University of Pavia

FACULTY OF ENGINEERING

Ph.D. School in Electronics, Computer Science and Electrical Engineering

Doctor of Philosophy

Optimal Dispatching Solutions for Industry 4.0

Supervisor

Prof. Davide M. Raimondo

Candidate

Alessio Mosca

Thesis submitted in 2019

Doctor of Philosophy Thesis

A.Y. 2018/19

COLOPHON

This thesis was typeset using the typographical language LaTeX.

alessio.mosca01@universitadipavia.it

“Alla mia famiglia,

Marzia e Khaleesi”

Contents

1 Introduction **1**

1.1 Thesis overview 9

2 Dispatching Problem **15**

2.1 Introduction 16

2.2 Preliminaries 17

2.2.1 Dynamic Programming 17

2.2.2 DP Basic Problem 17

2.3 Problem Formulation 18

2.3.1 Scenario 19

2.4 1 vehicle per order 20

2.4.1 Problem Definition 20

2.4.2 Solution 20

2.5 n vehicles per n orders 22

2.5.1 Problem Definition 22

2.5.2 Solution 22

2.5.3 Example 24

2.6 m vehicles per n orders 25

2.6.1 DP with Multiple Orders and 1 Vehicle 25

2.6.2 Extension to the multiple-vehicles case 28

2.7 DP on Dynamic Graph 33

2.7.1 Scenario 34

2.7.2 Order Priority 35

2.7.3 Battery Model 35

2.7.4 Problem Formulations 40
2.7.5 Solution 42
2.7.6 Results 55
2.8 Conclusion 61

3 Multi-robot Routing and Scheduling with Temporal Logic and Synchronization Constraints 63
3.1 Introduction 64
3.2 Preliminaries 65
3.3 Problem Formulation 68
3.3.1 Environment Model 69
3.3.2 Robot Model 69
3.3.3 Specification: Transportation Demands and Synchronization Rules 69
3.3.4 Problem Definition 70
3.4 Nominal Solution 71
3.5 Robust Solution 73
3.5.1 On-line Controller 73
3.6 Simulations and Results 78
3.6.1 Baseline 81
3.6.2 Online Controller 82
3.7 Conclusion 82

4 UNIPV Test-bed 83
4.1 Introduction 84
4.2 Infrared Camera 85
4.2.1 Hardware 85
4.2.2 Image Formation 86
4.2.3 Perspective Projection 87
4.2.4 Image Discretization 90
4.2.5 Rigid Transformation 91
4.2.6 Camera Calibration 92

4.2.7 3D-2D Transformation 95
4.2.8 Planar Calibration 97
4.2.9 Distortion Model 97
4.2.10 Calibration Method 98
4.2.11 Evaluation of the Calibration 101
4.3 Working Environment 105
4.3.1 Charging Station 106
4.4 Robots 108
4.4.1 Mobile Robot Description 109
4.4.2 Hardware 109
4.4.3 Battery Monitoring 122
4.4.4 Connections 122
4.5 Software 124
4.5.1 Introduction 124
4.5.2 Visual Studio 125
4.5.3 Matlab 127
4.5.4 Raspberry 128
4.6 Low Level Controllers 137
4.6.1 Pure Pursuit Approach 138
4.6.2 Model Predictive Control 144
4.7 Results 177
4.7.1 Path Following 177
4.7.2 Path Following on Graph 179
4.7.3 Normal Operation 180
4.7.4 Multiple Robots 181
4.8 Conclusion 182
4.8.1 Future Development 182

5 Battery Ageing-Aware Stochastic Management of Power Networks in Islanded Mode 185

5.1 Notation and Network Description 187
5.1.1 Conventional Generators 188

5.1.2 Renewable Generators 188
5.1.3 Loads 189
5.1.4 Energy Storage Systems 190
5.1.5 Costs 191
5.1.6 Network Constraints 192
5.2 Deterministic Optimization 194
5.3 Stochastic Optimization 195
5.3.1 Scenario Optimization with Certificates 195
5.3.2 Stochastic OPF Formulation 197
5.4 Closed Loop Solution 201
5.5 Case Study 202
5.5.1 Simulation Results 204
5.6 Conclusion 206

6 Conclusion and future work 207

Bibliography 211

List of Figures

1.1 The Evolution of Industry 1.0 to 4.0 [1] 1
1.2 Integration aspects in Industry 4.0 [2] 2
1.3 Internet of Things, Cyber Physical Systems and Smart Factory [3] . 4
1.4 Wafer [4] 5

2.1 Abstraction of the graph, plan taken from [5] 19
2.2 Graph $\mathcal{G}$ 24
2.3 Paths on graph 25
2.4 The environment $\mathcal{G}$ 32
2.5 The ECM of the DP model 35
2.6 Applied forces on the vehicle 37
2.7 Part 1: Optimization on dynamic graph 44
2.8 Part 2: Optimization on dynamic graph 45
2.9 The scenario given by Infineon Villach 55
2.10 The graph obtained by the fixed step discretization. 57
2.11 The MatLab simulator 59

3.1 Semiconductor fab 78
3.2 Robot TS 79
3.3 The environment graph (a) and the robot transition system (b) . . 79

4.1 UNIPV camera 86
4.2 The pinhole model [6] 87
4.3 Perspective projection 88
4.4 Perspective projection 89
4.5 Digitization of the image 90

4.6 World system, Camera system and Image system 91
4.7 Effects of radial distortion [6] 95
4.8 Pinhole model [7] 96
4.9 Checkerboard pattern with highlighted corners 99
4.10 Example of acquired image 100
4.11 Corner recognition in distorted image 100
4.12 Correct image 101
4.13 World reference system 102
4.14 Recognition software 102
4.15 Subdivision of worktop with reference to the coordinates corresponding to each zone 104
4.16 Accuracy of analyzed points 105
4.17 The worktop graph $\mathcal{G}$ 106
4.18 Support surface 106
4.19 Charging station 107
4.20 Charging station with robot 108
4.21 One of the mobile robots used in this project 109
4.22 Kit DFRobot 110
4.23 Robot clamp 111
4.24 Roof of the robot 1 111
4.25 Roof of the robot 2 112
4.26 Roof of the robot 3 112
4.27 Battery cases 112
4.28 Sliding contacts 113
4.29 Raspberry PI 3 Model B [8] 114
4.30 Arduino Bluno Nano [9] 115
4.31 PWM signal with its two basic time periods 116
4.32 DC motor control 116
4.33 *PWM signal* 117
4.34 Pinout L293D 118
4.35 H-bridge 119

4.36 H-bridge operation 120
4.37 Circuit diagram 121
4.38 Voltage divider 122
4.39 Simplified block diagram 123
4.40 USBBLE-LINK USB Bluetooth dongle 124
4.41 Test-bed communications block diagram 125
4.42 Screens generated by Visual Studio 126
4.43 Text file generated by MatLab 127
4.44 Algorithm scheme implemented on Raspberry 129
4.45 Possible values of the inclination angle of the robots 132
4.46 UNIPV robots 132
4.47 Geometric relationship of the algorithm 140
4.48 Pure Pursuit approach with different look ahead distances 141
4.49 Pure Pursuit flow chart 142
4.50 Robot differential-drive model 145
4.51 4.51a Pure translation occurs when both wheels move at the same angular velocity; 4.51b pure rotation occurs when the wheels move at opposite velocities 146
4.52 Control System represented by SISO model, for right (a) and left (b) motor 148
4.53 Sampled data plot 152
4.54 System Identification Toolbox: parameters estimation 154
4.55 Model-output plot 155
4.56 MPC algorithm operations 158
4.57 Receding horizon approach 159
4.58 Receding horizon control 159
4.59 Simulation on Raspberry 173
4.60 Simulation on Raspberry 174
4.61 *ARX model validation* 175
4.62 Piecewise linear motor model 176

4.63 Comparison between the reference trajectory (red) and the one really followed by the robot (blue) 177
4.64 The environment $\mathcal{G}$ 180
4.65 The environment $\mathcal{G}$ 181

5.1 IEEE 14 bus 202
5.2 Profiles with ESSs 204
5.3 Profiles without ESSs 204
5.4 Detail of demand profiles with ESSs 205
5.5 Detail of demand profiles without ESSs 205

List of Tables

2.1 Transport requests list 24

2.2 The shortest paths for v_1, v_2 and v_3 24

2.3 FCFS list 32

2.4 Suboptimal sequences π_{v1}, π_{v2} and π_{v3} obtained by using Algorithm 4 32

2.5 π_{v_1}, π_{v_2} and π_{v_3} 33

2.6 The main parameters of the DP battery model 57

2.7 Case 0: the order is assigned to an available vehicle which is capable of fulfilling the order without obstructing other vehicles to reach the charging station. 59

2.8 Case 1: the order can be assigned to a currently busy vehicle which respects the constraints. The charging time is not considered in the vehicle assignment 60

2.9 Case 2: Also the time to reach the charging station and to fully charge the battery are taken into account in the vehicle assignment 60

3.1 Transport Demands ($\mathcal{D}$) and Synchronization Rules ($\mathcal{R}$) 80

3.2 Optimal Nominal Solution $\mathbf{p}^*$ 81

3.3 Quantitative information on scalability 81

3.4 $\mathbf{sync}_v$ and $\mathbf{pick}_v$ for each $v \in \{1, 2\}$ 82

4.1 Correspondence between coordinates measured and found analytically 105

4.2 H-bridge operation 120

4.3 Radius of curvature and PWM values 139

4.4 FCFS list 179

4.5 FCFS list 181

Listings

4.1 Coordinates (X, Y) of graph nodes $\mathcal{Q}$ 129

4.2 Reading UDP packets 131

4.3 *struct* robot 131

4.4 Robot detection Algorithm 134

4.5 Syntax to send the data to the serial port 135

4.6 Arduino software: variable declaration 136

4.7 Arduino software: *void loop()* 137

Chapter 1

Introduction

The industrial world is going through a revolution. Indeed, worldwide, production systems are involved in a rapid wave of innovation which is expected to improve this field as never before. Technological advancements and continuous research are the driving forces of this fourth industrial revolution [10], which is also called "Industry 4.0" [11, 12]. Analyzing the industrial revolutions, it is easy to identify in each historical moment a driving event: the adoption of steam engines in the first revolution, the use of electricity in the second, the introduction of electronics and information technology in the lines of production in the third, while in the fourth the fusion of the physical world with the virtual one thanks to the cyber-physical production systems. Therefore, the fourth industrial revolution is subsequent to

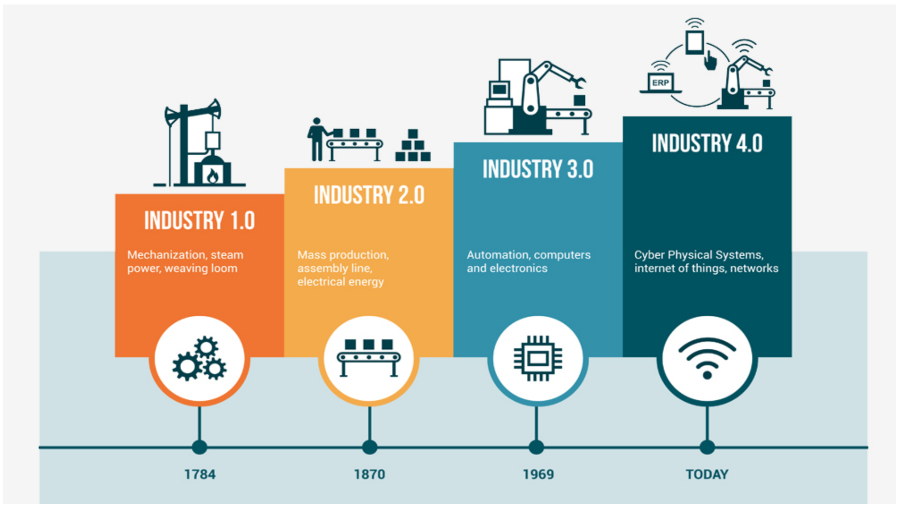

INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks
ERP
1784
1870
1969
TODAY

Figure 1.1: The Evolution of Industry 1.0 to 4.0 [1]

the advent of the Internet and related to the intuition that any device associated with its own identity and integrated with sensor networks can be programmed to interact or be part of even complex systems that can be reached remotely. In this context, factories have machines which are augmented with wireless connectivity and sensors, connected to a system that can visualise the entire production line and make decisions on its own.

The European scene is the main context where the fourth industrial revolution takes place. Indeed, the concept of Industry 4.0 was born in Germany in 2011 [13]. As described in [14, 15, 16, 17, 2], Industry 4.0 consists of three main features: *horizontal integration*, *vertical integration* and *end-to-end integration*, see Figure 1.2. *Horizontal integration* refers to the value creation network that includes both internal and external functions of the supply chain. This aims at establishing a collaborative network across the entire value creation network in order to create an efficient ecosystem where material, information, energy and finance can be exchanged quickly between several different companies [2, 16, 18, 19]. *Vertical integration* aims at the perfect integration of hierarchical systems within a company in order to obtain a flexible and reconfigurable manufacturing system [16, 2, 18, 20]. *End-to-end* integration describes the engineered integration across the entire product life cycle achieved through intelligent cross-linking and digitalization on the entire value chain: from product development to manufacturing system engineering, production, and services [16, 2, 21, 20].

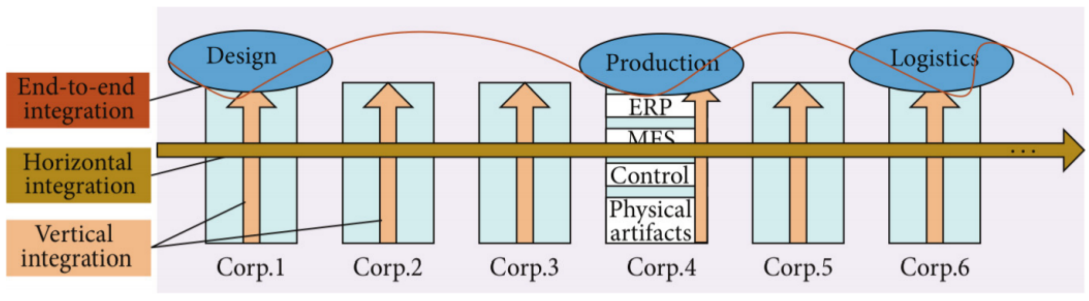


Figure 1.2: Integration aspects in Industry 4.0 [2]

Following an in-depth literature review conducted by authors in [22, 23], three key components of Industry 4.0 are identified to achieve the horizontal, vertical and end-to-end integrations. These components are: the *Internet of Things* (IoT) [24], *Cyber Physical System* (CPS) [25] and *Smart Factory* (SF) [26], see Figure 1.3. CPSs are systems that integrate physical components and processes with computing elements. These coordinate and communicate with sensors, which monitor cyber and physical indicators, and actuators, which modify the cyber and physical environment. To gain a deeper knowledge of the environment, which ensures more precise actions and tasks, CPSs rely on sensors to connect all distributed intelligence in the environment [27]. Note that the Cyber Physical Production Systems (CPPSs) are composed of several CPSs linked within digital networks [28]. The connection and data exchange with other embedded systems or clouds are achieved by using the IoT, while a human-machine interface is used for interacting with operators [17]. The IoT is the technology which allows the inter-connection of all types of devices through the Internet to exchange data, optimize processes and monitor devices. It is composed of a network of sensors, actuators, and devices, forming new systems and services [27]. Therefore, “things” and “objects” can interact with each other and cooperate with their neighboring “smart” components, to reach common goals [29]. The SF are factories where the CPSs interact and communicate with its environment through the IoT in order to assist humans and machines in task execution, and to ensure the collection, distribution and access of manufacturing relevant information from the physical as well as the virtual world in real-time [26]. Therefore, the combination of software, hardware and / or mechanics in the SF leads to an increase in process interoperability, allowing processes to be dynamically changed and adapted, and consequently in a reduction of unnecessary labour and waste of resource [26].

Thus, through the implementation of the concepts of Industry 4.0, it is possible to improve the entire production planning and optimize the flow of goods, optimize the quality of processes and products [18, 30]. These features are of crucial importance especially for the manufacturing industries. Indeed, these industries work in an intense competitive and highly sophisticated global environment, where, in order to survive, they must offer a higher product quality at lower costs and, at the same time, never exceed customer due date demands [31].

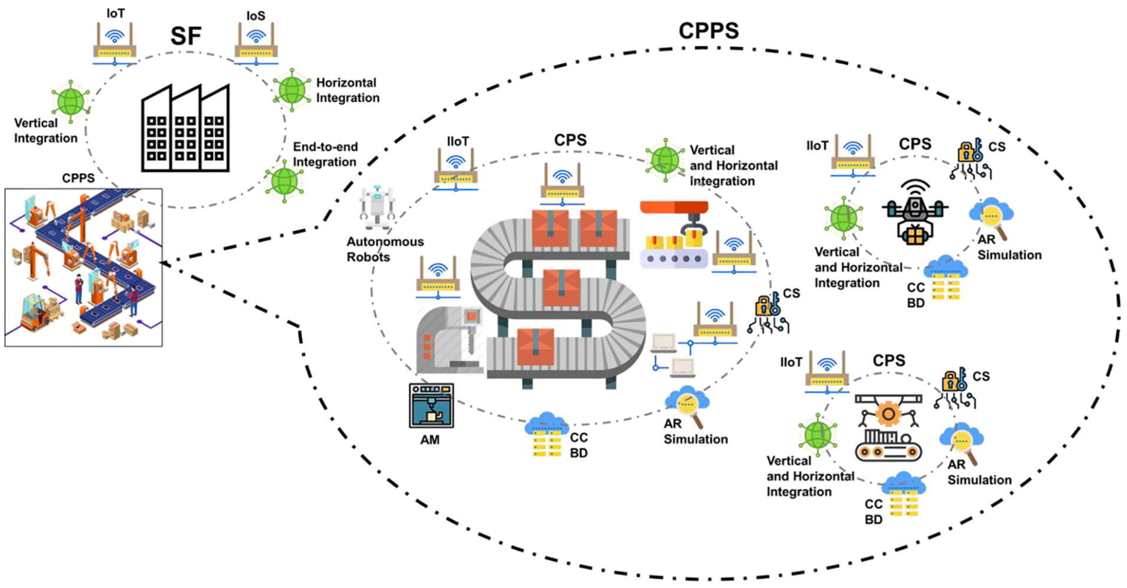


Figure 1.3: Internet of Things, Cyber Physical Systems and Smart Factory [3]

Since electronic components (chips) and systems (electronic boards) are the key drivers for Industry 4.0, one of the most involved manufacturing sectors in this revolution is electronics. One of the largest European projects focused on “smart production” and “cyber-physical production systems” is “Power Semiconductor and Electronics Manufacturing 4.0” (SemI40) [32]. This project aims to establish a smart, sustainable and integrated enterprise collaboration system manufacturing in order to increase the competitiveness of the Semiconductor manufacturing industry in Europe [32]. In particular, the final products of the manufacturing process of semiconductors are the integrated circuits of all electronic devices. A semiconductor is made in different stages using photo-lithographic and chemical techniques, and its circuits are gradually created on wafers (i.e. thin “slices” of semiconductor material, typically silicon, see Figure 1.4), see [33, 34] for a complete description of the process.

The processing is done layer by layer and all the production occurs in a clean room, so as to reduce particulate contamination and control other environmental parameters, such as temperature, humidity and pressure [35]. The transport of each batch of wafer can be done either by robots or by human operators. Several processes are needed to obtain a final product, for example a medium-complexity production involves 250-500 stages and uses from 50 to 120 different machines [36].

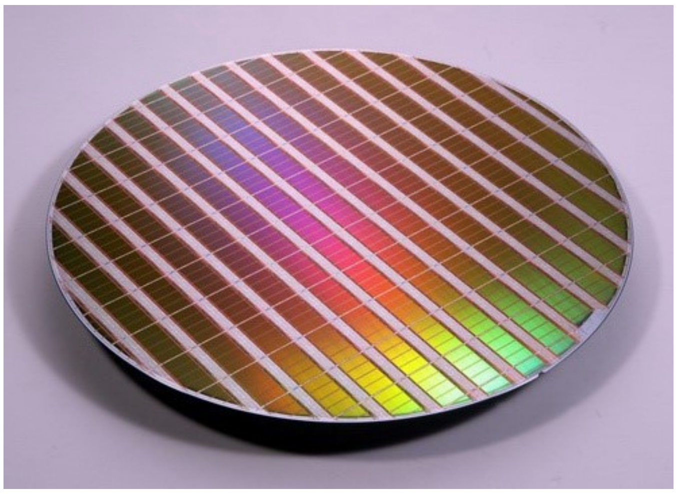


Figure 1.4: Wafer [4]

Furthermore, to survive in a competitive business environment, semiconductor manufacturing companies are required to meet Customer's demands in terms of time, quality and quantity. However, due to the ever growing demand for electronic products, see e.g. [37], this job has become increasingly difficult over the years. In order to cope with the complexity of the production process and satisfy Customer's demands, it is necessary to develop effective and efficient planning, scheduling and dispatching strategies. Thus, one of the goal of SemI40 is to design control strategies that optimize the robots behaviour in the Semiconductor Manufacturing factory. Several approaches have been considered in literature to solve this problem, see e.g. [37, 38, 39], for an overview. If all transport requests are known at the beginning of the process, it is possible to solve off-line the dispatching problem in an optimal way by using for example either the Dynamic Programming or the Hungarian method [40].

However, due to the dynamic and uncertain nature of the fabs, i.e. arrivals of urgent orders or unpredictable events (e.g. a machine out of order), it is impractical to use off-line dispatching methodologies [41]. A possible solution is to use real-time scheduling and dispatching policies, which allow to respond to unpredictable variations in system states in an efficient way. A number of real time approaches has been proposed in literature to deal with this problem. Most of the suggested methods rely on: dispatching rules [42], semi-Markov decision programming [43, 44, 45] and dynamic programming [46, 47]. Although dispatching rules have been widely used for scheduling because they are able to provide a very quick and sufficiently good solution, choosing the right rules is not an easy task [48, 49, 50]. Mathematical programming represents an alternative which can provide better solutions at the price of a higher computational complexity [51]. While such methods may provide interesting results and applications, they suffer from the the curse of dimensionality, which prevents their application to large scale systems, where many robots and many tasks are involved.

Recently, there has been an increasing interest in using approaches based on temporal logics in motion planning and robot control applications [52, 53, 54]. Temporal logics (TL) provides formal high-level languages to describe complex problems, such as routing and scheduling problems. When explicit time constraints are involved, e.g. when the pick up and deliver of a product need to be completed within 10 minutes or before time T=10 min, temporal logics with explicit time must be adopted. Examples include Metric Temporal Logic (MTL) [55], Signal Temporal Logic (STL) [56] Time Window Temporal Logic (TWTL) [52].

These properties make TLs useful for the development of dispatching strategies able to take into account, in addition to the classic transport demands, even more complicated transport requests, such as transport requests where the orders must be delivered in a synchronous way by the robot. Indeed, in the semiconductor manufacturing process, in order to maximize plant efficiency, some manufacturing steps require the simultaneous delivery of different components to start the processing of new goods, e.g. the diffusion furnaces [57]. The process can start only when all materials have arrived. Since components must be processed together, the maximum delay of a single component affects the overall process time. This can lead to a waste of time, money and efficiency. For this reason, taking into account any possible synchronization requests is of fundamental importance in order to provide the right components to the right machine/s at the right time. The authors in [54, 58] rely on Mixed Integer Linear Programming to obtain the vehicle trajectories that satisfies transport demands. In particular, transport requests are expressed using MTL. The problem of least-violating planning using linear programming is considered in [59, 60]. In [61] TWTL is defined and then used for solving the persistent vehicle routing problem. However, all the aforementioned papers do not take into account the possible presence of synchronization requests.

Despite the improvements made by the industrial revolutions, regarding the standard of living and the improvement in almost all aspects of society, there have also been important environmental problems, such as environmental pollution and global warming. These phenomena are mainly caused by the combustion of fossil fuels [62]. For this reason, Industry 4.0 aims both to reduce energy consumption by increasing energy efficiency in industrial processes and to promote and improve the use of renewable sources [63, 64, 65]. However, the increasing exploitation of renewables [66, 67, 68] has raised the need for more sophisticated power management systems able to cope with the intermittent and unpredictable nature of these sources [69].
In general, Optimal Power Flow (OPF) methods deal with the management of power networks by solving an optimization problem where constraints are used to guarantee safe and reasonable network operations [70, 71, 72]. Note that the OPF can be seen as an electric power dispatching problem, in which it is necessary to optimize the dispatch of generated power. The Alternating Current OPF (AC-OPF) formulation is non-convex and can be solved using off-the-shelf solvers for nonlinear programs, e.g. [73, 74], or relying on convex relaxations either by keeping the AC formulation, [75, 76, 77], or using a Direct Current (DC) approximation, [78, 79]. The authors propose in [80] two AC-based convex relaxations. The first method relies on a Semidefinite Programming (SDP) relaxation of the primal problem, while the second uses an SDP-type Lagragian dual of the OPF. The paper shows that the SDP relaxation provides the global optimal OPF solution if, and only if, the duality gap is zero. The authors also provide necessary and sufficient conditions for guaranteeing the zero duality gap. In particular, such conditions hold for the IEEE benchmark 14, 30, 57, 118, 300 buses. In [81, 82], the authors address the cases in which the zero duality gap cannot be obtained. In particular, they propose a penalized SDP relaxation, where a penalty term is introduced to obtain a low rank solution and, consequently, a near-to-optimal OPF solution.

However, the deterministic OPF methods have demonstrated to be successful in networks with limited load uncertainties and low renewable penetration. Stochastic OPF (SOPF) has been proposed as a possible solution in order to satisfy the network constraints with a desired probability and reduce the risk of dangerous phenomena (i.e. out of service lines and failure of the entire system). The authors in [83] propose a SOPF based on chance constraints and nonlinear programming. The work in [84] also relies on nonlinear programming to deal with the AC-SOPF but it addresses the stochasticity of the problem resorting to Polynomial Chaos Expansion and Galerkin-projection. Within the context of SOPF, convex relaxations have been adopted in e.g. [85, 86, 87]. In [86] the authors propose a DC relaxation and use a Scenario-based approach to deal with the stochasticities. A Scenario-based approach is also used in [85]. In this case, an SDP relaxation of the AC problem is considered and a Scenario with Certificates (SwCs, [88]) used to alleviate complexity. The authors in [87] propose an AC probabilistic OPF based on an SDP relaxation and chance constraints. In particular, the problem is solved using a combination of a Scenario-based approach and robust optimization.
The methods considered above are suitable for interconnected grids, where the power required by loads can always be satisfied. However, when the available power is limited (i.e. islanded mode grid), the node power balancing may not be guaranteed. In this case, Demand Response policies (DR) can be considered in SOPF formulations as a possible solution [89, 90]. In particular, DR is a class of demand side management programs [91], in which the utilities may curtail user loads in exchange for a bill reduction. While DR allows to respect the grid power balancing, it leads to lower income for the utilities and to a disruption of service for the end users. A significant improvement can be obtained using Energy Storage Systems (ESSs) that could result in a reduction of conventional power production and load curtailments by uncoupling the power production from the demand [92, 93, 94, 90]. ESSs have been used together with DR policies in the context of SOPF problems in e.g. [87, 95]. The method in [87] relies on a DC approximation and a scenario-based approach, while the work in [95] presents a SOPF based on a convex approximation and chance constraints. ESSs provide several benefits to the management of power

networks but they have non negligible fixed costs. Therefore, their proper use is important to avoid rapid degradation and their subsequent replacement.

In this thesis, optimal dispatching solutions for Industry 4.0 are proposed. In particular, two different dispatching problems are addressed. The main problem is the optimal dispatching problem in a semiconductor production site while the minor problem is the optimal dispatching of electric power in power networks operating in islanded mode.

1.1 Thesis overview

The first and main problem deals with the control of the fleets of autonomous vehicles used for semiconductor production. The production process requires wafers to undertake several steps, and robots or human operators are responsible for the dispatching. For this reason, this work aims to find dispatching strategies for an optimal management of the fleet and, consequently, an improved management of the production site. This type of problems can be seen as "pick-up and delivery problems", in which the orders can be seen as "passengers". Taking advantage of this particular vision of the problem, in this thesis appropriate algorithms based on Dynamic Programming or Time Window Temporal Logic are developed, presented in Chapters 2 and Chapter 3, respectively.
Besides, in order to evaluate the effectiveness of the strategies, it is necessary to develop a testbed composed of three small scale robots in the UNIPV laboratories. The position and orientation of the robots are detected by an infrared camera positioned on the ceiling of the lab. The robots are equipped with a clamp to simulate the drag and drop of the orders. The realization process, and the testbed operation are illustrated in detail in Chapter 4.
Finally, in Chapter 5, the optimal power flow in presence of Energy Storage Systems (ESSs) and Demand Response (DR) is studied. Given the nature of the network in question and the presence of uncertainties arising from both the use of renewable sources and unexpected variations in load, the problem is formulated as a Stochastic

OPF. Therefore, it is necessary to find a control in order to satisfy the constraints of the network with a desirable level of probability and to reduce the risks of any dangerous phenomena (failure of the entire network and / or out of service lines). Unlike other works in literature aimed at minimizing only production and curtailment costs, this thesis places greater emphasis on the use of energy storage systems, i.e. lithium ion batteries, taking into account also the functional cost of battery ageing.

Thesis structure

Chapter 2: Dispatching Strategy In this chapter solutions to the wafer dispatching problem based on Dynamic Programming are proposed. Due to the problem complexity, the dispatching problem will be addressed from the simplest case to the most complicated case.
In order to consider real scenarios, where walls or obstacles provide constraints to the movement of the robots/humans, all the proposed approaches rely on the Graph theory. In particular, a graph $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$ is used, which allows to abstract the operating environment, where: the nodes/vertexes ($\mathcal{Q}$) represent the pick-up/delivery locations (the machines where the wafers are processed); the arcs/edges (Δ) indicate the existence of a connection between different locations; the weights (ω) of these arcs represent the distances between the vertexes that the edges connect.
The first and second approach proposed aim to find the shortest route to pick up and deliver an order/orders by using one vehicle per order.
In the third developed approach, both the order priorities and the possibility to have a vehicle with variable transport capacity is considered by the introduction of appropriate constraints. This approach differs from the previous one also for the objective function, which is a weighted combination of the time to serve all customers and of the total degree of “dissatisfaction” experienced by them while waiting for service.
The fourth approach is the extension of the third approach to the case of multiple vehicles. In particular, a genetic algorithm is used to cope with the increase in complexity, while sacrificing optimality. This approach allows to find which is the

best assignment of n-orders to m-vehicles and the relative paths of the m-vehicles. The idea is to find the best n-assignments of the initial order list that minimizes the objective function given by the sum of m-vehicle cost functions.
In the fifth approach, the collision avoidance, the vehicle battery management and the possibility to assign an order to a currently busy vehicle are considered. Moreover, this approach deals also with the case of orders that arrive at different time instants in a scalable way. The complexity of a dynamic program approach grows with the number of orders and agents (e.g. vehicles) to be considered. For this reason, the dispatch of one order at a time is optimized and the graph is updated, abstracting the possible available routes and taking into account the dynamic presence of other vehicles on the way. Since orders come over time and are not known all a priori, it is necessary to find the path on the graph for a new order and its assigned vehicle without modifying the path of the existing ones. Even though this approach may lead to a suboptimal result, it allows to scale the optimization problem, thus providing a suitable solution for real time operation. In order to ensure obstacle avoidance, considering that some robots are already moving in the space, the optimization needs to operate on a dynamic graph. In particular, nodes occupied dynamically by previously assigned robots are omitted by the graph and the optimal dispatching is performed only on the residual graph. A vehicle can be assigned to an order only if it is able to satisfy the request and go back to the charging station with the available charge. This is achieved by using a non-linear battery model to simulate the battery discharge. Furthermore, in order to minimize the overall time of service, a task can be assigned for a future time to a currently busy vehicle rather than to a currently available one. This can happen when it is faster to fulfil the current order and pick-up the new one rather than having a currently available vehicle fulfilling the new order.
The results presented in Chapter 2 are published in:

- **A. Mosca**, G. De Nicolao, G. Schneider, T. Niekisch and D. M. Raimondo, "Optimal Wafer Dispatching based on Dynamic Programming", *European Advanced Process Control and Manufacturing*, Dresden, Germany, 2018.

- **A. Mosca** and D. M. Raimondo, “A low complexity wafer dispatching strategy for orders with different arrival times”, *European Advanced Process Control and Manufacturing*, Villach, Austria, 2019.

Chapter 3: TWTL This chapter focuses on the problem of scheduling and planning for a fleet of robots with motion uncertainty involved in manufacturing. The uncertainty concerns the duration of motion. Robots are tasked with transportation demands of goods that must be fulfilled according to given time constraints and synchronization rules. The demands and synchronization rules are specified using Time Window Temporal Logic (TWTL), which allows to express temporal properties with explicit time constraints. In addition, TWTL admits *temporal relaxation* semantics which enables a relaxation of time constraints in case of unsatisfiable tasks. In this chapter, a general solution to the nominal motion mode is developed, i.e., without robot motion uncertainty. Furthermore, an online controller is proposed in order to guarantee the satisfaction of the synchronization rules and to minimize deadlines relaxations in presence of robot motion uncertainties during deployment. The effectiveness of the approach is tested through simulations inspired by semiconductor manufacturing scenarios.
The results presented in Chapter 3 are published in:

- **A. Mosca**, C. Vasile, C. Belta and D. M. Raimondo, “Multi-robot routing and scheduling with temporal logic and synchronization constraints”, *accepted to 2nd International Conference on Control and Robots (ICCR)*, December 12-14, 2019, Jeju Island, Korea.

Chapter 4: Testbed In this chapter the activity carried out at the Process Control Laboratory of the University of Pavia is described. The objective of this activity is the design, implementation and control of a testbed for the validation of the proposed dispatching strategies. In particular, the goal is to simulate the operation of a semiconductor production plant in which it is necessary to transport semi-finished products between the various workstations by using a fleet of automated vehicles. The vehicles used for the testbed are laboratory-built robots, which are able to pick up and deliver the objects in the desired position. The position and orientation of the robots are detected by an infrared camera positioned on the ceiling of the lab.

Chapter 5: SOPF In this chapter, an Ageing-Aware Stochastic Optimal Power Flow in presence of Demand Response and Energy Storage Systems (ESSs) is presented. A proper use of these ESSs is fundamental in order to obtain high performance throughout their lifetime. For this reason, a degradation model of the ESSs in the optimization problem is considered. In this way, it is possible to optimize network management and to maximize their lifetime by taking into account ageing effects of the ESSs. In order to deal with network uncertainties, a stochastic optimization problem using a scenario approach which allows to provide a-priori level of constraint satisfaction is solved. The effectiveness of the proposed control strategy is tested by performing simulations on IEEE 14 bus.
The results presented in Chapter 5 are published in:

- **A. Mosca**, A. Pozzi and D. M. Raimondo, "Battery ageing-aware stochastic management of power networks in islanded mode", *22nd International Conference on System Theory,Control and Computing*, Sinaia, Romania , 2018.

Chapter 2

Dispatching Problem

Contents

2.1 Introduction 16

2.2 Preliminaries 17

2.2.1 Dynamic Programming 17

2.2.2 DP Basic Problem 17

2.3 Problem Formulation 18

2.3.1 Scenario 19

2.4 1 vehicle per order 20

2.4.1 Problem Definition 20

2.4.2 Solution 20

2.5 n vehicles per n orders 22

2.5.1 Problem Definition 22

2.5.2 Solution 22

2.5.3 Example 24

2.6 m vehicles per n orders 25

2.6.1 DP with Multiple Orders and 1 Vehicle 25

2.6.2 Extension to the multiple-vehicles case 28

2.7 DP on Dynamic Graph 33

2.7.1 Scenario 34

2.7.2 Order Priority 35

2.7.3 Battery Model 35

2.7.4 Problem Formulations 40

2.7.5 Solution 42

2.7.6 Results 55

2.8 Conclusion 61

2.1 Introduction

Planning, scheduling and dispatching are of critical importance in production systems. In particular, in the semiconductor manufacturing industry, the production process requires wafers to undertake several steps. For example, a medium- complexity production requires 250-500 steps and uses from 50 to 120 different machines [36]. The wafers dispatching can be done either manually (human operators) or automatically (by robots). In any case, designing optimal dispatching strategies is of major importance, since it can result in energy, time and cost savings. Several approaches have been considered in literature to solve this problem, see [37, 38, 39] for an overview. Many heuristic schemes have been developed over the years. In particular, dispatching rules have been widely used for real-time scheduling because they can provide a very quick and sufficiently good solution. On the other side, choosing the right rules is not an easy task. Mathematical programming represents an alternative which can provide better solutions at the price of higher computational complexity.

In this chapter, the developed solutions to tackle the dispatching problem in a semiconductor fab are shown. The presented approaches rely mainly on dynamic programming and graph theory. Tailored solutions (based on e.g. genetic algorithms and sub-optimal schemes) are also proposed to cope with the computational complexity of optimization-based approaches. The effectiveness of the proposed methodologies is demonstrated in simulation and on a testbed, developed at UNIPV, composed of robots with clamps aiming to simulate the real production site, see Chapter 4.

2.2 Preliminaries

2.2.1 Dynamic Programming

Dynamic programming (DP) is a method for solving a complex problem by breaking it down into a collection of simpler sub-problems where each subproblem gets solved only once and stored off-line. DP is based on the Principle of Optimality [96]: if an optimal trajectory exists, and a point on such trajectory is considered, then, by re-solving the optimal problem starting from this point, the remaining trajectory is still the same as before and it is still optimal.

2.2.2 DP Basic Problem

Considering the discrete-time dynamical system described by

$$x_{k+1} = f_k(x_k, u_k), \quad k = 0, 1, \cdots \tag{2.1}$$

where the input u_k and the state x_k verify the following constraints

$$x_k \in \mathcal{X} \qquad\qquad u_k \in U(x_k) \subseteq C \tag{2.2}$$

and $\mathcal{S}$ and C are finite sets, and $U(x_k)$ is the set of admissible input.
Let $\pi = \{u_0, \cdots, u_{N-1}\}$ be an admissible finite sequence of control laws, i.e

$$u_k \in U\left(x_k\right) \quad \forall x_k \in \mathcal{X} \tag{2.3}$$

Define $J_\pi(x_0)$ as the cost associated to the control policy π starting at x_0

$$J_\pi(x_0) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \tag{2.4}$$

where the functions $g_k(x_k, u_k)\, \forall k = 0, \cdots, N-1$ and $g_N(x_N)$ are given and represent the stage costs and the final cost, respectively.

The optimal control policy π^* is the policy that minimizes J_π, which is

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0) \tag{2.5}$$

where Π is the set of all admissible control laws [97].

The complete procedure for solving the Basic Problem is reported in Algorithm 1

Algorithm 1: Dynamic Programming

```
1 Set the solution of subproblem $N$ to $J_N(x_N) = g_N(x_N)$.
2 For $k = N-1, \cdots, 0$, for each possible $x_k$, solve the $k$-th “tail subproblem”.
      $J_k(x_k) = \min_{u_k \in U(x_k)} \{g_k(x_k, u_k) + J_{k+1}(f_k(x_k, u_k))\}$
  using the optimal cost-to-go of the $k+1$ tail sub-problem. Store the optimal
  cost and the optimal input
3 For a given $x_0$ at time $k = 0$, the optimal cost of the overall problem can be
  obtained as $J_{\pi^*}(x_0) = J_0(x_0)$. The corresponding optimal input sequence
  $\pi^* = \{u_0^*, \cdots, u_{N-1}^*\}$ is obtained by concatenating the optimal input
  elements of each subproblems.
```

2.3 Problem Formulation

Semiconductor production is the process used to make integrated circuits. The process requires several steps during which the circuits are gradually created on wafers of semiconductor material, typically silicon. All the production is carried out in a clean room, so to reduce particulate contamination and control other environmental parameters, such as temperature, humidity and pressure. The clean room is divided into bays (process work stations) and chases (service areas). The transport of each wafer batch can be done either by robots or by human operators. The production site contains several machines. Given the position of each machine and the relative position between bays, it is possible to abstract the environment where humans/robots can circulate to a graph in which the nodes/vertexes represent either the pick-up/delivery locations (the machines where the wafers are processed) or the interconnection nodes. The arcs/edges indicate the existence of a connection between the different locations of interest. The weights of these arcs represent the distances between the vertexes that the edges connect. Therefore, it is possible to define a digraph $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$ where: $\mathcal{Q}$ are vertices, Δ are edges and ω are the weights of the edges.

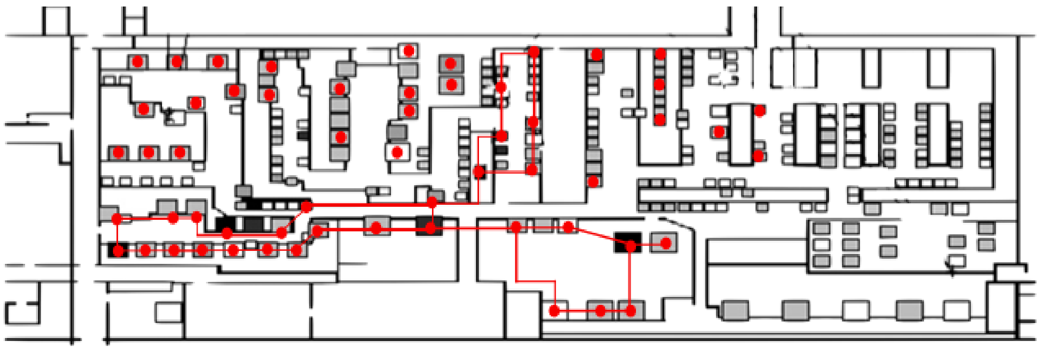


Figure 2.1: Abstraction of the graph, plan taken from [5]

2.3.1 Scenario

In this chapter, a semiconductor production site containing a set of machines ($\mathcal{M}$, each of which equipped with a buffer in and a buffer out of respectively size bin_m and $bout_m$), a charging station ($\mathcal{C}$), Transfer in (T_{IN}) and Transfer out (T_{OUT}) points are considered. Beside the above mentioned points of interest, a set of auxiliary nodes $\mathcal{I}$ is considered, which allows to discretize the routes available to the mobile robots for dispatching operations. Nodes in set $\mathcal{I}$ are the result of a space discretization performed with a fix discretization step. This is chosen in a way consistent with the dimension of the mobile robots and fine enough to avoid collision and maximize the exploitation of the available space by the fleet of vehicles.

Finally, the production site can be abstracted as a direct connected graph $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$, where $\mathcal{Q} = \mathcal{M} \cup \mathcal{C} \cup T_{IN} \cup T_{OUT} \cup \mathcal{I}$ is the set of nodes, $\Delta = \mathcal{Q} \times \mathcal{Q}$ the set of edges and ω the edge weights representing the travel time between nodes. Given $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$, the objective is to coordinate a team of robots $\mathcal{V}$ on the graph so to fulfil the required dispatching in a minimum time.

