV chart to start its operation (e.g. filling or mixing).

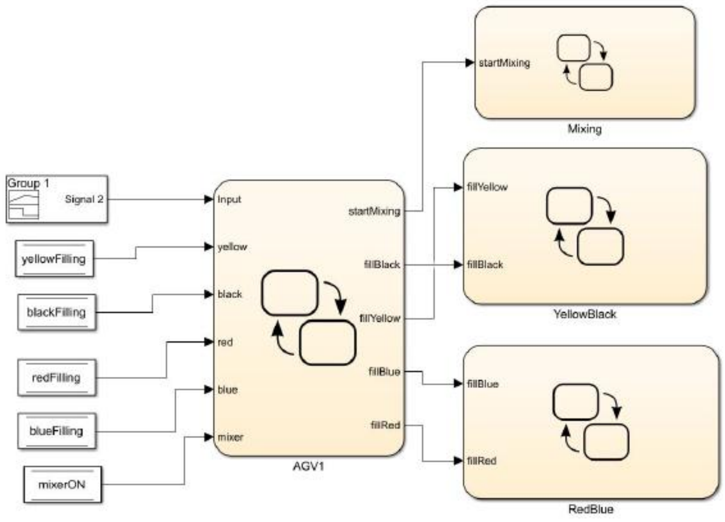

Figure 4.3: The simulation model for one AGV

For the AGV chart, Figure 4.4 shows the states inside AGV1 chart. The states of the AGV are the location of the AGV during the execution of the schedule (e.g. storage, mixing, filling stations). The input to the chart is carried out as a square signal with different values from zero till six. Each value represents an order for the AGV: zero is for going to the initial point, one is for the black filling station, two is for the yellow filling station, three is for blue filling station, four is for red filling station, five is for the mixing station and six is for storage. For each order for the AGV, there is a state inside the chart to represent it. In addition, there is a transition state to represent the transition from one station to another based on the time the AGV needs to reach the destination. Inside the *mixing* and *filling* states, there is a *send* command to send a signal from the AGV chart to the corresponding station chart to start operating (e.g. *send*(*fillBlue*)).

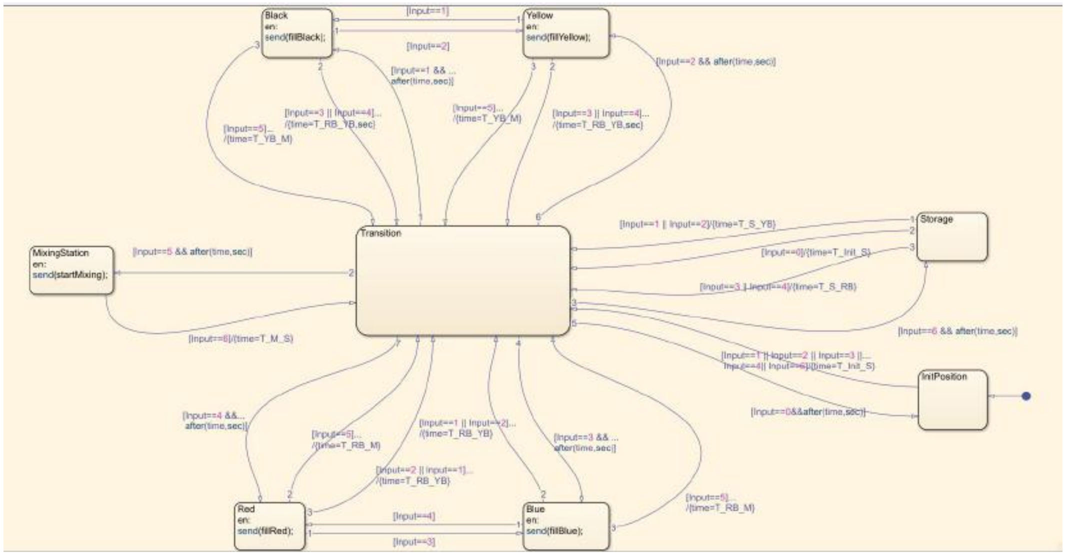

Figure 4.4: The states inside one AGV chart

For the filling stations charts, there are generally three states, no filling, filling color one or filling color two as shown in Figure 4.5 as an example for red and blue filling station and the same goes for yellow and black filling station. The transition between *nofilling* and *filling* states is carried out depending on the message received from the AGV chart (e.g. *fillRed*). After the filling time (assumed by 10 seconds) is over, the chart goes back to *nofilling* state.

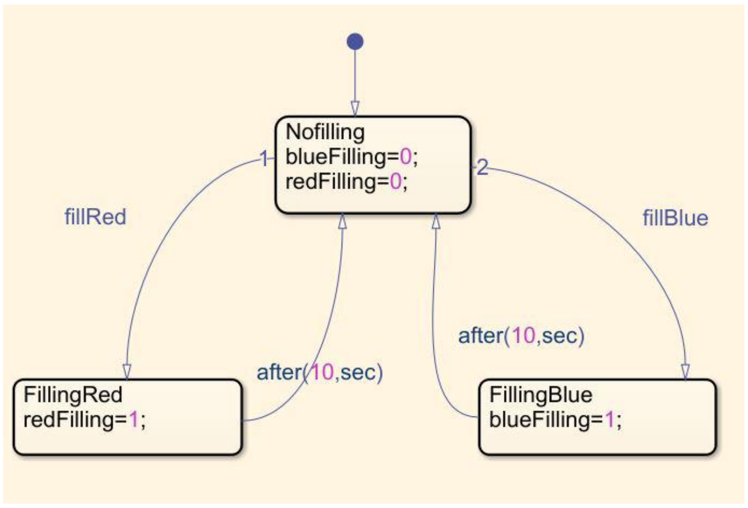

Figure 4.5: Chart for red and blue filling station

For the mixing station shown in Figure 4.6, it has two states only and the transition also depends on the message *startMixing* from the AGV chart. The mixing duration is assumed to be 15 seconds and the chart goes back to *MixerOFF* state afterwards.

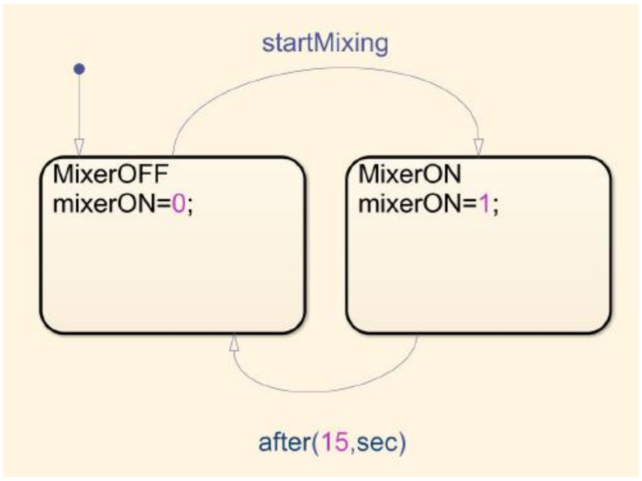

Figure 4.6: chart for mixing station

To test this model, the following recipe was implemented as an example to see the states sequence in the model. The task is to fill the vessel with the blue color, then going to the mixing station and then storage. As shown in the sequence view in Figure 4.7, The sequence for the AGV chart starts at the initial position, it then goes to transition to blue state where $fillBlue$ massage is sent and the chart $RedBlue$ goes to $fillingBlue$ state, it then goes to transition to mixing state where $startMixing$ message is sent and the Mixing chart goes to $MixerON$ state, then it goes back to storage.