Orders enter the production area from the Transfer in points T_{IN}. From there, they need to be transported to a set of predefined machines for elaboration. When the process is completed, the orders leave the site through the Transfer out points T_{OUT}. Let $\mathcal{D}(t)$ be the set of orders to be fulfilled at time t (i.e. orders present at the Transfer in or at the buffer out of one of the machines). The i-th order (d_i) is represented by a state a vector containing the following information: number of order (d_i^n), pick-up position (d_i^p) and delivery position (d_i^d).

$$d_i \;=\; [\; d_i^n \mid d_i^p \mid d_i^d \;] \tag{2.6}$$

2.4 1 vehicle per order

2.4.1 Problem Definition

Given the graph $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$ and a transport demand $d_i \in \mathcal{D}(t)$, the purpose of the following approach is to find the shortest path which allows to fulfil the transport demand d_i.

Problem 2.4.1. Given an environment $\mathcal{G}$, a transport demand d_i, i.e. the order that that must be transported from the position $d_i^p \in \mathcal{Q}$ to position $d_i^d \in \mathcal{Q}$, a robot v located in $q_v \in \mathcal{Q}$ finds the shortest path that starts from q_v, goes through to d_i^p and ends in d_i^d.

2.4.2 Solution

A possible solution to Problem 2.4.1 can be obtained by using DP with a finite horizon N. The finite horizon is computed as the maximum time required to achieve the required task, which is to reach the target node (q_t). The solution applies Algorithm 1 twice in order to find first the path from q_v to d_i^p and then the path from d_i^p to d_i^d. By considering the graph nodes as states ($\mathcal{X} = \mathcal{Q}$) and the graph arcs as control action ($C = \Delta$), it is possible to reformulate the problem within the framework of the DP basic problem and thus to apply Algorithm 1.
Let $J_k(x_i)$ be the optimal cost to reach the target node (i.e. x_t) in $N - k$ steps.

Then, the k-th tail sub-problem can be formulated as:

$$J_k(x_i) = \min_{u_k \in U(x_i)} \{g_k(x_i, u_k) + J_{k+1}(x_j)\} \quad k = N-1, N-2, \cdots, 0 \tag{2.7}$$

where: $U(x_i) \subseteq C$ is the set of the all possible control actions to apply from the state x_i (i.e. all the arcs of the graph that start from the state x_i), $u_k \in U(x_i)$ is a possible control action which corresponds to an arc from state x_i to the possible next state x_j (i.e. $u_k = (x_i, x_j)$), $g_k(x_i, u_k)$ is the stage cost which is defined as follows:

$$g_k(x_i, u_k) = \begin{cases} \omega(x_i, x_j), & \text{if } (x_i, x_j) \in \Delta \\ \infty, & \text{otherwise} \end{cases} \tag{2.8}$$

In order for the minimum path to be obtained, it is necessary to set the final cost as follows:

$$g_N(x_i) = \begin{cases} 0, & \text{if } x_i = x_t \\ \omega(x_i, x_t), & \text{otherwise} \end{cases} \tag{2.9}$$

As can be seen from Equation 2.9, the final cost $g_N(x_N)$ is set equal to 0 only for the target node, otherwise it is set equal to the cost of the transition to reach the target node.

Once the final cost is set, the optimal control policy $\pi^* = \{u_0, \cdots, u_{N-1}\}$ can be obtained by using Algorithm 1.

Suppose that a vehicle is in a node q_v and there is an order that must be picked up from the node d_i^p and must be released to the node d_i^d. To find the optimal path, it is necessary to solve two problems. The first problem aims to find the shortest path π_1^*, applying DP by setting as the target node $x_t = d_i^p$, starting from the starting node $x_0 = q_v$. Then, the second shortest path π_2^* is solved by setting as the target node $x_t = d_i^d$ and as the starting node $x_0 = d_i^p$. Finally, the shortest path which allows to satisfy the transport request d_i is given by the union of two paths found, i.e. $\pi^* = \pi_1^* \cup \pi_2^*$.

2.5 n vehicles per n orders

2.5.1 Problem Definition

Given the graph $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$, and a set of transport demands $\mathcal{D}$ that contains n transport requests, i.e. $|\mathcal{D}| = n$, the set of vehicles $\mathcal{V}$ which is composed by n vehicles, i.e. $|\mathcal{V}| = n$, the approach proposed in this section aims both to find the optimal vehicle assignment for each order and, at the same time, the optimal path for each vehicle. This is to minimize the overall transport time, which is given by the sum of the n transport times associated with the n vehicles. Note that in this approach it is assumed that each vehicle can only fulfill one order.

Problem 2.5.1. Given an environment $\mathcal{G}$, the set of n-transport demands $\mathcal{D}$, the set of the n-robots $\mathcal{V}$, find the optimal shortest paths for the n-vehicles $v \in \mathcal{V}$ that satisfy the n-transport demands $\mathcal{D}$ with the minimum overall time.

2.5.2 Solution

In order to consider the case of multiple orders and multiple vehicles, the previous approach needs to be modified. Indeed, in case of a single vehicle, the digraph is sufficient to find the shortest route, while in case of n vehicle, it is necessary to consider all the possible combinations of the possible positions of the different vehicles at any steps. In addition, to find a correct solution it is necessary to prevent both vehicles from being in the same vertex of the graph at the same time. Moreover, an order cannot be executed by more than a vehicle. To achieve this, in this approach a vector product graph is adopted, in which all the nodes (combinations) where the above mentioned anomalies occur are eliminated.

Definition 2.5.1 (Vector product Graph). Given n graph $\mathcal{G}_1, \dots, \mathcal{G}_n$, the vector product graph is a tuple $\mathcal{G}^n = \bigtimes_{v=1}^{n} \mathcal{G}_v = (\bar{\mathcal{Q}}^n, \Delta^n, \omega^n)$, where $\bar{\mathcal{Q}}^n = \mathcal{Q}^n \setminus \{(q_1, \dots, q_n) \mid \exists q_i = q_j, i \neq j\}$ is the set of joint states, $\Delta^n \subseteq \bar{\mathcal{Q}}^n \times \bar{\mathcal{Q}}^n$ is the set of joint transitions. A transition $(x, x') \in \Delta^m$ if $q_v \to_{\mathcal{G}_v} q'_v \in \Delta$, $\forall v \in \{1, \dots, m\}$, where $x = (q_1, \dots, q_m)$ and $x' = (q'_1, \dots, q'_m)$. The joint transition weights are given by $\omega^n(x, x') = \sum_{v=1}^{n} \omega(q_v, q'_v)$.

After having generated the vector product graph that contains all possible positions that the vehicles can assume while respecting the constraints, Algorithm 1 is applied. Similarly to the previous case, a 0-cost is given to the combination where all the vehicles are in the destination positions (i.e. all vehicles are in pick up positions or all vehicles are in delivery positions)

$$g_N(x_i) = \begin{cases} 0, & \text{if } x_i \in X_t \\ \omega^n(x_i, x_t), & \text{if } \exists x_t \in X_t \text{ such that } (x_i, x_t) \in \Delta^n \\ \infty, & \text{otherwise} \end{cases}$$

where: X_t are the set of nodes in which all the vehicles are in destination positions. Successively, the dynamic programming is applied from this position backwards. Note that two different dynamic programming need to be solved. The first DP allows to attain the pick-up positions starting from the current vehicles positions optimally, i.e π_1^*, in which the set X_t is composed of all the graph nodes where the vehicles are at pick-up positions; the second allows to achieve optimally the delivery positions starting from the pick-up position reached by π_1* , i.e π_2^*. In this case, the set X_t is composed of a single element, which is the node where each order is satisfied by the vehicle that picked it up. The complete procedure is summarized in Algorithm 2.

```
Algorithm 2: DP $n$ vehicles per $n$ requests
  Input  : $\mathcal{G}$ - the environment
  Input  : $\mathcal{D}$ - the set of demands to be satisfied
  Input  : $\mathcal{V}$ - the set of vehicles
  Output : the optimal path for each robot $v \in \mathcal{V}$
1 Construct the vector product graph $\mathcal{G}^n$ as defined in Definition 2.5.1.
2 Compute the set $X_t$ of all the vector product graph nodes where the vehicles
  are in pick-up positions.
3 Find the shortest path $\pi_1^*$ from the vehicle initial condition to the terminal
  state $X_t$ of $\mathcal{G}^n$ using Algorithm 1.
4 Compute the set $X_t$ of the single vector product graph node where the
  vehicles are in delivery position in accordance with $\pi_1^*$.
5 Find the shortest path $\pi_2^*$ from the vehicle pick-up position reached at the
  previous step to the terminal state $X_t$ of $\mathcal{G}^n$ using Algorithm 1.
```

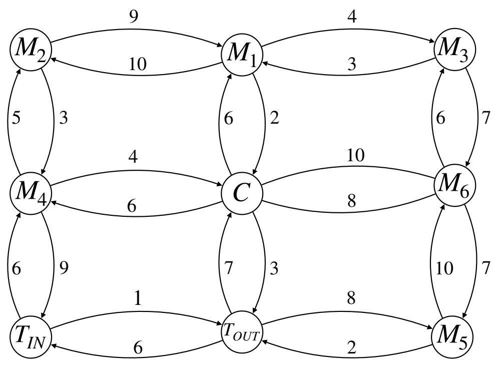


Figure 2.2: Graph $\mathcal{G}$

2.5.3 Example

The following is an example of optimization of pick-up and delivery of 3 customers by 3 vehicles using the described approach. In particular, at the initial time the three vehicles are located at nodes M_2, M_1, C. Transport requests are from node T_{IN} to node M_3, node M_4 to node T_{OUT}, and node M_5 to node M_6, as reported in Table 2.1.

Applying Algorithm 2, the shortest paths for the three vehicles are obtained, which

$\mathcal{D}$		
Number of order	Pickup Point	Delivery Point
1-st order	T_{IN}	M_3
2-nd order	M_4	T_{OUT}
3-rd order	M_5	M_6

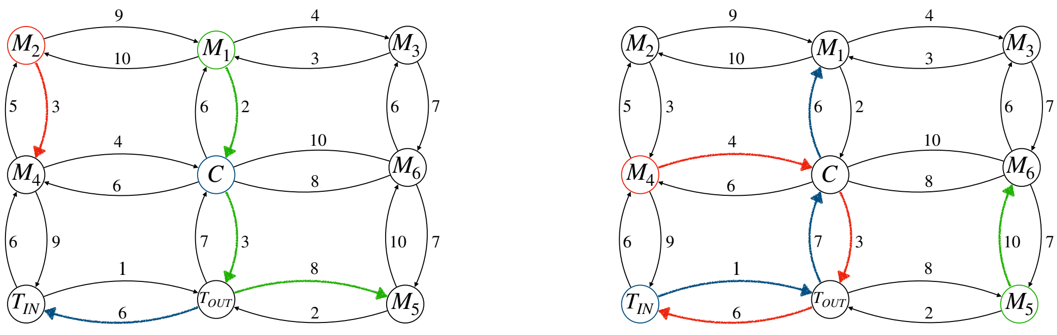
Table 2.1: Transport requests list

are summarized in the following Table 2.2 and they are shown in Figures 2.3.

Table 2.2: The shortest paths for v_1, v_2 and v_3

$\pi_1^* \cup \pi_2^*$								
v_1	M_2	M_4	M_4	M_4	C	T_{OUT}	T_{OUT}	T_{OUT}
v_2	M_1	C	T_{OUT}	M_5	M_6	M_6	M_6	M_6
v_3	C	T_{OUT}	T_{IN}	T_{IN}	T_{OUT}	C	M_1	M_3

While the described method is interesting, it still has the following issues:



(a) Pickup paths π_1^* (b) Delivery paths π_2^*

Figure 2.3: Paths on graph

- The complexity is still NP hard;
- It is not possible to use vehicles with a capacity greater than one, since the approach is restricted to the case of n vehicles per n orders;
- The priority of orders is not taken into account.

2.6 m vehicles per n orders

In this section, starting from the approach proposed in [98], which deals with the case where a single vehicle with multiple capacity is used for fulfilling multiple demands, an extension to the case of several vehicles is presented.

2.6.1 DP with Multiple Orders and 1 Vehicle

Before introducing the extension of several vehicles, in this section the approach proposed in [98] is summarized.
Assume that a list of orders $\mathcal{D}$ is given, where the i-th order (d_i) holds the i-th position (First Called First Served (FCFS) list). Suppose that at time $t = 0$, the position of the vehicle is known (q_v). Let d_i^p and d_i^d be the pick up position and the delivery position of i-th order, respectively. Moreover, for any pick-up and delivery positions, the optimal path (i.e. *path*) on the graph and its cost (i.e. *time*) have been precomputed (using e.g. Dijkstra). The algorithm proposed in [98] aims to find a vehicle path starting from q_v and ending in one of the delivery points that

minimize an objective function composed of a weighted combination of the time to serve all customers and of the total degree of their “dissatisfaction” waiting to be served

$$w_1 \cdot \sum_{j=1}^{2\cdot|\mathcal{D}|} path_j + w_2 \cdot \sum_{i=1}^{|\mathcal{D}|} [\alpha \cdot WT_i + (2-\alpha) \cdot RT_i] \tag{2.10}$$

where: i) w_1 and w_2 are given weights, ii) α is a customer's time preference constant $0 \leq \alpha \leq 2$, iii) $path_j$ is the duration of the j-th leg of the route, iv) WT_i is the waiting time of order i from $t = 0$ until its time of pick-up, v) RT_i is the riding time from its time of pick-up until its time of delivery, vi)$|\mathcal{D}|$ is the number of orders. The problem is subject to the following constraints:

- *vehicle capacity*: vehicles have a fixed given capacity v_c and thus cannot transport more than v_c orders at a time;
- *consistency*: the optimal route has to ensure that each order is first picked-up and then delivered;
- *priority constraint:* the path has to meet the priority constraint MPS (Maximum Position Shift). Let $z_{d_i}^{in}$ be the position of order d_i in the sequence of pickups and $z_{d_i}^{out}$ the d_i position in the sequence of delivery. The MPS constraint means that, in a valid dispatching sequence, an order d_i can be anticipated or delayed at most of MPS steps, i.e.

$$\begin{aligned} |i - z_{d_i}^{in}| \leq \text{MPS} \quad \forall i = 1, \cdots, |\mathcal{D}| \\ |i - z_{d_i}^{out}| \leq \text{MPS} \quad \forall i = 1, \cdots, |\mathcal{D}| \end{aligned} \tag{2.11}$$

The authors in [98] propose a solution which relies on state vectors for taking into account both the history of the path (addressed orders, available capacity, current position, etc.) and the constraints.

Let x be a state vector. Suppose to have $|\mathcal{D}|$ orders, the k-th state vector looks as follows:

$$x_k \; = \; [\; L_k, \; d_1\text{-condition}, \cdots, \; d_{|\mathcal{D}|}\text{-condition} \;]$$

where:

- L is the current position of vehicle $L \in \left\{d_1^p, \cdots, d_{|\mathcal{D}|}^p, d_1^d, \cdots, d_{|\mathcal{D}|}^d, q_v\right\}$
- d_i-condition represents the state of the order i, where the condition can be:
  - In Transport
  - Attending
  - Delivered

The procedure for generating the feasible set of state vectors (i.e. the set $\mathcal{X}$ where each element $x_i \in \mathcal{X}$ respects all the constraints) can be summarized as follows: i) Generate a list containing all the possible state vectors, i.e. $\mathcal{S}_{all}$; ii) Verify the satisfaction of the previously described constraints for each state vector. If at least one constraint is not respected, the state has to be deleted; iii) If a vector state does not have a successor which is feasible, then such state also has to be deleted. This holds unless the vector state is a "terminal state", which means a state where all the orders have been executed.
Once the algorithm finds a list of vectors that respect all the constraints, i.e. $\mathcal{X}$, they use DP on $\mathcal{X}$ in order to find the optimal sequence of pickups and deliveries. Due to possible presence of multiple terminal states, in order to apply Algorithm 1, it is necessary to first add a dummy state x_{dummy} to the set of state, i.e. $x_{dummy} \cup \mathcal{X}$, and then to create a dummy edge from each terminal state (i.e x_{N_i}) to x_{dummy} with a transition cost equal to $g_N(x_{N_i})$.
Let $J(x_k)$ be the optimal value of all subsequent decisions from x_k to the dummy state. The problem of finding the optimal path to reach a terminal state from the state x_k can be formulated as follows:

$$J(x_k) = \min_{L_{k+1} \in \mathcal{X}_{next}(x_k)} time(L_k, L_{k+1}) \cdot M + J(x_{k+1}) \tag{2.12}$$

where:

- $\mathcal{X}_{next}(x_k)$ is the set of the next possible position which is given by the union of the two following sets

  - $\mathcal{X}_{pick}(x_k) = \{d_i^p : 1 \leq i \leq |\mathcal{D}|$ with d_i condition = "Attending"$\}$
  - $\mathcal{X}_{del}(x_k) = \left\{d_i^d : N+1 \leq i \leq 2 \cdot |\mathcal{D}|\right.$ with d_i condition = "In transport"$\left.\right\}$

- $time(L_k, L_{k+1})$ is the precomputed time for reaching the position L_{k+1} starting from the position L_k
- M is the factor of proportionality which is defined as $M = w_1 + w_2 \{\alpha \cdot |\mathcal{X}_{pick}(x_k)| + (2-\alpha) \cdot |\mathcal{X}_{del}(x_k)|\}$

The optimal path π^* (i.e. the sequence of pickups and deliveries), which minimizes the cost function shown in Equation 2.10, is obtained by using Algorithm 1, where the target node is the dummy node $x_t = x_{dummy}$ and the terminal cost $g_N(x_N)$ is set equal to: i) 0 for the dummy state, i.e. $x_N = x_{dummy}$ ii) finite value for the vector states able to reach the dummy state in one step. iii) ∞ for the other vectors. The procedure is summarized in Algorithm 3.

Algorithm 3: Dynamic Programming 1 vehicle to multiple transport demand

```
Input  : $\mathcal{D}$ - the set of transport demands
Input  : $v_c$ - vehicle capacity
Input  : $q_v$ - vehicle position
Output : $\pi^*$ - optimal path composed of the sequences of pickups and
         deliveries
1 Generate the set of all possible states according to $\mathcal{D}$, i.e $\mathcal{S}_{all}$
2 Apply on $\mathcal{S}_{all}$ the constraints in order to obtain the set of feasible states, i.e.
  $\mathcal{S} \subseteq \mathcal{S}_{all}$
3 Add the dummy node and the dummy edges
4 Find $\pi^*$ by using Algorithm 1
```

2.6.2 Extension to the multiple-vehicles case

While the previous approach provides the optimal solution, it is restricted to the case of one vehicle only. In this section, the approach proposed in [98] is extended to the case of multiple vehicles. To cope with the increase of complexity, the new approach relies on genetic algorithm.

Problem 2.6.1. Given $\mathcal{D}$ (i.e. the FCFS list) consisting of $|\mathcal{D}|$ orders and m vehicles, the objective is to find the optimal assignment of the $|\mathcal{D}|$ orders (of the FCFS list) among the m vehicles which minimizes the objective function given by sum of m vehicle cost functions (fitness function)

$$\sum_{v=1}^{m} \left\{ w_1 \cdot \sum_{j=1}^{2\cdot|\mathcal{D}_v|} time_j + w_2 \cdot \sum_{i=1}^{|\mathcal{D}_v|} [\alpha \cdot WT_i + (2-\alpha) \cdot RT_i] \right\} \tag{2.13}$$

where: $\mathcal{D}_v \subseteq \mathcal{D}$ is the set of demands assigned to the vehicle v

Since finding an exact solution for this type of problem in a reasonable time is computationally too expensive when multiple vehicles are considered, an approximate method which provides a suboptimal solution in a compatible time with the order execution time is used for solving Problem 2.13. In particular, a genetic algorithm is considered. This type of algorithm ensures to find a suboptimal solution in a reasonable time. The idea is to let the genetic algorithm create possible assignments to which the approach of the previous Section 2.6.1 is then applied as fitness function.
To achieve this, it is necessary to set the $n_{\mathcal{Z}}$ chromosomes that compose the first population. A generic chromosome is encoded with $|\mathcal{D}|$ (number of transport demands) genes, in which each gene can assume values between 1 and m (number of vehicles). Then, it is possible to apply the genetic algorithm. This is an adaptive algorithm whose goal is to solve large global search problems. These types of algorithms are conceptually based on the principles governing the natural evolution of the species. Algorithm 4 shows the complete procedure in order to solve Problem 2.6.1.
The first operation of Algorithm 4 is the random generation of the first population of solutions/chromosomes $\mathcal{Z}$ which is composed of $n_{\mathcal{Z}}$ chromosomes, where:

- Each chromosome j is encoded with $|\mathcal{D}|$ genes, where $|\mathcal{D}|$ is the number of orders in the FCFS list (i.e. $\mathcal{D}$).
- Each i-th gene can assume values between 1 and m, where m is the number of vehicles.

$$\text{chromosome}_j = [\text{1st gene } | \text{2nd gene } | \cdots | \mathcal{D}\text{-th gene }]$$

Then, Algorithm 4 evaluates each chromosome by applying the fitness function (i.e. solving the m subproblem using Algorithm 3) and arranges the population according to the decreasing values of the fitness function. After that, Algorithm 4 finds the probability associated with the individual, which will be used for selection in cross-over processes. This is calculated as follows:

$$p_j = P_A + (P_B - P_A)\frac{j-1}{n_{\mathcal{Z}} - 1} \tag{2.14}$$

where:

- j is the position of the chromosome in the list made in the previous point
- P_A is equal to 0
- P_B is equal to $\frac{2}{\mathcal{Z}}$

Consequently, the j-th individual can then be associated with the j-th following cumulative probability:

$$P_j = \sum_{l=1}^{j} p_l \tag{2.15}$$

This will make the chromosomes with lower fitness function value more likely to be chosen in the crossover process. Then, depending on the probability set for both the crossover process (p_c typically close to 1) and the mutation process (p_m typically near to 0), chromosomes are chosen. The cross-over process generates new chromosomes by using the cross-over uniform technique. Each new chromosome inherits genes from the two chromosomes selected as “parents”. As concerns the mutation process, genes change in a random manner. A new population $\mathcal{Z}^{new}(t)$ is first obtained from the cross-over and mutation processes and then evaluated by using the fitness function. The last step of Algorithm 4 is to find the best $n_{\mathcal{Z}}$ chromosomes from the union of the two populations $\mathcal{Z}(t) \cup \mathcal{Z}^{new}(t)$, which will be used as the new population for the next iteration. The entire procedure is repeated until the algorithm converge to a suboptimal solution.
The iteration of the steps presented allows to find an optimized solution to the

considered problem. Since some good solutions might be lost during the course of evolution, and the evolution evolution may fall in to local optimum, the approach is often integrated with the techniques of “elitism” and with that of random mutations. The elitism is obtained by selecting only the best $n_{\mathcal{Z}}$ chromosomes in the list from the union of the initial population and the chromosomes obtained from the cross-over and mutation processes, i.e. $\mathcal{Z}(t) \cup \mathcal{Z}^{new}(t)$. Moreover, in order to avoid getting stuck in local solutions, it is necessary to introduce occasional random mutations, which allow to look at other solutions.

Algorithm 4: DP and GA multiple vehicles per multiple demand

```
Input  : $\mathcal{D}$ - the set of transport demands
Input  : $\mathcal{V}$ - the set of vehicles
Output : $\pi$ - sub-optimal sequences of pickups and deliveries for each vehicle
         v
1  k = 0
2  Set $n_{\mathcal{Z}}$
3  Generate the first population of chromosomes composed of $n_{\mathcal{Z}}$ chromosomes
   according to both $\mathcal{D}$ and $\mathcal{V}$
4  while True do
5      Evaluate each chromosome that belongs to $\mathcal{Z}(k)$ by using Algorithm 3
6      if stopping criteria then
7          break
8      Arrange the population $\mathcal{Z}(k)$ according to the decreasing values of the
       fitness function
9      Calculate the probability $p_j$ associated with the $j$-th individual for the
       cross-over process
10     Generate a new population $\mathcal{Z}^{new}(k)$ by using the Cross-over and
       Mutation processes
11     Evaluate each chromosome that belongs to $\mathcal{Z}^{new}(k)$ by using Algorithm 3
12     Generate $\mathcal{Z}(k+1)$ by selecting the best $n_{\mathcal{Z}}$ chromosomes from
       $\mathcal{Z}^{new}(k) \cup \mathcal{Z}^{best}(k)$
13     k= k+1
```

Example

Suppose that at time t, three single-capacity vehicles ($m = 3$) have to satisfy four demands of transport $|\mathcal{D}| = 4$ with MPS =2. The vehicles at time t are situated at node T_{IN}, M_2 and M_3. The list of demands $\mathcal{D}$ is shown in Table 2.3, while the environment graph $\mathcal{G}$ on which the solution is calculated is shown in Figure 2.4.

Since the genetic algorithm does not guarantee the attainment of an optimal

$\mathcal{D}$		
Number of order	Pickup Point	Delivery Point
1-st order	M_1	M_7
2-nd order	M_5	M_9
3-rd order	M_4	T_{OUT}
4-th order	M_2	M_3

Table 2.3: FCFS list

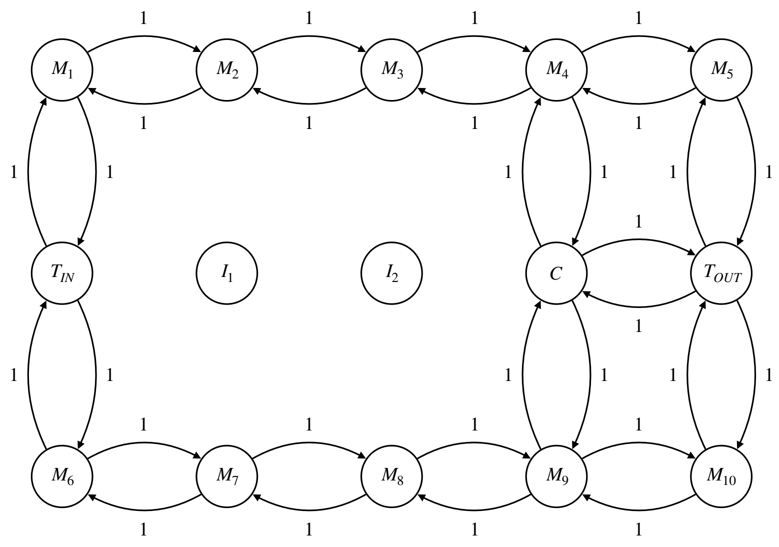


Figure 2.4: The environment $\mathcal{G}$

solution, only a suboptimal solution can be obtained. In this example, where three vehicles are considered, the genetic algorithm chooses to use only two of the three vehicles available to meet all transport requirements. In particular, the first vehicle satisfies only the first call and the second vehicle satisfies the rest. Table 2.4 reports the results of the approach based on the genetic algorithm.

Vehicle 1	Vehicle 2	Vehicle 3
1-st order	4-th order	Not use
	3-rd order	
	2-nd order	

Table 2.4: Suboptimal sequences π_{v1}, π_{v2} and π_{v3} obtained by using Algorithm 4

Table 2.5: π_{v_1}, π_{v_2} and π_{v_3}

k	1	2	3	4	5	6	7	8
v_1	M_1	T_{IN}	M_6	M_7	0	0	0	0
v_2	M_3	M_4	M_5	T_{OUT}	M_5	T_{OUT}	M_{10}	M_9
v_3	0	0	0	0	0	0	0	0

2.7 DP on Dynamic Graph

In the previous Section 2.6, an approach able to solve the dispatching problem of n orders with m vehicles is proposed. However, this is implies "wasting" some time in order to collect the requests of transport in a chronological way. Besides, collision avoidance cannot be guaranteed since the approach uses the pre-calculated shortest paths between the points of interest (i.e. pick-up points and delivery points) in the optimization for finding the optimal sequence of pickup and delivery positions.
For this purpose, in this section, the proposed approach takes into account the paths of the moving vehicles in the vehicle assignment to ensure collision avoidance. Moreover, as introduced in Section 2.3.1, the robot motions are captured by the environment graph $\mathcal{G}$, which is obtained through space discretization performed with a fixed discretization step. This implies that the vehicle paths found through the optimizations are obtained under the assumption that the vehicles always travel at the same fixed speed $\bar{v}_{robot}$. However, this capability of the vehicles depends on the charge stored in their batteries. Indeed, a vehicle is able to move at a given speed up to a certain battery charge level, after which the vehicle will move at reduced speed until it will stops because of its insufficient battery charge level. Thus, to guarantee both collision avoidance and the correct operation of the manufacturing site, it is of crucial importance to take into account the battery charge of the vehicles during the paths optimization. This can be achieved by using a battery model which allows to describe the battery behaviour.
However, the quality of the battery model adopted plays an important role. Indeed, using a "simple" battery model it is likely to overestimate or underestimate the charge of the battery, with dangerous consequences for the production. If the model

overestimates the battery charge, the robot might not have enough battery to fulfill the order moving at a constant speed. In best case scenario, this situation might lead to slow down the robot speed, while in the worst case scenario it might result in stopping the vehicles in the middle of the production site or in possible vehicle collisions. In addition, a complete discharge of the battery could reduce its useful life. Otherwise, if the battery charge is underestimated, the vehicle might be considered having a low battery level when it does not, leading to a worse exploitation of the vehicle fleet. For these reasons, in this section a viable dispatching strategy is proposed, which adopts a non-linear battery model for evaluating the vehicle assignment and ensuring the correct operation of the vehicle fleet.

2.7.1 Scenario

The scenario adopted for this approach is based on the scenario proposed in Section 2.3.1, to which the following characteristics are added. Robots have limited battery capacity. Besides, their energy consumption depends on the operation status: in transit, stand-by, recharging, as shown in Section 2.7.3. When the battery level goes below a certain threshold, robots are required to recharge at the charging station ($\mathcal{C}$).

Moreover, it is assumed that each robot can transport only one order at a time. Vehicles can be either available (ready to fulfil an order) or not available (busy or recharging/going to recharge). At each time instant t, each robot v comes with a vector state containing the following information:

1. $v.state$ is the vehicle condition, i.e. in transit, stand-by, recharging;
2. $v.bat$ is the the battery status vector, i.e. $\mathbf{x_{bat}}$, where $v.bat.soc = SOC$, $v.bat.v_1 = V_1$ and $v.bat.v_2 = V_2$;
3. $v.position$ is the current location;
4. $v.orders$ are the assigned orders;
5. $v.path$ are the assigned routes.

2.7.2 Order Priority

Orders must be fulfilled according to their priorities. Top priority is given to orders at buffers out which need to be transported to other machines or to the Transfer out. If several orders of this kind exist, higher priority is given according to the age of the order. Orders at the Transit in are given a lower priority. Also in this case, orders are served according to their age.

2.7.3 Battery Model

The battery behaviour is described by using the Double Polarization model which is proposed in [99]. The continuous time, non-linear Equivalent Circuit Model (ECM) used is shown in Fig 2.5. As can be noticed, the ECM is composed of i) the internal resistance R_0; ii) the polarization resistances R_1 and R_2; iii) the effective capacitances C_1 and C_2; iv) the open circuit potential v_{ocp}.

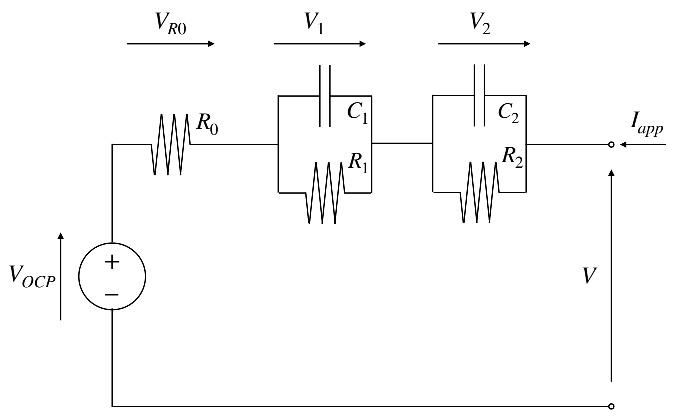


Figure 2.5: The ECM of the DP model

The relations between voltage at the terminals of the battery pack $V(t)$, the applied current $I_{app}(t)$ and the state of charge $SOC(t)$ are shown in Figure 2.5, whose equations are:

$$\begin{cases} \dot{x}_1(t) & = \frac{u(t)}{3600 \cdot Q_{tot}} \\ \dot{x}_2(t) & = \frac{x_2(t)}{R_1 C_1} + \frac{u(t)}{C_1} \\ \dot{x}_3(t) & = \frac{x_3(t)}{R_2 C_2} + \frac{u(t)}{C_2} \\ y_1(t) & = v_{ocp}(x_1(t)) + x_2(t) + x_3(t) + R_0 u(t) \\ y_2(t) & = x_1(t) \end{cases} \tag{2.16}$$

where:

$$\mathbf{x_{bat}}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} SOC(t) \\ V_1(t) \\ V_3(t) \end{bmatrix}, \qquad \mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} V(t) \\ SOC(t) \end{bmatrix},$$

$v_{ocp}(x_1(t))$ is a polynomial function of the SOC that must be identified experimentally, see Equation 2.17

$$v_{ocp}(SOC(t)) = \sum_{i=0}^{n} b_i \left(SOC(t)\right)^i \tag{2.17}$$

where b_i is the i-th polynomial parameter.

The SOC is the charge indicator and is measured in percentage, where the $SOC = 100\%$ if the battery is completely charged, while $SOC = 0\%$ if the battery is completely discharged. A problem in the online control of lithium ion batteries is the correct estimate of the SOC, as its direct measurement is not possible. For the different types of SOC estimates, see [13]. In this work, it is assumed that it is always possible to detect the initial value of the SOC.

Aiming to find the power that must be supplied by the battery to the electric vehicle motors for maintaining the speed constant, a study on the forces to which the vehicle is subjected during its motion is conducted in the following section.

Vehicle Model

The vehicle model adopted is inspired from [100]. The vehicles have three wheels. Two wheels are on the rear part and one is on the front part. The driven wheels are the rear wheels. Each wheel is powered by an electric motor. The force that

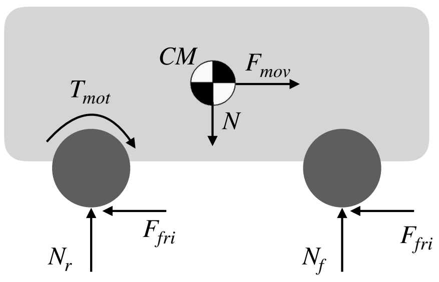


Figure 2.6: Applied forces on the vehicle

the electric motors have to apply in order to move the vehicle depends on different resistance forces, such as areo drag, weight, wheel friction etc. Therefore, in order to move the vehicle, electric motors have to overcome the resulting sum of all forces, see Equation 2.18

$$F_v = F_{mov} + F_{fri} + F_{drag} + F_{weight} \tag{2.18}$$

where: F_{mov} is the driving force; F_{fri} is the friction force; F_{drag} is the drag force; F_{weight} is the weight force.
For simplicity, in this work the following assumptions are made:

1. the vehicles are moving always at the same low constant speed, i.e. $\frac{d\bar{v}_{robot}(t)}{dt} = a_{robot} = 0$;

2. the vehicle paths are in a flat indoor track and the floor is regular;

3. the mass of the order m_o is negligible when compared to the vehicle mass m_r, i.e. $m_r >> m_o$.

Due to the first and the second assumptions, both drag and weight forces can be neglected. Therefore, the resulting force is given by the sum of the moving force

F_m and the friction force F_{fri}

$$F_v = F_m + F_f = m_v * a_{robot} + f * N \tag{2.19}$$

where m_v is the vehicle mass that includes also the mass of the battery; a_{robot} is the acceleration ($[\frac{m}{s^2}]$); f is the friction coefficient between wheels and floor; N is the normal force.
Once the force that the electric motors must apply to move the vehicle at a constant speed is obtained, see Equation 2.19, the electrical power required by the electric motors from the battery (P_b) is obtained as follows:

$$P_b(t) = \frac{1}{\eta_b \eta_m \eta_e} F_v(t) * \bar{v}_{robot}(t) \tag{2.20}$$

where: η_b is the efficiency of the battery; η_m is the efficiency of motor and η_e is the overall efficiency of the electric drive.
Note that, since the vehicles travel always at the same constant speed $\bar{v}_{robot}$, both $F_v(t)$ and $\bar{v}_{robot}(t)$, are constant and thus also the battery electric power $P_b(t)$ will be constant for each time instant t.
Since the total electric power load requested from the battery is constant, the following algebraic constraint is added to System 2.16

$$V(t) * I_{app}(t) = K \tag{2.21}$$

where K indicates the amount of power with which the battery is charged ($K > 0$) or discharged ($K < 0$). Thus, in the vehicle assignment, in order to evaluate the capability of the robot to move at a constant speed, the following battery initial value problem is solved:

$$\begin{cases} \dot{x}_1(t) & = \frac{u(t)}{3600 \cdot Q_{tot}} \qquad \mathbf{x_{bat}}(t_0) = \mathbf{x}_0 \\ \dot{x}_2(t) & = \frac{x_2(t)}{R_1 C_1} + \frac{u(t)}{C_1} \\ \dot{x}_3(t) & = \frac{x_3(t)}{R_2 C_2} + \frac{u(t)}{C_2} \\ y_1 & = v_{ocp}(x_1(t)) + x_2(t) + x_3(t) + R_0 u(t) \\ y_2 & = x_1(t) \\ 0 & = (v_{ocp}(x_1(t)) + x_2(t) + R_0 u(t))\, I_{app}(t) - K \end{cases} \tag{2.22}$$

where $\mathbf{x}_0$ is the initial condition at time t_0.

Note that, since the system dynamics is discretized, also the continuous time System 2.22 will be discretized, according to the fixed discretization step used to discretize the routes. Thus, the applied input $\mathbf{u_{bat}}$ will be a piece-wise constant with sampling time equal to the fix discretization step.

Let $\mathbb{T}_{ch,v}$ be the time required for a full battery charge of vehicle v from the battery condition $\mathbf{x_{bat}}(t)$ at time t. Then, $\mathbb{T}_{ch,v}$ is obtained by evolving the system from the initial condition $\mathbf{x_{bat}}(t)$ until the battery state of charge reaches its maximum allowed value. Algorithm 5 allows to compute a single step evolution (i.e. $\mathbf{x_{bat}}(t+1)$) of System 2.22 under the power algebraic constraint starting from the battery initial condition $\mathbf{x_{bat}}(t)$.

```
Algorithm 5: Single step evolution of the DP battery - evolutionBattery($\mathbf{x_0}$,K)
   Input  : $\mathbf{x_0}$ - battery initial condition
   Input  : $K$ - electric power required/absorbed by the battery
   Output : $\mathbf{x_{bat}}(t+1)$ - battery state at $t+1$
 1 Evolve the System 2.22 from the initial condition $\mathbf{x_0}$ for one step, with $K$
   constant.
 2 $\begin{cases} \dot{x}_1(t) = \frac{u(t)}{3600 \cdot Q_{tot}} \quad \mathbf{x_{bat}}(t_0) = \mathbf{x}_0 \\ \dot{x}_2(t) = \frac{x_2(t)}{R_1 C_1} + \frac{u(t)}{C_1} \\ \dot{x}_3(t) = \frac{x_3(t)}{R_2 C_2} + \frac{u(t)}{C_2} \\ y_1 = v_{ocp}(x_1(t)) + x_2(t) + x_3(t) + R_0 u(t) \\ y_2 = x_1(t) \\ 0 = (v_{ocp}(x_1(t)) + x_2(t) + R_0 u(t))\, I_{app}(t) - K \end{cases}$
```

2.7.4 Problem Formulations

In this section, different formulations for the same dispatching problem are proposed in order to identify which formulation allows to process the largest number of orders at the same time.

First, a general formulation is provided where the assignment of the order to the available vehicle is made only taking into account the total time needed by the available vehicles to satisfy the order. This formulation is considered as the baseline formulation that this work aims to improve in terms of number of orders elaborated during in the same time span.

In the second formulation, in order to minimize the overall time of service, a task can be assigned for a future time to a currently busy vehicle rather than to a currently available one. This can happen when it is faster to fulfil the current order and pick-up the new one, rather than having a currently available vehicle fulfilling the new order.

The last formulation tries to improve also the exploitation of the vehicle fleet by taking into account the time needed to recharge the vehicle battery.

Note that in all the formulations a vehicle can be assigned to an order only if able to fulfill the request and go back to the charging station with the available charge. Moreover, in order to find a viable solution to the scenario described above, in all formulations only one order at a time is optimized. The order with the highest priority still to be served will be assigned to the vehicle which minimizes its delivery time.

Let $d_i(t)$ be the order with the highest priority at time t. Define $d_i^p(t)$ and $d_i^d(t)$ the pickup and delivery positions of order $d_i(t)$. Let $\mathcal{V}_{av}(t) \subseteq \mathcal{V}$ be the subset of available vehicles at time t, i.e the vehicles that are neither in delivery nor in charging at time t.

Case 0

For each $v \in \mathcal{V}_{av}(t)$, indicate with q_v, the position of the vehicle v at time t. Denote with $\mathbb{T}_v(t, q_v, t, d_i^p(t), d_i^d(t))$ the shortest time required by vehicle v to reach $d_i^p(t)$ and then deliver the order to $d_i^d(t)$ starting from q_v at time t. The overall time for service can be obtained as follows:

$$\mathbb{T}_{p,v} = \mathbb{T}_v(t, q_v, t, q_v, d_i^p(t)) \tag{2.23}$$

$$\mathbb{T}_{d,v} = \mathbb{T}_v(\mathbb{T}_{p,v}, d_i^p(t), \mathbb{T}_{p,v}, d_i^p(t), d_i^d(t)) \tag{2.24}$$

$$\mathbb{T}_v(t, q_v, t, d_i^p(t), d_i^d(t)) = \mathbb{T}_{p,v} + \mathbb{T}_{d,v} \tag{2.25}$$

$\mathbb{T}_{p,v}$ and $\mathbb{T}_{d,v}$ are obtained using dynamic programming on the graph $\mathcal{G}$ by taking into account the moving vehicles. The problem can be summarized as follows.