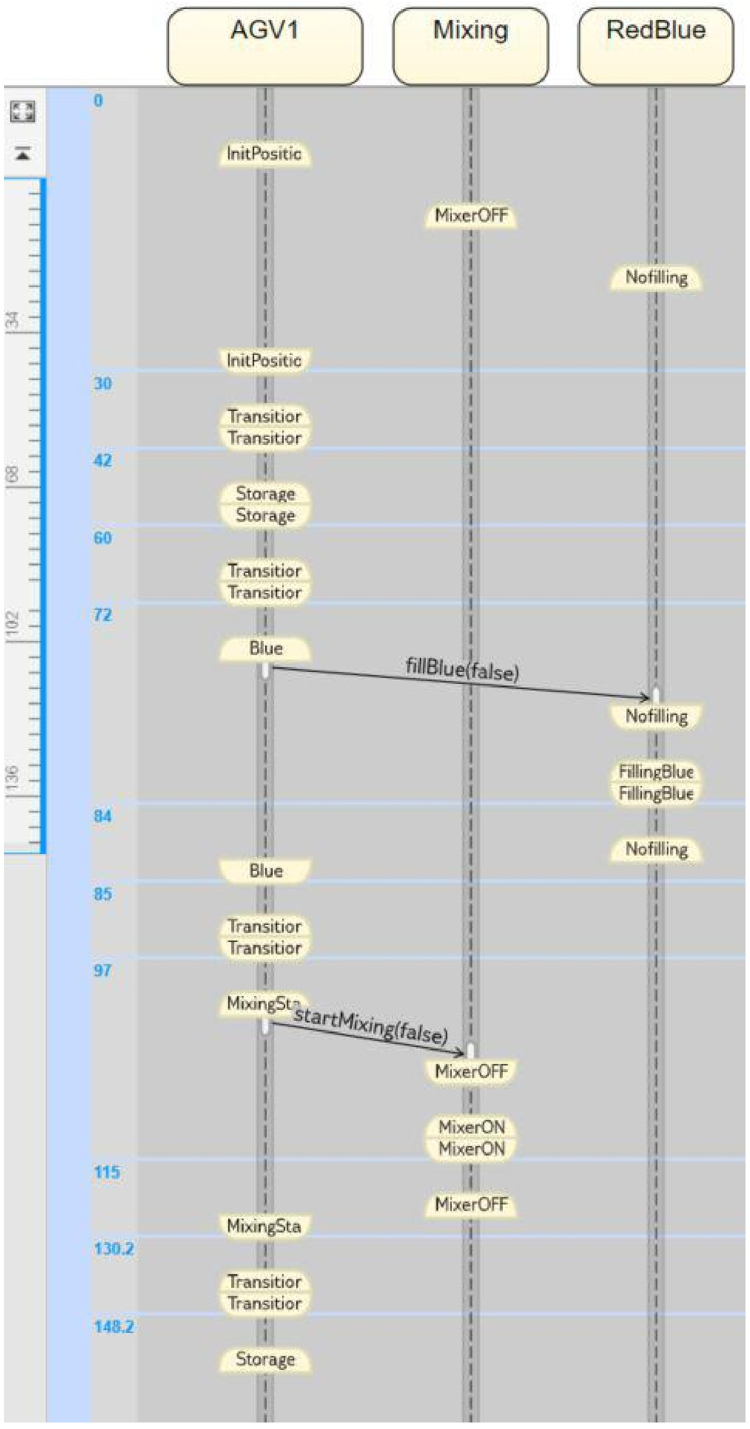

Figure 4.7: Sequence view for one recipe

Further improvement is done to the AGV chart by adding a state called *InvalidTransition* that the system goes to and terminates the simulation whenever an invalid transition between stations is committed by the schedule. The invalid transitions are as follows:
1) From the storage, the AGV cannot go directly to the mixing station without filling the vessels first.
2) From the filling stations, the AGV cannot go directly to storage without mixing the liquids first.
3) From the mixing station, the AGV cannot go back to filling stations.
By adding these conditions, the model will check the schedule for any invalid transition, sending a signal to terminate the simulation and show a message including the mistake made. Figure4.8 shows an example of invalid transition. The order is to go from storage to filling blue and back to storage without going to the mixing station. The chart went to *InvalidTransition* state directly after blue.

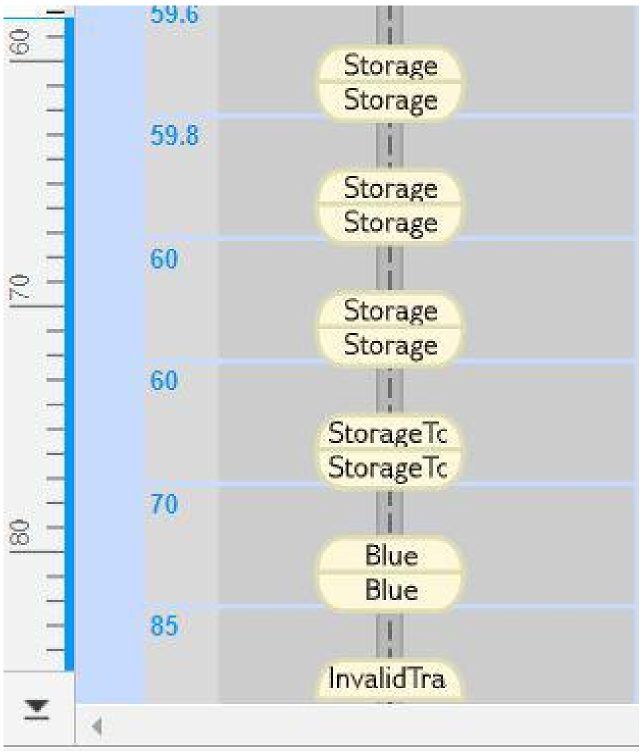

Figure 4.8: Snippet from Sequence View shows an invalid Transition state

For the full schedule check with multiple AGVs, the same AGV chart is used four times each represents an AGV while using the same charts for stations. The only modification made is to add a *StationError* state in each AGV chart to represent an error whenever two AGVs are trying to access the same station at the same time. Inside each AGV chart, a modification is made to check whether the station is occupied or not before reaching it. In case the station is already occupied, the chart terminates the simulation and goes to *StationError* state. Videos for some test cases and the full model files can be found in the GitHub link provided in Appendix.

4.3 AGV trajectory simulation

The second part of simulation is to draw a real time plot of the AGVs trajectory over time and to detect any collision between the AGVs. In this part, a Simulink model is created to take the velocity and angular velocity values for each AGV from the optimizer solution, run the simulation and give each AGV coordinates over time during operation. The simulation stops if any collision between AGVs is detected. After that, these coordinates are saved to MATLAB and then a MATLAB script runs to show a plot of the trajectory. The plotting is done in MATLAB instead of Simulink as plotting in MATLAB gives access to a lot of plotting options (e.g. including a map).

4.3.1 Implementation

Figure 4.9 shows the model used for the implementation. The velocity and angular velocity values are read from an excel sheet that is produced by the optimizer. From Robotic System Toolbox, Differential Drive Kinematic Model block is used to represent the kinematic model of the AGV. This block takes the velocity and angular velocity as inputs and gives the xy-coordinates and robot heading as outputs. Some parameters have also to be set for the block for calculations of position, for example the wheel radius r, the length between two wheels l and initial position and heading values. In this model, the radius is set to be 0.01 m and the track length to be 0.15 m for both AGVs. The initial position is set to be $[0.8; 0.5; 0.655043]$ for the x,y and heading values for AGV1 and $[1.13; 0.4; -0.13643]$ for AGV2.