Problem 2.7.1. At time t, considering an environment $\mathcal{G}$, the order with the highest priority $d_i(t)$ and a set of available vehicles $\mathcal{V}_{av}(t)$, the suboptimal vehicle assignment is given by finding which vehicle (v^*) guarantees the fastest delivery, i.e. the one with the lowest $\mathbb{T}_v$

$$v^* = \arg\min_{v \in \mathcal{V}_{av}(t)} \mathbb{T}_v \tag{2.26}$$

Case 1

For each $v \in \mathcal{V}$, indicate with q_v^*, the position that vehicle v will have when available for fulfilling a new order and $\mathbb{T}_{av,v}(t)$ ($\mathbb{T}_{av,v}(t) \geq t$) the corresponding time. Denote with $\mathbb{T}_v(t, q_v^*, \mathbb{T}_{av,v}(t), d_i^p(t), d_i^d(t))$ the shortest time required by vehicle v to reach $d_i^p(t)$ and then deliver the order to $d_i^d(t)$ starting from q_v^* at time $\mathbb{T}_{av,v}(t)$. The overall time for service can be obtained as follows:

$$\mathbb{T}_{p,v} = \mathbb{T}_v(\mathbb{T}_{av,v}(t), q_v^*, \mathbb{T}_{av,v}(t), q_v^*, d_i^p(t)) \tag{2.27}$$

$$\mathbb{T}_{d,v} = \mathbb{T}_v(\mathbb{T}_{p,v}, d_i^p(t), \mathbb{T}_{p,v}, d_i^p(t), d_i^d(t)) \tag{2.28}$$

$$\mathbb{T}_v(t, q_v^*, \mathbb{T}_{av,v}(t), d_i^p(t), d_i^d(t)) = (\mathbb{T}_{av,v}(t) - t) + \mathbb{T}_{p,v} + \mathbb{T}_{d,v} \tag{2.29}$$

The problem can be summarized as follows.

Problem 2.7.2. At time t, considering an environment $\mathcal{G}$, the order with the highest priority $d_i(t)$ and a set of vehicles $\mathcal{V}$, the suboptimal assignment is given by finding which vehicle (v^*) guarantees the fastest delivery, i.e. the one with the lowest $\mathbb{T}_v$

$$v^* = \arg\min_{v \in \mathcal{V}} \mathbb{T}_v \tag{2.30}$$

Case 2

Denote with $\mathbb{T}_v(t, q_v^*, \mathbb{T}_{av,v}(t), d_i^p(t), d_i^d(t), C)$ the shortest time required by vehicle v to pick up the order in $d_i^p(t)$, deliver it to $d_i^d(t)$, reach the charging station $\mathcal{C}$ and then fully charge its battery, starting from q_v^* at time $\mathbb{T}_{av,v}(t)$. The overall time for service can be obtained as follows:

$$\mathbb{T}_{p,v} = \mathbb{T}_v(\mathbb{T}_{av,v}(t), q_v^*, \mathbb{T}_{av,v}(t), q_v^*, d_i^p(t)) \tag{2.31}$$

$$\mathbb{T}_{d,v} = \mathbb{T}_v(\mathbb{T}_{p,v}, d_i^p(t), \mathbb{T}_{p,v}, d_i^p(t), d_i^d(t)) \tag{2.32}$$

$$\mathbb{T}_{c,v} = \mathbb{T}_v(\mathbb{T}_{d,v}, d_i^d(t), \mathbb{T}_{d,v}, d_i^d(t), \mathcal{C}) \tag{2.33}$$

$$\begin{split}\mathbb{T}_v(t, q_v^*, \mathbb{T}_{av,v}(t), d_i^p(t), d_i^d(t), C) = (\mathbb{T}_{av,v}(t) - t) + \\ + \mathbb{T}_{p,v} + \mathbb{T}_{d,v} + \mathbb{T}_{c,v} + \mathbb{T}_{ch,v}\end{split} \tag{2.34}$$

$\mathbb{T}_{p,v}$, $\mathbb{T}_{d,v}$ and $\mathbb{T}_{c,v}$ are obtained using dynamic programming on the graph $\mathcal{G}$, while $\mathbb{T}_{ch,v}$ is obtained by solving the non linear System 2.22.

Problem 2.7.3. At time t, considering an environment $\mathcal{G}$, the order with the highest priority $d_i(t)$ and a set of available vehicles $\mathcal{V}$, the suboptimal assignment is given by finding which vehicle (v^*) guarantees the fastest delivery, i.e. the one with the lowest $\mathbb{T}_v$

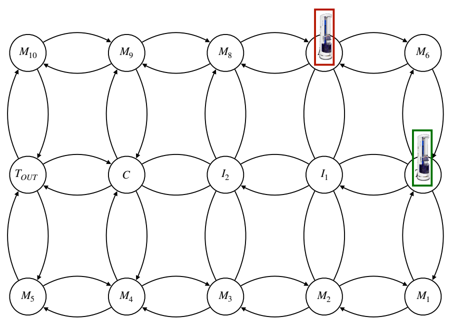
$$v^* = \arg\min_{v \in \mathcal{V}} \mathbb{T}_v \tag{2.35}$$

2.7.5 Solution

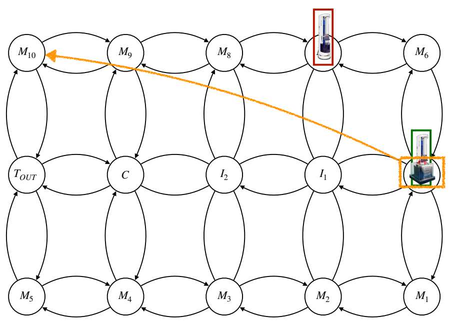
The optimum for sub-tasks $\mathbb{T}_{p,v}$, $\mathbb{T}_{d,v}$ and $\mathbb{T}_{c,v}$ can be found using DP. Since orders come over time and are not known a priori, it is necessary to find the path on the graph $\mathcal{G}$ for the new order and its assigned vehicle without modifying the path of the existing ones. Although this approach may lead to a suboptimal result, it allows to scale the optimization problem, thus providing a suitable solution for real time operation. Since some robots are already moving in the space, the dynamic programming needs to operate on a dynamic graph. In particular, nodes occupied dynamically by the previously assigned robots will be omitted from the graph and the optimization will be performed only on the residual graph.

Figure 2.7 and Figure 2.8 provide an example of this problem, where the path of the green vehicle is first optimized. Then, the path for the red vehicle is obtained by taking the green one into account. The dynamic programming algorithm on the dynamic graph is reported in Algorithm 6, where for each time step k the available control action set $\tilde{U}_k(x_k)$ is obtained from $U(x_k)$ by removing all the control actions in $U(x_k)$ that lead to occupied nodes at time k. This makes an occupied node at a specific given time instant unreachable, preventing two or more vehicles from being in the same node at the same time, see Algorithm 6 row 7.

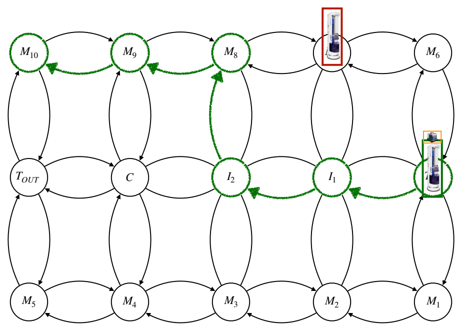
Note that for the vehicles which are currently not available, this problem will be addressed starting only from time $\mathbb{T}_{av,r}(t)$ and taking into account the vehicles moving on the graph only from that time onwards. Another issue which needs to be addressed is the future position of the vehicles after fulfilling all the assigned orders. In order to develop a dispatching strategy that is as close as possible to the real requests of a semiconductor production site, the following assumptions are made: the vehicle stays in the delivery position of the last order and this location is used by subsequent optimizations in order to account for the node occupied by such vehicle. In case the vehicle in stand-by is needed for a new order, this will be involved in a new optimization which will guarantee collision avoidance with other vehicles without modifying the other pre-defined paths. In case the pick-up/delivery point of a new order is a node occupied by a vehicle in stand-by, before assigning the new order, this vehicle is sent back to the Transfer In point by solving another dynamic program. Keep in mind that the state of charge of each vehicle is updated taking into account the length of the paths to be followed and the relative power consumption. Only if the vehicle has enough battery first to fulfill the order (the task), and then to reach the charging station within the battery limits, the task is



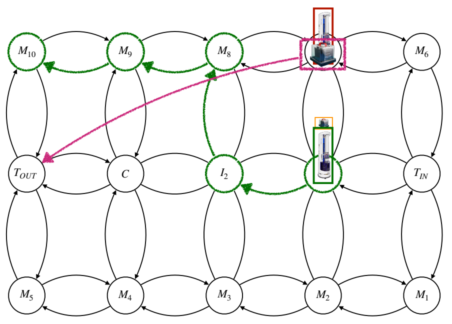
(a) Vehicles initial condition.



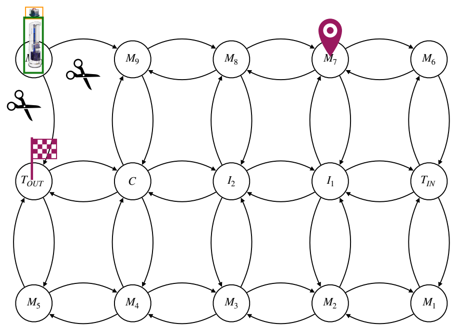
(b) The orange order appears. No moving vehicles are present.



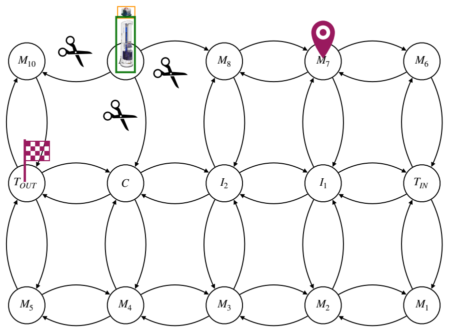
(c) The orange order is assigned to the green vehicle by using Algorithm 6.



(d) At time t, a second order appears (purple). Thus, Algorithm 6 is applied to the red vehicle by taking into account the future positions of the green vehicle.

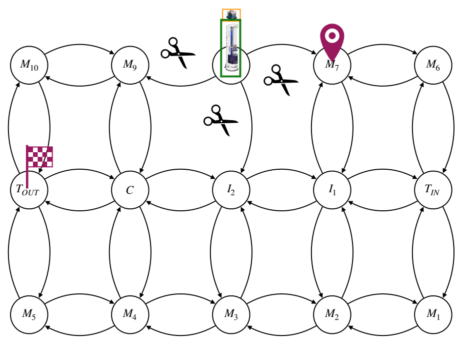


(e) Algorithm 6 goes backward and finds the positions of the red vehicle for each $k \in \{N, \ldots, 0\}$ by making the node occupied by the green vehicle unreachable only for the instant in question. For each time instant $t + k$ with $k \in \{N, \ldots, 4\}$, the node M_{10} is unreachable, where 4 is the number of steps required by the green vehicle to reach its goal.

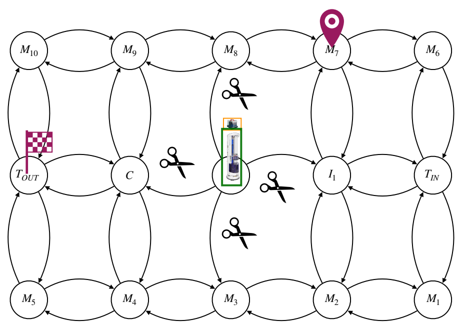


(f) At time $t+k$ with $k = 3$ the unreachable node is M_9

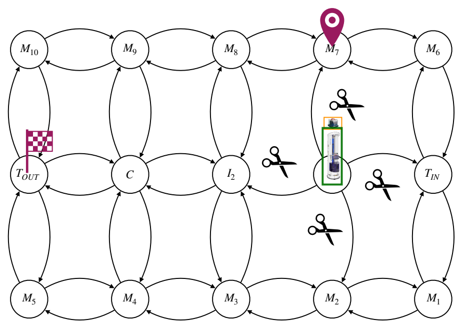
Figure 2.7: Part 1: Optimization on dynamic graph



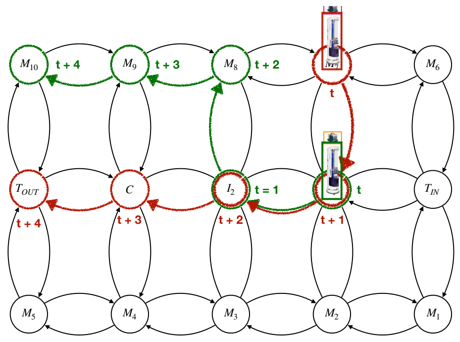
(a) At time $t+k$ with $k=2$ the unreachable node is M_8



(b) At time $t+k$ with $k=1$ the unreachable node is I_2



(c) At time $t+k$ with $k=0$ the unreachable node is I_1



(d) The path for the red vehicle is found.

Figure 2.8: Part 2: Optimization on dynamic graph

Algorithm 6: DP on dynamic graph for solving vehicle v_i

```
   Input  : $\mathcal{V}$ - vehicles set
   Input  : $v_i$ - vehicle considered
   Input  : $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$ - the directed connected graph
   Input  : $q_{start}$ - the start position of vehicle $v_i$
   Input  : $q_{end}$ - the end position of vehicle $v_i$
   Input  : $shift$ - the constant to update the vehicles path
   Input  : $N$ - Horizon Dynamic Programming
   Output : $path_{v_i}$ - the collision-free path for vehicle $v_i$
1  Set $last\ position = 0$
2  while $last\ position \neq q_{end}$ do
3      Set the solution of subproblem $N$ to $J_N(x_N) = g_N(x_N)$
4      for $k = N-1, \cdots, 0$, do
5          $\mathcal{O} = \{v.path(k + shift) \mid v \in \mathcal{V} \setminus v_i\}$
6          for $x_i \in \mathcal{Q} \setminus \mathcal{O}$ do
7              $\tilde{U}_k(x_i) = \{(x_i, x_j) \in \Delta \mid x_j \in \mathcal{Q} \setminus \mathcal{O}\}$
8              $\tilde{J}_k(x_i) = \min_{u_k \in \tilde{U}_k(x_k)} \{g_k(x_i, u_k) + J_{k+1}(x_j)\}$
9              Store the optimal cost and the optimal input.
10     The optimal cost of the overall problem can be obtained as
       $\tilde{J}(q_{start}) = J_0(q_{start})$.
11     The corresponding optimal input sequence $\tilde{\mathbf{u}} = \{\tilde{u}_0, \cdots, \tilde{u}_{N-1}\}$ is
       obtained by concatenating the optimal input elements of each subproblem.
12     Find the path on graph $path_{v_i}$ obtained by using $\tilde{\mathbf{u}}$
13     $last\ position = path_{v_i}(end)$
14     $N = N + N$
```

actually assigned to the vehicle.

Algorithm 7 shows the new Dispatching Strategy (DS) which allows to solve all cases. This is made possible by choosing which Problem to solve in Algorithm 8 using the variable *case*, i.e. $case \in \{Case\,0,\ Case\,1,\ Case\,2\}$, .

Algorithm 7: Dispatching Strategy - DS($\mathcal{V}$, $d_i(t)$, *case*)

```
   Input  : $\mathcal{V}$ - vehicles set
   Input  : $d_i(t)$ - order to be assigned
   Output : $v^*$ - vehicle chosen to satisfy $d_i(t)$
   Output : $path_{v^*}$ - the path for fulfilling $d_i(t)$
 1 $[\mathcal{V}]$=moveVehicles($\mathcal{V}$,$d_i(t)$)
 2 while $True$ do
 3     $[v^s, path_{v^s}]$ = findVehicle($\mathcal{V}$, $d_i(t)$,$case$)
 4     if $case \in \{``case\,0'', ``case\,1''\}$ then
 5         $[chekBattery]$ = checkBattery($\mathcal{V}$, $v^s$, $path_{v^s}$)
 6     else
 7         $chekBattery = True$
 8     if $chekBattery$ then
 9         $\mathcal{V}_{bk} = \mathcal{V}$
10         Assign $d_i(t)$ to vehicle $v^s$, i.e. $v^s.orders = v^s.orders \cup d_i^n(t)$ and
           $v^s.path = path_{v*}$
11         $[move, checkAssigmnet]$ = assignmentVerification($\mathcal{V}$,$v^s$)
12         if $checkAssigmnet$ then
13             confirm the assignment
14             break
15         else
16             $\mathcal{V} = \mathcal{V}_{bk}$
17             for $v \in move$ do
18                 Find $p_{last}$, i.e. $p_{last} = path_v(end)$
19                 Update the routes of moving vehicles, i.e.
                   $shift = length(path_v)$
20                 Find $path_{ch,v}$ from $p_{last}$ to $\mathcal{C}$ by using Algorithm 6
21                 Set $v.path = v.path \cup path_{ch,v}$
22     else
23         $\mathcal{V} = \mathcal{V} \setminus v^s$
24
```

The dispatching strategy can be summarized as follows. First, it checks through Algorithm 14 if the new order ($d_i(t)$) has its pick-up ($d_i^p(t)$) or its delivery ($d_i^d(t)$) nodes occupied by a vehicle in stand-by or will be occupied by a vehicle in delivery.

If so, before assigning the new order, the vehicles that are or will be in pickup or delivery positions are sent back to the Transfer In point by solving another DP program, see Algorithm 7.

Then, the DS finds the possible candidate vehicle (v^s) to which the order could be assigned by using Algorithm 8. Here, depending on the value of the variable *case*,

Algorithm 8: Vehicle Dispatching strategy - findVehicle($\mathcal{V}$, $d_i(t)$, *case*)

```
Input  : $\mathcal{V}$ - vehicles set
Input  : $d_i(t)$ - order to be assigned
Input  : $case$ - variable
Output : $v^*$ - vehicle chosen to satisfy $d_i(t)$
Output : $path_{v^*}$ - the path for fulfilling $d_i(t)$
1 if $case \in \{``Case\,0''\}$ then
2     $[v^s, path_{v^s}]$ = findVehicleC0($\mathcal{V}$, $d_i(t)$)
3 if $case \in \{``Case\,1''\}$ then
4     $[v^s, path_{v^s}]$ = findVehicleC1($\mathcal{V}$, $d_i(t)$)
5 if $case \in \{``Case\,2''\}$ then
6     $[v^s, path_{v^s}]$ = findVehicleC2($\mathcal{V}$, $d_i(t)$)
```

three different problems can be solved using three different algorithms.

Algorithm 9 aims to find the vehicle assignment for Problem 2.7.1. This algorithm finds the shortest path for each available vehicle (i.e. $v \in \mathcal{V}_{av}(t)$) to fulfill the order $d_i(t)$. Consequently, the algorithm assigns the order to the vehicle v that has the minimum time $\mathbb{T}_v$.

Algorithm 10 is used to solve Case 2, in which the possibility to assign an order to a currently busy vehicle is added. The algorithm optimizes the route of each vehicle taking into account both the future position of the moving vehicles and if the vehicle v is currently in delivery or not. In particular, if vehicle v is occupied, the algorithm identifies when $\mathbb{T}_{av,v}$ and where q_v^* the vehicle becomes available for fulfilling the order $d_i(t)$. Successively, the algorithm finds the shortest path to satisfy the order $d_i(t)$ starting from q_v^* by taking into account the future position of the moving vehicles from the $\mathbb{T}_{av,v}$ onwards, i.e. $t \geq \mathbb{T}_{av,v}$. As it is necessary to take into consideration the time that vehicle v may require before becoming available, the term $(\mathbb{T}_{av,v} - t)$ is added to the necessary time of vehicle v to pickup and to deliver the order $d_i(t)$, where $\mathbb{T}_{av,v} \geq t$ if the vehicle at time t is busy while

Algorithm 9: Vehicle Dispatching strategy Case 0 - findVehicleC0($\mathcal{V}$, $d_i(t)$)

```
    Input  : $\mathcal{V}$ - vehicles set
    Input  : $d_i(t)$ - order to be assigned
    Output : $v^*$ - vehicle chosen to satisfy $d_i(t)$
    Output : $path_{v^*}$ - the path for fulfilling $d_i(t)$
 1  for $v \in \mathcal{V}$ do
 2      if $v.state \in \{$"in stand-by"$\}$ then
 3          Find $q_v{}^* = v.position$
 4          Find $\mathbb{T}_{p,v}$ and $path_{p,v}$ from $q_v$ to $d_i^p(t)$ by using Algorithm 6
 5          Update the routes of moving vehicles, i.e. $shift = length(path_{p,v})$
 6          Find $\mathbb{T}_{d,v}$ and $path_{d,v}$ from $d_i^p(t)$ to $d_i^d(t)$ by using Algorithm 6
 7          $\mathbb{T}_v = (\mathbb{T}_{av,v} - t) + \mathbb{T}_{p,v} + \mathbb{T}_{d,v}$
 8          $path_v = path_{p,v} \cup path_{d,v}$
 9      else
10          $\mathbb{T}_v = \infty$
11  Find $v^*$ and $path_{v^*}$, i.e. $v^* = \arg\min_{v \in \mathcal{V}_{av}(t)} \mathbb{T}_v$
```

is set $\mathbb{T}_{av,v} = t$ if vehicle v is available at time t.

Otherwise, if *case* is set as *Case* 2, the vehicle assignment is made by taking into account also the time to fully recharge the vehicle battery. To achieve this, Algorithm 11 first checks the vehicle condition and, based on this status, it finds: the remaining time $\mathbb{T}_{av,v}$, the remaining path $path_v$ and the position q_v^* where the the analyzed robot becomes available to fulfill order $d_i(t)$. Then, the path to fulfill the order $d_i(t)$ and reach the charging station $\mathcal{C}$ are computed taking into consideration the future position of the moving vehicles by using Algorithm 6.

Once this first path is computed (i.e. $path_v = path_{av,v} \cup path_{p,v} \cup path_{d,v} \cup path_{c,v}$), the algorithm finds the battery state of charge of vehicle v at the end of $path_v$ by evolving the System 2.22 from the initial condition $v.bat$ for the entire length of $path_v$ by using Algorithm 15.

If the state of charge obtained from Algorithm 15 is greater than a given threshold, first the algorithm finds the necessary time steps to fully recharge the vehicle battery by using Algorithm 16, then it computes the overall time $\mathbb{T}_v$. Otherwise, if the state of charge is less than the given threshold, since the battery constraint is not respected, Algorithm 11 recomputes the path for the considered vehicle. In this case the algorithm finds the path for vehicle v where, as soon as v becomes available to fulfill the order, first it is sent to fully recharge its battery and successively to

Algorithm 10: Vehicle Dispatching strategy Case 1 - findVehicleC1($\mathcal{V}$, $d_i(t)$)

```
Input  : $\mathcal{V}$ - vehicles set
Input  : $d_i(t)$ - order to be assigned
Output : $v^*$ - vehicle chosen to satisfy $d_i(t)$
Output : $path_{v^*}$ - the path for fulfilling $d_i(t)$
1  for $v \in \mathcal{V}$ do
2      if $v.state \in \{$"in transit", "stand-by"$\}$ then
3          if $v.state \in \{$"in transit"$\}$ then
4              Find $q_v^* = v.path(end)$
5              Find $path_{av,v} = v.path$
6              Find $\mathbb{T}_{av,v}$
7              Update the routes of moving vehicles, i.e. $shift = length(v.path)$
8          else
9              $q_v^* = v.position$
10             Set $\mathbb{T}_{av,v} = t$
11             Set $path_{av,v} = \{\}$
12         Find $\mathbb{T}_{p,v}$ and $path_{p,v}$ from $q_v^*$ to $d_i^p(t)$ by using Algorithm 6
13         Update the routes of moving vehicles, i.e.
           $shift = shift + length(path_{p,v})$
14         Find $\mathbb{T}_{d,v}$ and $path_{d,v}$ from $d_i^p(t)$ to $d_i^d(t)$ by using Algorithm 6
15         $\mathbb{T}_v = (\mathbb{T}_{av,v} - t) + \mathbb{T}_{p,v} + \mathbb{T}_{d,v}$
16         $path_v = path_{av,v} \cup path_{p,v} \cup path_{d,v}$
17     else
18         $\mathbb{T}_v = \infty$
19 Find $v^*$ and $path_{v^*}$, i.e. $v^* = \arg\min_{v \in \mathcal{V}} \mathbb{T}_v$
```

fulfill order $d_i(t)$.

Once time $\mathbb{T}_v$ is computed according to the problem to be solved, the dispatching strategy finds the best vehicle candidate v^s to which to assign the order (i.e. the vehicle with the minimum overall time), see Algorithm 7 row 10. However, in order to ensure the proper functioning of the site, the dispatching strategy must verify that vehicle v^s respects the following constraints:

- the selected vehicle v^s has to be able to first satisfy order $d_i(t)$ and then reach the charging station $\mathcal{C}$ with the charge level of its battery;
- the order assignment to vehicle v^s must not prevent other vehicles from reaching the charging station with enough charge in their battery.

Algorithm 7 verifies these constraints by using both Algorithm 12 and Algorithm 13. The first algorithm evaluates if the vehicle has enough battery to satisfy the order and then to reach the charging station with enough battery charge. This is achieved by discharging the battery with a constant rate ($K > 0$) for the entire path composed of the path to satisfy the order $d_i(t)$ and the one to reach the charging station. This is obtained by taking into account the future position of the moving vehicles through Algorithm 6. Note that since Algorithm 11 allows to find a path that respects the battery constraint, Algorithm 12 is used only for *case* 0 and *case* 1.

The second algorithm evaluates if the vehicle assignment obstructs some vehicles from reaching the charging station with enough battery charge level. In particular, the algorithm relies on Algorithm 12 in order to evaluate for each vehicle v the battery charge that enables v to reach the charging station by taking into account the path of the vehicle v^s. Furthermore, the algorithm computes the set *move* which is onlu composed of the vehicles that are not able to reach the charging station with enough battery.

After the dispatching strategy has verified the constraints, if these are respected, the algorithm confirms the assignment of order $d_i(t)$ to vehicle v^s. Otherwise,

Algorithm 11: Vehicle assignment Case 2 - findVehicleC2($\mathcal{V}$, $d_i(t)$)

```
   Input  : $\mathcal{V}$ - vehicles set
   Input  : $d_i(t)$ - order to be assigned
   Output : $v^*$ - vehicle chosen to satisfy $d_i(t)$
   Output : $path_{v^*}$ - the path for fulfilling $d_i(t)$
1  for $v \in \mathcal{V}$ do
2      if $v.state \in \{$"in transit", "stand-by"$\}$ then
3          if $v.state \in \{$"in transit"$\}$ then
4              Find $q_v^* = v.path(end)$
5              Find $path_{av,v} = v.path$
6              Find $\mathbb{T}_{av,v}$
7              Update the routes of moving vehicles, i.e. $shift = length(v.path)$
8          else
9              $q_v^* = v.position$
10             Set $\mathbb{T}_{av,v} = t$
11             Set $path_{av,v} = \{\}$
12         Find $\mathbb{T}_{p,v}$ and $path_{p,v}$ from $q_v^*$ to $d_i^p(t)$ by using Algorithm 6
13         Update the routes of moving vehicles, i.e.
           $shift = shift + length(path_{p,v})$
14         Find $\mathbb{T}_{d,v}$ and $path_{d,v}$ from $d_i^p(t)$ to $d_i^d(t)$ by using Algorithm 6
15         Update the routes of moving vehicles, i.e.
           $shift = shift + length(path_{d,v})$
16         Find $\mathbb{T}_{c,v}$ and $path_{c,v}$ from $d_i^d(t)$ to $\mathcal{C}$ by using Algorithm 6
17         $[\mathbf{x_{end}}]$ = dischargeBattery($v.x_{bat}$, $path_v$), see Algorithm 15
18         if $\mathbf{x_{end}}(1) \geq threshold$ then
19             $[\mathbb{T}_{ch,v}]$ = chargeBattery($x_{end}$), see Algorithm 16
20             $\mathbb{T}_v = (\mathbb{T}_{av,v} - t) + \mathbb{T}_{p,v} + \mathbb{T}_{d,v} + \mathbb{T}_{c,v}$
21             $path_v = path_{av,v} \cup path_{p,v} \cup path_{d,v} \cup path_{c,v}$
22         else
23             Find $\mathbb{T}_{c,v}$ and $path_{c,v}$ from $q_v^*$ to $\mathcal{C}$ by using Algorithm 6
24             $[\mathbf{x_{end}}]$ = dischargeBattery($v.x_{bat}$, $v.path \cup path_{c,v}$), see
               Algorithm 15
25             $[\mathbb{T}_{ch,v}]$ = chargeBattery($\mathbf{x_{end}}$), see Algorithm 16
26             Update the routes of moving vehicles, i.e.
               $shift = length(v.path \cup path_{c,v}) + \mathbb{T}_{ch,v}$
27             Find $\mathbb{T}_{p,v}$ and $path_{p,v}$ from $\mathcal{C}$ to $d_i^p(t)$ by using Algorithm 6
28             Update the routes of moving vehicles, i.e.
               $shift = shift + length(path_{p,v})$
29             Find $\mathbb{T}_{d,v}$ and $path_{d,v}$ from $d_i^p(t)$ to $d_i^d(t)$ by using Algorithm 6
30             $\mathbb{T}_v = (\mathbb{T}_{av,v} - t) + \mathbb{T}_{p,v} + \mathbb{T}_{d,v} + \mathbb{T}_{c,v}$
31             $path_v = path_{av,v} \cup path_{c,v} \cup path_{p,v} \cup path_{d,v}$
32     else
33         $\mathbb{T}_v = \infty$
34 Find $v^*$ and $path_{v^*}$, i.e. $v^* = \arg\min_{v \in \mathcal{V}} \mathbb{T}_v$
```

Algorithm 12: Check Battery - checkBattery($\mathcal{V}$, v^*, $path_{v^*}$)

```
   Input  : $\mathcal{V}$ - vehicles set
   Input  : $v^*$ - considered vehicle
   Input  : $path_{v^*}$ - path of the considered vehicle $v^*$
   Output : $check$ - the path for fulfilling $d_i(t)$
1  Find $p_{last}$, i.e. $p_{last} = path_{v^*}(end)$
2  Update the routes of moving vehicles, i.e. $shift = length(path_{v^*})$
3  Find $path_{ch,v}$ from $p_{last}$ to $c \in \mathcal{C}$ by using Algorithm 6
4  for $i \in 0, \cdots, length(path_{v^*} \cup path_{ch,v})$ do
5      [$\mathbf{x_{bat}}$] = evolutionBattery($v^*.bat$, K)
6      $v^*.bat = \mathbf{x_{bat}}$
7  if $v^*.bat.soc \geq threshold$ then
8      check = True
9  else
10     check = False
```

Algorithm 13: Assignment verification - assignmentVerification($\mathcal{V}$,v^*)

```
   Input  : $\mathcal{V}$ - vehicles set
   Input  : $v^*$ - assigned vehicle
   Output : $move$ - the set of vehicles to be moved
   Output : $checkAssigmnet$ - check
1  $move = \{\}$
2  $checkAssigmnet = True$
3  for $v \in \mathcal{V} \setminus v^*$ do
4      [$chekBattery$] = checkBattery($\mathcal{V}$, $v$, $v.path$)
5
6      if $chekBattery \neq True$ then
7          $move = move \cup v$
8          $checkAssigmnet = False$
```

Algorithm 14: Move obstruction vehicle- moveVehicles($\mathcal{V}$,$d_i(t)$)

```
   Input  : $\mathcal{V}$ - vehicles set
   Input  : $d_i(t)$ - order to be fulfilled
   Output : $\mathcal{V}$ - the set of vehicles
1  for $v \in \mathcal{V}$ do
2      if $v.state \in \{"stand\text{-}by"\} \wedge v.position \in \{d_i^p(t), d_i^d(t)\}$ then
3          Find $path_{mov,v}$ from $v.position$ to $T_{IN}$ by using Algorithm 6
4          $v.path = v.path \cup path_{mov,v}$
5      if $v.state \in \{"in\ transport"\} \wedge v.path(end) \in \{d_i^p(t), d_i^d(t)\}$ then
6          Update the routes of moving vehicles, i.e. $shift = length(v.path)$
7          Find $path_{mov,v}$ from $v.path(end)$ to $T_{IN}$ by using Algorithm 6
8          $v.path = v.path \cup path_{mov,v}$
```

depending on which constraint is violated, the algorithm acts as described below.
If vehicle v^s does not have enough battery, the DS finds another candidate by reapplying the previous steps without considering vehicle v^s.
On the other hand, if vehicle v^s has enough battery but it prevents other vehicles from reaching the charging station, first the DS sends all the vehicles who cannot reach the charging station (i.e. all the vehicles that belong to the set *move*) back to the charging station by solving several DP programs, then it reapplies the previous steps.
This procedure is performed as long as the algorithm does not find a vehicle that meets all the requirements described above. If it is not possible to find a vehicle v^s that satisfies the order $d_i(t)$, this is frozen and, if a second order $(d_{i+1}(t))$ with higher priority exists, it is taken into consideration. Otherwise, order $d_i(t)$ remains on hold until there is a vehicle capable to fulfill it.
Note that all the paths are collision-free as a result of the use of Algorithm 6, where the future position of the moving vehicles is taken into account. However, since Algorithm 6 is used for finding paths in a future time, it is necessary to consider the future position of moving vehicles starting from the correct time instant. To achieve this, the variable $shift$ is used in Algorithm 7, which allows to consider the paths of moving vehicles only from time $shift$ onwards.
In Algorithm 12, in order to find the level of the battery, Algorithm 5 is used, which relies on the non-linear battery model described in Section 2.7.3. Both Algorithm 15 and Algorithm 16 are used in Algorithm 11. The first algorithm is used for obtaining the battery charge level at the end of the $path_v$, while the second algorithm is used for finding the time required to fully recharge the vehicle battery from the initial condition $\mathbf{x_0}$.

Algorithm 15: Discharge non-linear battery - dischargeBattery($\mathbf{x_0}$, $path_v$)

```
    Input  : $\mathcal{V}$ - vehicle
    Input  : $path_v$ - path of vehicle $v$
    Output : $\mathbf{x}_{end}$ - State of charge of vehicle $v$ at the end of the $path_v$
1   $n = length(path_v)$
2   for $k \in \{1, \cdots, n\}$ do
3       Find $\mathbf{x_{bat}}$ by using Algorithm 5 starting from $\mathbf{x}_0$ with $K < 0$
4       $\mathbf{x}_0 = \mathbf{x_{bat}}$
5   $\mathbf{x}_{end} = \mathbf{x_{bat}}$
```

Algorithm 16: Charge non-linear battery - chargeBattery($\mathbf{x_0}$)

```
    Input  : $\mathbf{x}_0$ - battery initial condition
    Output : $\mathbb{T}_{ch}$ - Time necessary to fully recharge the battery
1   $\mathbb{T}_{ch} = 0$
2   while $\mathbf{x}_0(1) < 100\%$ do
3       Find $\mathbf{x_{bat}}$ by using Algorithm 5 starting from $\mathbf{x}_0$ with $K > 0$
4       $\mathbf{x}_0 = \mathbf{x_{bat}}$
5       $\mathbb{T}_{ch} = \mathbb{T}_{ch} + 1$
```

2.7.6 Results

In order to show the effectiveness of the proposed dispatching strategy, a simulator in MatLab is developed. This is based on the scenario suggested by Infineon Austria, see Figure 2.9. The scenario is composed of 20 machines, in which for each

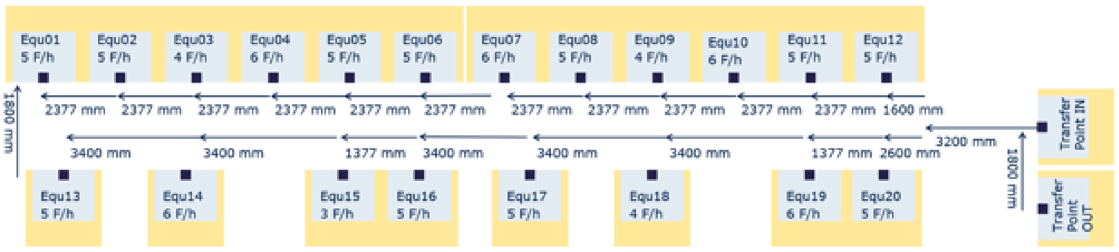


Figure 2.9: The scenario given by Infineon Villach

equipment there is a *Buffer In* and *Buffer Out*. The buffer (In-Out) dimensions are set equal to the processing capacity of the machines. The charge station is positioned at the *Transfer Point In* and manages to charge all vehicles at the same time. Both the *Transfer Point In* and the *Transfer Point Out* are supposed to have an limitless capacity.

A list of wafers entering the site is generated in accordance with the supposed average number of Foups per hour (λ = [F/h], a Poisson distribution is adopted). A Foup is Front Opening Unified Pod, which is a container where wafers are stored to be transported safely.

Order priority is assigned with the following rules. The orders that must be transported from *Buffer out* to *Transfer Point Out* have a higher priority than the orders in *Transfer Point IN*. In particular, the older requests have a higher priority. The machine to which the wafer is destined is extracted randomly (uniform distribution). Wafers move towards their destination only when the machine *Buffer In* has space available. Otherwise, they wait at the *Transfer Point In*.

The elaboration time of a machine is assumed deterministic and equal to 1/(machine processing capacity) (e.g. 1/(5F/h)=12 minutes per wafers). After an order has been processed, it goes to the machine *Buffer out*. If the *Buffer Out* is full, the Foup stays in the machine until the *Buffer out* becomes available again. When a vehicle is available, then the Foup is transferred to the *Transfer Point Out*. The vehicle assignment is made by using one of the dispatching strategies proposed in Section 2.7 through Algorithm 7.

Each vehicle has a limited battery capacity and consumes energy unless it is located at the charging station.

Since detailed information on the mechanical (weight, coefficients of friction, etc.) and electrical (nominal power of the engines and battery characteristics) characteristics of the vehicles used are not available, the following assumptions are made:

- each vehicle has a battery pack which is composed of three cells in series and with a total capacity of $Q_{tot} = 10.12\ Ah$
- each vehicle consumes 30% of its battery capacity per hour when it is moving, while 5% when it is in stand-by. Moreover, it is assumed that the battery station allows to recharge all vehicles at the same time with a rate of 50% per hour;
- robots are required to recharge when their battery level goes below 30%.

In order to identify the battery parameters, the COETA [101] program is used to generate the following profiles: a random profile for identifying the battery parameters (R_0, R_1, R_2, C_1, C_2) and a Hybrid Pulse Power Characterization (HPPC) that allows to find the polynomial parameters b_i relative to the v_{ocp}. Using both MatLab and HPPC the v_{ocp} profile, which is approximated with a ninth degree polynomial curve, is found. Then, the identification of the parameters for a non-linear battery system is performed through the least square methods by using CasADi [102]. In Table 2.6 the polynomial parameters b_i and the battery parameters are reported.

Name	Value	Unit	Name	Value	Unit
Q_{tot}	10.12	Ah	b_1	-0.0069	-
R_0	42.2	$m\Omega$	b_2	-0.0264	-
R_1	6.45	$m\Omega$	b_3	0.0243	-
R_2	6.53	$m\Omega$	b_4	0.0313	-
C_1	1.957	kF	b_5	-0.0561	-
C_2	7.284	kF	b_6	0.1261	-
V_{max}	12.448	V	b_7	0.0860	-
V_{min}	10.0331	V	b_8	0.3100	-
b_0	0.0063	-	b_9	11.4173	-

Table 2.6: The main parameters of the DP battery model

The machines buffer (In-Out) dimension is set equal to the processing capacity of the machines. The graph is obtained in accordance with the maximum robot size increased by 20%. A discretization step larger than the real size of the vehicle is chosen in order to have a safety margin to avoid collisions. The resulting Graph has 1089 (33x33) nodes and 120 edges as shown in Figure 2.10. The simulations are

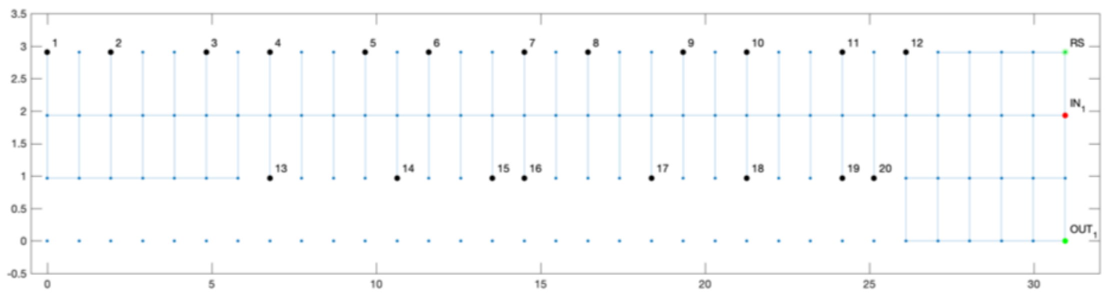


Figure 2.10: The graph obtained by the fixed step discretization.

conducted using 5 vehicles and the mean time needed for resolving the dispatching problem using our proposed methods is equal to 2.011 s, which is compatible with the desired sampling time adopted in a real scenario. At the following link, `https://youtu.be/kIojrm5dRfU` it is possible to see a short video of a simulation. As can be seen in the video, it is possible to divide the simulator graphics into two images, upper and lower. The analyzed environment is reported on the upper image, in which it is possible to notice the paths followed at each time instant by vehicles. These are represented by rectangles. In particular, each vehicle has its assigned color, as reported in the legend. The number above each robot, when it is present, is the order number actually transported. Moreover, the pickup and delivery positions of the transported order are shown below the vehicle. In case, the order is assigned to a busy robot, the number of the assigned order is reported on the upper right corner of the vehicle, as can be seen at 2.21 min of the video for the 4-th vehicle. As concerns the current time and the battery level, they are shown for each vehicle on the lower side of the upper image. The lower image shows the states of each machine (equipment, T in and T out) in the considered scenario. In particular, for the equipment it is possible to see the order number that is processing on the upper right corner of the box which represents each machine. Therefore, for each machine the conditions of both buffer-in and out are reported. In particular, the orders that are in the buffer-in and buffer out are arranged in order of arrival on the left and on the right column respectively for each machine. The counter on transfer point in give us the number of orders that are waiting in the transfer-in, while the counter in transfer out shows the fulfilled order number. Note that an order is considered fulfilled when it is delivered at the transfer point-out. In Table 2.7, Table 2.8 and Table 2.9 show the results of 6 simulated days of the production site obtained by using the three different proposed strategies. As can be noticed from the comparison of the results, the possibility to assign the order to a busy vehicle allows to improve the performance of the site in terms of number of fulfilled orders. From the comparison between *case 1* and *case 2*, it is possible to notice that, when the time required to reach the charging station and then to fully recharge the battery is taken into account in the vehicle assignment, the performance of the production cite worsens. This is mainly due to the fact that the time required to recharge the battery is much longer than the time required to

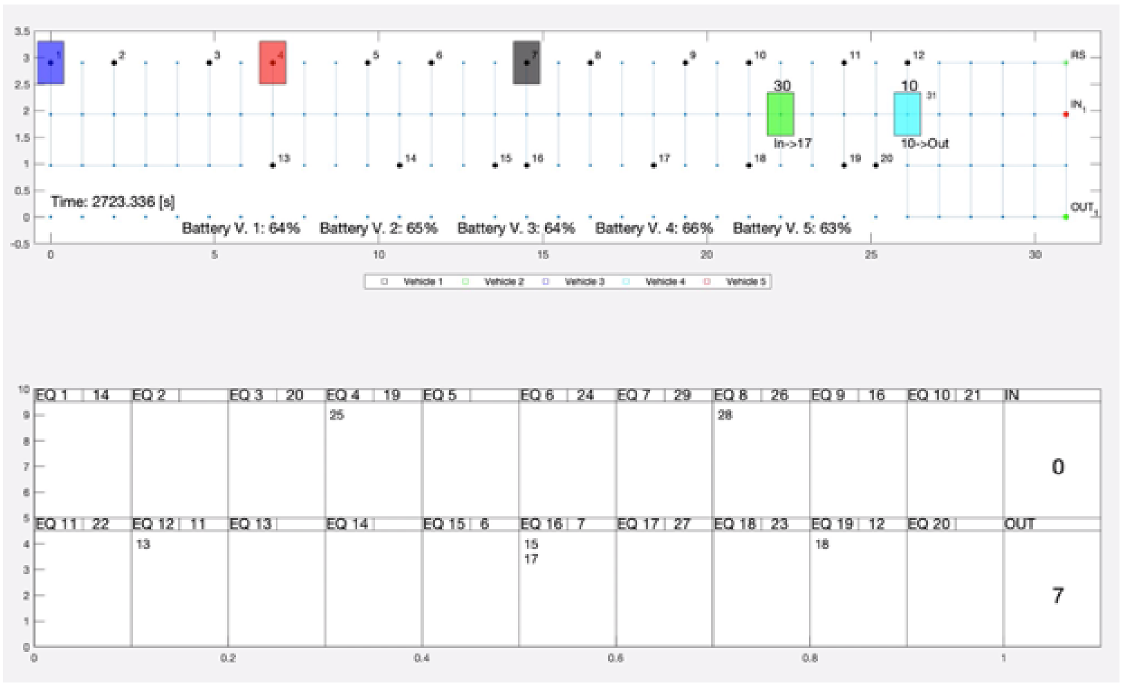


Figure 2.11: The MatLab simulator

fulfill the order. This means that an order is always assigned to the vehicle that has the highest battery charge. Furthermore, this leads to a more equitable assignment of orders to vehicles. As a consequence, all vehicles are likely to need charging at the same time, which may result in worsening the performance of the production site.