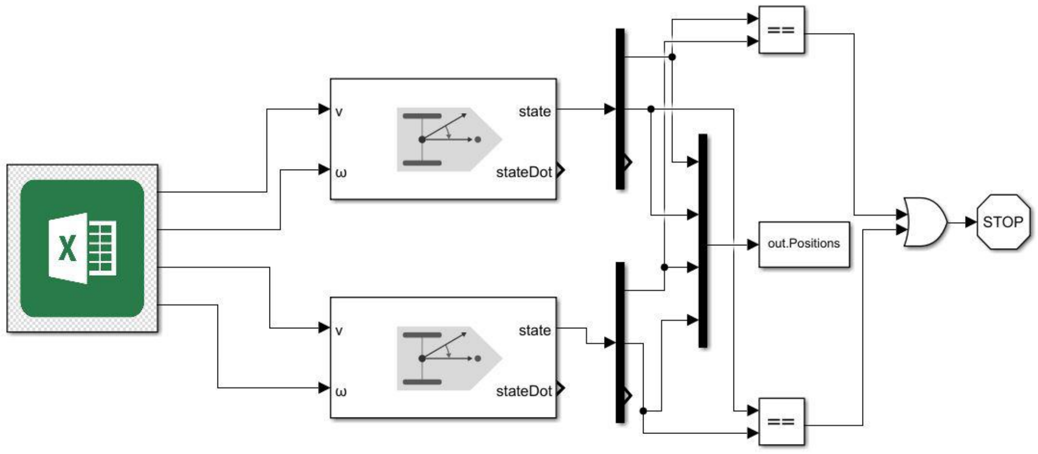

Figure 4.9: The AGV trajectory plotting model

The recipe used for this simulation is as following: for the first AGV, initial, Storage, fillBlue, Mixing, Store, initial and for the second AGV , initial, Storage, fillingYellow, fillingRed, mixing, store. Figure 4.10 shows the result of the simulation where the upper station is the storage, the lower station is the mixing station, left station is yellow and black color station and right station is for blue and red color. As seen in the figure, the AGV starts from the initial point to the storage, then goes to filling blue color station, afterwards it goes to the mixing station and then back to storage to store the recipe then back to initial point.

For the second AGV, Figure 4.11 shows the trajectory of the AGV as it follows the recipe. The AGV starts from initial position to storage, picks up a vessel and then goes to filling yellow color station, afterwards it goes to filling red color station and then to mixing station to mix the recipe then back to storage to store the recipe then back to initial point. Note that both AGVs are working simultaneously but the figures show the trajectory individually as showing both trajectories together will look like they are colliding while in reality the time for every trajectory is different and it has to be shown in a video to spot the live plot. However, the model is uploaded to GitHub and the link to it is available in the appendix.

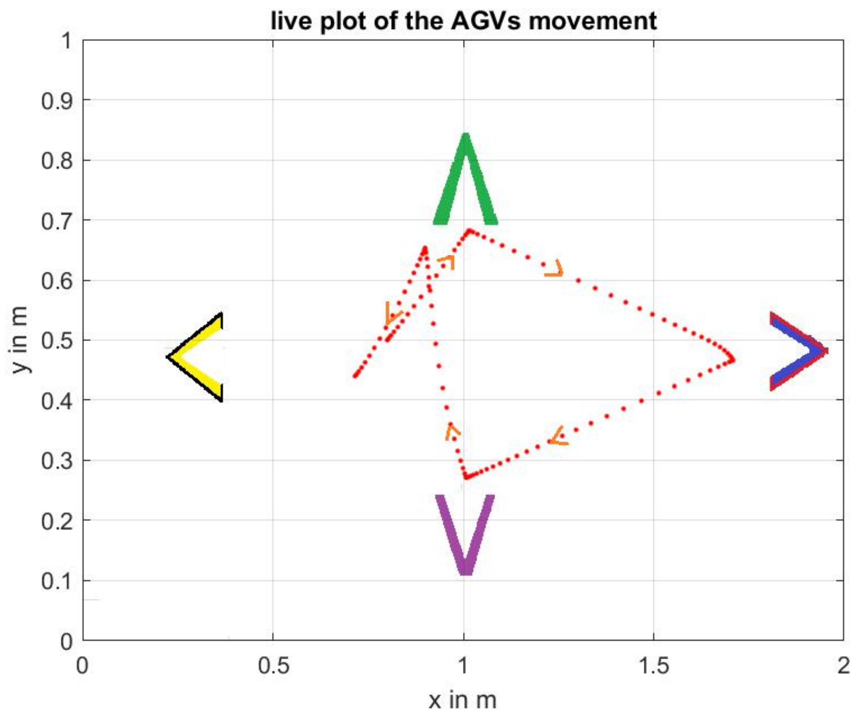

Figure 4.10: AGV1 Trajectory

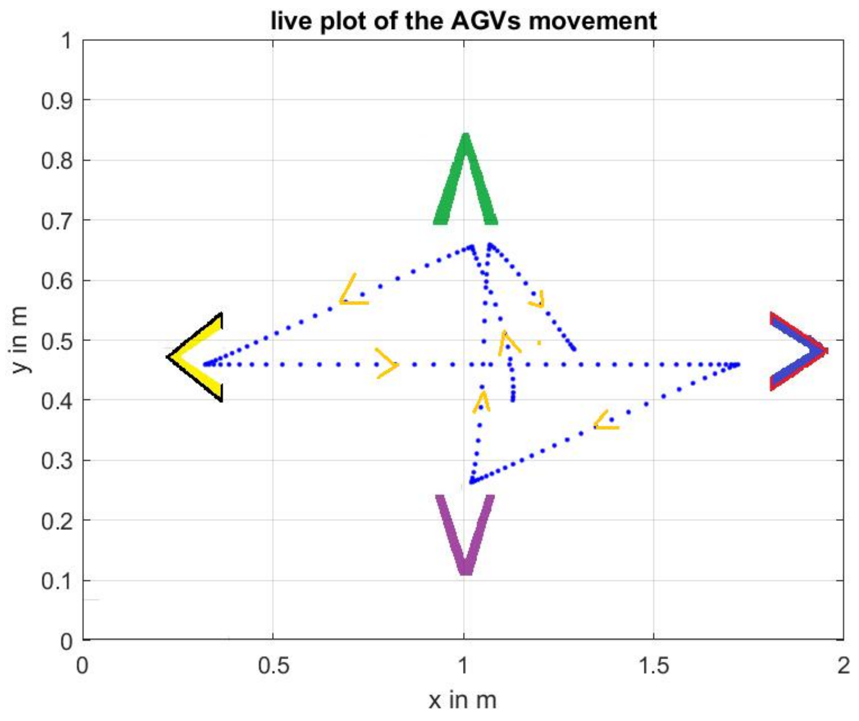

Figure 4.11: AGV2 Trajectory

CHAPTER

FIVE

CONCLUSION AND FUTURE WORK

This project has shown that the Resource-Task Network discrete time formulation can efficiently capture the scheduling constraints of the proposed batch processing plant. It is a general approach and can be used for other applications with industrial relevance. An optimal schedule was obtained for given number of resources fulfilling all the given recipes. Time and position of the AGVs were then fetched from the obtained schedule and were used as inputs for trajectory planning. Trajectories were obtained using dynamic optimization approach by obeying static and dynamic obstacle avoidance constraints. The trueness of schedule and trajectories were verified using matlab - simulink simulation.

Future suggested work for finishing the design of a scheduler for the pipe-less production plant would be to implement online trajectory planning and to implement it on the real plant.

BIBLIOGRAPHY

[1] Gurobi. https://en.wikipedia.org/wiki/Gurobi.

[2] Ipoptwiki. https://en.wikipedia.org/wiki/IPOPT.

[3] irobot_specifications. https://www.irobot.lv/uploaded_files/File/iRobot_Roomba_500_Open_Interface_Spec.pdf.

[4] Path planning for a simple car. https://nbviewer.jupyter.org/github/jckantor/ND-Pyomo-Cookbook/blob/master/notebooks/06.03-Path-Planning-for-a-Simple-Car.ipynb.