Case 0	
No. of order assignments to v_1	3222
No. of order assignments to v_2	3374
No. of order assignments to v_3	3234
No. of order assignments to v_4	3170
No. of order assignments to v_5	3194
No. of order assignments to vehicles with insufficient battery charge	0
No. of order assignments to busy vehicles (future assignments)	0
Total number of transport requests processed	8055
Total number of transport requests	14375

Table 2.7: Case 0: the order is assigned to an available vehicle which is capable of fulfilling the order without obstructing other vehicles to reach the charging station.

Case 1	
No. of order assignments to v_1	3576
No. of order assignments to v_2	3581
No. of order assignments to v_3	3535
No. of order assignments to v_4	3503
No. of order assignments to v_5	3467
No. of order assignments to vehicles with insufficient battery charge	0
No. of order assignments to busy vehicles (future assignments)	10510
Total number of transport requests processed	8773
Total number of transport requests	14375

Table 2.8: Case 1: the order can be assigned to a currently busy vehicle which respects the constraints. The charging time is not considered in the vehicle assignment

Case 2	
No. of order assignments to v_1	3464
No. of order assignments to v_2	3442
No. of order assignments to v_3	3472
No. of order assignments to v_4	3439
No. of order assignments to v_5	3444
No. of order assignments to vehicles with insufficient battery charge	149
No. of order assignments to busy vehicles (future assignments)	10121
Total number of transport requests processed	8570
Total number of transport requests	14375

Table 2.9: Case 2: Also the time to reach the charging station and to fully charge the battery are taken into account in the vehicle assignment

2.8 Conclusion

In this chapter several approaches are proposed in order to solve the dispatching problem in a semiconductor manufacturing process. In particular, the dispatching problem has been analyzed from the simplest problem, finding the shortest route to satisfy the order for the only available vehicle, to the most complicated one, finding the shortest route when orders appear at different times, with different priorities and a with fleet of vehicles, taking into account also collision avoidance and the vehicle battery constraints. As highlighted in the chapter, as the complexity of the problem analyzed increases, in order to obtain a solution in a reasonable time sub-optimal approaches have been proposed. Among all the wafer dispatching strategies proposed in this chapter, only the approach described in the Algorithm 4 has been tested on a real application in Infineon Dresden.

Chapter 3

Multi-robot Routing and Scheduling with Temporal Logic and Synchronization Constraints

Contents

3.1 Introduction 64

3.2 Preliminaries 65

3.3 Problem Formulation 68

3.3.1 Environment Model 69

3.3.2 Robot Model 69

3.3.3 Specification: Transportation Demands and Synchronization Rules 69

3.3.4 Problem Definition 70

3.4 Nominal Solution 71

3.5 Robust Solution 73

3.5.1 On-line Controller 73

3.6 Simulations and Results 78

3.6.1 Baseline 81

3.6.2 Online Controller 82

3.7 Conclusion 82

3.1 Introduction

The manufacturing industry has become increasingly competitive over the last decades. In order to survive in this environment, manufactures must satisfy customer demands exactly in time, quality and quantity. Generally, the production of goods is a complex task which requires several steps. In a medium-complexity semiconductor fab, processes need 250-500 steps and use from 50 to 120 different machines [5]. Furthermore, in order to maximize plant efficiency, several goods are usually processed/transported at the same time. In this challenging scenario, the optimal routing and scheduling of goods are of major importance. While in the past the dispatching was performed mainly by human operators, progress in technology allows to move towards fully automated transportation (e.g. fleet of robots). In this case, the routing and scheduling problem consists in finding the optimal robot routes able to minimize transport time while respecting constraints (i.e. deadlines, priorities, etc.). For this reason, in this chapter, a temporal logic with explicit time is used to solve routing and scheduling problems [103, 104] in case some demands must be delivered in a synchronous way by the robots. Moreover, an online controller is proposed in order to guarantee the synchronization demands even in the presence of uncertainties that can affect the motion of the robots (i.e. travel time). Synchronization demands are justified by the fact that some manufacturing steps require the simultaneous delivery of different components for starting the processing of new goods. The process can start only when all materials have arrived. Since the components must be processed together, the maximum delay of a single component affects the overall process time. This can lead to a waste of time, money and efficiency. Therefore, the synchronization rules are of fundamental importance in order to provide the right components to the right machine/s at the right time. In this chapter, the motions of the robots without uncertainties are captured by using weighted Transition Systems (TS)[105]. The transport demands within the production site are expressed by using TWTL. This choice is motivated by the

TWTL capability to deal with both time constraints and possible unsatisfiable tasks. Because of, it may unfortunately occur that some demands cannot be satisfied due to possible strict time constraints. TWTL provides *temporal relaxation*, which offers the possibility to find the minimally relaxed formulae (in terms of time constraints) that the robots can satisfy. The TWTL formulae are translated to finite state automata which are then composed with the TS in order to obtain Product Automata (PA)[105]. On the resulting PA, a nominal (no uncertainties) optimal routing and scheduling solution is found by using the Dijkstra's algorithm and selecting among the accepting states of PA the one that can be reached in the shortest time, thus minimizing time constraints relaxation.

In real scenarios, robots movements may be affected by structural uncertainty, such as the weight variation of the robots and/or dynamic uncertainties, e.g. battery state of charge. These uncertainties can lead to significant variations in travelling times between points of interest. In this section an online approach able to guarantee the synchronization demands even in presence of uncertainties is presented. The proposed method is finally validated on a case study representing a semiconductor production site where the dispatching is assigned to a fleet of autonomous robots.

The contributions of this work can be summarized as follows: i) a routing and scheduling problem with synchronization rules and time constraints for vehicles in presence of structural and dynamic uncertainties is formulated; ii) a general (baseline) solution for nominal motion models is proposed; iii) an online method for solving the problem in the presence of uncertainties while minimizing constraints relaxation and still guaranteeing synchronization is proposed.

3.2 Preliminaries

For a finite set Σ, denote with 2^{Σ} and $|\Sigma|$ the set of all subsets, and the cardinality of Σ, respectively. A *word* $\boldsymbol{\sigma}$ is a finite or infinite sequence of elements from Σ. Let $|\boldsymbol{\sigma}|$ indicate the length of a word $\boldsymbol{\sigma}$. The repetition of symbol σ d times is denoted by $\sigma^{\{d\}}$. A *language* is a set of words over the *alphabet* Σ.

Definition 3.2.1 (Deterministic Transition System, DTS). A (weighted) Deterministic Transition System is a tuple $\mathcal{T} = (Q, q_0, \Delta, AP, h, \omega)$, where Q is the finite set of states; $q_0 \in Q$ is the initial state; $\Delta \subseteq Q \times Q$ is the set of transitions; AP is the set of observations (atomic propositions); $h : Q \to 2^{AP}$ is the labeling function and; $\omega : \Delta \to \mathbb{Z}_{>0}$ is a map that assigns a positive integer weight to each transition.

A transition $(q, q') \in \Delta$ of $\mathcal{T}$ is also denoted by $q \to_{\mathcal{T}} q'$. Define *trajectory* of the system as a sequence of states $\mathbf{q} = q_0, q_1, \ldots$ such that $q_k \to_{\mathcal{T}} q_{k+1}$ for all $k \geq 0$. A *trajectory* generates an *output word* $\mathbf{o} = \mathbf{o}_0 \cdot \mathbf{o}_1 \cdot \mathbf{o}_2 \ldots$, where $\mathbf{o}_0 = h(q_0)$, $\mathbf{o}_k = h(q_k)^{\{\omega((q_k,q_k))\}}$ if $q_k = q_{k-1}$, and $\mathbf{o}_k = \emptyset^{\{\omega((q_{k-1},q_k))-1\}} h(q_k)$ if $q_k \neq q_{k-1}$ for all $k \geq 1$. The sub-word $\mathbf{o}_k$ corresponds to the observations generated along the transition q_{k-1}, q_k of duration $\omega((q_{k-1}, q_k))$. Note that, as opposed to state trajectories, output words are defined at each discrete time $k \in \mathbb{Z}_{\geq 0}$, where the weights of $\mathcal{T}$ are interpreted as transition durations. Thus, no observations (i.e., $\emptyset$) are considered along transitions. It also denote $\mathbf{o}$ by $h(\mathbf{q})$. Let $\mathcal{L}(\mathcal{T})$ be the set of all output words generated by $\mathcal{T}$, i.e., its *generated language*. Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ be trajectories of $\mathcal{T}_1, \ldots, \mathcal{T}_m$ with the same alphabet AP, and $\mathbf{o}_\ell = h_\ell(\mathbf{q}_\ell) = o_{\ell,0}, o_{\ell,1} \ldots$ the corresponding output words for all $\ell \in \{1, \ldots, m\}$. The joint output word generated by all trajectories is $h(\times_{\ell=1}^{m} \mathbf{q}_\ell) = o_0, o_1, \ldots$, where $o_k = (\bigcup_{\ell=1}^{m} o_{\ell,k}) \in 2^{AP}$, and $k \in \{0, 1, \ldots\}$ indicates the time instants at which observations occur.

Definition 3.2.2 (Time Window Temporal Logic). A Time Window Temporal Logic formula over a set of atomic propositions AP is defined as follows

$$\varphi ::= \mathbf{H}^d s \,|\, \mathbf{H}^d \neg s \,|\, \varphi_1 \wedge \varphi_2 \,|\, \varphi_1 \vee \varphi_2 \,|\, \varphi_1 \cdot \varphi_2 \,|\, [\varphi_1]^{[a,b]},$$

where $s \in AP \cup \{\top\}$ is either an atomic proposition or the "true" constant $\top$; $\neg, \wedge, \vee$ are the negation, conjunction, and disjunction Boolean operators, respectively; $\cdot$ is the concatenation operator; $[\varphi_1]^{[a,b]}$ with $0 \leq a \leq b$ is the *within* operator, and $\mathbf{H}^d$ with $d \geq 0$ is the *hold* operator. When $d = 0$, $\mathbf{H}$ is dropped from the notation, e.g., $s \equiv \mathbf{H}^0 s$. The satisfaction of an TWTL formula φ is defined with respect to finite output words $\mathbf{o}$ over 2^{AP}. The hold operator $\mathbf{H}^d s$ is satisfied if $s \in AP$ is repeated for d time units. Instead, the $\mathbf{H}^d \neg s$ requires that for d time units only symbols from $AP \setminus \{s\}$ appear. The fomulae $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ or $\neg\varphi$ are satisfied if

$\mathbf{o}$ satisfies both formulae, at least one, or does not satisfy the formula, respectively. The within operator $[\varphi]^{[a,b]}$ is satisfied if the formula φ becomes true in the given time window $[a, b]$. The concatenation operator $\varphi_1 \cdot \varphi_2$ requires that formula φ_1 is first satisfied and then φ_2 is satisfied immediately after.

A complete description of the semantics of TWTL can be found in [106].

The satisfaction of a TWTL formula can be decided in bounded time. Let $\|\phi\|$ be the maximum time needed to satisfy ϕ, which can be computed as follows:

$$\|\phi\| = \begin{cases} d & \text{if } \phi \in \{\mathbf{H}^d s, \mathbf{H}^d \neg s\} \\ \max(\|\phi_1\|, \|\phi_2\|) & \text{if } \phi \in \{\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2\} \\ \|\phi_1\| & \text{if } \phi = \neg\phi_1 \\ \|\phi_1\| + \|\phi_2\| + 1 & \text{if } \phi = \phi_1 \cdot \phi_2 \\ b & \text{if } \phi = [\phi_1]^{[a,b]} \end{cases} \tag{3.1}$$

Let $\phi = \left[M_1 \cdot [M_2]^{[c,d]}\right]^{[a,b]}$ be a TWTL formula that describes, for example, a possible transport demand from the pickup point M_1 to the delivery point M_2. Note that every time there is a concatenation between two formulae, the time constraints of the second formula are related to the time when the first one was satisfied. In the previous example one has "satisfy M_2 between c and d time instants after the satisfaction of M_1". The formula is satisfied if and only if all the sub-tasks (pickup and delivery) are fulfilled within the time window expressed by the external *within* operator, i.e. [a,b]. What if ϕ can not be fulfilled in the given time window? In order to cope with this possible problem, in [106] the authors propose a *temporal relaxation* of TWTL formulae. The *temporal relaxation* introduces the possibility to relax the deadlines for the time windows, which are expressed by the *within* operator. Thus, the relaxed version of ϕ is $\phi(\boldsymbol{\tau}) = \left[M_1 \cdot [M_2]^{[c,d+\tau_2]}\right]^{[a,b+\tau_1]}$, where $\boldsymbol{\tau} = (\tau_1, \tau_2) \in \mathbb{Z}^2$. Furthermore, the $\phi(\boldsymbol{\tau})$ has to preserve the feasibility of ϕ, i.e. every time window of a *within* operator has to be greater or equal than the time needed to satisfy the task enclosed by the within operator. Note that, in this chapter only the relaxation of the deadlines (upper bound)is allowed because the relaxation of the lower bound of a within operator would correspond to anticipating the pickup time of an order. Unfortunately, this is not possible in general since the order may not be already available.

Definition 3.2.3 (τ-relaxation of ϕ). Given a TWTL formula ϕ with m *within* operators, the feasible τ-relaxation of ϕ is defined as $\phi(\boldsymbol{\tau})$, where $\boldsymbol{\tau} \in \mathbb{Z}^m$ and each subformula of the form $[\phi_i]^{[a_i,b_i]}$ is replaced with $[\phi_i]^{[a_i,b_i+\tau_i]}$ for all $i \in \{1,\ldots,m\}$.

Definition 3.2.4 (Linear Temporal Relaxation). Given ϕ, let $\phi(\boldsymbol{\tau})$ be the feasible relaxation of ϕ. The linear temporal relaxation of ϕ is $|\boldsymbol{\tau}|_{LTR} = \sum_{i=1}^{m} \tau_i$.

Definition 3.2.5 (Deterministic Finite State Automaton). A Deterministic Finite State Automaton (DFA) is a tuple $\mathcal{A} = (S_{\mathcal{A}}, s_0, \delta_{\mathcal{A}}, 2^{AP}, F_{\mathcal{A}})$, where $S_{\mathcal{A}}$ is a finite set of states; $s_0 \in S_{\mathcal{A}}$ is the initial state; $\delta_{\mathcal{A}} : S_{\mathcal{A}} \times 2^{AP} \to S_{\mathcal{A}}$ is the transition function; 2^{AP} is the input alphabet; and $F_{\mathcal{A}} \subseteq S_{\mathcal{A}}$ is the set of (final) accepting states.

Let $s \xrightarrow{\sigma}_{\mathcal{A}} s'$ be the transition from s to $s' = \delta_{\mathcal{A}}(s,\sigma)$ with input symbol σ. A finite sequence of symbols $\boldsymbol{\sigma} = \sigma_0, \sigma_1, \ldots, \sigma_n$ generates a trajectory of the DFA $\mathbf{s} = s_0, s_1, \ldots, s_n$ such that $s_0 \in S_{\mathcal{A}}$ is the initial state of $\mathcal{A}$, and $s_k \xrightarrow{\sigma}_{\mathcal{A}} s_{k+1}$ denotes the transition from time k to $k+1$. The word $\boldsymbol{\sigma}$ is accepted by the DFA if and only if the corresponding trajectory ends in the final automaton state, i.e., $\sigma_{n+1} \in F_{\mathcal{A}}$. The accepted language of the DFA $\mathcal{A}$ is defined as $\mathcal{L}(\mathcal{A})$.

Formulae expressed in TWTL can be captured by DFAs as shown in [106]. Methods to compute DFAs accepting possible deadline relaxations, and to perform synthesis and verification using a *bottleneck* temporal relaxation cost have been proposed in [106]. Here, the automata construction methods is employed, but considering a linear temporal relaxation cost instead.

3.3 Problem Formulation

In this section, the environment, the robot model, and the transport demands that characterize, for example, the dispatching problem in a semiconductor manufacturing fab, are defined.

3.3.1 Environment Model

Let $\mathcal{G} = (Q, \Delta, \overline{\omega})$ denote a weighted directed connected graph, where Q represents the set of locations of interest (machines location, charging stations, interconnection nodes) labeled with observations from AP as given by map $h : Q \to 2^{AP}$. The edges $\Delta \subseteq Q \times Q$ capture feasible motions between locations with nominal travel times given by $\overline{\omega} = \Delta \to \mathbb{Z}_{\geq 1}$. Travel times are expressed in terms of a global discrete clock with time step Δt.

3.3.2 Robot Model

Consider a team of m robots moving in an environment $\mathcal{G}$. The motion model of each robot $v \in \{1, \ldots, m\}$ is captured by a TS $\mathcal{T}_v = (Q, q_{v,0}, \Delta, AP, h, \omega_v)$, where $q_{v,0} \in Q$ is the initial state of the v-th robot, and ω_v is its non-deterministic travel time function such that $\omega_v(e) \in [\underline{\rho}\,\overline{\omega}, \overline{\rho}\,\overline{\omega}] \cap \mathbb{Z}_{>0}$ for all $e \in \Delta$, where $\underline{\rho}\,, \overline{\rho}\, \in \mathbb{R}_{>0}$. In this chapter it is assumed that all robots can communicate with all other robots. Furthermore, each robot is able to detect its position when it reaches a node of interest $q \in Q$. In the following section, it is assumed that each robot can transport (fulfill) at most one transport demand at a time, i.e., robots have single capacity.

3.3.3 Specification: Transportation Demands and Synchronization Rules

Let $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ be the set of the n transport demands that must be satisfied. The i-th demand is defined as the tuple $\mathcal{D}_i = (\phi_i, \pi_i^{start})$, where ϕ_i is a TWTL formula, and $\pi_i^{start} \in AP$ indicates the start proposition of the formula ϕ_i, i.e., the pick-up location. For brevity, it is assumed that transportation demand formulae include the pick-up specification, i.e., are of the form $\phi_i = [\pi_i^{start} \wedge \phi_i']^{[0, \|\phi_i'\|]}$, where ϕ_i' is a TWTL formula.

Since some elements of $\mathcal{D}$ may require to be fulfilled (i.e. demands delivered) at the same time, a set $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots\}$ of synchronization rules is defined. The j-th rule is defined as tuple $\mathcal{R}_j = (\psi_j, I_j)$, where ψ_j is the TWTL formula of the task to be performed in a synchronous way, and $I_j \subseteq \{1, \ldots, n\}$ indicates which elements of

$\mathcal{D}$ are involved in ψ_j. Let $\widehat{\psi_j} = \left(\bigwedge_{\ell \in I_j} [\pi_\ell^{start}]^{[0,\|\phi_\ell\|]}\right) \cdot \psi_j$ with the meaning that the synchronization task ψ_j of rule $\mathcal{R}_j$ must be satisfied after the start of all associated transportation demands in I_j.

The overall specification, in which all transport demands and synchronization rules are considered, is expressed as

$$\varphi = \Phi \wedge \Psi, \tag{3.2}$$

where $\Phi = \bigwedge_{i=1}^{|\mathcal{D}|} \phi_i$, and $\Psi = \bigwedge_{j=1}^{|\mathcal{R}|} \widehat{\psi_j}$.

3.3.4 Problem Definition

Given a set of demands to be satisfied and a list of synchronization rules, our goal is to find the optimal assignment of demands and the corresponding optimal paths for the different robots. Optimality is with respect to the total deadline deviations over all demands and rules.

Problem 3.3.1. Given φ, i.e. the specification of transportation demands and synchronization rules as in (3.2), an environment $\mathcal{G}$, m robots modelled as $\mathcal{T}_1, \ldots, \mathcal{T}_m$, find trajectories Traj $= \{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ such that φ is satisfied with minimum temporal relaxation

$$\begin{aligned} \min_{\text{Traj}} \quad & |\boldsymbol{\tau}|_{LTR} \\ \text{subject to} \quad & \forall \mathcal{D}_i \in \mathcal{D}, \exists \mathbf{q}_{r_i} \in \text{Traj} \ : \ h(\mathbf{q}_{r_i}) \models \phi_i(\boldsymbol{\tau}_{\phi_i}) \\ & \forall \mathcal{R}_j \in \mathcal{R} \ : \ h(\times_{\ell \in I_j} \mathbf{q}_{r_\ell}) \models \widehat{\psi_j}(\boldsymbol{\tau}_{\psi_j}), \end{aligned}$$

where $\boldsymbol{\tau} = [\boldsymbol{\tau}_{\phi_1}, \ldots, \boldsymbol{\tau}_{\phi_{|\mathcal{D}|}}, \boldsymbol{\tau}_{\psi_1}, \ldots, \boldsymbol{\tau}_{\psi_{|\mathcal{R}|}}]$ is the vector of all deadline relaxation variables for transportation demands and synchronization rules.

In the following section, first a general nominal solution is provide, where robot motion models with nominal travel time are considered. For deployment (i.e. the real scenario), the computed trajectories are enforced by synchronizing the transitions of all robots at all time steps. This solution is considered as the baseline approach that wants to improve in terms of optimal total deadline deviations.

For this purpose, an online controller able to satisfy the TWTL formula $\phi(\boldsymbol{\tau})$ in the presence of the uncertainties in robot travel times in a more optimal way is proposed.

3.4 Nominal Solution

In this section, the general solution to Problem 3.3.1 for the nominal case without motion uncertainty is introduced. First, for each robot v the deterministic motion transition system $\mathcal{T}_v$ is created. In particular, this is defined to have all edges with weight one. This is achieved by replacing the original transitions $e \in \Delta$ with $\overline{\omega}(e)$ transitions. Secondly, the transportation demands and synchronization rules are translated to DFAs. The i-th automaton obtained from ϕ_i is denoted by $\mathcal{A}_{\phi_i}$, while $\mathcal{A}_{\widehat{\psi}_j}$ refers to the j-th automaton of Ψ obtained from $\widehat{\psi}_j = \left(\bigwedge_{\ell \in I_j} [\pi_\ell^{start}]^{[0,\|\phi_\ell\|]}\right) \cdot \psi_j$. Then, a product automaton $\mathcal{P}$ is constructed as defined in Definition 3.4.1 that captures the motion of the m robots and the satisfaction of the specification φ from (3.2). On the resulting product automaton, the solution in terms of optimal assignments and shortest paths is then found using Dijkstra's algorithm.

Definition 3.4.1 (Product Automaton). Given the product transition system $\mathcal{T}^m = \bigtimes_{v=1}^m \mathcal{T}_v = (Q^m, q_0^m, \Delta^m, 2^{AP}, h^m)$, the automata $\mathcal{A}_{\phi_i} = (S_{\mathcal{A}_{\phi_i}}, s_{0,i}, \delta_{\mathcal{A}_{\phi_i}}, 2^{AP}, F_{\mathcal{A}_{\phi_i}})$, for all $i = 1, \ldots, |\mathcal{D}|$, and the automata $\mathcal{A}_{\widehat{\psi}_j} = (S_{\mathcal{A}_{\widehat{\psi}_j}}, \widehat{s}_{0,j}, \delta_{\mathcal{A}_{\widehat{\psi}_j}}, 2^{AP}, F_{\mathcal{A}_{\widehat{\psi}_j}})$, for all $j = 1, \cdots, |\mathcal{R}|$, the product automaton (PA) is a tuple $\mathcal{P} = (S_\mathcal{P}, s_{0,\mathcal{P}}, \Delta_\mathcal{P}, F_\mathcal{P}, \omega_\mathcal{P})$, where

- $S_\mathcal{P} = Q^m \times \left(S_{\mathcal{A}_{\phi_i}} \times \{0, \cdots, m\}\right)_{i=1}^{|D|} \times \left(S_{\mathcal{A}_{\widehat{\psi}_j}}\right)_{j=1}^{|R|}$ is the finite set of states;
- $s_{0,\mathcal{P}} = \left(x_{-1}, (s_{0,i}, 0)_{i=1}^{|D|}, (\widehat{s}_{0,j})_{j=1}^{|R|}\right)$ is the initial state;
- $\Delta_\mathcal{P} \subseteq S_\mathcal{P} \times S_\mathcal{P}$ is a transition relation. Let r_i be the robot assigned to i-th demand. Then $\left(x, (s_i, r_i)_{i=1}^{|D|}, (\widehat{s}_j)_{j=1}^{|R|}\right) \to_\mathcal{P} \left(x', (s_i', r_i')_{i=1}^{|D|}, (\widehat{s}_j')_{j=1}^{|R|}\right) \in \Delta_\mathcal{P}$ iff:
  - $x = (q_1, \ldots, q_m)$, $x' = (q_1', \ldots, q_m')$, $q_v \to_{\mathcal{T}_v} q_v' \in \Delta$, $\forall v \in \{1, \ldots, m\}$;
  - $\left(r_i = 0 \ \wedge \ s_i = s_{0,i} \ \wedge \ s_i \xrightarrow{\sigma_i} s_i' \ \wedge \ \sigma_i = h(q_v') = \pi_i^{start} \ \wedge \ r_i' = v\right) \ \vee$ $\left(r_i = v \ \wedge \ s_i \xrightarrow{\sigma_i} s_i' \ \wedge \ \sigma_i = h(q_v') \ \wedge \ r_i' = v\right)$, $\forall i \in \{1, \ldots, |D|\}$;

  - $\widehat{s}_j \xrightarrow{\widehat{\sigma}_j} \widehat{s}'_j \;\wedge\; \widehat{\sigma}_j = \{h(q'_{r_i}) \mid i \in I_j \wedge r_i > 0\}, \forall j \in \{1, \cdots, |R|\}$;

- $\omega_{\mathcal{P}}(s_{\mathcal{P}}, s'_{\mathcal{P}}) = |D| - \sum_{i=1}^{|D|} |\{s'_i\} \cap F_{\phi_i}| + c(x, x')$ is the weight function, where $\sum_{i=1}^{|D|} |\{s'_i\} \cap F_{\phi_i}|$ is the number of fulfilled demands in s', and $c(x, x')$ is a cost used to penalize the number of robots changing positions in the transition from x to x' so to avoid unnecessary movements;
- $F_{\mathcal{P}} = Q^m \times (F_{\phi_i} \times \{1, \dots, m\})_{i=1}^{|D|} \times \left(F_{\widehat{\psi}_j}\right)_{j=1}^{|R|}$ is the set of accepting states.

For simplicity, the initial state $x_{-1} = (q_{1,-1}, \dots, q_{m,-1}) \in Q^m$ is introduced such that, for all $v \in \{1, \dots, m\}$, the only transition available from $q_{v,-1}$ is $q_{v,-1} \rightarrow_{\mathcal{T}_v} q_{v,0}$.

Similar to TS, a trajectory of $\mathcal{P}$ is a sequence $\mathbf{p} = p_0, p_1, \dots$ such that $p_0 = s_{0,\mathcal{P}}$ and $(p_k, p_{k+1}) \in \Delta_{\mathcal{P}}$ for all $k \geq 0$. Any satisfying (accepted) trajectory of $\mathcal{P}$ ends in a state of $F_{\mathcal{P}}$. The solution to the nominal version of Problem 3.3.1 is obtained by computing an optimal satisfying trajectory $\mathbf{p}^*$ using Dijkstra's algorithm and selecting among the accepting states of PA the one that can be reached in the shortest time, thus minimizing time constraints relaxation. By construction, $\mathbf{p}^*$ encodes valid movements of robots in the environment $\mathcal{G}$, i.e., transitions in Δ, and satisfies all transportation demands and synchronization rules. The trajectories that robots have to follow are obtained by projecting $\mathbf{p}^*$ onto each $\mathcal{T}_v$, $v \in \{1, \dots, m\}$, as given by Definition 3.4.2.

Definition 3.4.2. (Projection of a trajectory of $\mathcal{P}$ onto $\mathcal{T}_v$). Let $\mathbf{p} = p_0, p_1, \dots$ be a trajectory of $\mathcal{P}$, where $p_k = \left(x_k, (s_{i,k}, r_{i,k})_{i=1}^{|D|}, (\widehat{s}_{j,k})_{j=1}^{|R|}\right)$ and $x_k = (q_{1,k}, \dots, q_{m,k})$. The projection of $\mathbf{p}$ onto $\mathcal{T}_v$ is the trajectory $\mathbf{q}_v = q_{v,0}, q_{v,1}, \dots$ for all $v \in \{1, \dots, m\}$.

Algorithm 17 summarizes the nominal solution to Problem 1. Note that the nominal solution does not guarantee the satisfaction of the specifications in presence of uncertainties (i.e. robots travel time). In this case, it may be not possible to complete the mission specification φ without violations.

Remark. For the nominal case, it is possible to impose collision avoidance by removing unwanted states and transitions of $\mathcal{P}$ (Definition 3.4.1). When considering uncertain travel times, additional synchronization is needed, which can be computed using, again, the nominal model $\mathcal{P}$. For simplicity, in this work, collision avoidance is not considered, but left for future work.

Algorithm 17: Nominal Solution

```
Input  : $\mathcal{G}$ - the environment
Input  : $\mathcal{D}$ - the set of demand to be satisfied
Input  : $\mathcal{R}$ - the set of synchronization rules
Output : The optimal run $\mathbf{q}_v$ for each robot $v \in \{1, \ldots, m\}$
1 Construct the TSs $\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ for all robots
2 Construct the FSAs $\{\mathcal{A}_{\phi_1}, \cdots, \mathcal{A}_{\phi_{|D|}}\}$ corresponding to $\mathcal{D}$
3 Construct the FSAs $\{\mathcal{A}_{\psi_1}, \cdots, \mathcal{A}_{\psi_{|R|}}\}$ corresponding to $\mathcal{R}$
4 Construct the product automaton $\mathcal{P}$ as defined in Definition 3.4.1
5 Find the shortest trajectories from the initial condition $s_{0,\mathcal{P}}$ to the accepting
  states $F_{\mathcal{P}}$ of PA using Dijkstra's algorithm and select the optimal one $\mathbf{p}^*$ in
  terms of time relaxations
6 Project the optimal trajectory $\mathbf{p}^*$ onto $\mathcal{T}_1, \ldots, \mathcal{T}_m$ as defined in
  Definition 3.4.2
```

3.5 Robust Solution

In this section, the case of uncertain robots travel time is considered. The robots' motion along a transition e is uncertain with respect to its duration $\omega_v(e)$, and it is modeled as a non-deterministic value from the set $[\underline{\rho}\overline{\omega}, \overline{\rho}\overline{\omega}] \cap \mathbb{Z}_{>0}$ for $v \in \{1, \ldots, m\}$, where $\underline{\rho}, \overline{\rho} \in \mathbb{R}_{>0}$. Considering this uncertainty in the robot transition systems leads to an explosion in the number of states [53]. Thus, the construction of a PA and the solution of a game against the non-deterministic motion of the robots [107] becomes intractable even for small problems with few agents, demands, and rules. To avoid this issue, the nominal solution is taken and augmented with a synchronization procedure that guarantees the satisfaction of the specification φ at deployment. The procedure takes the form of a centralized on-line controller that produces a small number of synchronization events. In Section 3.6 it is shown that our on-line controller outperforms a baseline solution that enforces synchronization at each step.

3.5.1 On-line Controller

An on-line controller that exploits the robots' capabilities to recognize their arrival at states, and to communicate with the other robots, is proposed. The optimal satisfying trajectory obtained in the nominal case provides: (a) the individual robot trajectories, (b) for each transportation demand ϕ_i the robot it is assigned to and the state trajectory of $\mathcal{A}_{\phi_i}$ to satisfy it, and (c) for each synchronization rule $\widehat{\psi}_j$ the state trajectory of $\mathcal{A}_{\widehat{\psi}_j}$ to enforce it. Thus, from $\mathbf{p}^*$ it is possible to identify when, where (i.e., in which state), and which robots have to synchronize in accordance with both the demand assignments and the synchronization rules. Consequently, in order to satisfy φ, it is enough that when a robot reaches a point of interest (i.e $q \in Q$), it first communicates its position and then it awaits instructions to/from the online controller. Instructions can be either to proceed towards the next point of interest or to wait at the reached point. The latter occurs when either a robot that is involved in a synchronization rule reaches its synchronization point before the other robots involved in the same synchronization, or when a robot reaches a pick up position before a demand is ready to be transported.

Given the nominal solution obtained by Algorithm 17, extract $\mathbf{pick}_v$ and $\mathbf{sync}_v$ are extracted for each robot v. Vector $\mathbf{pick}_v$ contains as elements the pick up lower bound time in positions corresponding to the stages of $\mathbf{p}^*$ when a demand is supposed to be picked up, and zeros in all other positions. On the other side, $\mathbf{sync}_v$ is a finite sequence of sets. This contains as elements which robots v needs to synchronize with, in positions corresponding to stages of $\mathbf{p}^*$ when synchronization involving robot v is supposed to happen. All other positions contain an empty set. In the following, $\mathbf{sync}_{v,k}$ and $\mathbf{pick}_{v,k}$ denote the k-th elements of $\mathbf{sync}_v$ and $\mathbf{pick}_v$, respectively. The complete procedure for obtaining **sync** and **pick** is reported in Algorithm 18. In particular, lines 1-10 are used to compute **sync**. The variable $robotsInvolved$ is used to store the indexes of the robots involved in the j-th synchronization rule. Line 7 checks if the state $\widehat{s}_{j,k}$ belongs to accepting states $F_{\widehat{\psi}_j}$. In the affirmative case, for all robots involved in $\mathcal{R}_j$, $\mathbf{sync}_{v,k}$ is computed as reported in line 9. Note that, starting from the first k satisfying condition at line 7, the state $\widehat{s}_{j,k}$ will be an accepting state and therefore will not change. For this reason, a break is imposed at Line 10. Lines 11-16 describe the procedure to obtain **pick**. In particular, lines 13-16 look for the first k at which the i-th demand is assigned to a robot and compute $pick_{v,k}$ accordingly.

In order to satisfy φ during deployment in presence of uncertainty, at the k-th stage, if $\mathbf{sync}_v$ is not empty, the v-th robot is supposed to synchronize with robots in the set $\mathbf{sync}_{v,k}$. In case the other robots have not arrived yet, v is required to wait. Similarly, at the k-th stage, if $\mathbf{pick}_{v,k}$ is different from zero, this implies that the v-th robot is required to pick up a demand. In case the demand is not ready yet, v is required to wait. In all the other cases, robot v is free to go ahead and continue with the remaining states of $\mathbf{q}_v^*$. The complete procedure for computing the solution in presence of uncertainties is reported in Algorithm 19. In particular, the online controller waits continuously for events coming from robots which reached a state of interest, i.e. $q \in Q$ (see line 3). Whenever an event is detected, the corresponding robot number v and event time t are obtained and counter cnt_v increased (line 4). In case the corresponding $\mathbf{sync}_{v,cnt_v}$ is empty, the $\mathbf{pick}_{v,cnt_v}$ is checked and a proper command sent to the robot v, see lines 5-9. If, on the other hand, $\mathbf{sync}_{v,cnt_v}$ is not empty, the online controller has to check the position of all the other robots involved in the synchronization. If even just one of them is not at the synchronization position yet, the online controller sends the command to wait at the current position to robot v, see line 13-18. Thanks to the procedure summarized in Algorithm 19, it is possible to guarantee synchronization and demand satisfaction even in presence of robot travel time uncertainties.

Algorithm 18: Synchronization and Pick up Sequences

```
Input  : $\mathbf{p}^*$ - the nominal optimal trajectory of PA
Input  : $\mathcal{R}$ - the set of synchronization rules
Input  : $\mathcal{D}$ - the set of transport demands
Output : The sequence $\mathbf{sync}_v$ for each robot $v \in \{1, \ldots, m\}$
Output : The vector $\mathbf{pick}_v$ for each robot $v \in \{1, \ldots, m\}$
1  Set $\mathbf{sync}_{v,k} = \{\emptyset\}$ for $v \in \{1, \cdots, m\}$ and $k = 0, \cdots, |\mathbf{p}^*|$
2  for $j = 1, \cdots, |\mathcal{R}|$ do
3      robotsInvolved $= \emptyset$
4      for $i \in I_j$ do
5          robotsInvolved = robotsInvolved $\cup \{r_{i,|\mathbf{p}^*|}\}$
6      for $k = 0, \cdots, |\mathbf{p}^*|$ do
7          if $\widehat{s}_{j,k} \cap F_{\widehat{\psi}_j} \neq \emptyset$ then
8              for $v \in$ *robotsInvolved* do
9                  $\mathbf{sync}_{v,k}$ = robotsInvolved $\setminus \{v\}$
10             break
11 Set $\mathbf{pick}_v = \mathbf{0} \in \mathbb{R}^{|\mathbf{p}^*|}$ for $v \in \{1, \cdots, m\}$
12 for $i = 1, \cdots, |\mathcal{D}|$ do
13     for $k = 0, \cdots, |\mathbf{p}^*|$ do
14         if $r_{i,k} \cap \{1, \cdots, m\} \neq \emptyset$ then
15             pick$_{v,k}$ = the lower bound time of the demand $\mathcal{D}_i$
16             break
```

Algorithm 19: On-line controller

```
   Input  : The sequence $\mathbf{sync}_v$ for each robot $v \in \{1, \ldots, m\}$
   Input  : The vector $\mathbf{pick}_v$ for each robot $v \in \{1, \ldots, m\}$
1  Set a counter $cnt_v = 0$ for each robot $v \in \{1, ..., m\}$
2  while $True$ do
3      $[v, t] = waitForEvent()$ :
4      $cnt_v = cnt_v + 1$
5      if $\boldsymbol{sync}_{v,cnt_v} = \emptyset$ then
6          if $pick_{v,cnt_v} \leq t$ then
7              Send to $v$ the permission to move to the next node
8          else
9              Send to $v$ the command to wait at current node until time
               $\mathrm{pick}_{v,cnt_v}$
10     else
11         temp = True
12         for $r_v \in \boldsymbol{sync}_{v,cnt_v}$ do
13             if $cnt_{r_v} < cnt_v$ then
14                 Send to $v$ the command to wait at current node
15                 temp = False
16                 break
17         if $temp$ then
18             Send to $v$ the permission to move to the next node
```

3.6 Simulations and Results

The algorithms presented in this work are implemented in Python2.7 using the PyTWTL package [106]. All simulations have been running on a MacBook Pro equipped with i5 @2.09 GHz 64bit CPU system, 8 Gbytes of RAM and MacOS Mojave.

In this section, the simulation results in a semiconductor manufacturing fab setting are presented. A fab sector is shown in Figure 3.1. The end products of the semiconductor manufacturing process are integrated circuits. Chips are composed of several layers of chemical patterns that are imprinted on silicon wafers by machines. To obtain a layer, it is necessary that the wafer undertakes several steps, e.g., deposition, photolithography, and etching, performed by the machines. Since the cost of the machines is prohibitive, the wafers must revisit the machines multiple times to obtain the end products. Moreover, some steps must be performed at the same time on a machine (e.g. the diffusion furnaces [57]). In this scenario, a fleet of robots is employed to perform transportation demands between machines, and satisfy the synchronization rules of the fabrication process. These rules aims at avoiding time waste and thus maximizing efficiency in the fabrication process. The challenge in the semiconductor fab is to find the optimal routing and scheduling for meeting transport demands and the synchronization rules. Robots are responsible for the demand transportation among the various machines in the fab. The

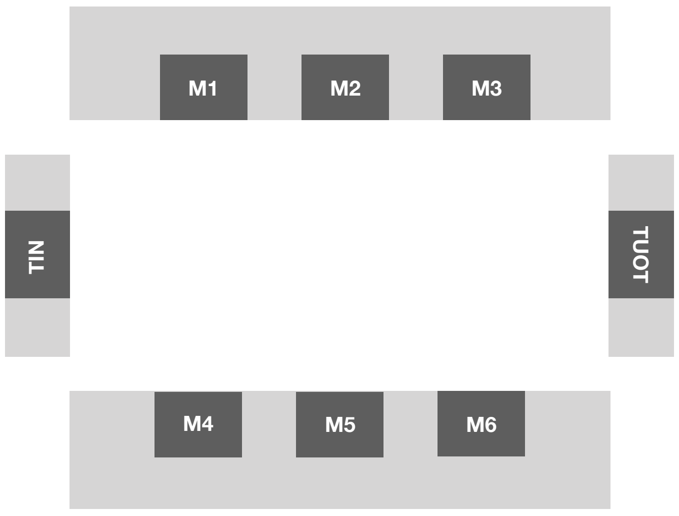


Figure 3.1: Semiconductor fab

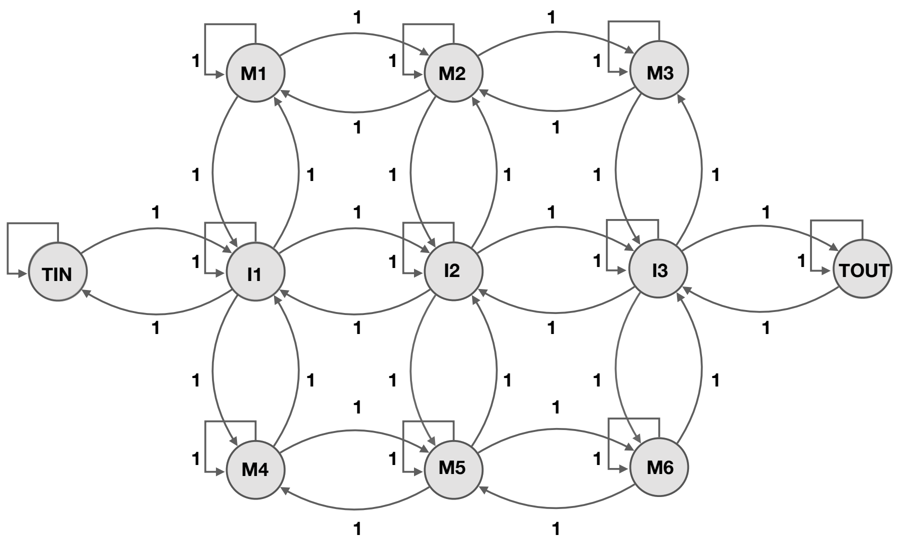


Figure 3.2: Robot TS

Figure 3.3: The environment graph (a) and the robot transition system (b)

scenario in Figure 3.3 is composed of 6 machines $M_1, \ldots, M_6$, 1 Transfer Point In T_{IN}, and 1 Transfer Point Out T_{OUT}. Demands processed in the fab sector appear at T_{IN} and are released at T_{OUT}, respectively. Each demand must follow its specific recipe based on following information available at its arrival: (a) pickup position, (b) delivery position, (c) wafer transportation time window within, and (d) any synchronization requirements with other demands.

The fab sector environment, which is shown in Figure 3.1, is abstracted into a weighted graph, where the nodes represent points of interest (machines and transfer points), the edges indicate the possibility of motion between nodes, and the weights represent the travel times associated with the edges. The motion model of each robot is abstracted as a transition system obtained from the environment graph by splitting each edge into a number of transitions equal to the corresponding edge's nominal travel time, see Figure 3.2. The set of propositions (AP) is $AP = \{M_1, M_2, M_3, M_4, M_5, M_6, T_{IN}, T_{OUT}\}$.

Two robots are used. These must fulfill five transportation demands subject to two synchronization rules shown in Table 3.1. All transportation demands and synchronization rules are captured by TWTL formulae, and translated to FSAs. In the nominal case, where there is no uncertainty in the robots travel times, the optimal

Table 3.1: Transport Demands ($\mathcal{D}$) and Synchronization Rules ($\mathcal{R}$)

$\mathcal{D}$	$\mathcal{R}$
$\phi_1 = \left[T_{IN} \cdot [M_5]^{[0,4]}\right]^{[0,6]}, \pi_1^{start} = T_{IN}$	$\psi_1 = [M_5]^{[0,6]}, I_1 = \{1, 3\}$
$\phi_2 = \left[M_6 \cdot [T_{OUT}]^{[0,5]}\right]^{[0,7]}, \pi_2^{start} = M_6$	$\psi_2 = [T_{OUT} \wedge M_3]^{[0,7]}, I_2 = \{2, 5\}$
$\phi_3 = \left[M_4 \cdot [M_5]^{[0,4]}\right]^{[0,6]}, \pi_3^{start} = M_4$	
$\phi_4 = \left[M_2 \cdot [M_4]^{[0,2]}\right]^{[0,4]}, \pi_4^{start} = M_2$	
$\phi_5 = \left[M_5 \cdot [M_3]^{[0,5]}\right]^{[0,7]}, \pi_5^{start} = M_5$	

trajectory $\mathbf{p}^*$ of $\mathcal{P}$ satisfying the transportation demands and synchronization rules is computed using Algorithm 17 and shown in Table 3.2. The minimal temporal relaxation vector associated with $\mathbf{p}^*$ is:

$$\boldsymbol{\tau} = \left(\boldsymbol{\tau}_{\phi_1}, \ldots, \boldsymbol{\tau}_{\phi_5}, \boldsymbol{\tau}_{\widehat{\psi}_1}, \boldsymbol{\tau}_{\widehat{\psi}_2}\right) = (1, 4, 1, 1, 4, 1, 4)$$

and the minimum linear temporal relaxation is $|\boldsymbol{\tau}|_{LTR} = 16$. Table 3.2 shows that the transportation demands ϕ_4, ϕ_3 and ϕ_5 are satisfied by robot 1, while ϕ_1 and ϕ_2 are satisfied by robot 2.

Table 3.3 shows the runtime performance of this approach. When the travel time of the robots is an uncertain quantity, the online controller is used to guarantee the satisfaction of the formula φ. In the following section, to evaluate the effectiveness of the robust solution to cope with travel time uncertainty, the online controller is compared to the baseline approach. In the latter, robots apply the nominal optimal solution and synchronize at each state during deployment. Both approaches are tested 20 times on the same specification φ. Each test is characterized by different uncertainties regarding travel time, i.e., each robot has its own travel time uncertainty. The upper and lower deviation values for each robot are $\overline{\rho} = 1.2$ and $\underline{\rho} = 0.7$, respectively.

Table 3.2: Optimal Nominal Solution $\mathbf{p}^*$

$\mathcal{T}^2$		ϕ_1		ϕ_2		ϕ_3		ϕ_4		ϕ_5		$\widehat{\psi}_1$	$\widehat{\psi}_2$
$\mathbf{q}_1$	$\mathbf{q}_2$	$\mathcal{A}_{\phi_1}$	r_i	$\mathcal{A}_{\phi_2}$	r_i	$\mathcal{A}_{\phi_3}$	r_i	$\mathcal{A}_{\phi_4}$	r_i	$\mathcal{A}_{\phi_5}$	r_i	$\mathcal{A}_{\widehat{\psi}_1}$	$\mathcal{A}_{\widehat{\psi}_2}$
M_1	T_{OUT}	s_0	0	s_0	0	s_0	0	s_0	0	s_0	0	s_0	s_0
M_1	I_3	s_0	0	s_0	0	s_0	0	s_0	0	s_0	0	s_0	s_0
M_2	I_2	s_0	0	s_0	0	s_0	0	s_1	1	s_0	0	s_0	s_0
I_2	I_1	s_0	0	s_0	0	s_0	0	s_1	1	s_0	0	s_0	s_0
I_2	T_{IN}	s_1	2	s_0	0	s_0	0	s_1	1	s_0	0	s_0	s_0
M_4	I_1	s_1	2	s_0	0	s_0	0	s_F	1	s_0	0	s_0	s_0
M_4	I_2	s_1	2	s_0	0	s_1	1	s_F	1	s_0	0	s_0	s_0
M_5	M_5	s_F	2	s_0	0	s_F	1	s_F	1	s_0	0	s_F	s_0
M_5	M_5	s_F	2	s_0	0	s_F	1	s_F	1	s_1	1	s_F	s_0
I_2	M_6	s_F	2	s_1	2	s_F	1	s_F	1	s_1	1	s_F	s_0
I_3	I_3	s_F	2	s_1	2	s_F	1	s_F	1	s_1	1	s_F	s_0
M_3	T_{OUT}	s_F	2	s_F	2	s_F	1	s_F	1	s_F	1	s_F	s_F

Table 3.3: Quantitative information on scalability

	Number of states	Number of transitions	Computational time
$\mathcal{T}$	11	39	8 ms
$\mathcal{T}^2$	121	1521	16 ms
$\mathcal{A}_{\phi_i}$	3	4	11 ms
$\mathcal{A}_{\psi_j}$	2	2	16 ms
$\mathcal{P}$	23185	295802	58.4 s

3.6.1 Baseline

The baseline approach enforces the nominal solution by synchronizing the transitions of all vehicles at all time steps. This implies that when a robot reaches the k-th state of its trajectory, it will remain there until all the robots assume their k-th trajectory position. When all the robots are on their k-th state, they can move to their $(k+1)$-th state.

The mean value of the linear temporal relaxation is 18.2, and its variance is 6.55 using the baseline approach.

3.6.2 Online Controller

Using the online controller we can minimize the synchronization points along the robots paths. The control algorithm exploits the $sync_v$ sequences and $pick_v$ vectors in order to guarantee the satisfaction of the global task. These are obtained using Algorithm 18, and are shown in Table 3.4. Once these quantities are obtained, Algorithm 19 is employed for deployment.

Table 3.4: $\mathbf{sync}_v$ and $\mathbf{pick}_v$ for each $v \in \{1, 2\}$

k	0	1	2	3	4	5	6	7	8	9	10	11
$sync_1$	{}	{}	{}	{}	{}	{}	{}	{2}	{}	{}	{}	{2}
$sync_2$	{}	{}	{}	{}	{}	{}	{}	{1}	{}	{}	{}	{1}
$pick_1$	0	0	0	0	0	0	0	0	0	0	0	0
$pick_2$	0	0	0	0	0	0	0	0	0	0	0	0

As can be seen in Table 3.4, robot 1 has to synchronize with robot 2 at $k = 7$ and $k = 11$. The mean value of the linear temporal relaxation using the online controller is 14.9, and its variance is 4.96. Thus, the robust approach provide a better performance than the baseline method: a 22% reduction in overall deadlines relaxation cost.

3.7 Conclusion

The problem of planning and scheduling for a fleet of robots with travel time uncertainty operating in manufacturing systems where some demands require to be satisfied at the same time is studied. Inspired by model checking techniques, a general nominal solution is proposed, where the nominal motion models are adopted. An online controller able to ensure synchronization during deployment have been proposed which improve on the performance of the nominal approach. The online controller is compared with the baseline nominal controller on simulated scenarios from the semiconductor manufacturing domain. The results show that we are able to reduce tasks violation by 22% using the online controller during deployment.

Chapter 4

UNIPV Test-bed

Contents

4.1 Introduction 84

4.2 Infrared Camera 85

4.2.1 Hardware 85

4.2.2 Image Formation 86

4.2.3 Perspective Projection 87

4.2.4 Image Discretization 90

4.2.5 Rigid Transformation 91

4.2.6 Camera Calibration 92

4.2.7 3D-2D Transformation 95

4.2.8 Planar Calibration 97

4.2.9 Distortion Model 97

4.2.10 Calibration Method 98

4.2.11 Evaluation of the Calibration 101

4.3 Working Environment 105

4.3.1 Charging Station 106

4.4 Robots 108

4.4.1 Mobile Robot Description 109

4.4.2 Hardware 109

4.4.3 Battery Monitoring 122
4.4.4 Connections 122
4.5 Software 124
4.5.1 Introduction 124
4.5.2 Visual Studio 125
4.5.3 Matlab 127
4.5.4 Raspberry 128
4.6 Low Level Controllers 137
4.6.1 Pure Pursuit Approach 138
4.6.2 Model Predictive Control 144
4.7 Results 177
4.7.1 Path Following 177
4.7.2 Path Following on Graph 179
4.7.3 Normal Operation 180
4.7.4 Multiple Robots 181
4.8 Conclusion 182
4.8.1 Future Development 182

4.1 Introduction

In order to evaluate the effectiveness of the developed strategies, a test-bed composed of small scale robots is developed at UNIPV laboratories. The position and orientation of the robots is detected by an infrared camera positioned on the ceiling of the lab. The robots are equipped with a clamp to simulate the drag and drop of the orders and two connectors located on the bottom of the vehicle to allow it to recharge the batteries to simulate the vehicle's recharge. The environment represents a graph with several pick-up and delivery locations. The work described in this chapter is the result of my activity as a co-supervisor, under the supervision of Prof. Raimondo, of the Bachelor and Master theses of: F. Napoli, L. Borrelli,

I. Triggiani, G. Saccani, S. Mercanti, F. Capelli, R. Aguzzi and S. Fasolato. In particular, my main contribution was both the design and programming of the testbed.

4.2 Infrared Camera

Artificial Vision Systems are devices able to capture images from the real world and to process them in order to obtain information. The main components of a system are: a lighting device, a camera equipped with an appropriate lens and an acquisition sensor, a computer, a software for processing and storing images, and a display monitor.
The lighting, which aims to highlight the object characteristics so that they are clearly seen by the camera, is an essential and often critical component of the system. In fact, an inadequate lighting may affect the subsequent operations.
The camera lens captures the images and presents them to the sensor in the form of light. The sensor converts the light into a digital image which is then sent to the processor to analyse them. An analytical model of the process described above is necessary in order to obtain information about the scene.
Once the model is defined, the calibration method will be studied. This is a method for the estimation, as accurate as possible, of the parameters that define the camera model. These parameters are necessary in order to correlate the points of the real world with the points of the image captured through the camera.

4.2.1 Hardware

The camera used is a Point Grey Grasshopper, see Figure 4.1, which has the following features: i) maximum resolution: 640 x 480 pixels; ii) frame rate up to 60 fps. The camera is equipped with an infrared filter that allows only the detection of infrared light. For the acquisition of the images a program is developed using Visual Studio and OpenCV libraries. The OpenCV library package (Open Source Computer Vision Library) is the most used in the field of artificial vision to process and recognize objects captured by visual sensors, and it is of crucial importance in

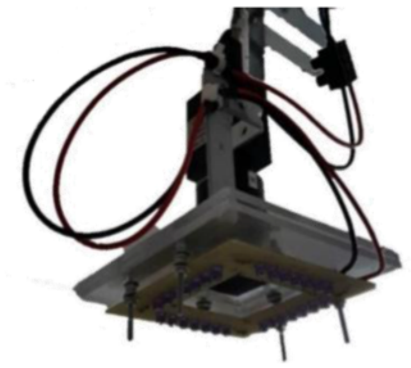


Figure 4.1: UNIPV camera

order to derive the two-dimensional coordinates in pixels of a point in space.

4.2.2 Image Formation

In the process of image formation, the two-dimensional image of 3D objects of the world is generated on a plane covered with photosensitive material (i.e. image plane) by the optical system [108]. The optical system of any image acquisition system characterizes the quality of the image generated, in terms of:

- *geometric resolution*: the ability to reproduce geometric details of objects;
- *geometric project*: the mode of propagation of light rays incident on the optical system;
- *intensity*: the intensity of projected light;
- *sharpness*: the image sharpness.

The transition from the three-dimensional world scene to an image seen from the camera is obtained by using the *perspective transformation.* This process can be modeled in a simple and effective way through the *pinhole* theoretical approximation, see Figure 4.2, which is the camera model adopted in this thesis. In the human eye, the optical system is the crystalline and the image plane is the retina, while in a camera the optical system is the lens and the image plane is the film or digital sensor.

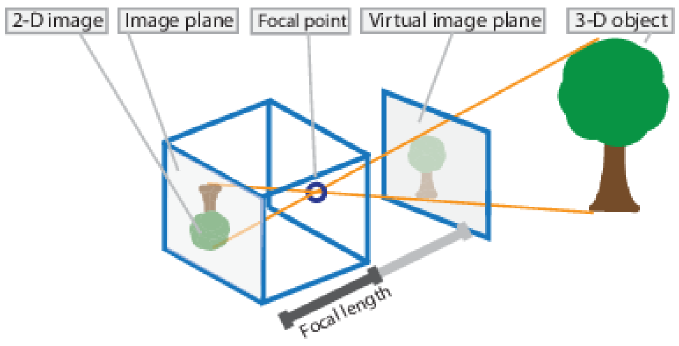


Figure 4.2: The pinhole model [6]

Pinhole Model

Generally, a camera is modeled by using the pinhole camera model, whose principle scheme is illustrated in Figure 4.2. The theoretical approximation of the pinhole model consists in a camera without lenses where light passes through an infinitesimal hole. The light rays coming from the objects in the 3D word pass through the optical system, which modifies their optical path and generates the image of the scene projected in the image plane, which is perpendicular to the optical axis.
The pinhole model is the simplest and ideal but, at the same time, it provides an acceptable approximation of the image formation process. Furthermore, it is convenient also from the mathematical and computational point of view.
Indeed, it is a simple perspective projection (or central projection) that constitutes the geometric model suited to schematize the formation of the image in a pinhole type camera.

4.2.3 Perspective Projection

Considering a reference system positioned in the optical center, i.e. the origin of the world system coincides with the center of projection C and the world's z-axis is aligned with the camera axis (optical axis), as it is shown in Figure 4.3, the following parameters are defined:

- M: world point of coordinates $[x, y, z]^T$
- m: perspective projection of M with image coordinates $[u, v]^T$
- I: image plane

- C: optical center
- *optical axis*: the line through C and perpendicular to the image plane I
- c: the intersection of the optical axis with the image plane is called *principal point* or *image center*
- f: focal length

The nonlinear equations that describe the projection of a 3D world point onto the image plane are obtained from a simple similarity between triangles and are given by:

$$\frac{u}{x} = \frac{v}{y} = -\frac{f}{z} \Longrightarrow \begin{cases} u = -\frac{fx}{z} \\ v = -\frac{fy}{z} \end{cases} \tag{4.1}$$

However, it is preferable to eliminate the inversion of the sign of the coordinates

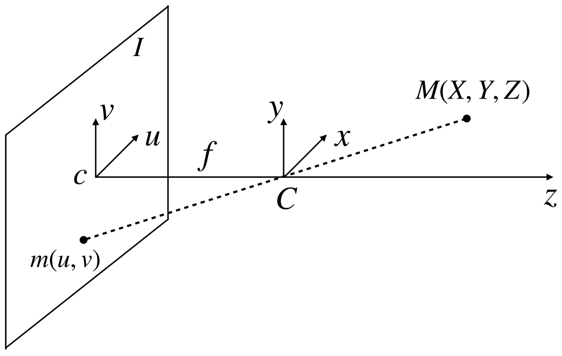


Figure 4.3: Perspective projection

which characterizes Equations 4.1. To achieve this, it is assumed that the image plane is located in front of the center of projection C at distance f, see Figure 4.4. Thus, Equations 4.1 can be rewritten as:

$$u = \frac{fx}{z}, \quad v = \frac{fy}{z} \tag{4.2}$$

Since the projection from the 3D world space to the 2D image plane is non-linear (due to the presence of z in the denominator, see Equations 4.4), it is possible to

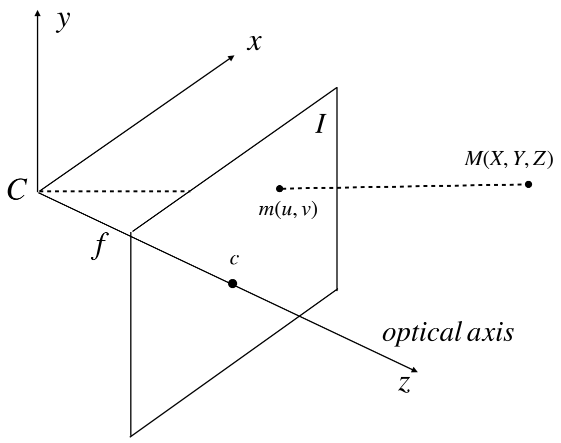


Figure 4.4: Perspective projection

express both points M and m using homogeneous coordinates, as:

$$M|1 = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad m|1 = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.3}$$

In this way, through Equations 4.2

$$z \cdot (m|1) = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = MPP \cdot M \tag{4.4}$$

where: MPP is the *perspective projection matrix.*

A realistic camera model that describes the transformation from 3D coordinates to pixel coordinates, as well as the perspective projection, must take into account the following two processes:

- The discretization (sampling) due to the sensor (seen as a two-dimensional array of pixels) and its position in relation to the optical axis;

- The isometric transformation between the world and the camera reference system.

4.2.4 Image Discretization

In order to process an image through a calculator, the continuous image must be converted into image points on a sampling grid. Thus, a rectangular array of equi-spaced samples called pixels is generated.

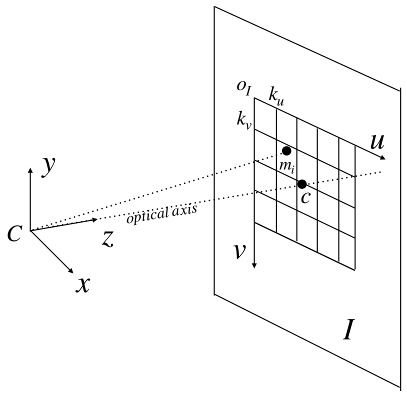


Figure 4.5: Digitization of the image

Discretization must take into account the fact that:

1. The optical center of the camera does not coincide with the physical center of the sensor but it has coordinates (c_x, c_y) in pixels;
2. The coordinates of a point in the standard reference system of the camera are measured in pixels. Therefore, a scale factor is introduced;
3. The shape of the pixels is not square. Thus, it is necessary to consider two different scale factors along the x and y axes which are indicated respectively with $k_u = \frac{1}{s_u}$ and $k_v = \frac{1}{s_v}$.

The above three points are taken into account by introducing both the translation of the optical center and the independent scaling of the u, v axes in Equation 4.2:

$$u = \frac{k_u f x}{z} + c_x \tag{4.5}$$

$$v = \frac{k_v f y}{z} + c_y \tag{4.6}$$

where: (c_x, c_y) are the the coordinates of the image center c, k_u and k_v are the units of the image reference system $o_i uv$.

As it can be noticed in Figure 4.5, the coordinates of a generic point m_i on the image plane I are expressed in pixels, while those of the point that generated m_i are expressed in meters. Therefore, the MPP can be rewritten as follows:

$$A = \begin{bmatrix} fk_u & 0 & c_x & 0 \\ 0 & fk_v & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.7}$$

where: $f_x = fk_u$ and $f_y = fk_v$ are the the size of the focal distance in terms of horizontal and vertical pixels, respectively.

4.2.5 Rigid Transformation

Generally, the world reference system does not coincide with the standard reference system of the camera. Therefore, it is necessary to introduce a rigid transformation that links the two reference systems, or a roto-translation. In general, three reference systems are identified:

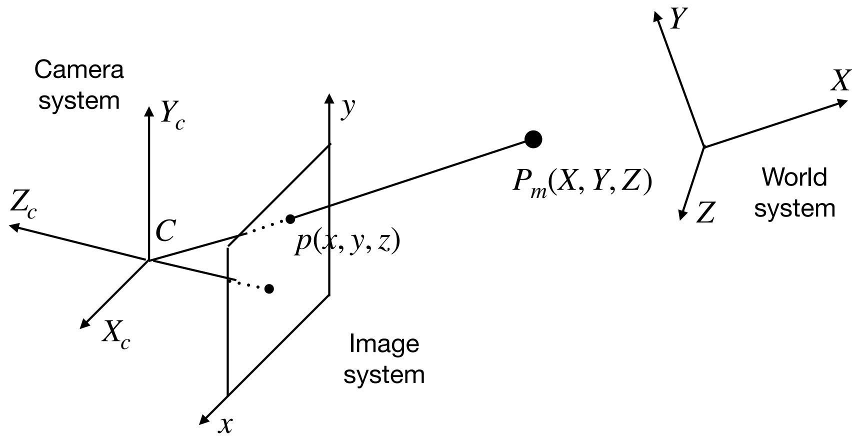


Figure 4.6: World system, Camera system and Image system

- *World system*: it is the reference system relative to the scene. It expresses the world coordinates of the points $P_m = [X, Y, Z]^T$ in space;

- *Camera system*: it is the reference system relative to the camera where the X_c axis represents the horizontal axis of the image plane while, Y_c represents the vertical axis. The Z_c axis coincides with the optical axis of the camera. The X_c, Y_c and Z_c axes form a right-handed reference system and are arranged so that the camera is oriented towards negative coordinates of the Z_c axis. The optical center C, located on the optical axis, represents the origin of the camera reference system. The optical center C is the origin of the system;

- *Image system*: it is the reference system of the camera sensor, in which the two-dimensional coordinates of the camera are expressed in pixel (u, v).

The world reference system is linked to the standard one by a rotation around the optical center (which can be modeled by an orthogonal matrix R) and by a shift (which can be modeled by a translation vector t).

4.2.6 Camera Calibration

Camera calibration is a necessary step in 3D computer vision to extract metric information from 2D images. Approximately three calibration categories can be considered:

- *3D reference object based calibration*: the camera calibration is performed by observing a 3D calibration object whose geometry in 3D space is known with great precision;

- *2D plane based calibration*: the techniques of this category require to observe a planar model shown in some guidelines;

- *1D object based calibration*: calibration objects are composed of a set of collinear points.

In this work, the 2D plane based calibration is adopted.

Calibration Parameters

The purpose of camera calibration is to determine the parameters of the transformation between an object in 3D space and the 2D image observed by the camera. In particular, this transformation depends on both the intrinsic and the extrinsic parameters.

Intrinsic Parameters They are parameters that characterize the single camera and do not depend on the view scene. Therefore, once they are estimated, they can be reused as long as the focal distance is fixed. These parameters, which are used to pass from the image plane to the camera reference system, are identified by the intrinsic matrix A

$$A = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.8}$$

Extrinsic Parameters They are parameters that define the position and orientation of the camera reference system in relation to the world reference system. Therefore, they introduce the roto-translation between the world system and the one integral with the camera. The rotation and translation matrices R and t are defined as follows

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

where:

$$
\begin{aligned}
r_{11} &= cos(\varphi)cos(\theta)\\
r_{12} &= cos(\varphi)sin(\theta)sin(\psi) - sin(\varphi)cos(\psi)\\
r_{13} &= cos(\varphi)sin(\theta)cos(\psi) + sin(\varphi)sin(\psi)\\
r_{21} &= sin(\varphi)cos(\theta)\\
r_{22} &= sin(\varphi)sin(\theta)sin(\psi) + cos(\varphi)cos(\psi)\\
r_{23} &= sin(\varphi)sin(\theta)cos(\psi) - cos(\varphi)sin(\psi)\\
r_{31} &= -sin(\theta)\\
r_{32} &= cos(\theta)sin(\psi)\\
r_{33} &= cos(\theta)cos(\psi)
\end{aligned}
$$

φ, θ, ψ are the Euler angles of rotation around the three axes

t_x, t_y, t_z are the 3D translation parameters in the three directions from the world system to the camera system

Although, the rotation matrix is composed of 9 elements, it has only 3 degrees of freedom, while the translation vector t has 3 parameters. Therefore, there are 6 extrinsic parameters and 4 intrinsic parameters. Hence, the total number of parameters to be identified are 10.

Distortion Coefficients

Camera imperfections due to individual lenses and how they are mounted produce distortions in the acquired images.
There are two types of distortions:

- **Radial Distortion**
  It is a symmetrical distortion by which the points of the image are distorted along the radial directions from a point called the center of distortion. This is caused by the imperfect shape of the lenses ("curvature" of the lenses). In the following Section, the parameters k indicate the distortion.
  The magnification produced by the peripheral parts of a lens is different from

that produced by the central parts. The parts of an object that are seen through the periphery of positive lenses appear more enlarged, while they appear smaller in case of negative lenses. The overall effect is to make the lines that are actually straight appear curved.

In particular, a square seen through a positive lens takes the form of a "pincushion" ($k_1 < 0$), while seen through a negative lens it takes the form of a "barrel" ($k_1 > 0$), see Figure 4.7.

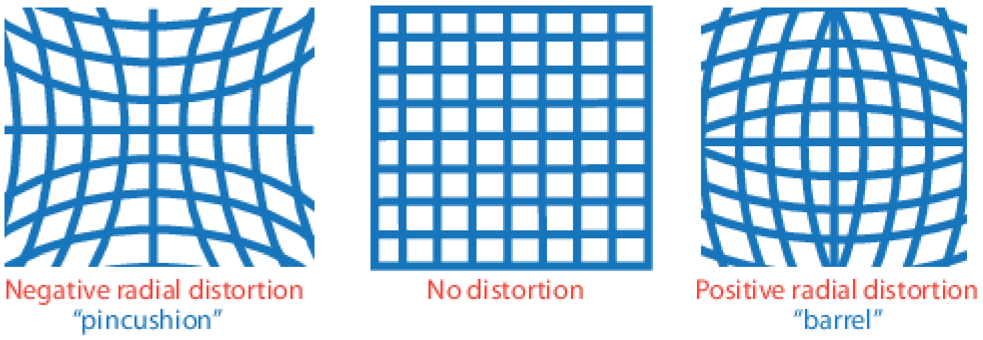


Figure 4.7: Effects of radial distortion [6]

- **Tangential Distortion**

  This is usually caused by an improper assembly of the lenses; the parameters are indicated with the letter p.

Generally, the image points are distorted both in the radial and tangential direction. It is very common that the mathematical function that models distortion is totally dominated by radial components, especially by the first term, while secondary effects are introduced by tangential distortions. Distortion coefficients do not depend on the scene displayed. Therefore, they belong to the intrinsic parameters of the camera and they remain the same regardless of the resolution of the acquired image.

4.2.7 3D-2D Transformation

A 2D point m is indicated by $[u, v]$. A 3D M point is indicated by $[X, Y, Z]$. According to the *pinhole* model, initially neglecting the distortion, the image of a 3D point, which is detected in the corresponding 2D point, is formed by a straight line that passes from M through the optical center C and intersects the image plane. In Figure 4.8, the virtual image plane, which is mathematically equivalent to the image plane, is considered as the image plane.

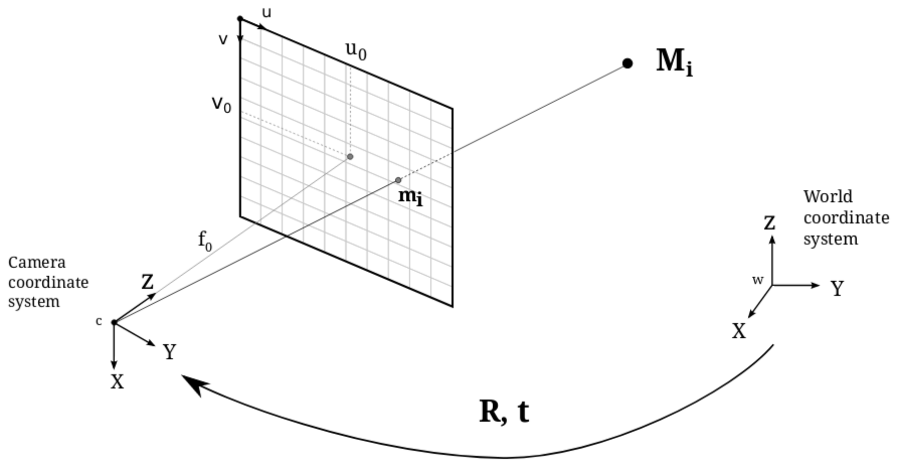


Figure 4.8: Pinhole model [7]

The relation between the 3D point M and its projection onto the image plane m is given by:

$$s\begin{bmatrix} m \\ 1 \end{bmatrix} = A[R|t]\begin{bmatrix} M \\ 1 \end{bmatrix} = P\begin{bmatrix} M \\ 1 \end{bmatrix} \tag{4.9}$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{array}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.10}$$

where s is an arbitrary scale factor, $[R|t]$ is the overall matrix of extrinsic parameters, A is the intrinsic matrix of the camera, (c_x, c_y) the coordinates in pixels of the image center and (f_x, f_y) are the focal lengths expressed in the pixels.
The matrix P is called *camera projection matrix*, which mixes intrinsic and extrinsic parameters.
By using Equation 4.4, for the case where $z \neq 0$, the transformation that is described

by Equation 4.10 can be rewritten as:

$$u = f_x \cdot x' + c_x \tag{4.11}$$

$$v = f_y \cdot y' + c_y \tag{4.12}$$

where $x' = \frac{x}{z}$ and $y' = \frac{y}{z}$.

4.2.8 Planar Calibration

Without loss of generality, it is assumed that the model plane is at the global coordinate $Z = 0$. The transformations are therefore:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{array} \right] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{4.13}$$

4.2.9 Distortion Model

Sometimes the linear projective Equation 4.12 is not enough. Especially when low-end cameras (such as WebCams) or wide-angle cameras are used, it is necessary to take into account the lens distortion.

These phenomena are modeled through a non-linear relationship between the points actually observed on the image plane

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = L(r) \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} dx' \\ dy' \end{bmatrix} \tag{4.14}$$

where: $L(r)$ is the radial distortion function which depends on the distance (r) from the center of distortion, i.e. $r = \sqrt{(x' - x_c)^2 + (y' - y_c)^2}$ for $r > 0$. This non-linear function $(L(r))$ is typically approximated by using Taylor's series development (up to the n-th order, depending on the desired precision):

$$L(r) = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots \tag{4.15}$$

Instead, the tangential distortion vector is approximated with:

$$\begin{bmatrix} dx' \\ dy' \end{bmatrix} = \begin{bmatrix} 2p_1 x'y' + p_2(r^2 + 2x'^2) \\ p_1(r^2 + 2\tilde{y'}^2) + 2p_2 x'y' \end{bmatrix} \tag{4.16}$$

The coefficients for radial distortion correction (i.e. $k_1, k_2, \cdots, k_n$) together with the center of radial distortion (x_c, y_c) and the two tangential distortion coefficients p_1 and p_2 widen and complete the set of intrinsic parameters of the standard model of a camera. Typically, for simplicity, it is assumed that the center of radial distortion coincides with the center of the image.
Optical distortion is usually modeled as a transformation that occurs after projecting 3D coordinates onto the image plane. After that, the intrinsic matrix applies an affine transformation to the image, translating the physical coordinates on the image plane into pixel coordinates.

4.2.10 Calibration Method

In order to reconstruct the computerized three-dimensional scene, the camera calibration is of crucial importance. Solving a calibration problem means determining all the geometric parameters of the camera such as:

- focal length;
- coordinates of the principal point;
- distortion coefficients;
- position of the camera reference system in relation to the absolute world reference system.

The idea behind each calibration algorithm is to rewrite the equations of the perspective projection as a linear system with the parameters as unknowns, since

the correspondence between 2D projections and 3D points of known coordinates is known. In particular, if OpenCV is used, in order to know a certain number of correspondences in advance, a pattern of known form is used. In this case a chessboard is used, on which it is easy, once the image is captured, to identify some particular points.
Some sources of systematic error, however, cannot be eliminated, so the identification is never exact but it consists of an estimate to be made as accurately as possible.

OpenCV

The calibration methodology used through *OpenCV* is the 2D type. Indeed, it is based on the recognition of some characteristic points of a simple geometric element, in this case a chessboard, which is captured in different positions on the worktop. The points in question are the internal vertices (corner) of the cells of the board itself.

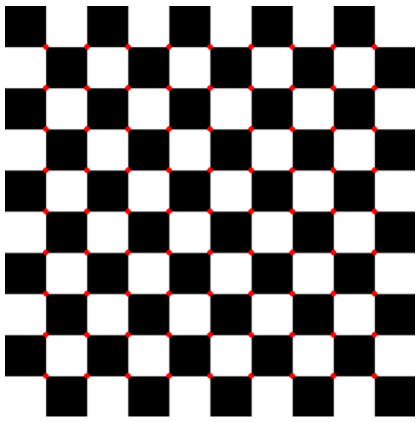


Figure 4.9: Checkerboard pattern with highlighted corners

Image Acquisition

The image acquisition and corner recognition operation can be performed in two ways. The first relies on a “live view”, which consists in using the calibration program to detect several frames of the checkerboard. This is repositioned after each time the corner detection is performed.
The second alternative is to take a series of photos of the checkerboard in different positions (usually at least 10), which must be taken before the calibration program

is run. Then, the program is run and it takes care of recognizing the corners. Since

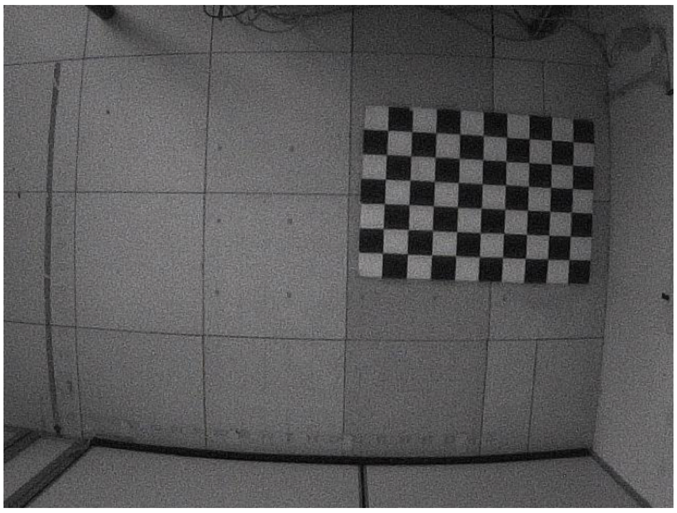


Figure 4.10: Example of acquired image

the movement of the checkerboard should be manual and the presence of the subject would alter some characteristics of the image, such as brightness, in this work the second alternative is used.

Sofware

As already mentioned, to recognize the corners a C ++ language code is used, which is available as a *source code* and uses the libraries provided by *OpenCV*.

The calibration is performed using the *"calibrateCamera"* function, the virtual

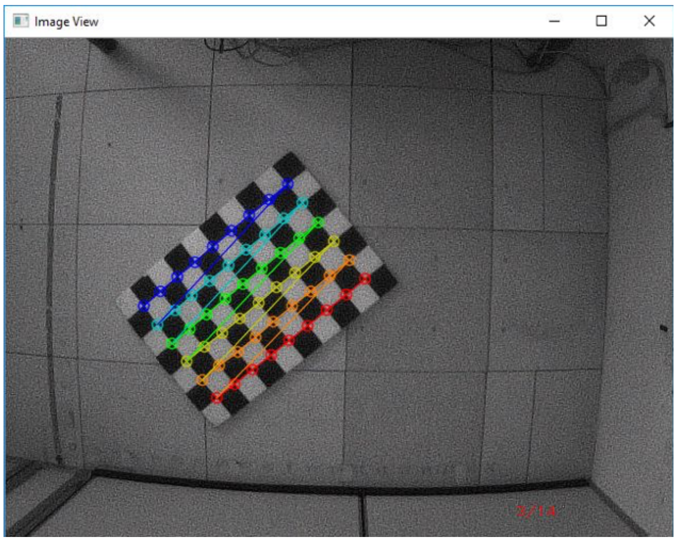


Figure 4.11: Corner recognition in distorted image

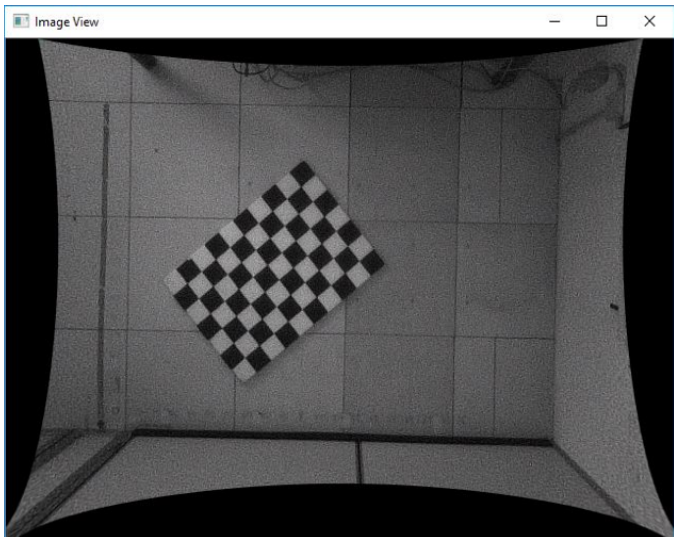


Figure 4.12: Correct image

points *"objectPoints"* are automatically referenced to the real points *"imagePoints"* in order to extract both the intrinsic parameters *"cameraMatrix"* and the distortion parameters *"distCoeffs"*. The position (orientation and position respectively) of the camera in relation to each checkerboard detected are contained in the vectors *"rvecs"* and *"tvecs"*.
By entering some data such as the number of corners in the long and short side of the board and the .xml file with the list of images to be used, the program performs the calibration and returns a .yml file that contains:

- the intrinsic and extrinsic parameters of the camera;
- the coordinates in pixels of the corners and their calculation error.

4.2.11 Evaluation of the Calibration

Thanks to what has been described above, taking a system oriented like the checkerboard in one of the images as a world reference system, it is possible to obtain a 2D reconstruction of the plan. Indeed, by manually measuring the spatial coordinates of some points on the plane, it is possible to find the pixel coordinates by using Equation 4.2 and vice versa. Therefore, in order to evaluate the camera calibration the following procedure can be adopted:

- *choice of the world reference system*: The world reference system is chosen as in Figure 4.13;

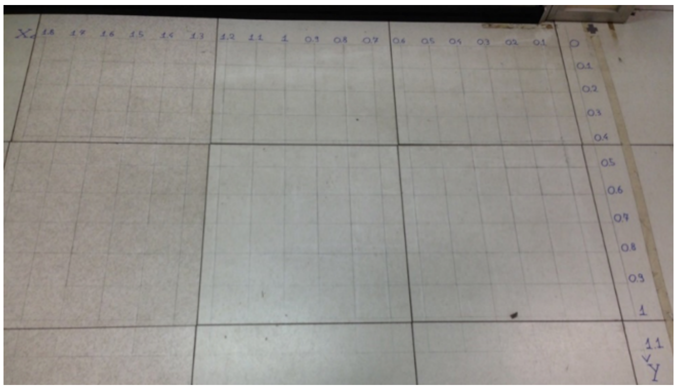
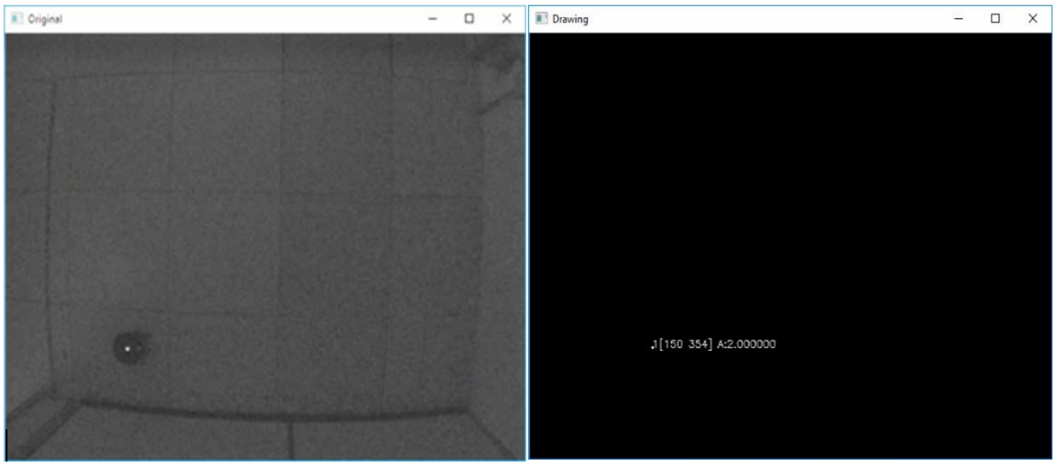


Figure 4.13: World reference system

- *find the coordinates (u, v) of the point by software*: to obtain the coordinates (u, v) of the points a program implemented in C ++ is used, through which the camera identifies an infrared LED located at the point of interest on the plane, see Figure 4.14;



```
1[150 354] A:2.000000
```

Figure 4.14: Recognition software

- *find (X, Y, Z) through equations*: as regards the setting of Equations 4.2, different values of t_z from *OpenCV* are obtained, the resolution field of the camera is divided into nine parts, each of which has its own value of t_z. This is obtained by arithmetic mean of the values provided by the program only for frames where the chessboard is contained in the part of interest. Thus, the adopted method for calibration is summarized as follows:

- divide the viewing space into portions: in the case in question it is divided into nine parts as shown in Figure 4.15;
- capture the board at least twice per portion, so that it is possible to calculate an arithmetic average of the values provided by the program and obtain a better approximation;
- apply, in Equations 4.2, the values of t_z based on the part in which the point of interest is located.