[5] Pyomo colllocation. https://pyomo.readthedocs.io/en/stable/modeling_extensions/dae.html#collocation-transformation.

[6] Pyomo overview. https://pyomo.readthedocs.io/en/stable/pyomo_overview/.

[7] Stateflow mathworks. https://de.mathworks.com/help/stateflow/index.html.

[8] Behrens, M., Khobkhun, P., Potschka, A., and Engell, S. (2014). Optimizing set point control of the mcsgp process. In *Control Conference (ECC), 2014 European*, pages 1139–1144.

[9] C. C. Pantelides, M.J. Realff, N. S. (1997). Short-term scheduling of pipeless batch plants. *JUBILEE SUPPLEMENT-Trans IChemE*, (3).

[10] David Alexandre, Larsen Kim, L. A. M. M. P. D. (2015). Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer.*

[11] Duinkerken, M. B., van der Zee, M., and Lodewijks, G. (2006). Dynamic free range routing for automated guided vehicles.

[12] Engell, S. (2007). Feedback control for optimal process operation. *Journal of Process Control*, 17(3):203 – 219.

[13] Furfaro Angelo, N. L. (2007). Modelling and schedulability analysis of real-time sequence patterns using time petri nets and uppaal.

[14] Jianbin Xin, Rudy R. Negenborn, G. L. (2014). Trajectory planning for agvs in automated container terminals using avoidance constraints: a case study. pages 9828–9833.

[15] Luiz Felipe Verpa Leite, Robson Marinho A. Esposito, A. P. V. F. L. (2015). Simulation of a production line with automated guided vehicle: A case study. *Independent Journal of management production (IJMP)*, 6(2).

[16] Nocedal, J. and Wright, S. (1999). *Numerical Optimization (Springer Series in Operations Research and Financial Engineering).* Springer, 1st. edition.

[17] Pantelides, C. C. (1994). Unified frameworks for the optimal process planning and scheduling. *Cache Publications: New York*, (3).

[18] Pedro M. Castro, Lige Sun, I. H. (2013). Resourcetask network formulations for industrial demand side management of a steel plant. *IEC research*, 52(3):13046 – 13058.

[19] Sebastian Panek, Sebastian Engell, C. L. (2005). Scheduling of a pipeless multi-product batch plant using mixed-integer programming combined with heuristics. *European Symposium on Computer Aided Process Engineering-15*, (3):1033 – 1038.

[20] Sharratt, P. N. (1997). *Chemicals manufacture by batch processes.* Springer, 1st. edition.

[21] Srinivasan, B., Palanki, S., and Bonvin, D. (2000). Bonvin d. dynamic optimization of batch processes: I. characterization of the nominal solution. comp chem eng.

[22] Williams, T. J. and Otto, R. E. (1960). A generalized chemical processing model for the investigation of computer control. *AIEE Trans*, 79(5):458 – 473.

[23] Yao Weijia, Dai Wei, X. J. L. H. Z. Z. (2015). A simulation system based on ros and gazebo for robocup middle size league.

[24] Yi-fei Tao, Jun-ruo Chen, M.-h. L. X.-x. L. Y.-l. F. (2010). Research of unidirectional automated guided vehicles system based on simulation. *2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management*, pages 1564–1567.

APPENDIX

The softwares used for the implementation of the project are:
1. Anaconda (Python IDE) with python version 3.7
2. Pyomo (Python software packages) with version 5.6.9
3. Gurobi (Optimization tool) with version 9.0
4. IPOPT (optimization Nonlinear Solver) with version 3.12.11
5. MATLAB/Simulink 2019 toolboxes: Stateflow and Robotics systems toolbox

The program files along with the step by step guidelines (readme files) are provided in the GitHub community : `https://github.com/anayghatpande/DYN_project`

Eidesstattliche Versicherung

Name, Vorname	Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum	Unterschrift