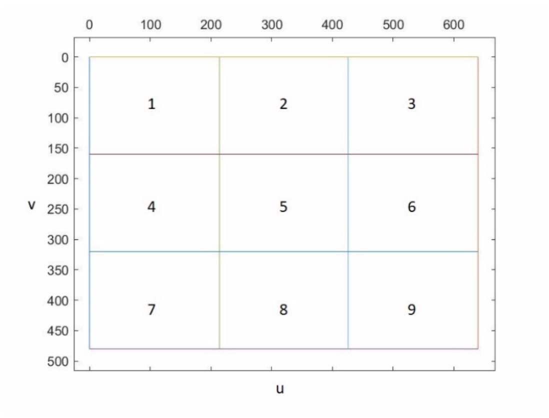
Then, the non linear equations that allow the inverse transformation from (u, v) to (X, Y, Z) are solved by imposing as (u, v) the values obtained in the previous step. Note that it is possible to solve the inverse equations 4.2 since in the considered case it is assumed that the worktop is set at $Z = 0$;

- *manual measurement of the actual coordinates (X, Y, Z)*
- *analysis of the correspondences between the two sets of values found*

Analysis of the Results

In this section an analysis of the accuracy of the procedure followed in Section 4.2.11 is conducted. In particular, the manual measurements and those obtained with the program will be compared.
As can be noticed from Table 4.1, good correspondences are obtained. Moreover, it is possible to note that the error is distributed quite randomly in the worktop area. This means that the obtained discrepancy does not depend on the distance from the image center. However, it is true that the points that present a major error are located in areas where the distortion is considerable.
In general, the deviation may depend on the fact that the measure is affected by some uncertainties:

- average error of point projection due to the calibration program;



$1\begin{cases} 0 \leq u \leq 214 \\ 0 \leq v \leq 160 \end{cases}$ $2\begin{cases} 214 < u < 426 \\ 0 \leq v \leq 160 \end{cases}$ $3\begin{cases} 226 \leq u \leq 640 \\ 0 \leq v \leq 160 \end{cases}$

$4\begin{cases} 0 \leq u \leq 214 \\ 160 < v < 320 \end{cases}$ $5\begin{cases} 214 < u < 426 \\ 160 < v < 320 \end{cases}$ $6\begin{cases} 426 \leq u \leq 640 \\ 160 < v < 320 \end{cases}$

$7\begin{cases} 0 \leq u \leq 214 \\ 320 \leq v \leq 480 \end{cases}$ $8\begin{cases} 214 < u < 426 \\ 320 \leq v \leq 480 \end{cases}$ $9\begin{cases} 426 \leq u \leq 640 \\ 320 \leq v \leq 480 \end{cases}$

Figure 4.15: Subdivision of worktop with reference to the coordinates corresponding to each zone

- error in the calculation of the coordinates (u, v) through the LED recognition program;
- error in the measurement of the coordinates in cm of the points;
- error in the positioning of the LEDs, which may not coincide perfectly with the point in question;
- approximation of the value t_z, obtained by arithmetic mean.

The following table shows the matches obtained for some of the considered points. The position of the points (u, v), on the worktop, whose coordinates have been calculated (X, Y, Z) is shown in Figure 4.16. These points, indicated in pixel coordinates, are distinguished by a color based on the precision obtained (see the legend on the image). The yellow point is the image center of coordinates (c_x, c_y) calculated with the calibration.

(u, v)	measured (X, Y, Z)	obtained (X, Y, Z)
455, 410	30, 110	30, 111
156, 83	210, 280	212, 279
183, 300	90, 260	91, 260
70, 340	0, 340	0, 340
614, 66	230, 10	229, 9
406, 137	180, 140	180, 140
605, 230	130, 20	130, 22
226, 471	-10, 240	-10, 240

Table 4.1: Correspondence between coordinates measured and found analytically

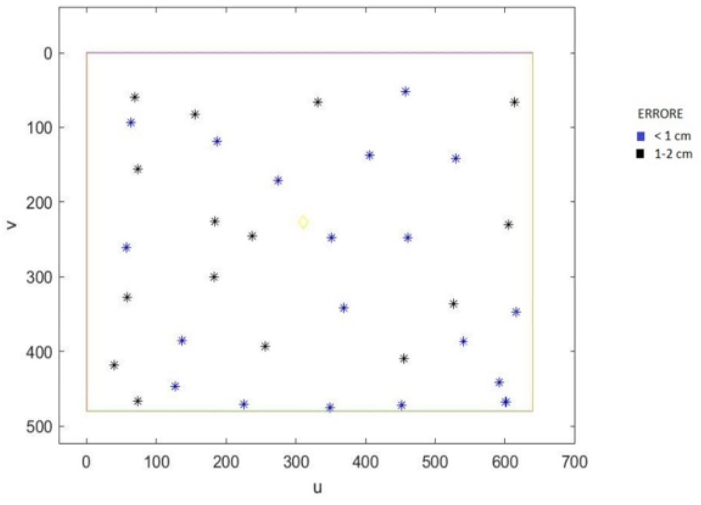


Figure 4.16: Accuracy of analyzed points

4.3 Working Environment

As previously mentioned, the testbed consists in controlling the trajectory of some robots, which have the task of picking up the objects from an initial position and of delivering them to a final position. All the available paths can be abstracted by using a graph $\mathcal{G} = (\mathcal{Q}, \Delta, \omega)$, where $\mathcal{Q} = \{1, \cdots, 15\}$ is the set of graph nodes that represents the location of interest (i.e. machines, interconnection nodes etc.) while, Δ is set of edges that captures the feasible motions between the locations of interest with the fixed travel time ω. For each node of the graph the corresponding world coordinates (x, y) are known. In particular, our scenario is composed of 15 nodes, organized in 3 rows of 5 nodes, as can be seen from Figure 4.17. The nodes represent the different workstations where vehicles must pass and collect or deposit objects in the production plant.

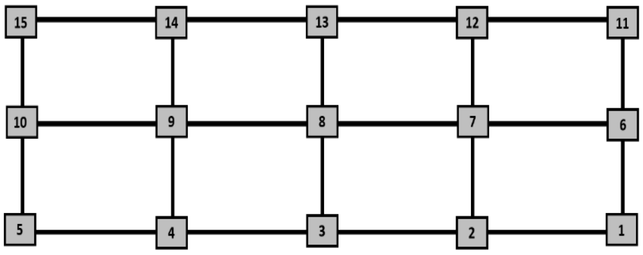


Figure 4.17: The worktop graph $\mathcal{G}$

The working grid is obtained on a plywood panel with dimensions 2 x 1 m. The plywood panel is covered with black sheets so as to eliminate the possibility that the camera incorrectly detects reflections. The graph node are 30 cm distant from one another, both vertically and horizontally. In Figure 4.18 how the worktop actually looks is shown.

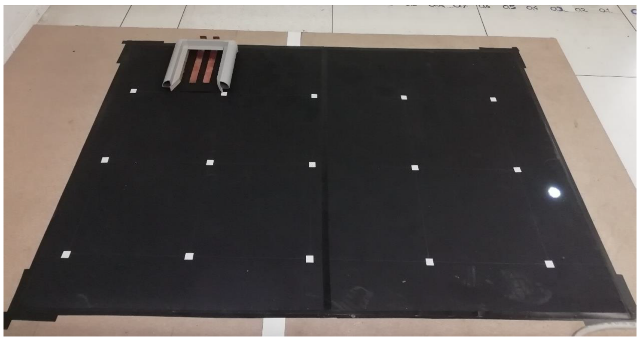


Figure 4.18: Support surface

4.3.1 Charging Station

As stated, the purpose of this project is to simulate the activity of vehicles used for transporting semi-finished products within the semiconductor production process where it is necessary to use robots as efficiently as possible.

For these reasons, in order to reduce the wasting time inside the plants, it is convenient to automate the activities, which can be carried out more efficiently.

In particular, the recharging station solves the inconvenience of manually recharging the exhausted batteries.

Charging Station Structure

The purpose of the charging station, as previously mentioned, is to recharge the batteries present on the vehicles. The connection between the poles of the battery and the special recharging device is achieved by using the sliding contacts under the vehicles and the copper slides in the charging station, see Figure 4.19 and Figure 4.28. The connection that is created between the stripped contacts on the

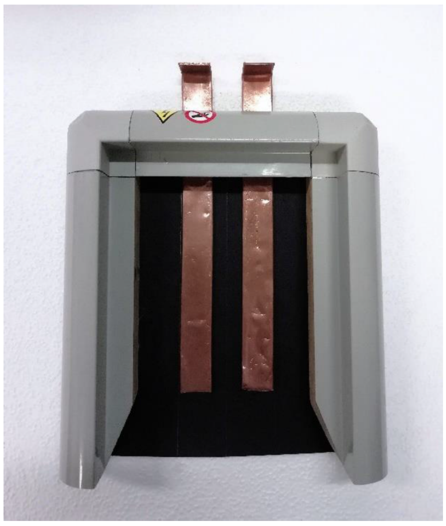


Figure 4.19: Charging station

bottom of the frame and the slides in the structure is used, when the robot enters the station.
Essentially, the charging station is composed of two elements: two copper slides and an external structure.
The external structure of the station is built using pieces of aluminum section, its task is to convey the vehicle to the copper slides, so as to compensate for any inaccuracies in the trajectory.
The station is positioned in a corner of the work area, so as not to occupy the space that is normally used to perform the simulations, see Figure 4.18.

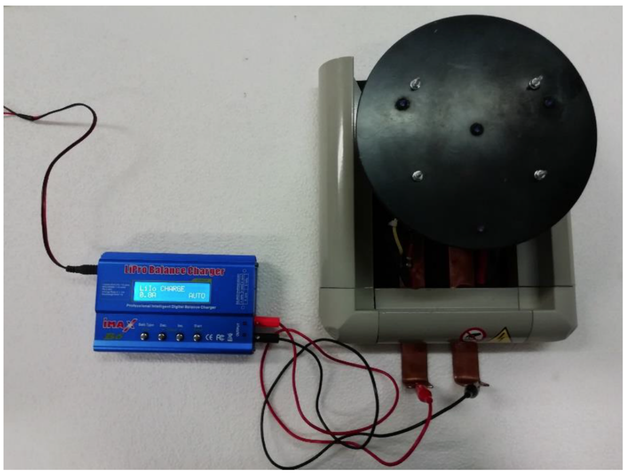


Figure 4.20: Charging station with robot

4.4 Robots

The design of autonomous mobile robots capable of intelligent motion and action without requiring a teleoperator control involves the integration of different hardware devices and software implementation.

The mobile robots used for this project are small scale robots and they have two motors, one per wheel. The Arduino Bluno Nano is used to control engine speed. In this work, the Arduino programmable card is used to correlate the PWM impulses sent to the motors and their speed.

The work described in this chapter relies on the capability to move the robot in a controlled space. A camera, some markers and the Visual Studio integrated development environment (IDE) are used. This make it possible to calculate the X, Y coordinates of the markers placed on the robot in real time. The PC on which Visual Studio is installed is connected to an infrared camera, placed on the ceiling in order to completely frame the worktop.

In this section, the features of the mobile robot, the Visual Studio development environment and the hardware devices that allow to recognize and track the robot in real time will be described.

4.4.1 Mobile Robot Description

A robot can be defined as "a mechanical device which performs automated tasks, either according to direct human supervision, a pre-defined program, or a set of general guidelines, using artificial intelligence techniques" [109]. In this project a *differential drive* robot is used. Mobile robots are, by definition, those which have the ability to move in terrestrial, marine or air environments. The ones used in this case are shown in Figure 4.21.

The indoor mobile robot has two main wheels, each of them is linked to its own

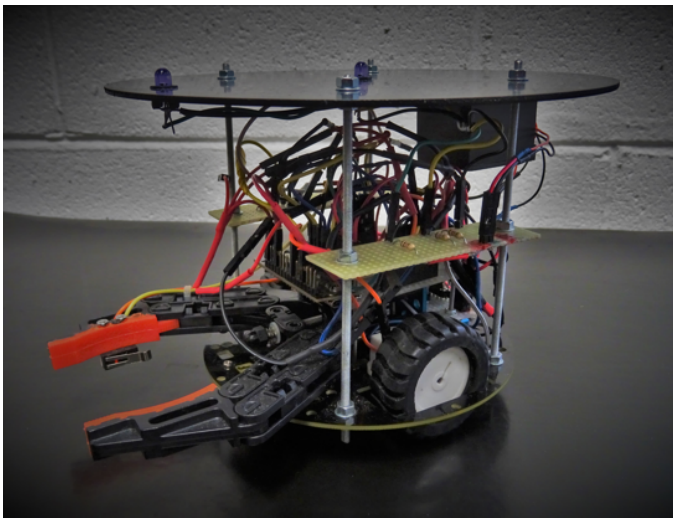


Figure 4.21: One of the mobile robots used in this project

motor and two other specific wheels placed in the lower part to passively roll along while preventing the robot from falling over. The batteries needed by the robot are placed in a resin-coated glass battery box, located below the roof of the robot.

4.4.2 Hardware

Each robot required for the test-bed is made by using the *"2WD miniQ Robot Chassis"* kit, an Arduino Bluno Nano board for logic control and a clamp for gripping the objects, from a Makeblock kit. The robot kit and the Arduino board are produced by DFRobot and they are purchased online. The power is supplied by two 3.7 V lithium-ion (Li-ion) batteries and 1000 mAh each, the batteries are connected in series in order to have a supply voltage of 7.4 V. An on / off switch is added to turn the control board on and off. The clamp is equipped with a switch to get feedback when a object is taken. Copper brushes are mounted on each robot to allow it to recharge automatically. For the detection by the camera, infrared leds are used that emit an infrared light, fixed on the robot roof. The control of the robot trajectories is implemented on a Raspberry, which is a single-board computer appropriately interfaced with the other systems (camera and Arduino) in order to ensure the correct functioning of the robots.

Kit DFRobot

The kit used for building the UNIPV robots is the DFRobot kit, which is composed of:

- two wheels which have a 42 mm diameter and 19 mm width;
- two electric direct current motors with nominal voltage equal to 6 V. The rotation speed of the shaft at 6 V is 13000 RPM. A speed reducer is used with the gear ratio equal to 50:1. The speed reducer allows to decrease the rotation speed to 260 RPM.
- The maximum torque at blocked rotor is 3.82 N·cm.
- One chassis. The main role of the chassis is to provide support and attachment for the principal functional components of the robot. The chassis diameter is 122 mm without wheel and balls bearing.

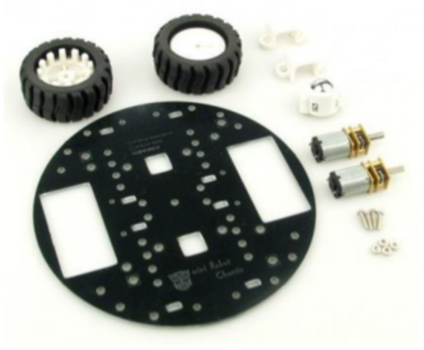


Figure 4.22: Kit DFRobot

Clamp

In order to simulate the drag and drop of orders inside an industrial plant, it is necessary to equip the robots with a clamp. To this end, a clamp is mounted on the chassis. The clamp is driven by an electric motor controlled by Arduino. A Normally Open limit switch (NO) is inserted in one of the two clamp arms. It acts as feedback to know when the package is collected. Indeed, when the clamp is closing on the package the switch is in contact with the package causing it to close. Thanks to the connection between Arduino and the limit switch, when this switch is closed it sends Arduino an electric input in order to deactivate the clamp.

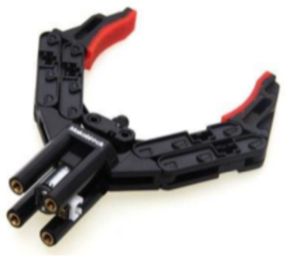


Figure 4.23: Robot clamp

Roof

The robots detection relies on infrared light. For this purpose, leds are positioned on the top of the robots. In order to have a good camera detection, a dark color is chosen for the roofs. These are made of fiberglass and have small differences to distinguish the configuration of leds of different vehicles, see Figure 4.24 4.25 4.26

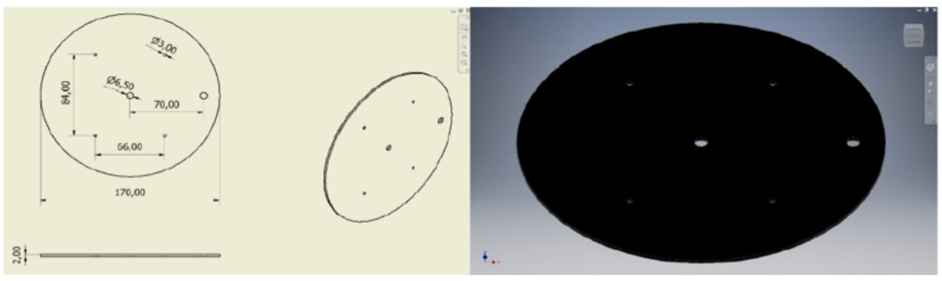


Figure 4.24: Roof of the robot 1

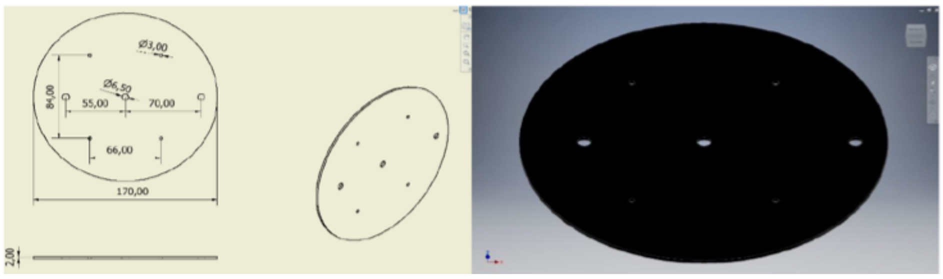


Figure 4.25: Roof of the robot 2

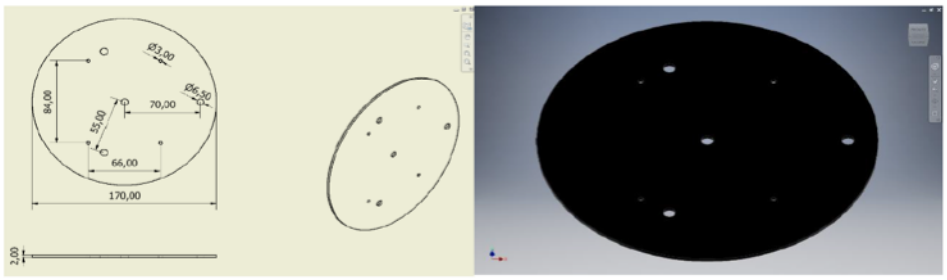


Figure 4.26: Roof of the robot 3

Battery Cases

A battery case made of fiberglass is necessary in order to guarantee the safety and correct use of the battery. Power is supplied by two lithium-ion (Li-ion) batteries of 3.7V and 1000 mAh each, connected in series to have a supply voltage of 7.4V.

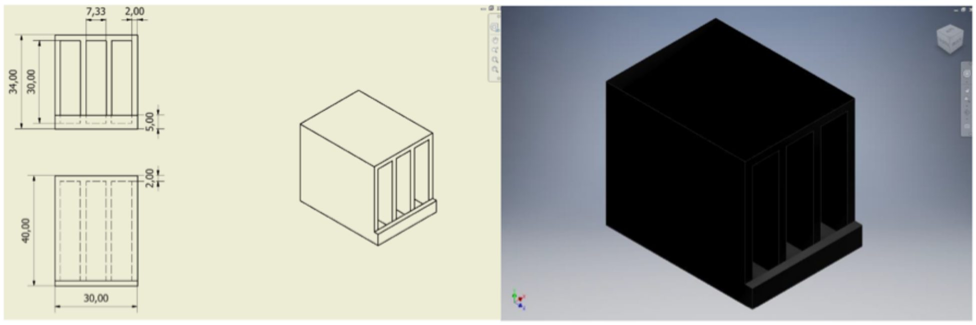


Figure 4.27: Battery cases

Sliding Contacts

The sliding contacts are made with a copper braid with a section of about 2 mm^2 and they allow the physical connection of the robot with the copper strips in the charging station, see Figure 4.28. The contacts are fixed in the lower part of the chassis, and are connected to the two poles of the batteries. In this way, when the robot enters the charging station, a potential difference is applied directly to the battery poles and then the battery starts to recharge.

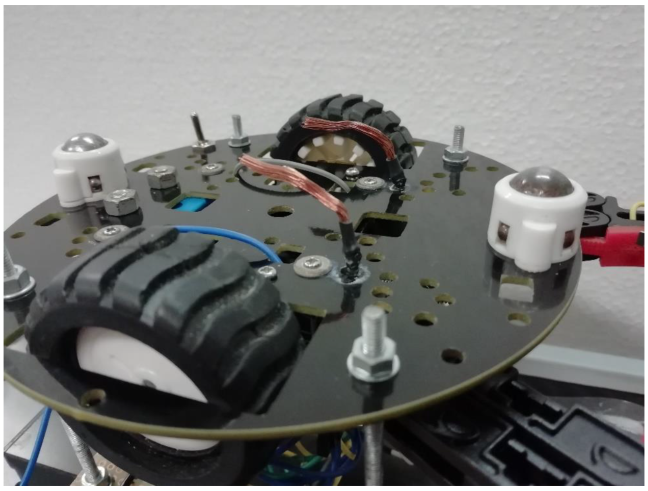


Figure 4.28: Sliding contacts

Raspberry PI 3 Model B

The Raspberry PI 3 is a single-board computer, which is an electronic card that implements an entire computer. It is equipped with:

- a microSD port where the appropriate card is inserted, on which the chosen operating system (generally Raspbian) has previously been saved;
- an Ethernet port, as it will be seen in Section 4.4.4, is indispensable for the purposes of this project because from this port the data coming from the camera are received;
- four USBs 2.0 to which mouse, keyboard and a 4.0 USB Bluetooth dongle are connected. These are used to send data to Arduino, as can be seen in Section 4.4.4;
- a HDMI port to which a monitor is connected;
- 40 GPIOs (General Purpose Input/Output), which are a series of pins that can be used as inputs, to read digital signals sent from other devices or from other parts of the circuit, or as an output to control other devices

Figure 4.29: Raspberry PI 3 Model B [8]

Arduino Bluno Nano

The Arduino Bluno Nano board is based on the Arduino Uno: more precisely, the BT 4.0 module (BLE - Bluetooth Low Energy) is integrated in the Arduino Uno board. This feature is very important for the purposes of this project, indeed data can be exchanged between Raspberry and Arduino via Bluetooth, without, the presence of annoying and cumbersome cables. Arduino Bluno Nano can be easily programmed using an IDE (Integrated Development Environment), which uses a programming language called Wiring, based on C ++. Arduino must be powered in DC voltage between 6 V and 12 V through the appropriate pins (positive terminal to the pin V_{in}, negative terminal to the pin GND). The other pins on the electronic board are:

- $5\ V \,\&\, 3.3\ V$: stabilized output voltage at 5 V and 3.3 V, respectively;
- $Analog(A0-A7)$ analog pins, they can be used to read the signal, for example, from a sensor and convert it to a digital signal. For the purposes of this project an analog pin is used to read the voltage value of the Arduino power supply battery and to read the status of the limit switch installed on the clamp;
- $Digital(D1-D13)$ digital pins used both as input and output. For the purposes of this project, some of these pins are particularly important, i.e.

the PWM (Pulse Width Modulation) pins, which are the pins through which it is possible to vary the speed of rotation of the robot motors.

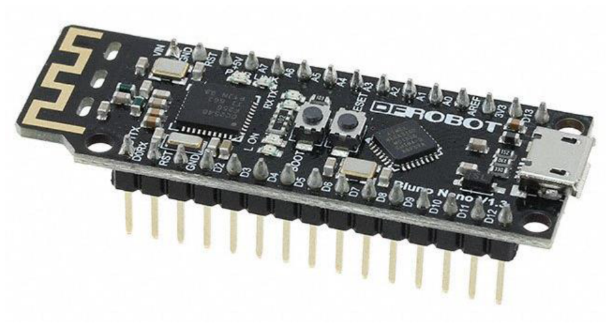


Figure 4.30: Arduino Bluno Nano [9]

Pulse Width Modulation - PWM

The term Pulse Width Modulation refers to a power regulation technique, consisting in modulating the width, the time duration of a series of pulses. For this project PWM signals are used to provide variable speed control for the two robot motors. By using the classic linear adjustment systems, the speed of a motor is regulated by acting on the supply voltage. This system has two drawbacks: the first is that, in order to reduce the power reaching the motor, a resistance in series that causes the necessary voltage drop, with relative waste of power and unwanted heat production. The second is that, at very low voltages, the motor might not produce sufficient torque and therefore it would not start.
The PWM technique acts not on tension, but on time; indeed an electric motor works according to average value of the power that reaches it. If the motor is powered with a series of impulses that follow one another with sufficient speed, thanks to its mechanical inertia, it will run with a continuous motion, proportional to the average value of these pulses [110].
Figure (4.31) shows a voltage signal comprised of pulses of duration t_0 that are repeated every t_c units of time. The output of a PWM channel is either V_s volts during the pulse or zero volts; in this case, the maximum voltage that can be supplied to the engine battery is 7.5 V. If this signal is supplied as input to a device that has a response time much longer than t_c, the device will experience the signal

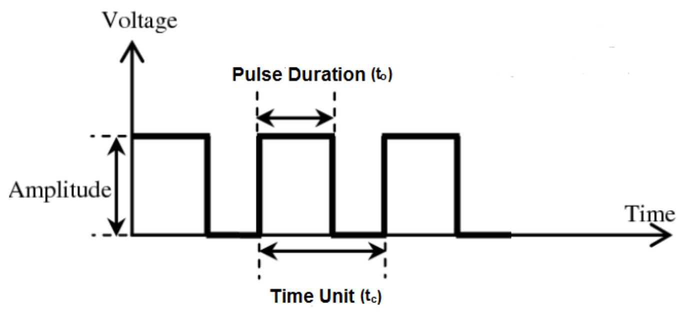


Figure 4.31: PWM signal with its two basic time periods

as an approximately DC input with an effective voltage of:

$$V_{\text{eff}} = V_s \frac{\tau_o}{\tau_c} \tag{4.17}$$

The ratio t_0/t_c is called the *duty cycle* of the square wave pulses, where t_c is the opposite of the signal frequency ($t_c = 1/f$).

To apply this type of control to our robot, it is sufficient to drive a H bridge input with the duty cycle square wave, variable provided by the oscillator (see Figure 4.32). The variation of the duty cycle allows to control the speed, while by changing the logic level applied on the other bridge input the direction of motor rotation is controlled.

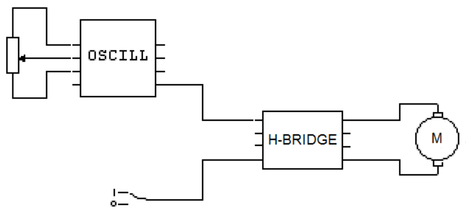


Figure 4.32: DC motor control

As shown in figure 4.33, digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (7.5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values,(i.e. effective DC voltage supplied to the load), the pulse width is changed or modulated.

In Figure 4.33, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to $analogWrite()$ is on a scale of 0 - 255, for example $analogWrite(255)$ requests a 100% duty cycle (always on), and $analogWrite(127)$ is a 50% duty cycle (on half the time).

In our case, the robot is supplied with two identical sinusoidal PWM signals, one

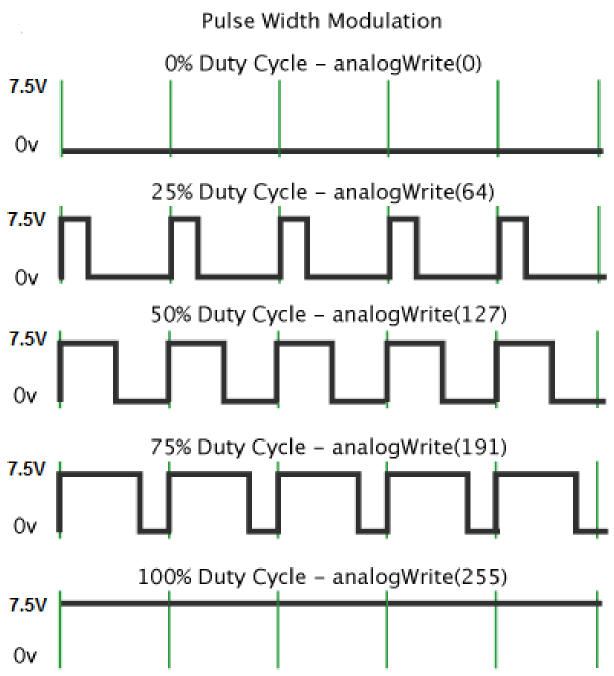


Figure 4.33: *PWM signal*

per motor.

Motor Drive L293D

In order to be able to control both wheel and clamp motor, the L293D integrated circuit is used, which is a motor driver specifically designed for controlling the direct current motors.

L293D consists of a double H bridge on two independent and symmetrical circuits, so as to be able to control 2 motors simultaneously.

The L293D is a 16 pin integrated circuit, as shown in Figure 4.34, where the pins are:

- Enable: it is connected to an Arduino digital pin. It provides a logical consent to the motor without which no current flows to the motor;

- Input: it is connected to an Arduino PWM digital pin, it receives the PWM signal;
- Output: it is connected to a motor terminal;
- GND: it is connected to the Arduino GND;
- Vcc 1: it is connected to the Arduino 5 V stabilizer output, it supplies power to the logic circuits of the chip;
- Vcc 2: it is connected to the positive pole of the battery, it allows to supply the motors with a different voltage from the one the logic of the circuit is supplied with.

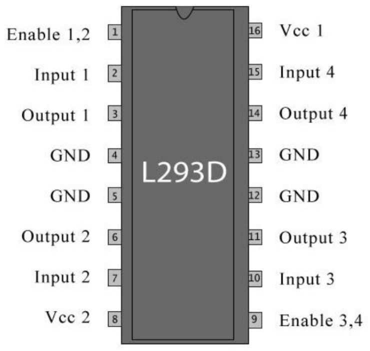


Figure 4.34: Pinout L293D

H-bridge

The H-bridge is an electronic circuit that can operate in the four quadrants of the load current - load voltage plane. In particular, it is used to switch the polarity of the voltage applied to the motor and then to reverse the direction of the motor current. As shown in Figure 4.35, this circuit is composed of four MOSFETs (used for the high switching speed and low energy loss), each of which has its own free wheeling diode.

The two lower MOSFETs (Q2,Q4) are called low side sink and they absorb the current from the motor while the upper MOSFETs (Q1,Q2) are named high side

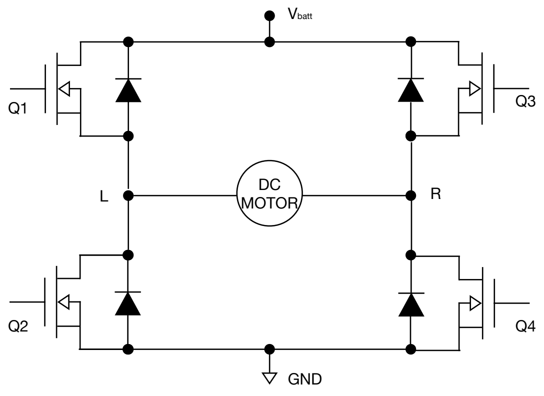


Figure 4.35: H-bridge

switch and they are connected to the voltage source directly.

The H-bridge operation can be summarized as follows:

- if Q1 and Q4 are turned on, the L-node of the motor is connected to power supply while the R-node is connected to the ground. In this way, the current I_{dc} starts to flow through the motor from the L-node to the R-node and the motor shaft starts spinning in a certain direction, for simplicity the forward direction, see Figure 4.36a;
- if Q2 and Q3 are turned on, the current I_{DC} flows through the motor in the reverse way and then the shaft of the motor will start spinning backwards, see Figure 4.36b.

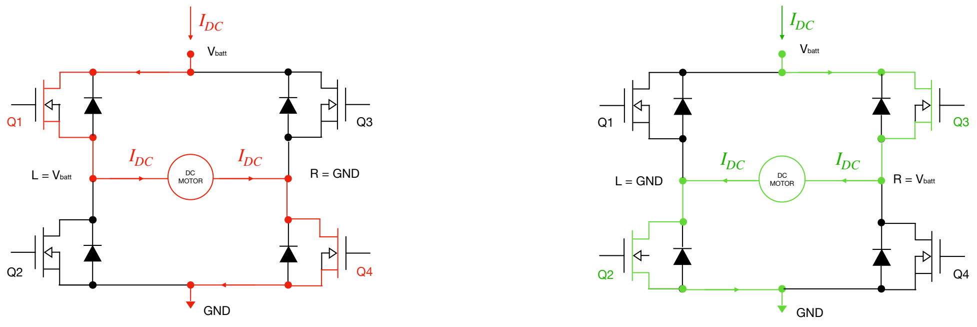
However, Q1 and Q2 or Q3 and Q4 must not be turned on at the same time, since a very low resistance path between the voltage source and the ground would be created and it would produce a short circuit.

Lock Anti-Phase Drive

The previous section shows how to control the power transferred to the motor with the PWM technique. However, in order to manage the robot's movements, it is necessary to be able to reverse the rotation direction of the motors. For this purpose, the lock anti-phase drive is used, where the 4 MOSFETs are opened and closed in pairs arranged diagonally. However, since the motor is mainly an inductive load, it is not possible to change the current that flows through the motor instantaneously. For this reason, it is necessary to guarantee that the inductor-current can continue to flow in some way. To achieve this, it is necessary to have one switch closed on both legs of the motor. Thus the H-bridge operates as follows

Mapping	Q1	Q2	Q3	Q4
Forward direction	on	off	off	on
Backwards direction	off	on	on	on

Table 4.2: H-bridge operation



(a) H-bridge with Q1 and Q4 on (b) H-bridge with Q2 and Q3 on

Figure 4.36: H-bridge operation

Electrical Connections

The circuit used is built on matrix boards. The power supply of the Arduino, the motors and the two L239D chips is produced by two 3.7 V Li-ion batteries connected in series, obtaining an input voltage of approximately 7.4 V. The L239D requires a power supply of 5 V for the internal circuitry: this voltage is supplied by the 5 V pin on the Arduino board. A voltage divider is designed to monitor the value of the input voltage so as to warn when the voltage falls below a preset threshold. The limit switch installed on an arm clamp uses an Arduino analog pin set as an input able to read the voltage difference between a reference value, in this case 0 V, and the input. All electrical connections are shown in Figure 4.37.

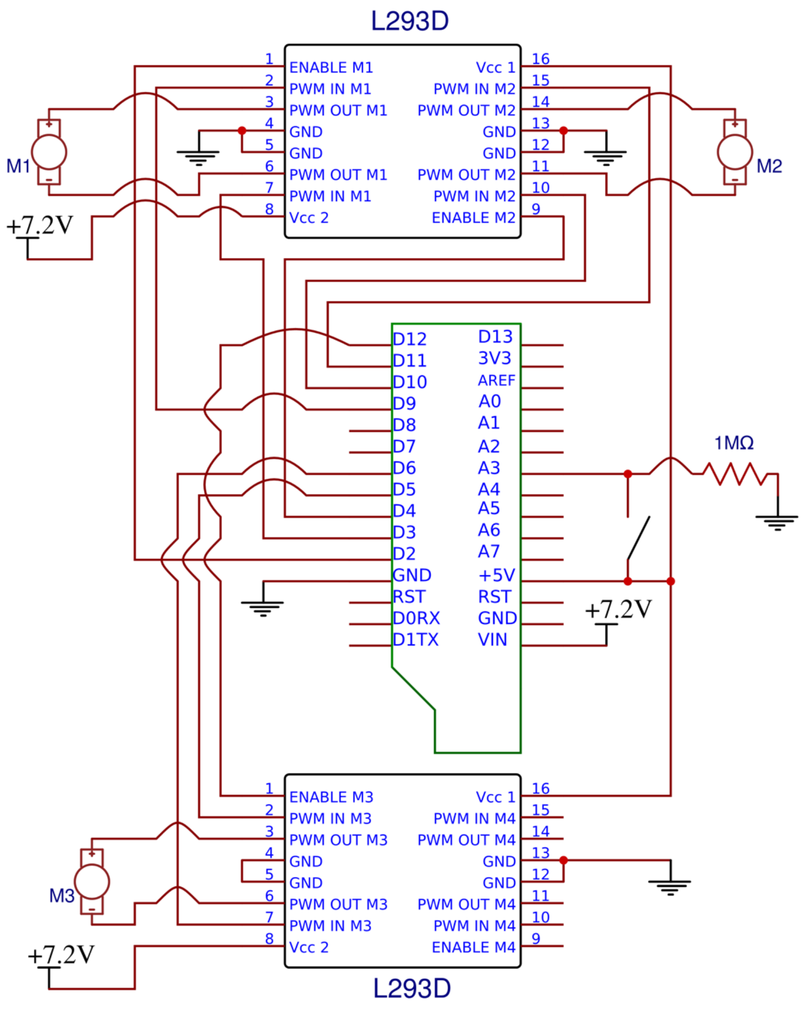
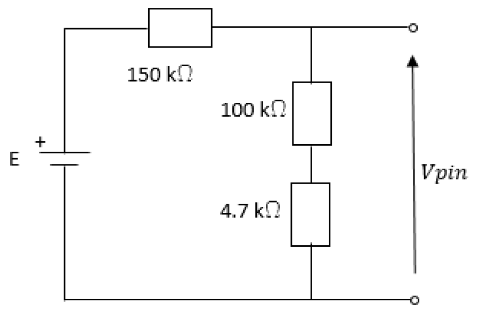


Figure 4.37: Circuit diagram

4.4.3 Battery Monitoring

In order to use the charging station, the batteries must never completely discharge. This means that the supply voltage must never fall below a minimum threshold so that the robots can reach the structure autonomously. To monitor the state of the batteries an Arduino analog pin is used, which allows to receive an analog signal between 0 V and 5 V, to convert it into 10-bit digital and to send the digital signal to Raspberry. Since two 3.7 V batteries are connected in series, in order to avoid that the voltage applied to the Arduino pin is higher than 5 V, a simple voltage divider is realized. The configuration of the divider is shown in Figure 4.38, with these resistance values the voltage applied to the Arduino pins will vary between 0 V and 3.5 V depending on the state of charge of the batteries.



$$V_{pin} = E\frac{(100 + 4.7)}{(100 + 150 + 4.7)}$$

Figure 4.38: Voltage divider

4.4.4 Connections

For the purposes of this project it is essential to put in communication the different systems which are in the block diagram shown in Figure 4.39. To do this, the technical features of Raspberry are exploited:

- the connection between Computer and Raspberry is made using the Ethernet port provided by the electronic card;
- the connection between Raspberry and Arduino is made using the USB 2.0 ports, where the Bluetooth USB dongle is installed, allowing the Bluetooth connection between the two electronic boards.

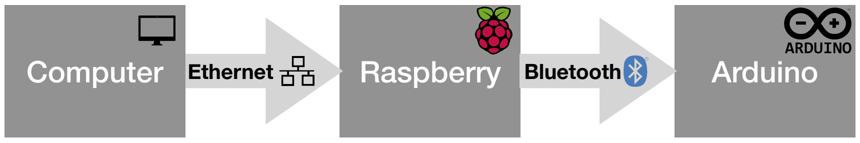


Figure 4.39: Simplified block diagram

Connection between Computer and Raspberry

The connection between Computer and Raspberry is made using an Ethernet cable. For the purposes of the project it is of crucial importance that this connection happens correctly, otherwise the robots will malfunction. The camera video tracking software, see Section 4.5.2, detects the presence of infrared LEDs, hence the robot position, and compiles an array that contains:

- X coordinate of the detected LED;
- Y coordinate of the detected LED;
- area A of the detected LED.

Then, this data are written in three special arrays and sent via UDP packets (User Datagram Protocol) to the port where the Raspberry is connected. This is responsible for the "unpacking" of what it receives in order to proceed with the subsequent parts of the implemented algorithm.

Connection between Raspberry and Arduino

The connection between Raspberry and Arduino is made using Bluetooth technology. Bluetooth is a standard wireless technology for exchanging data between fixed and mobile devices over short distances through short-wavelength UHF radio waves, usually up to 20-30 m [111]. Radio waves that operate around the 2.45 GHz band are used; to reduce interference, the Bluetooth protocol divides the band into channels and switches between the various channels 1600 times per second (frequency hopping). For the purposes of this project, the Bluetooth USB dongles are used to connect Raspberry and Arduino via Bluetooth. In particular, the

USBBLE-LINK is used, which is able to transmit at a distance of about 20 m using the Bluetooth 4.0 technology.

Figure 4.40: USBBLE-LINK USB Bluetooth dongle

4.5 Software

4.5.1 Introduction

The block diagram shown in Figure 4.41 shows how the various components of the test-bed communicate. Essentially, the work cycle is the following:

- Paths and positions where the robots must drag and drop orders are assigned by using one of the dispatch strategies implemented in Matlab, see Section 2 and Section 3. These information is then saved in a text file;
- The camera detects the infrared LEDs installed on the roofs of the robots;
- The program written in Visual Studio, in C ++, assigns to each LED, previously detected, the coordinates (X, Y) and the area A. Then, these data are written in arrays that will be sent to Raspberry
- Raspberry performs the following operations:
  - initializing the connection via serial port with Arduino
  - reading the text file in which both the trajectories that the robots must follow and the relative actions that the vehicles must execute are written;
  - unpacking arrays containing LED data;
  - recognizing the robots in the work area;
  - finding the PWM values to be sent to Arduino by using either the Pure Pursuite controller or the MPC controller. This data is written in an array that is sent to Arduino.

- Arduino reads the array sent to it by Raspberry and commands the three electric motors.

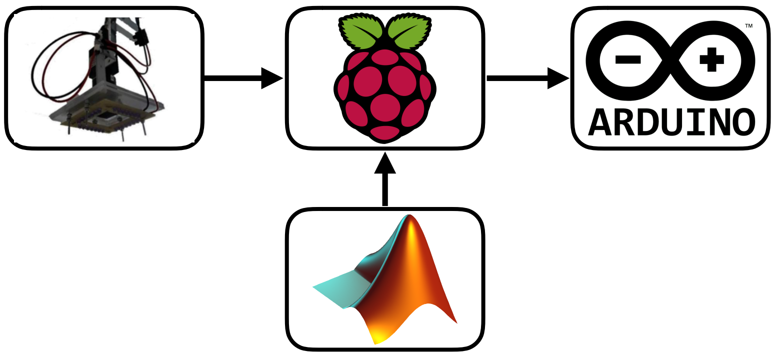


Figure 4.41: Test-bed communications block diagram

4.5.2 Visual Studio

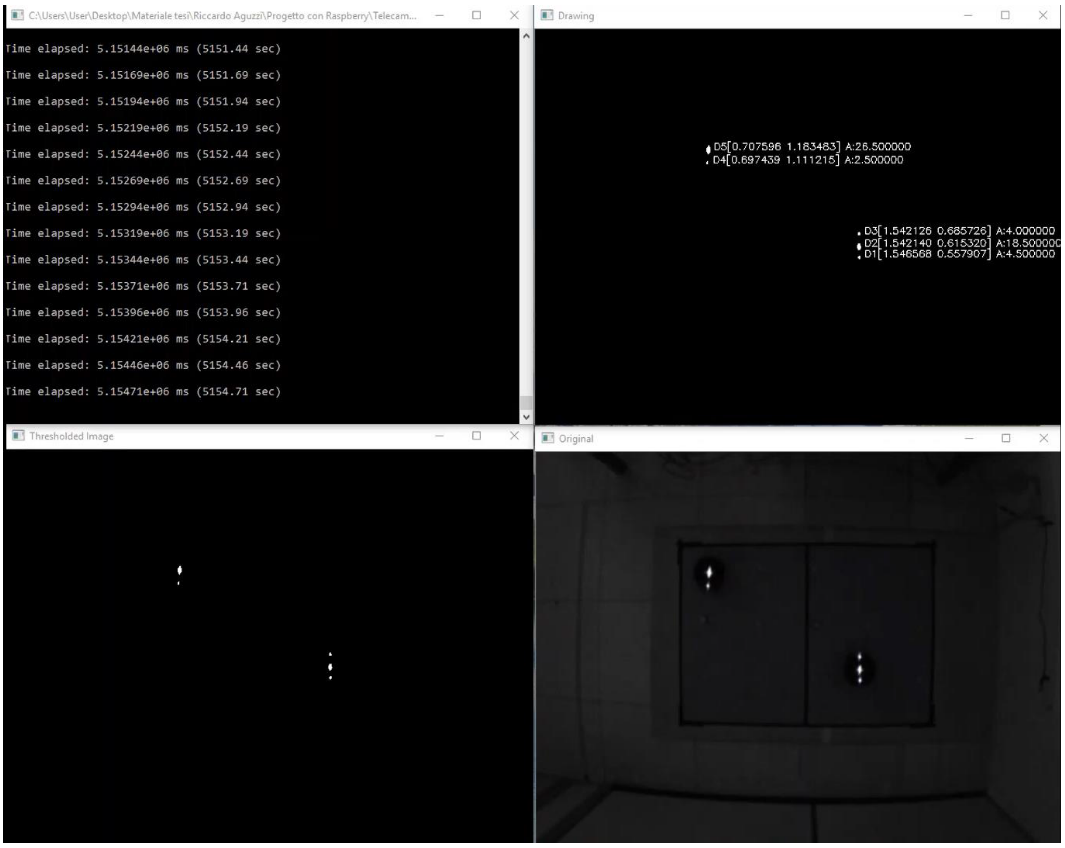
In this section, the operation of the camera will be briefly explained, since all that concerns the operating principle, the calibration and the modeling of the various parameters are explained in Section 4.2.

Video Tracking

The program used for the robot tracking is written using Visual Studio. For graphic image processing the libraries of OpenCV (Open Source Computer Vision Library) are used, while FlyCapture SDK for image acquisition is used and conversion into variables compatible with OpenCV itself. The program performs image manipulation to obtain the coordinates (X, Y) and area A of the infrared LEDs, using the following functions:

- $cvtColor()$: converts the image to grayscale;
- $threshold()$: performs a binary screening;
- $findContours()$: given a binary image, it returns the position of the outer contours of white pixels;

- *findDrawMarkers*(): for each contour found by the *findContours* function, it finds the moments and the position of the center of mass in pixel coordinates. The moment of order 0 represents the area enclosed by the outline of the LEDs placed above each individual robot. The pixel coordinates are then transformed into coordinates (X, Y, Z) by means of conversion matrices.


C:\Users\User\Desktop\Materiale tesi\Riccardo Aguzzi\Progetto con Raspberry\Telecam...
Drawing
Thresholded Image
Original

```
Time elapsed: 5.15144e+06 ms (5151.44 sec)
Time elapsed: 5.15169e+06 ms (5151.69 sec)
Time elapsed: 5.15194e+06 ms (5151.94 sec)
Time elapsed: 5.15219e+06 ms (5152.19 sec)
Time elapsed: 5.15244e+06 ms (5152.44 sec)
Time elapsed: 5.15269e+06 ms (5152.69 sec)
Time elapsed: 5.15294e+06 ms (5152.94 sec)
Time elapsed: 5.15319e+06 ms (5153.19 sec)
Time elapsed: 5.15344e+06 ms (5153.44 sec)
Time elapsed: 5.15371e+06 ms (5153.71 sec)
Time elapsed: 5.15396e+06 ms (5153.96 sec)
Time elapsed: 5.15421e+06 ms (5154.21 sec)
Time elapsed: 5.15446e+06 ms (5154.46 sec)
Time elapsed: 5.15471e+06 ms (5154.71 sec)
```

```
D5[0.707596 1.183483] A:26.500000
D4[0.697439 1.111215] A:2.500000

D3[1.542126 0.685726] A:4.000000
D2[1.542140 0.615320] A:18.500000
D1[1.546568 0.557907] A:4.500000
```

Figure 4.42: Screens generated by Visual Studio

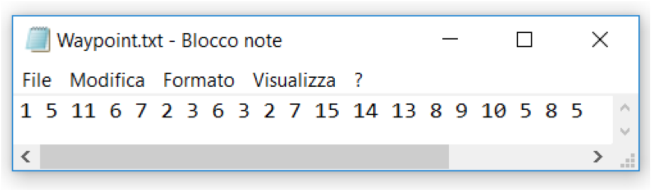
Figure 4.42 shows:

- at the bottom right, the screen of the Original image, or what is generated by the *cvtColor*() function;
- at bottom left, the screen of the *Thresholds Image*, or what is generated by the *threshold*() function;
- at the top right, the *Drawing* image, which is an image where it is possible to read both the coordinates (X, Y) and the area of each identified LED. In particular, the presence of five LEDs that emit infrared rays can be noticed.

These infrared rays are first identified by the camera and then processed by the software so as to be assimilated to “dots” having different areas. This is fundamental, as will be seen in Section 4.5.4, in order to correctly recognize the different robots present on the worktop.

4.5.3 Matlab

Once the selected dispatching strategy has found the paths, which consist of a sequence of the graph nodes, and the assignments for each robot, a text file is saved where all the useful information for the Raspberry is stored. An example of text file is shown in Figure 4.43. The series of numbers shown in Figure 4.43 has the


Waypoint.txt - Blocco note
File Modifica Formato Visualizza ?

```
1 5 11 6 7 2 3 6 3 2 7 15 14 13 8 9 10 5 8 5
```

Figure 4.43: Text file generated by MatLab

following meaning:

- the first number represents the number of the robot to which the path written later is assigned;
- the second number represents the length of the path in terms of graph nodes (η_p), i.e. the number of nodes the path is formed of;
- the successive η_p-numbers are the graph nodes that form the path
- the next two numbers indicate where to pickup and where to deliver the object, respectively.

The sequence is repeated a number of times equal to the number of robots that must work simultaneously. Thus, in Figure 4.43 the path of robot 1 is composed of 5 points (11,6,7,2,3), it has to pick up the object at node 6 and deliver it at node 3. Instead, robot 2 has a path composed of 7 points (15,14,13,8,9) and it has to transport an order from point 8 to point 5.

However, if the MPC is used as the lower controller, instead of the Pure Pursuit, one more step is needed. In this case, each edge that makes up the path which the robot must follow, is first defined in a parametric way. Then, in accordance with the sampling time adopted by the MPC, each edge is discretized. In this way, the resulting path result composed of a finite number of discrete points, for example the trajectory of robot 1, with a sampling time equalt to 0.1, will be:

$$x = [0.670, 0.671, \ldots, 0.679, 0.680, 0.711, 0.742, \ldots, 0.959, 0.990, \ldots]$$
$$y = [0.610, 0.640, \ldots, 0.880, 0.910, 0.908, 0.906, \ldots, 0.892, 0.890, \ldots]$$

4.5.4 Raspberry

As can be seen in Figure 4.41, Raspberry is the block to which a greater number of operations to be performed belongs. The algorithm implemented on the electronic card is written in C. When the program is started, the algorithm implemented in Raspberry must first carry out the initialization of the connection with Arduino. Then, it has to read the text file generated from MatLab and for each robot convert the graph nodes that constitute the trajectory into (X, Y) coordinates. Finally, the algorithm has to initialize the connection with Visual Studio. After completing these startup operations, the algorithm must perform the following functions cyclically:

- acquiring the data sent to it by the camera program and “unpacking” the data contained in the received array;
- recognizing the robot;
- finding the PWM values to be sent to Arduino by using either the Pure Pursuit controller or MPC controller;
- sending the new PWM values to Arduino

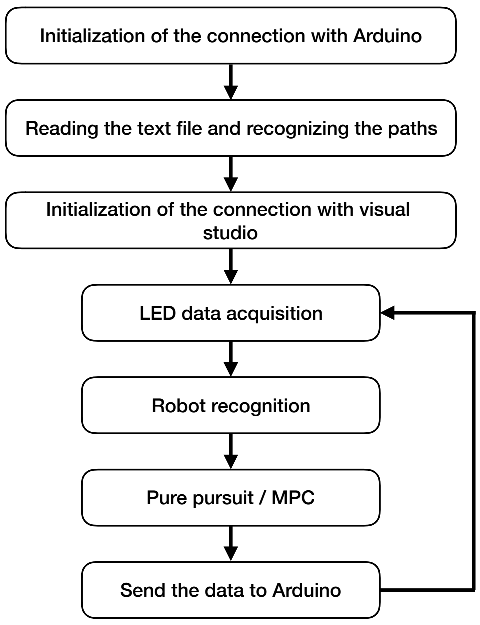

Initialization of the connection with Arduino
Reading the text file and recognizing the paths
Initialization of the connection with visual studio
LED data acquisition
Robot recognition
Pure pursuit / MPC
Send the data to Arduino

Figure 4.44: Algorithm scheme implemented on Raspberry

Trajectory Reading and Conversion in *(X, Y)* Coordinates

This operation is not performed by Raspberry cyclically, but only when the program is started via compiler. As explained in Section 4.5.3, the text file generated by Matlab contains all the information necessary to ensure that the robots work according to a precise scheme. This text file must be read by the algorithm implemented on Raspberry. The algorithm is responsible for writing a special array that contains both the coordinates (X, Y) of the robot trajectory and the actions that the robot must execute once it has reached every graph node. To achieve this, it is necessary to know the coordinate values of all fifteen graph nodes. In our case, this values are stored in two vectors, see Listing 4.1.

Listing 4.1: Coordinates (X, Y) of graph nodes Q

```
float XWP[15] = {0.69, 1.00, 1.30, 1.59, 1.87, 0.68, 0.99, 1.29, 1.59, 1.87, 0.67, 0.98, 1.28, 1.58, 1.86};
float YWP[15] = {1.21, 1.19, 1.17, 1.14, 1.12, 0.91, 0.89, 0.88, 0.85, 0.83, 0.61, 0.58, 0.58, 0.57, 0.55};
```

Once the robot path is known, an algorithm is implemented to create three arrays for each robot involved. The first two arrays are the *x-coordinate* array and the *y-coordinate* array. As suggested by their name, they contain the coordinates of the graph nodes that the robot must follow. Instead, the third arrays, which is called *actions*, contains a sequence of numbers that indicates the action that the robot must execute in a specific graph node. This number is equal to: $+1$ where the robot must pickup the object, -1 where the robot must deliver the object, 0 if no actions are required. As shown in Figure 4.43, the algorithm produces the following arrays:

$$\begin{aligned}
\text{robot 1:}\quad X - coordinate &= [0.67, 0.68, 0.99, 1.00, 1.30]\\
Y - coordinate &= [0.61, 0.91, 0.89, 1.19, 1.17]\\
actions &= [0, 1, 0, 0, -1]\\
\text{robot 2:}\quad X - coordinate &= [1.86, 1.58, 1.28, 1.29, 1.59, 1.87, 1.87]\\
Y - coordinate &= [0.55, 0.57, 0.58, 0.88, 0.85, 0.83, 1.12]\\
actions &= [0, 0, 0, 1, 0, 0, -1]
\end{aligned}$$

Reading a UDP packet

This operation is performed cyclically by Raspberry. From Visual Studio a 216 byte array of doubles is sent. This value is obtained by taking into account that: i) a double is bytes, ii) at most there can be nine LEDs in the working area. Thus:

$$\text{array size sent to Raspberry} = 8 \cdot 9 \cdot 3 = 216\, byte$$

The reason why nine LEDs can be detected at most is that, as better explained in Section 4.5.4, robot 1 has two LEDs, robot 2 has three LEDs and robot 3 has four. Listing fig:udp-read shows the rows of the algorithm by which the "unpacking" of the incoming array from Visual Studio is performed. In the $X[]$, $Y[]$ and $A[]$ double arrays, the values of the coordinates and the area of each LED detected by the camera are saved.

Listing 4.2: Reading UDP packets

```
for (i=0; i<9; i++){
    for (j=0; j<8; j++){
        aus[j] = message[j+i*8];
    }
    X[i] = (*(double*)aus);
}
for (i=9; i<18; i++){
    for (j=0; j<8; j++){
        aus[j] = message[j+i*8];
    }
    Y[i−9] = (*(double*)aus);
}
for (i=18; i<27; i++){
    for (j=0; j<8; j++){
        aus[j] = message[j+i*8];
    }
    A[i−18] = (*(double*)aus);
}
```

Robot Detection

One of the fundamental steps of the algorithm consists of the part of the code dedicated to the recognition of the different robots on the worktop and to the compilation of the different parameters of the *struct* robots. As can be noticed from Listing 4.3, the values that characterize these *struct* are:

- (X, Y) coordinates of the head
- (X, Y) coordinates of the tail, i.e. the center of gravity of the vehicle
- the angle of inclination expressed in degrees of the robot in relation to the general reference system $X, Y,$, see Figure 4.45

Listing 4.3: *struct* robot

```
typedef struct robot{
    double X_Head;
    double Y_Head;
    double X_Tail;
    double Y_Tail;
    double Angle;
}Robot;
```

A different number of infrared LEDs are positioned on the robot roofs. The LEDs

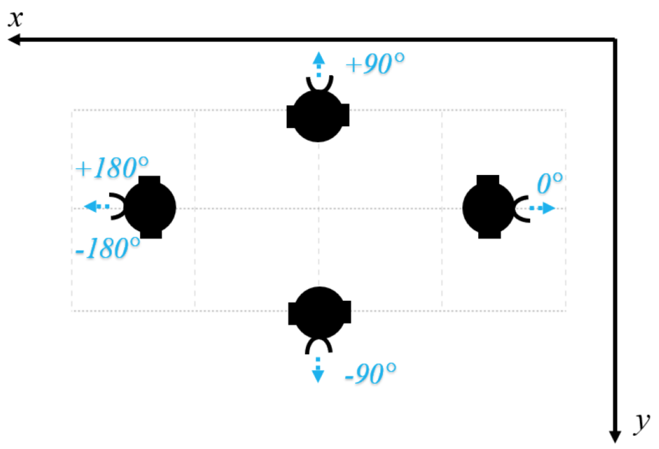


Figure 4.45: Possible values of the inclination angle of the robots

have two types of configuration:

- area greater than 15 cm^2 (large)
- area less than 10 cm^2 (small)

Each robot is equipped with a different number of LEDs as shown in Figure 4.46. For each robot the led placed at the center is the one with the largest area while

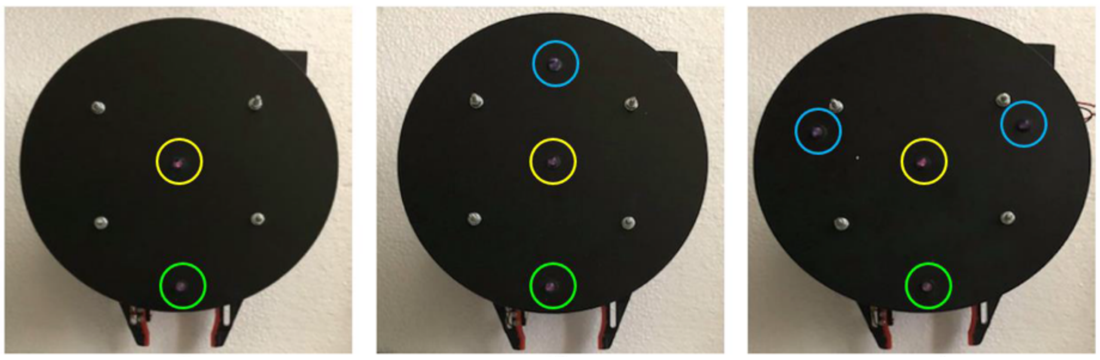


Figure 4.46: UNIPV robots

all the others are smaller. The "small" LEDs, are placed at different distances in relation to the "big" LED placed at the center of each robot, in particular:

- 0.055 m if it represents a control led
- 0.007 m if it represents a "head" led

In particular, the following configuration is used

- 1 small led for the first robot (robot head)
- 2 small LEDs for the second robot (a "head" LED and a control LED)
- 3 small LEDs for the third robot (a "head" LED and two control LEDs)

The operating principle of the robot recognition algorithm can be summarized as follows: first in the array the algorithm finds $A = []$, which is obtained by the "unpacking" of the UDP data, a LED that represents the robot center of gravity, i.e. $(A[i] > BIGAREA)$. Then, the algorithm looks for LEDs whose area is both small $(A[i] < BIGAREA)$ and its distance from the tail is less than 8 cm $(dist < BIGRADIUS)$. At the same time, the number of LEDs with a small area is counted. Moreover, if the LED with a small area has a distance from the center of gravity that is lower than $BIGRADIUS$, but, at the same time, greater than $SMALLRADIUS$, the head of the robot is found. Finally, knowing the number of LEDs with a small area found, the struct robots can be initialized (robot 1 has only one small are LED, robot 2 has two, while robot 3 has three). In Listing 4.4 the $atan2$ function is used in order to obtain the robot angle of inclination. This function always returns a result which is included between $-\pi$ and π, as reported in Equation 4.18.

$$atan2(X,Y) = \begin{cases} tan^{-1}(\frac{Y}{X}) & \text{if } X > 0 \\ tan^{-1}(\frac{Y}{X}) & \text{if } X > 0 \text{ and } Y \geq 0 \\ tan^{-1}(\frac{Y}{X}) & \text{if } X \leq 0 \text{ and } Y < 0 \\ \frac{\pi}{2} & \text{if } X = 0 \text{ and } Y > 0 \\ \frac{-\pi}{2} & \text{if } X = 0 \text{ and } Y < 0 \end{cases} \tag{4.18}$$

Listing 4.4: Robot detection Algorithm

```
for (i−0;i<9;i++){
    nSmall−0;
    if(A[i]> BIGAREA){
        for(j+0; j<9; j++){
            if(A[j]< BIGAREA){
                dist = sqrt((X[i]−X[j])∗(X[i]−X[j])+(Y[i]−Y[j])∗(Y[i]−Y[j]));
                if(dist < BIGRADIUS){
                    nSmall ++;
                    if(dist>SMALLRADIUS){
                        AusHeadX = X[j];
                        AusHeadY = Y[j];
                    }
                }
            }
        }
        if(nSamll == 1){
            robot1.X_Head = AusHeadX; robot1.Y_Head = AusHeadY; robot1.X_Tail = X[i]; robot1.Y_Tail = Y[i];
            AusAngle = atan2((robot1.Y_Tail − robot1.Y_Head), (robot1.X_Tail − robot1.X_Head));
            robot1.Angle = AusAngle∗180/pi;
        }else if(nSmall == 2){
            robot2.X_Head = AusHeadX; robot2.Y_Head = AusHeadY; robot2.X_Tail = X[i]; robot2.Y_Tail = Y[i];
            AusAngle = atan2((robot2.Y_Tail − robot2.Y_Head), (robot2.X_Tail − robot2.X_Head));
            robot2.Angle = AusAngle∗180/pi;
        }else if(nSmall == 3){
            robot3.X_Head = AusHeadX; robot3.Y_Head = AusHeadY; robot3.X_Tail = X[i]; robot3.Y_Tail = Y[i];
            AusAngle = atan2((robot3.Y_Tail − robot3.Y_Head), (robot3.X_Tail − robot3.X_Head));
            robot3.Angle = AusAngle∗180/pi;
        }
    }
}
```

Sending data to Arduino

Once the control action is found, as will be explained in Section4.6, the control action must be sent to Arduino. The connection between Raspberry and Arduino is achieved by using the USB Dongle Bluetooth, see Section 4.4.4. These dongles are seen as a normal serial port by Raspberry. Thus, by using the special library ($< wiringSerial.h >$), it is possible to write or read some data. In particular, it is possible to send Arduino the previously obtained PWM values through the $serialPutchar()$ command, while the $serialGetchar()$ allows to read the values sent by Arduino (the state (ON / OFF) of the limit switch installed on the clamp and the percentage of the batteries). Listing 4.5 shows the code used for sending the data through the USB Dongle Bluetooth. Moreover, it is possible to note that the sent data has a fixed sequence. In particular, the first datum is the PWM value for the right motor wheel, the second is the PWM value for the left motor wheel. The third datum corresponds to the PWM value which controls the motor of the clamp, while the fourth datum corresponds to a control number that will be used by Arduino to verify the integrity of the received data packet: this data is always equal to 60.

Listing 4.5: Syntax to send the data to the serial port

```
for(i=0: i<3; i++)
   serialPutchar(fd1, sendPWM1[i]);
serialPutchar(fd1,60);
StatusPliers1 = serialGetchar(fd1);
StatusBattery1 = serialGetchar(fd1);

for(i=0: i<3; i++)
   serialPutchar(fd2, sendPWM2[i]);
serialPutchar(fd2,60);
StatusPliers2 = serialGetchar(fd2);
StatusBattery2 = serialGetchar(fd2);
```

Arduino

Arduino is responsible both for commanding the robot motors (right and left motors and clamp motor) and for sending Raspberry the states of the limit switch and of the battery. The Arduino program consists of three main parts: the definition of the variables, a cycle performed only when the program starts ($voidsetup()$) and a cycle performed cyclically ($voidloop()$)"

- variable declaration: in this part of the program all the variables used in the *void loop()* are initialized. As shown in Listing 4.6, it is possible to see that these variables can be assigned a number corresponding to the Arduino pin to which the input or output is connected (for example, the pin for enabling the left wheel, which is represented by the Enable_Motor_SX variable, is connected to the pin with the number 2);

Listing 4.6: Arduino software: variable declaration

```
#define Enable_Motor_SX 2
#define Enable_Motor_DX 4
#define Enable_Motor_Pliers 12
#define Pin1_Motor_SX 3
#define Pin2_Motor_SX 9
#define Pin1_Motor_DX 10
#define Pin2_Motor_SX 11
#define Pin1_Motor_Pliers 5
#define Pin2_Motor_Pliers 6
#define Status_Pliers 17
#define ZERO 127

char incomingByte[] = [0,0,0,0];
int n = 0;
int PinBatt = A0;
int ValPinBatt;
float Vpin_perc;
float Vbat;
float calc_res;
float R1 = 109000;
float R2 = 150000;
byte val_perc;
```

- *void setup()*: in this phase of the program, the state (input / output) of the previously initialized variables are set. Furthermore, since data are transmitted via serial port, in the *void setup()*, it is necessary to set the data transmission speed;

- *void loop()*: it contains the main body of the program which is repeated cyclically. Initially, the values received from Raspberry are read and stored in the *incomingByte* vector: the program receives four values, the first three are PWMs for the engines while the fourth corresponds to the control byte (set as a constant number equal to 60). When Arduino receives four data, and the last has the value of 60, the algorithm assigns the received PWMs. Thus, the program writes the PWMs to the corresponding outputs, adding and subtracting to ZERO (127, the PWM value corresponding to the stationing) the data received, so as to carry out the anti-phase drive lock. This procedure is performed for all three motors (wheel left, right wheel, clamp). Finally, using the *analogWrite* function, the values of the limit switch on the clamp and of the battery state are read, which are sent to Raspberry via serial communication.

Listing 4.7: Arduino software: *void loop()*

```
void loop(){

  while(Serial.available()>0){
    digitalWrite(Enable_Motor_SX, HIGH);
    digitalWrite(Enable_Motor_DX, HIGH);
    digitalWrite(Enable_Motor_Pliers, HIGH);
    incomingByte[n] = Serial.read();
    n++;

    if(n==4){
      analogWrite(Pin1_Motor_SX, ZERO + incomingByte[0]);
      analogWrite(Pin2_Motor_SX, ZERO + incomingByte[0]);
      analogWrite(Pin1_Motor_DX, ZERO + incomingByte[1]);
      analogWrite(Pin2_Motor_DX, ZERO + incomingByte[1]);
      analogWrite(Pin1_Motor_Pliers, ZERO + incomingByte[2]);
      analogWrite(Pin2_Motor_Pliers, ZERO + incomingByte[2]);
      int Status_Read = analogRead(Status_Pliers);
      if(Status_Read >950){
        Serial.write(0)
      }else{
        Serial.write(1)
      }
      delay(10);
      ValPinBatt = analogRead(PinBatt);
      Vpin_perc = map(ValPinBatt,0,1023,0,500);
      Vbatt = Vpin_perc * calc_res/100;
      Vbat -= 7
      val_perc = 100 * Vbatt/1.4;
      Serial.write(val_perc);
      delay(10)

      n = 0;

    }
  }
}
```

4.6 Low Level Controllers

In this section, the controllers of the robots are described in order to ensure that they follow a given trajectory and perform actions initially set. In particular, the controllers are organized in a hierarchical way. At the higher level, there is the optimal dispatching, which is written in Matlab and relies on the algorithms mentioned in Section 3 and Section 2. At the lower level, a control algorithm is employed to guarantee the robots movement on the graph. At this level two different controllers are developed: a pure pursuit approach and model predictive control.

4.6.1 Pure Pursuit Approach

In order to apply the Pure Pursuit control, it is necessary to find the relationship between the applied PWM and robot dynamic. First, in this section the relationship between the PWM and the radius of curvature is introduced. Then, the section describes how it is possible to find the correct input (PWM) in order to follow the given trajectory. Finally, the pure pursuit algorithm and its characteristic operations are analyzed.

Robot Dynamic

The robot movement, as seen in the previous chapter, is ensured by two motors controlled by PWM technique. Thus, modifying these values, the robot can follow different trajectories, which can be of two types:

- Rectilinear trajectory: in order to follow a straight line, in theory, it would be sufficient to assign the same PWM value to the wheel motors, but this is practically impossible due to inevitable inaccuracies in the assembly and transmission frictions;

- Curvilinear trajectory: by assigning different PWM values to the two motors, the robot will follow a curvilinear trajectory, characterized by a radius of curvature.

It is therefore necessary to find a correspondence between the PWM values sent to the wheel motors and the radius of curvature of the trajectory that the robot follows.

Radius of Curvature and PWM Values

A fixed PWM value is assigned to one of the two wheel motors and, by varying the PWM sent to the other motor, the robot follows a circumference. On this circumference, 3 points (p_1, p_2 and p_2) of known coordinates are identified. A generic circumference that pass through the three points can be derived by solving the following system of equations:

$$\begin{cases} x_{p_1}^2 + y_{p_1}^2 + ax_{p_1} + by_{p_1} + c = & 0 \\ x_{p_2}^2 + y_{p_2}^2 + ax_{p_2} + by_{p_2} + c = & 0 \\ x_{p_3}^2 + y_{p_3}^2 + ax_{p_3} + by_{p_3} + c = & 0 \end{cases} \tag{4.19}$$

Once the circumference parameters a, b and c are obtained, it is possible to find the radius as

$$Radius = \sqrt{\left(\frac{-a}{2} + \frac{-b}{2}\right)^2}$$

Therefore, by fixing the PWM of a motor equal to 18 and using the Table 4.3, it is possible to determine the PWM value to send the other motor in order to have a given radius of curvature. However, there are situations where the robot

Radius	0.4668	0.3701	0.3114	0.2687	0.2304	0.2212	0.2066	0.1837	0.1764	0.1642	0.1468
PWM	22	24	26	28	30	32	34	36	38	40	42

Table 4.3: Radius of curvature and PWM values

does not have to follow a predetermined trajectory, but it only needs to change its orientation. To achieve this, the best way is to rotate the robot around its vertical axis. This is done by assigning the motors two equal PWM values in the module but opposite in sign, so that the motors rotate in the opposite direction.

Geometric Relationships

In Figure 4.47 the geometric relationship of the Pure Pursuit approach is shown.

Let $G(x, y)$ be the *goal point*. Define l as the *look ahead distance*. Considering

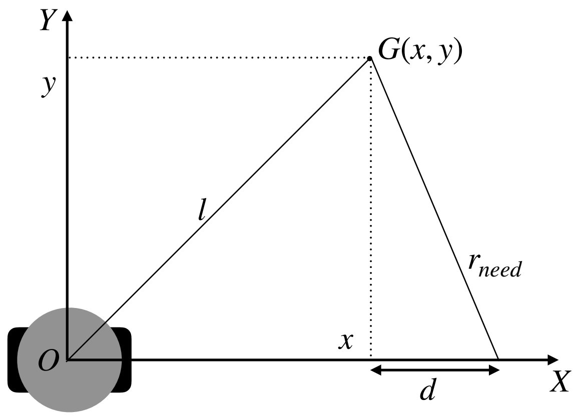


Figure 4.47: Geometric relationship of the algorithm

the right triangle GOB

$$x^2 + y^2 = l^2 \tag{4.20}$$

and the sum of the segments on the x axis *goal point*

$$x + d = r_{need} \tag{4.21}$$

from which

$$d = r_{need} - x \tag{4.22}$$

From the right-angled triangle of sides $y, x, x + d$::

$$(r - x)^2 + y^2 = r_{need}^2 \tag{4.23}$$

where it is possible to find the radius of curvature needed to reach the *goal point* as:

$$\begin{aligned} 2 \cdot r_{need} \cdot x =& l^2 \\ r_{need} =& \frac{l^2}{2 \cdot x} \end{aligned}$$

The geometric relationships described above are obtained in the robot reference system. However, the *goal point* is expressed in the world reference system. Thus, in order to obtain the goal point coordinates expressed in the robot reference system (i.e. X_v, Y_v), it is necessary to convert them as follows:

$$\begin{aligned} X_v =& (X_g - X_r)cos(\theta) + (Y_g - Y_r)sin(\theta) \\ Y_v =& -(X_g - X_r)sin(\theta) + (Y_g - Y_r)cos(\theta) \end{aligned}$$

where: θ is the orientation angle of the robot, $(X_g - Y_g)$ and $(X_r - Y_r)$ are the goal coordinate and the robot coordinate in the world reference system, respectively.

Pure Pursuit

Pure pursuit is an algorithm developed for the control of autonomous driving vehicles. It is based on the calculation of the radius of curvature of the trajectory that the vehicle must follow to move from the position in which it is to another point, called goal point. Given the position of the robot and the trajectory to follow, the closet point to the vehicle and the goal point are identified. The goal is a future point on the trajectory whose distance from the current position of the vehicle depends on the so called "look ahead distance window". Once such point has been identified, the curvature necessary to attain it is computed and the corresponding inputs applied. The algorithm is applied simultaneously to each individual robot,

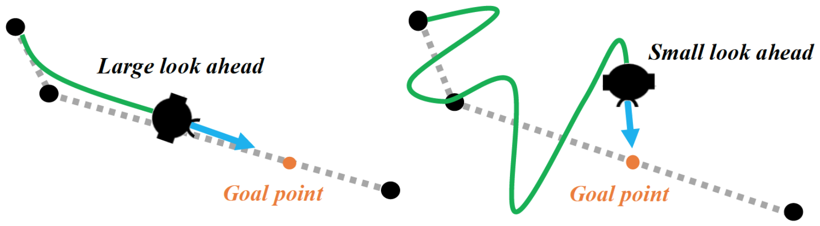


Figure 4.48: Pure Pursuit approach with different look ahead distances

but it is tailored to each of them. Indeed, the robots can have different starting positions and trajectories. The pure pursuit program is divided into 3 main modes, each of which has a different function:

- Cruise: it deals with moving the robot between two different points on the graph, using the Pure Pursuit algorithm;

- Action: it takes care of the actions to be performed once the robot has arrived at the points of the graph, for example taking an object, leaving an object or simply passing through the point without taking any actions;
- Reorientation: it deals with correcting the orientation of the robot.

The operations is schematized in the flow chart reported in Figure 4.49.

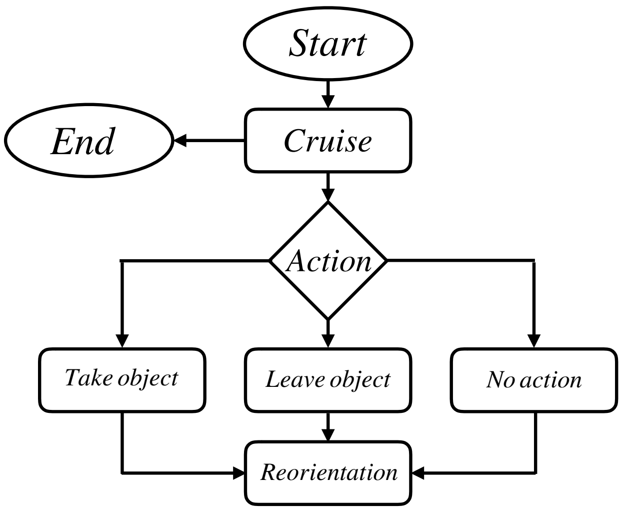


Figure 4.49: Pure Pursuit flow chart

Cruise mode

The "Cruise" mode allows the robot to move from one point to another of the graph, following a trajectory optimized by a Pure Pursuit algorithm. The first operations of the algorithm are to find the graph node to be reached and to calculate the distance between the robot and the selected graph node. Successively, the approach finds the radius of curvature for reaching the node and it compares this value with the values of radius curvatures reported in Table 4.3. From this comparison, the radius of curvature that results as close as possible to the theoretical one is chosen. Finally, the approach checks if the distance between the robot's center of gravity and the graph node to be reached is less than 0.075 m. If so, it means that the robot has reached its target graph node. Therefore, the algorithm will exit from the cruise mode and will enter the action mode, according to Figure 4.49.

Otherwise, the robot has not reached its target node. Thus, it will remain in cruise mode and it will repeat all the previous operation until it reaches its target node.

Action Mode

The “Action” mode allows the robot to perform an action every time it reaches a graph node in its trajectory. As soon as the program enters this cycle, in order to prevent the robot from moving away from the target node it assigns null PWM values to both motors. In this mode, three possible actions can be performed alternatively:

- Take object: the robot picks up the object. In particular, the program checks the clamp status. If no object is detected, the clamp is closed and thus the object is taken.
- No action: this is the simple passage through the node graph.
- Leave object: the robot delivers the object, which means the clamp will be opened.

Once one of the aforementioned actions has been completed, if the vehicle has not yet completed its trajectory, the algorithm exits from the “Action” mode and enters the “Reorientation” mode, otherwise it exits “Action” mode and stops.

Reorientation Mode

Once a point on the route has been reached and the necessary action has been completed, this mode allows the robot to orient itself towards the next graph node to be reached. Therefore, PWM values are found to be sent to the robot motors so that the vehicle is aligned with the next target. Note that a tolerance is provided to prevent the robot from remaining in Rotation mode for a long time.
The pure pursuit algorithm is extremely simple but it may be not very accurate nor does it allow to consider important features such as collision avoidance. For this reason, the Model Predictive Control is investigated as an alternative. MPC is an advanced control based on the solution of an optimization at each time step.

The method relies on a model which is employed to predict the future behavior of the vehicles. In particular, such prediction allows to improve the performance. By looking ahead, it is, for example, possible to slow down the vehicle so as to perform a curve staying within lane (something that traditional controllers find quite difficult). By relying on optimization, MPC allows to include explicitly the presence of states and input constraints. Although it appears to be computationally heavier than the pure pursuit, MPC is able to guarantee a better performance.

4.6.2 Model Predictive Control

In order to apply a control on the mobile robot, it is necessary to model the dynamic process under exam. First, a mathematical model that describes the robot dynamic is introduced. Then, the continuous time equations of the angular speeds of the motors are defined, and, by considering the coordinates of the points that describe the trajectory, the section shows how to calculate the speed values needed by the vehicle to follow the default route. However, the inputs of the system under consideration are not the speeds, but the PWM signals. Therefore, it is necessary to establish a relationship between these two variables: PWM speeds and signals, as well as motor inputs and outputs. Once this relationship is found, the MPC is first introduced and then applied in order to follow the trajectory generated by the high level control explained in Chapter 2 and Chapter 3.

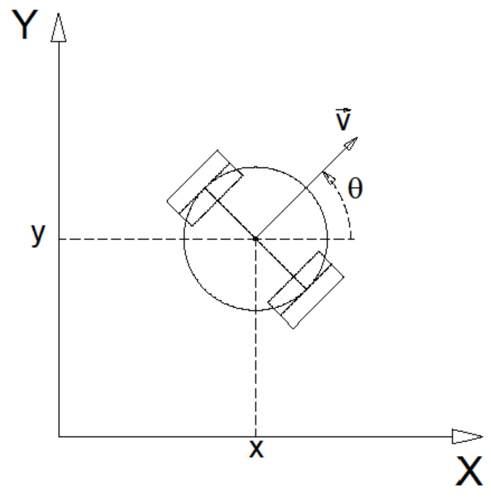
Robot Model

In order to create a good controller, it is necessary to determine a model that describes the kinematics of the robot. Kinematics is the branch of classical mechanics which describes the motion of a point without considering the mass of the objects or the forces that may have caused the motion. The kinematic equations are used to transform the motion from polar coordinates (r, θ) to a rectangle coordinate system (x, y) system.

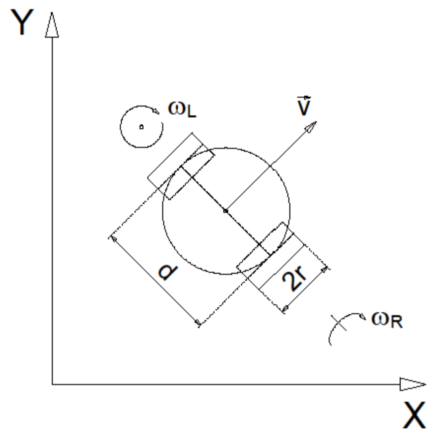
The robot system, see Figure 4.50a, can be described by the non-linear continuous-time model characterized by the following equations of state:

$$\begin{cases} \dot{x}_1 = v_{robot}\cos(x_3) \\ \dot{x}_2 = v_{robot}\sin(x_3) \\ \dot{x}_3 = w \end{cases} \tag{4.24}$$

where the model inputs are the linear velocity v_{robot} of the robot and the angular velocity w while, the state variables are x_1, x_2, x_3, the world coordinate ($x_1 = x$, $x_2 = y$) and the angle ($x_3 = \theta$) that is formed with the X axis, respectively, as shown in Fig 4.50a.



(a) Robot model



(b) Differential-drive model

Figure 4.50: Robot differential-drive model

To allow the robot to reach every point of the plane it is necessary that it can make the curves of the trajectory even though it is not equipped with a steering. For this reason, a differential-drive model is used. This technique consists in providing each wheel of the vehicle with a speed that is independent of the other, so as to allow the robot to follow any radius of curvature.
Figure 4.50b shows the differential drive robot layout, where w_R is the velocity of the right wheel, w_L is the velocity of the left wheel, d is the distance between the center of the wheels and r is the radius of the wheel. A differential drive robot is a mobile robot with two driving wheels in which the overall velocity is split between left and right wheels, in w_L and w_R.

To construct a simple model of the constraints that arise from the *differential drive*, only the distance d between the two wheels and the wheel radius r, are necessary. If $w_{\mathrm{R}} = w_{\mathrm{L}} > 0$, then the robot moves forward in the direction that the wheels are pointing, if $w_{\mathrm{L}} = -w_{\mathrm{R}} \neq 0$, then the robot rotates clockwise because the wheels are turning in opposite directions, see Figure 4.51. In general, if $w_{\mathrm{L}} = w_{\mathrm{R}}$, then the distance covered over a duration t of time is $r \cdot t \cdot w_{\mathrm{R/L}}$, because $t \cdot w_{\mathrm{R/L}}$ is the total angular displacement of the wheels, for more information see [112]. Using this

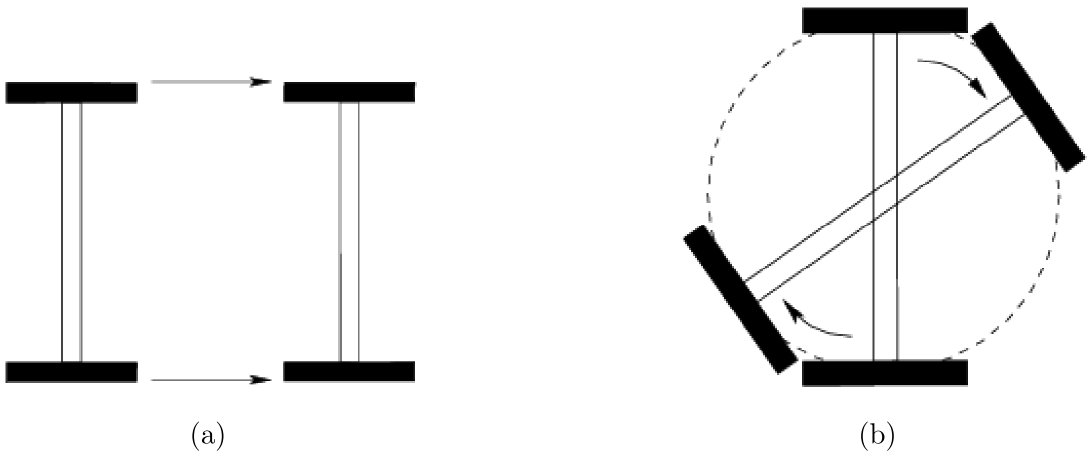


Figure 4.51: 4.51a Pure translation occurs when both wheels move at the same angular velocity; 4.51b pure rotation occurs when the wheels move at opposite velocities

model based on the difference in angular velocity of the two wheels, it is appropriate to take the angular velocities as new inputs of System 4.24 the angular velocities w_{R} e w_{L}.

To make the change of variable possible and pass from the linear speed v_{robot} and angular w of the robot to the angular velocities of each wheel, the following matrix is used:

$$\begin{bmatrix} w_{\mathrm{R}} \\ w_{\mathrm{L}} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{r}}{2} & \frac{\mathrm{r}}{2} \\ \frac{\mathrm{r}}{\mathrm{d}} & -\frac{\mathrm{r}}{\mathrm{d}} \end{bmatrix}^{-1} \begin{bmatrix} v_{robot} \\ w \end{bmatrix} \tag{4.25}$$

In this way, it is possible to rewrite the linear velocity v_{robot} and the angular w in terms of w_{R} and w_{L} as follows:

$$\begin{cases} v_{robot} = \frac{r}{2}w_{\mathrm{R}} + \frac{r}{2}w_{\mathrm{L}} \\ w = \frac{r}{d}w_{\mathrm{R}} - \frac{r}{d}w_{\mathrm{L}} \end{cases}$$

Replacing these values in the System 4.24, the equations that describe the vehicle kinematics become:

$$\begin{cases} \dot{x}_1 = \frac{r}{2}(w_{\mathrm{R}} + w_{\mathrm{L}})\cos(x_3) \\ \dot{x}_2 = \frac{r}{2}(w_{\mathrm{R}} + w_{\mathrm{L}})\sin(x_3) \\ \dot{x}_3 = \frac{r}{d}(w_{\mathrm{R}} - w_{\mathrm{L}}) \end{cases} \tag{4.26}$$

As can be noticed from System 4.26, the translation speed depends on the average of the angular wheel velocities, while the robot's rotation rate grows linearly with the wheel radius but it decreases linearly in relation to the distance between the wheels.
Moreover, from the third equation of System 4.26, it is possible to note that the robot can follow curves depending on the speed difference between the two wheels. Thus, the angular velocities w_L and w_R are seen directly as system inputs. However, the speed cannot be set directly, but through the PWM signals.

Motor Model

In this section, the problem of motor modeling is analyzed, as to define a relationship between the angular velocities, the inputs of the robot model, and the voltage signals (PWM) which are the signals actually supplied to the motor.
The best solution to this problem would be to directly apply encoders on the axis of each motor, so as to have the two speeds available in real time. Unfortunately, this solution is not viable due to the reduced dimensions of the motors of the vehicles, which do not allow the coupling of an encoder to its own axis. Thus, it is

decided to try to obtain them analytically.
In particular, in this thesis, this work tries to use a parametric ARX (AutoRegressive with eXogeneous input) model for describing the dynamics between the inputs (i.e. PWM) and outputs (i.e angular speed) of the motors.

Modelling Method System modelling plays an important role in the application of the control on the robot and therefore on the achievement of the objective of this thesis. However, it is also one of the more complicated tasks, as it is more closely connected with reality.
In control applications is not necessary to have a deep mathematical knowledge of the system under study, but it is sufficient to predict the system evolution and obtain sufficient robustness to parameter uncertainty. Therefore the study uses a black-box model. This model is used when either the internal structure of the system is unknown or there are no first principles available, the only chance is to collect data and use them to guess the links between inputs and outputs [113].
In a SISO model, as shown in Figure 4.52 where: input (u) is the PWM signal sent to the motor and output (y) is the relative angular speed. In this study, the method

Input
PWM(RIGHT)
Control System
Output
W(RIGHT)
(a)
Input
PWM(LEFT)
Control System
Output
W(LEFT)
(b)

Figure 4.52: Control System represented by SISO model, for right (a) and left (b) motor

adopted in order to model the process is based on a parametric identification of an ARX model.
The evolution of the estimated output allows to follow the dynamic evolution of the process and to detect the presence of fault from the variation of the estimated parameters of the identified model. An interesting insight into system identification using ARX model is reported in articles [114] [115].
When attempting to identify a model from extracted data it is common practice to follow four basic steps:

- the acquisition of dataset;
- a set of candidate models and the determination the best model in the set;
- parameters estimation;
- validation model;

Data acquisition is made possible by using a code written on Arduino, the infrared camera and Visual Studio (an integrated development environment). The other three steps are performed by using System Identification toolbox on Matlab. Before proceeding with the analysis of the model identification, a detailed description of the ARX model is given below.

ARX Model Structure The ARX structure describes the input effects $u^{arx}(t)$ on the process output $y^{arx}(t)$; in this case, the effects that the two PWM signals sent to Arduino have on the angular velocities w_{L} and w_{R}. By using this model, motor dynamic is defined as a discrete-time system.
The ARX model can be described as follows:

$$\begin{aligned} y^{arx}(t) = &- a_1 y^{arx}(t-1) - \dots - a_{n_a} y^{arx}(t-n_a) + b_1 u^{arx}(t-n_k) + \dots \\ &\dots + b_{n_b} u^{arx}(t-n_b-n_k+1) + e^{arx}(t) \end{aligned} \tag{4.27}$$

where $e^{arx}(t)$ refers to the noise supposed to be Gaussian, a_{n_a} and b_{n_b} are the model parameters, n_a indicates the polynomial degree of the output (system order) and n_b is the polynomial degree of the input, n_k is the time delay between $y^{arx}(t)$ and $u^{arx}(t)$.
This construction, of a discrete-time linear system is suitable for a punctual analysis o the process behavior (i.e. to evaluate the output value for each single instant of time).
The representation of the ARX model, given by the corresponding differential equation, is now defined in a simple way to allow its manipulation and implementation on a processor. For this purpose, it is possible to use the polynomial representation:

$$A^{arx}(z)y^{arx}(t) = B^{arx}(z)u^{arx}(t - n_k) + e^{arx}(t) \tag{4.28}$$

where $A^{arx}(z)$ and $B^{arx}(z)$ are given by:

$$A^{arx}(z) = 1 + a_1 z^{-1} + \cdots + a_{n_a} z^{-n_a} \tag{4.29}$$

$$B^{arx}(z) = b_1 z^{-1-n_k} + \cdots + b_{n_b} z^{-n_b-n_k} \tag{4.30}$$

and z^{-1} is the delay operator such as $y^{arx}(t) = z^{-1}u^{arx}(t)$.
$A^{arx}(z)$ and $B^{arx}(z)$ are estimated through the SystemIdentification toolbox on Matlab.

Data Collection As mentioned in the previous paragraph, the first step to identify the model is to collect the data. Inputs of this model are the Pulse Width Modulation (PWM) signals of both motors (left and right); while outputs are the relative angular velocities (w_{L} and w_{R}).
The purpose of this step is to collect a set of data that describe how the system behaves over its entire range of operation. The idea is to vary the inputs and observe the impact on the outputs.
By using a code written on Arduino, the robot is supplied with two sinusoidal PWM signals, (same signal for both motors). These sinusoidal signals (with time period of $t = 0.250\ s$) are sampled in order to collect 80 PWM values. Thanks to the infrared camera and a code written in Visual Studio, many values of the X and Y world coordinates, related to the positions assumed by the robot in the world reference system, are collected.

PWM Input Signals In our case, the robot is supplied with two identical sinusoidal PWM signals, one for each motor. As written above, the maximum value of the PWM signal is 255 and the minimum value is 0. However, a code is used where the signals sent to Arduino could assume a maximum value of +127 and a minimum of -127. Thus, PWM values are included in the interval [-127, + 127].

Since the inputs of the ARX model are voltage PWM signals, before proceeding with the identification of the model it is necessary to make the following proportion:

$$PWM_{\mathrm{Volt}} : 7.5 = PWM_{\mathrm{Arduino}} : 127 \tag{4.31}$$

The PWM signals sent to Arduino can assume, as already mentioned, negative values since they vary in the range [-127, +127], therefore also the voltage values can assume negative values, in particular the minimum allowed value is -7.5 volts.

Velocity Output Signals Once the PWM signals have been sent to Arduino, the robot moves on the plane and it is possible to visualize the x and y coordinates (expressed in meters) on Visual Studio. However, the outputs to be considered in the ARX model under consideration are, as already mentioned, the angular speeds of both motors, i.e. w_{R} and w_{L}.
First of all, the distance between each sampling point is calculated as:

$$d(P_1, P_2) = \sqrt{(x_1(t+1) - x_1(t))^2 + (x_2(t+1) - x_2(t))^2} \tag{4.32}$$

Then, the linear velocity v_{robot} can be obtained as $v_{robot} = d(P_1, P_2)/t$ while the angular speed w as $w = v_{robot}/$r.
It should be remarked that the kinematics of the mobile robot is specified by *differential drive* model equations. Therefore, the relation between linear velocity v and the angular velocities can be expressed through the equations (previously submitted):

$$\begin{cases} v_{robot} = \frac{r}{2} w_{\mathrm{R}} + \frac{r}{2} w_{\mathrm{L}} \\ w = \frac{r}{d} w_{\mathrm{R}} - \frac{r}{d} w_{\mathrm{L}} \end{cases}$$

So, it is possible to obtain the angular speeds of both motors by changing the variables, using the matrix:

$$\begin{bmatrix} w_{\mathrm{R}} \\ w_{\mathrm{L}} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{r}}{2} & \frac{\mathrm{r}}{2} \\ \frac{\mathrm{r}}{\mathrm{d}} & -\frac{\mathrm{r}}{\mathrm{d}} \end{bmatrix}^{-1} \begin{bmatrix} v_{robot} \\ w \end{bmatrix}$$

where d represents the distance between the wheels.

Therefore, once a sufficient number of experimental data is collected, see Figure 4.53,

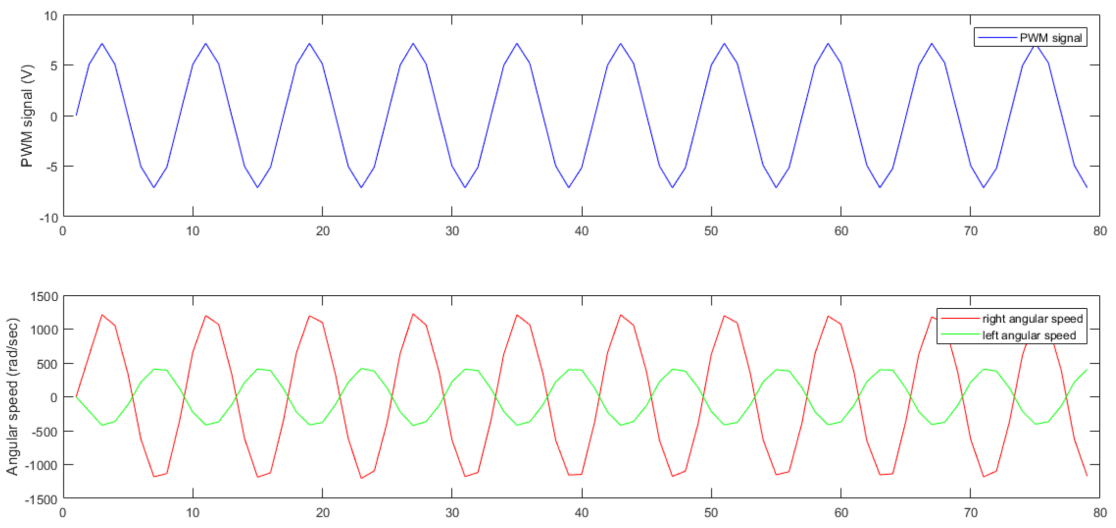


Figure 4.53: Sampled data plot

obtained by considering as inputs the PWM signals (voltage) and as outputs the angular velocities w_{L} and w_{R} (rad/sec), it is possible to use a toolbox on Matlab in order to identify the parameters of the ARX model.

System Identification It is possible to talk about parametric identification when the class of models can be represented by a structure that allows to describe the dynamics of input-output of the system with a finite number of numerical values, called parameters.

The identified model falls into the class of time-invariant linear models and it can be rewritten as:

$$y^{arx}(t) = G(z, \vartheta)u^{arx}(t) \tag{4.33}$$

where the term $G(z, \vartheta)$ represents the deterministic component of the model and describes the relation between the input $u^{arx}(t)$ and the output $y^{arx}(t)$, ϑ is the vector of the coefficients of the polynomials at the numerator and at the denominator of the rational function *G(z)*.
The transfer function of the LTI (4.33) model can be written as:

$$G(z, \vartheta) = \frac{B^{arx}(z)}{A^{arx}(z)} \tag{4.34}$$

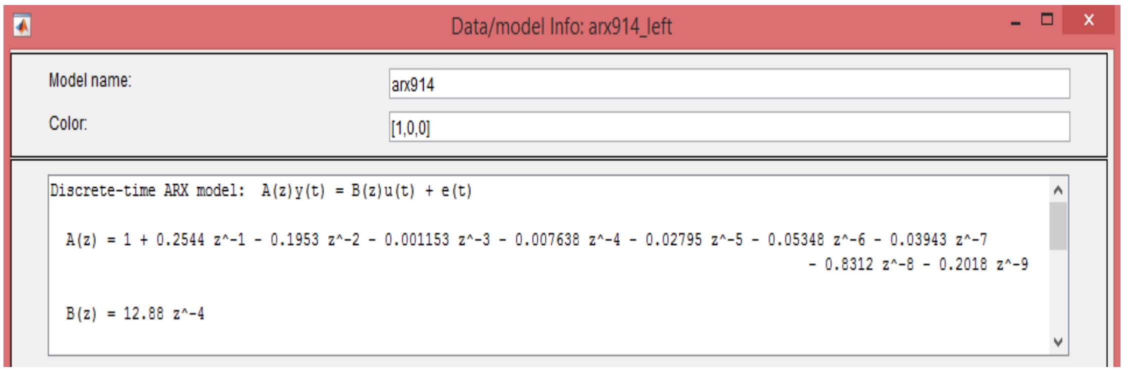
where A(z) and B(z) are the polynomials explained above (equations (4.29) and (4.30)), while ϑ is the vector of the parameters:

$$\vartheta = \Big[a_1 + \cdots + a_{n_a} + b_1 + \cdots + b_{n_b}\Big]'$$

Considering Equation 4.27 of the ARX model, since the error $e^{arx}(t)$ is filtered by a transfer function that has the same transfer function denominator of *G(z)*, i.e the deterministic channel, it is neglected in Equation 4.33.
In order to estimate the value of the parameters and validate the model in question, Matlab SystemIdentification toolbox is used.

Parameter Estimation Sampled data are imported into the System Identification app. The offsets are removed from the data by subtracting the mean values of the input and the output. The mean values are subtracted from each signal because linear models that describe the responses to deviations from a physical equilibrium are built. With steady-state data, it is reasonable to assume that the mean levels of the signals correspond to such an equilibrium.
Removing means, adds a new data set to the System Identification app, which is used for estimating models. These data are split into two parts, the first part for model estimation and the second part for model validation.
Then it is possible to specify the model orders n_a, n_b, and n_k to estimate ARX models.
At first the orders are not specified, but a range of [1:10] is set for each order. In this way simple polynomial models are estimated for a range of orders and delays. By

comparing the obtained models, the best model that fits the data is chosen. Then, the System Identification Toolbox estimate the parameters $a_1 \dots a_n$ and $b_1 \dots b_n$ from the data.
The parameters obtained are the same for both models (i.e. right and left motor models).
In particular, as shown in Figure 4.54, nine alpha parameters (the first one, always equal to one, is not considered because a unitary coefficient of $y^{arx}(t)$) and only one beta parameter are obtained.


Data/model Info: arx914_left

Model name: arx914

Color: [1,0,0]

```
Discrete-time ARX model:  A(z)y(t) = B(z)u(t) + e(t)

  A(z) = 1 + 0.2544 z^-1 - 0.1953 z^-2 - 0.001153 z^-3 - 0.007638 z^-4 - 0.02795 z^-5 - 0.05348 z^-6 - 0.03943 z^-7
                                                                                      - 0.8312 z^-8 - 0.2018 z^-9

  B(z) = 12.88 z^-4
```

Figure 4.54: System Identification Toolbox: parameters estimation

So, considering the values of the estimated parameters and that $n_a = 9$, $n_b = 1$ and $n_k = 4$, it is possible to write ARX model as:

$$\begin{aligned} y^{arx}(t) =& 0.2544 \cdot y^{arx}(t-1) - 0.1953 \cdot y^{arx}(t-2) - 0.001153 \cdot y^{arx}(t-3)+ \\ & - 0.007638 \cdot y^{arx}(t-4) - 0.02795 \cdot y^{arx}(t-5) - 0.05348 \cdot y^{arx}(t-6)+ \\ & - 0.03943 \cdot y^{arx}(t-7) - 0.8312 \cdot y^{arx}(t-8) - 0.2018 \cdot y^{arx}(t-9)+ \\ & + 12.88 \cdot u^{arx}(t-4) \end{aligned} \tag{4.35}$$

Model Validation The last step to be taken, during this identification study, is the analysis of the results, checking the accuracy of the proposed model.
Figure 4.55 shows the model response to the input in validation data. The fit values for each model are summarized in the Best fit area of the Model Output window. The models in the Best Fits list are ordered from the best at the top to worst at the

bottom. The fit between the two curves is computed so that 100 means a perfect fit, and 0 indicates a poor fit.
In this case, the output of the model matches the validation data output, therefore the model seems to capture the main system dynamics and thus it highlights that the use of linear modeling is more than enough.

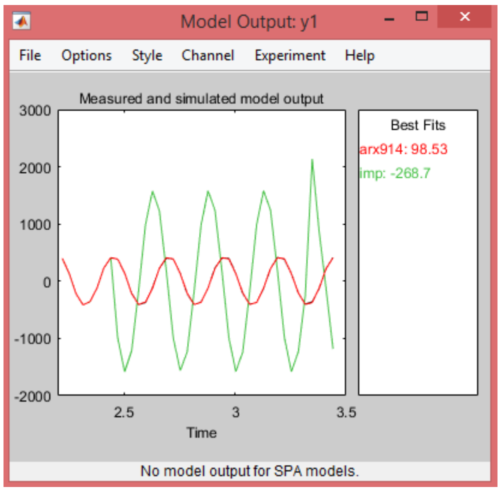


Figure 4.55: Model-output plot

As can be noticed from Figure 4.55 a satisfactory agreement between identified and experimental data is found. The ARX 914 predicts the evolution of motor angular velocity in a successfully way.

Robot and Motor Model

Once a model that describes the motor dynamic is obtained, it is possible to include it in the robot model, which is explained in Section 4.6.2, in order to get a more complete robot model.
In this way, the new state variables are the outputs of the ARX model, the $x_1 = x$ $x_2 = y$ coordinates and the angle $x_3 = \theta$ that describe the position and orientation of the vehicle in the two-dimensional space.
Furthermore, in order to apply the predictive control, a discretization of the continuous-time model that describes the dynamics of the robot (Equation 4.26) is necessary.

$$\begin{cases} \frac{x_1(t)-x_1(t-1)}{t_s} = \frac{r}{2}\left(w_{\mathrm{R}}(t)+w_{\mathrm{L}}(t)\right)\cos(x_3(t)) \\ \frac{x_2(t)-x_2(t-1)}{t_s} = \frac{r}{2}\left(w_{\mathrm{R}}(t)+w_{\mathrm{L}}(t)\right)\sin(x_3(t)) \\ \frac{x_3(t)-x_3(t-1)}{t_s} = \frac{r}{d}(w_{\mathrm{R}}(t)-w_{\mathrm{L}}(t)) \end{cases}$$

$$\begin{cases} x_1(t) = x_1(t-1) + t_s\frac{r}{2}(w_{\mathrm{R}}(t)+w_{\mathrm{L}}(t))\cos(x_3(t)) \\ x_2(t) = x_2(t-1) + t_s\frac{r}{2}(w_{\mathrm{R}}(t)+w_{\mathrm{L}}(t))\sin(x_3(t)) \\ x_3(t) = x_3(t-1) + t_s\frac{r}{d}(w_{\mathrm{R}}(t)-w_{\mathrm{L}}(t)) \end{cases} \tag{4.36}$$

where t_s is the sampling time. The same sampling time is considered to identify both the points within optimal trajectory and model.
Integrating Equations 4.36 with those related to the ARX 914 model (i.e. Equation 4.27), the robot discrete-time system used in this work can be defined as follows:

$$\begin{cases} y_{\mathrm{R}}(t) = -\alpha_1 y_{\mathrm{R}}(t-1) - \alpha_2 y_{\mathrm{R}}(t-2) - \alpha_3 y_{\mathrm{R}}(t-3) \cdots - \alpha_9 y_{\mathrm{R}}(t-9) + \beta_4 u_{\mathrm{R}}(t-4) \\ y_{\mathrm{R}}(t-1) = y_{\mathrm{R}}(t-1) \\ \vdots \\ y_{\mathrm{L}}(t) = -\alpha_1 y_{\mathrm{L}}(t-1) - \alpha_2 y_{\mathrm{L}}(t-2) - \alpha_3 y_{\mathrm{L}}(t-3) \cdots - \alpha_9 y_{\mathrm{L}}(t-9) + \beta_4 u_{\mathrm{L}}(t-4) \\ y_{\mathrm{L}}(t-1) = y_{\mathrm{L}}(t-1) \\ \vdots \\ x_1(t) = x(t-1) + t_s\frac{\mathrm{r}}{2}(y_{\mathrm{R}}(t)+y_{\mathrm{L}}(t))cos(x_3(t)) \\ x_2(t) = x_2(t-1) + t_s\frac{\mathrm{r}}{2}(y_{\mathrm{R}}(t)+y_{\mathrm{L}}(t))sin(x_3(t)) \\ x_3(t) = x_3(t-1) + t_s\frac{\mathrm{r}}{d}(y_{\mathrm{R}}(t)-y_{\mathrm{L}}(t)) \end{cases} \tag{4.37}$$

The number of state variables depends on α and β parameters estimated with the model identification. Note that the $y_{L,R}$ and $u_{L,R}$ are the outputs and the inputs of the ARX models. However, for simplicity in the following sections the apex *arx* is not used.

Predictive Control

Fundamental to the successful autonomous operation of the mobile robot is the motion control algorithm. The goal of the controller is to let the vehicle stay as close as possible to the reference route.
Although in literature there are different approaches to trajectory control of a mobile robot, in this work a Model Predictive Control (MPC) is used. In the vehicle control field, this technique is already used with success to achieve various objectives.
MPC control is the result of iterative application of the Optimal Control Problem (OCP). Predictive control uses a system model to predict the process output over some future horizon (Prediction Horizon) of length N. By applying the MPC algorithm, a series of manipulated inputs ($U_o = [u_0, u_1 \dots u_{N-1}]$) is computed so that the predicted future output is as precise as possible. Only the input related to the first period (u_0) of the open loop solution is applied to the system.
Once the OPC has been solved and new measurements become available, the estimation and regulation windows are shifted and the estimation and optimization procedures are repeated. Since MPC solves a new optimization problem at each time step, it requires a high computational effort.
In the following sections, the MPC features will be introduced with reference to [116].

MPC Technique

The Model Predictive Control (MPC) is a particular control architecture widespread in the industrial field, to which in recent years, numerous studies have been dedicated in order to determining conditions that guarantee stability properties and robustness to uncertainties.
The basic idea of this technique is to transform the classic control problem into a problem of mathematical optimization, so as to be able to easily insert constraints and limitations of the real system.

In particular, the model of the entire system is used to predict, on a finite horizon, the evolution of the state variables, starting from the values that they assume at the current instant and according to the sequence of future control inputs. In this way, since this sequence of inputs is a degree of freedom in the considered problem, the prediction can be used as a figure of merit to ensure that the state properly follows a given objective, producing in output exactly the optimum input values that allow to achieve the desired performances.
In accordance with the Receding Horizon (RH) technique, only the first inputs calculated by the optimizer are applied to the system and the procedure is repeated again based on the updated measurement of the current state. This approach allows to formulate the control problem by taking into account constraints on the control, states and outputs variables throughout the predictive horizon, in a simple way.
All the operations carried out by the MPC controller are schematized in Figure 4.56.
Therefore, the main ingredients of an MPC algorithm are:

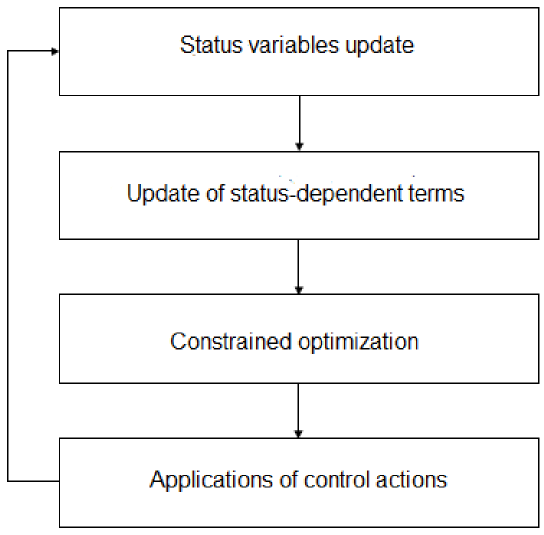

Status variables update
Update of status-dependent terms
Constrained optimization
Applications of control actions

Figure 4.56: MPC algorithm operations

- a process model, usually in discrete-time;
- a cost function J defined, at any time instant k, over a finite horizon $[k, k+N]$;
- an optimization algorithm computing the future optimal control sequence;
- input, output and state constraints;

- the so called RH principle.

Receding Horizon Principle

An aspect of the MPC control that deserves special attention is the *receding horizon approach*, figure 4.57.

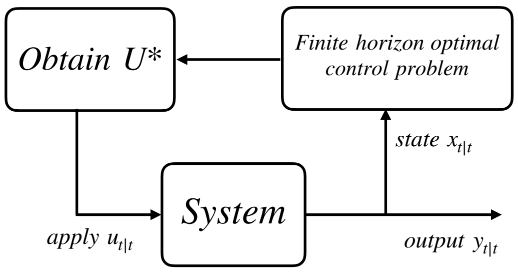


Figure 4.57: Receding horizon approach

At any time instant t, based on the available process information, the controller solves the optimization problem in relation to the future control sequence [$u(t), \ldots, u(t+N-1)$] and applies only its first element $u^o(t)$. Then, at the next time instant $t+1$, a new optimization problem is solved, based on the process information available at time $t+1$, along the prediction horizon $[t+1, \ldots, t+N]$ [116]. This technique is also defined "mobile horizon", see Figure 4.58, since the prediction horizon is always of the same duration but it is moved one step forward at each iteration.

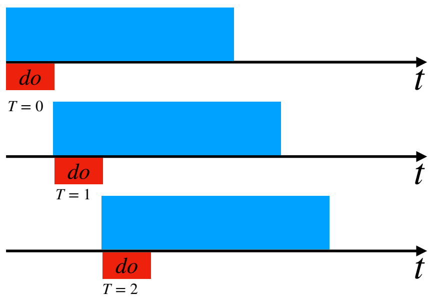


Figure 4.58: Receding horizon control

The new control sequence is generally different from the previous one, therefore, since the predictive horizon is moved forward and new measures of the state of the system $x(t+1)$ are used, a good robustness is obtained against possible modeling errors and disturbances.

This technique can be compared to a chess game, in which each player elaborates a strategy according to the current scenario and the possible choices of the opponent; however if the opponent responds to an unexpected move it is necessary to re-evaluate your position by rescheduling the choices to be made.
Obviously, good players are able to foresee the evolution of the game thinking about different advanced scenarios over time (long prediction horizons). It is clear that the longer the chosen horizon is, the more precise the obtained control results will be; however, the overall computational burden becomes heavier.
By extending the prediction horizon, all the operations performed by the controller become more time-consuming.
The horizon must be chosen coherently with the available computing resources since it is obvious that greater iterations imply greater computational costs.

MPC with Transfer Function Models

The system that describes the dynamic process of the robot motors can be rewritten by the transfer function:

$$G(z) = \frac{B^{arx}(z)}{A^{arx}(z)} = \frac{b_{n-1}z^{n-1} + b_{n-2}z^{n-2} + \cdots + b_0}{z^n + a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + \cdots + a_0} \tag{4.38}$$

Therefore, to implement the motor dynamics in the predictive control algorithm in a simple way, it is possible to consider the single motor system in the following non minimal state-space form:

$$x^{arx}(t+1) = A^{arx} \cdot x^{arx}(t) + B^{arx} \cdot u^{arx}(t)$$

where

$$x^{arx}(t+1) = \begin{bmatrix} y^{arx}(t+1) \\ y^{arx}(t) \\ \vdots \\ y^{arx}(t-n+2) \\ u^{arx}(t) \\ u^{arx}(t-1) \\ \vdots \\ u^{arx}(t-n+2) \end{bmatrix}$$

$$A^{arx} = \begin{bmatrix} -a_{n-1} & -a_{n-2} & \dots & -a_0 & b_{n-2} & b_{n-3} & \dots & b_0 \\ 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B^{arx} = \begin{bmatrix} b_{n-1} \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

LTV Approximation of the Robot Non-Linear System

In this study a linear MPC is used, in order to have a quadratic cost function, which is easily minimizable as it has to be simply set to zero, and consequently it streamlines the optimization calculations.
A problem immediately arises regarding the compatibility and applicability of this type of MPC to the non-linear model to be used. Since a model-based control is used there are two possibilities: changing the MPC and making it non-linear, or linearizing the model. The second option is more in line with the set objectives, namely the reduction of calculation times in an attempt to make a real-time

implementation feasible.
Then, the system is linearized around the predefined trajectory $(\hat{X}, \hat{U})$ obtaining matrices $\hat{A}(t)$ and $\hat{B}$, where only $\hat{A}(t)$ is time-variant.
Simple linearization around the initial state is not sufficient because the position and orientation of the robot vary significantly when following a given trajectory. It is therefore necessary to linearize around the trajectory to obtain references both to the states and to the inputs.
For example, the system model, based on ARX 914 and linearized around a given optimal trajectory $(\hat{X}, \hat{U})$, in this case a Lemniscata, can be written as follows:

$$\delta x(t+1) = \hat{A}(t)\delta x(t) + \hat{B}\delta u(t) \tag{4.39}$$

where:

$$\delta x(t) = x(t) - \hat{x}(t) \qquad \hat{x}(t) \in \hat{X}$$

$$\delta u(t) = u(t) - \hat{u}(t) = \begin{bmatrix} u_R(t) - \hat{u}_R(t) \\ u_L(t) - \hat{u}_L(t) \end{bmatrix} \qquad \hat{u}(t) \in \hat{U}$$

$$\hat{B} = \begin{bmatrix} 0 & 0 & \dots & 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

$$\delta x(t+1) = \begin{bmatrix} y_{\mathrm{R}}(t+1) - \hat{y}_{\mathrm{R}}(t+1) \\ y_{\mathrm{R}}(t) - \hat{y}_{\mathrm{R}}(t) \\ \vdots \\ u_{\mathrm{R}}(t) - \hat{u}_{\mathrm{R}}(t) \\ u_{\mathrm{R}}(t-1) - \hat{u}_{\mathrm{R}}(t-1) \\ u_{\mathrm{R}}(t-2) - \hat{u}_{\mathrm{R}}(t-2) \\ y_{\mathrm{L}}(t+1) - \hat{y}_{\mathrm{L}}(t+1) \\ y_{\mathrm{L}}(t) - \hat{y}_{\mathrm{L}}(t) \\ \vdots \\ u_{\mathrm{L}}(t) - \hat{u}_{\mathrm{L}}(t) \\ u_{\mathrm{L}}(t-1) - \hat{u}_{\mathrm{L}}(t-1) \\ u_{\mathrm{L}}(t-2) - \hat{u}_{\mathrm{L}}(t-2) \\ x_1(t+1) - \hat{x}_1(t+1) \\ x_2(t+1) - \hat{x}_2(t+1) \\ x_3(t+1) - \hat{x}_3(t+1) \end{bmatrix}, \delta x(t) = \begin{bmatrix} y_{\mathrm{R}}(t) - \hat{y}_{\mathrm{R}}(t) \\ y_{\mathrm{R}}(t-1) - \hat{y}_{\mathrm{R}}(t-1) \\ \vdots \\ u_{\mathrm{R}}(t-1) - \hat{u}_{\mathrm{R}}(t-1) \\ u_{\mathrm{R}}(t-2) - \hat{u}_{\mathrm{R}}(t-2) \\ u_{\mathrm{R}}(t-3) - \hat{u}_{\mathrm{R}}(t-3) \\ y_{\mathrm{L}}(t) - \hat{y}_{\mathrm{L}}(t) \\ y_{\mathrm{L}}(t-1) - \hat{y}_{\mathrm{L}}(t-1) \\ \vdots \\ u_{\mathrm{L}}(t-1) - \hat{u}_{\mathrm{L}}(t-1) \\ u_{\mathrm{L}}(t-2) - \hat{u}_{\mathrm{L}}(t-2) \\ u_{\mathrm{L}}(t-3) - \hat{u}_{\mathrm{L}}(t-3) \\ x_1(t) - \hat{x}_1(t) \\ x_2(t) - \hat{x}_2(t) \\ x_3(t) - \hat{x}_3(t) \end{bmatrix}$$

$$\hat{A}(t) = \begin{bmatrix}
-a_1 & \dots & -a_9 & 0 & 0 & b_4 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & 0 & 0 & 0 \\
0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \dots & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \dots & 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \dots & 0 & 0 & 0 & 0 & -a_1 & \dots & -a_9 & 0 & 0 & b_4 & 0 & 0 & 0 \\
0 & \dots & 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & 0 & 0 & 0 \\
0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
t_s \frac{r}{2}\cos(\hat{x}_3(t)) & \dots & 0 & 0 & 0 & 0 & t_s \frac{r}{2}\cos(\hat{x}_3(t)) & \dots & 0 & 0 & 0 & 0 & 1 & 0 & -t_s \frac{r}{2}(\hat{w}_{\mathrm{L}}(t) + \hat{w}_{\mathrm{R}}(t))\sin(\hat{x}_3(t)) \\
t_s \frac{r}{2}\sin(\hat{x}_3(t)) & \dots & 0 & 0 & 0 & 0 & t_s \frac{r}{2}\cos(\hat{x}_3(t)) & \dots & 0 & 0 & 0 & 0 & 0 & 1 & t_s \frac{r}{2}(\hat{w}_{\mathrm{L}}(t) + \hat{w}_{\mathrm{R}}(t))\cos(\hat{x}_3(t)) \\
t_s \frac{r}{d} & \dots & 0 & 0 & 0 & 0 & -t_s \frac{r}{d} & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \tag{4.40}$$

Model Predictive Control Application

This section describes the procedures and the choices made to implement the MPC control, in order to compute the *path tracking*.
As already mentioned, in this work the trajectory to be followed is given by the dispatching strategy described in Chapter 2 and Chapter 3.
In the optimization problem, the controller makes a prediction of the output variables of the system. Considering these variables, it defines the values of the control variables so that a cost function is minimized. This is typically represented by the error in relation to a given reference.
The result is therefore a sequence of optimal control PWM signals. The length of the sequence depends on the value of the prediction horizon (N). For the instant when the optimization is performed only the first element is actually applied to the system.

Nominal Input Sequence

To control the mobile robot, once the trajectory has been established ($\hat{X}$), it is necessary to calculate a correct sequence of impulses to be sent to the two motors ($\hat{U}$).
Given the kinematics equations and the points belonging to the trajectory, to calculate the actual angular velocities $\hat{w}_{\mathrm{R}}(t)$ and $\hat{w}_{\mathrm{L}}(t)$, it is necessary to impose the trajectory coordinates $(\hat{x}_1(t), \hat{x}_2(t), \hat{x}_3(t))$ as equilibrium points for the robot model 4.26. Since the trajectory is a finite sequence of points, it is possible to find the relative input sequence that ensures to reach the reference trajectory.
However, in practice, it is not sufficient to know the value of the angular speeds since the motor inputs are the PWM voltage signals.
Based on the linear dynamic relationship created between the two variables (i.e. input and output of the motor), it is possible to trace two different sequences of PWM values (one for each motor) necessary for the vehicle to follow the optimal trajectory.
It is assumed that the robot starts from stationary condition, with angular speeds equal to zero. At the next time instant, when the first speed is different from zero,

the relative voltage signal is obtained according to the inverse formula:

$$\hat{u}(t-n_k) = \frac{\hat{y}(t)}{b_1} \qquad (t=1) \tag{4.41}$$

note that n_k is the delay between input and output signals previously identified.
At the time t=2 the input signal is given by:

$$\hat{u}(t-n_k) = \frac{\hat{y}(t) + a_1\hat{y}(t-1)}{b_1} \qquad (t=2) \tag{4.42}$$

and so on until as many PWM values as the points describing the trajectory are obtained:

$$\hat{u}(t-n_k) = \frac{\hat{y}(t) + a_1\hat{y}(t-1) + \cdots + a_{n_a}\hat{y}(t-n_a)}{b_1} \quad (t>0) \tag{4.43}$$

In particular, the inputs just calculated are considered piecewise constants, therefore:

$$\hat{u}(t) = \hat{u}(k) \qquad with \quad kt_s \le t < (k+1)t_s$$

where t_s is the sampling time.
When the optimization is performed, it is possible to consider a sequence of nominal input and states variables, respectively:

$$\hat{U}(t) = [\hat{u}(t), \cdots, \hat{u}(t+N-1)]^T \tag{4.44}$$

and

$$\hat{X}(t) = [\hat{x}(t), \cdots, \hat{x}(t+N-1)]^T \tag{4.45}$$

both vectors have a size equal to N (i.e. the prediction horizon).

QP Formulation

The MPC linear control configures an optimal control problem with a quadratic cost function. In this section, the formulation of constructing the matrices in order

to solve the quadratic programming (QP) problem is described. The QP matrices, of appropriate size, depend directly on the matrices $\hat{A}$ and $\hat{B}$ and must take into account the constraints imposed in the design phase, which will be discussed in more detail in the following paragraph.
For the MPC application, an alternative quadratic problem formulation is used, based on which both δX and δU are optimized simultaneously.
The prediction model is determined by the following equation:

$$\delta X(t) = \hat{\mathcal{A}}(t)\delta X(t) + \hat{\mathcal{B}}\delta U(t) + \mathcal{H}\delta x_{t|t} \tag{4.46}$$

This linear discrete time approximation is called Linear Time Variant (LTV) model, over a period of time $t \in [k_0 t_s \dots (k_0 + N)t_s]$.
In particular:

$$\delta X(t) = X(t) - \hat{X}(t) = [\delta x_{t|t}^T, \delta x_{t+1|t}^T, \cdots, \delta x_{t+N-1|t}^T]^T$$

$$\delta U(t) = U(t) - \hat{U}(t) = [\delta u_{t|t}^T, \delta u_{t+1|t}^T, \cdots, \delta u_{t+N-1|t}^T]^T$$

where: $\delta u_{t+k|t} = \begin{bmatrix} u_R(t+k|t) - \hat{u}_R(t+k|t) \\ u_L(t+k|t) - \hat{u}_L(t+k|t) \end{bmatrix} \quad \forall k \in \{0, \cdots, N-1\}$

$$\hat{\mathcal{A}}(t) = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ \hat{A}(t) & 0 & 0 & \dots & 0 \\ 0 & \hat{A}(t+1) & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \hat{A}(t+N-1) & 0 \end{bmatrix}$$

$$\hat{\mathcal{B}} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ \hat{B} & 0 & 0 & \dots & 0 \\ 0 & \hat{B} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \hat{B} & 0 \end{bmatrix}, \qquad \mathcal{H} = \begin{bmatrix} I \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The final optimization problem is:

$$\begin{aligned} \underset{\mathcal{Z}(t)}{\text{minimize}} \quad & \mathcal{Z}^T(t)\bar{\mathcal{Q}}\mathcal{Z}(t) \\ \text{subject to} \quad & \varepsilon_z \mathcal{Z}(t) \le \mathcal{F}_z(t) \\ & [I - \hat{\mathcal{A}}(t), \, -\hat{\mathcal{B}}]\mathcal{Z}(t) = \mathcal{H}\delta x_{t|t} \end{aligned} \tag{4.47}$$

where

$$\bar{\mathcal{Q}} = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}, \qquad \varepsilon_z = \begin{bmatrix} \varepsilon_x & 0 \\ 0 & \varepsilon_u \end{bmatrix}, \qquad \mathcal{F}_z(t) = \begin{bmatrix} \mathcal{F}_x(t) \\ \mathcal{F}_u(t) \end{bmatrix}$$

$$\mathcal{Z}(t) = \begin{bmatrix} \delta X(t) \\ \delta U(t) \end{bmatrix} = \begin{bmatrix} X(t) - \hat{X}(t) \\ \delta U(t) - \hat{U}(t) \end{bmatrix}$$

and $\hat{U}(t)$ and $\hat{X}(t)$ are vectors composed of inputs and states related to the reference trajectory, defined respectively in (4.44) and (4.45).

The matrices Q and R are positive-defined symmetric matrices and represent the weights on states and on inputs, respectively.

At this point, it is possible to use the *quadprog* function in order to solve the optimization problem. Note that if $\bar{\mathcal{Q}}$ is not symmetrical, it must be made symmetrical by using the following simple formula: $\frac{\bar{\mathcal{Q}}+\bar{\mathcal{Q}}^T}{2}$.

In conclusion, at each time step t, the optimization problem 4.47 is solved and the optimal value of the decision variable $\mathcal{Z}$ is found, i.e $\mathcal{Z}^o(t) = [\delta X^o(t), \delta U^o(t)]^T$.

Hence, only $u^o(t) = \begin{bmatrix} u^o_R(t) \\ u^o_L(t) \end{bmatrix} = \delta u^o_{t|t} + \hat{u}(t)$ is then applied to the robot.

State and Control constraints

All processes are subject to constraints, necessary for constructive reasons, industrial and environmental safety or simply purely economic considerations. In the considered case, both the control variable and the states related to the x_1, x_2 coordinates and angle x_3 are constrained, because of the physical characteristics of the wheel motors and the workplace, respectively.
In particular, the control variables are limited both in amplitude and variation (i.e. slew rate). The first is due to the maximum capacity of the available robot motor battery (7.5 V), while the latter allows to achieve a better performance.
The x_1 and x_2 coordinates can assume a maximum value, as the mobile robot is forced to move in a limited space.
It is important to note that the MPC is able to manage rigid constraints on the system in a non-conservative way, so the $\mathcal{F}_z(t)$ matrix, necessary for entering the constraints, is recalculated at every moment.
Since all constraints are linear inequalities, it is possible to write:

$$U \in \mathbb{U}^N = \{U : \varepsilon_u \delta U(t) \leq \mathcal{F}_u(t)\}$$
$$X \in \mathbb{X}^N = \{X : \varepsilon_x \delta X(t) \leq \mathcal{F}_x(t)\}$$

where:

$$\varepsilon_u = \begin{bmatrix} \bar{\varepsilon}_u & 0 & 0 & 0 \\ 0 & \bar{\varepsilon}_u & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \bar{\varepsilon}_u \\ \bar{\varepsilon}_u & 0 & 0 & 0 \\ -\bar{\varepsilon}_u & \bar{\varepsilon}_u & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -\bar{\varepsilon}_u & \bar{\varepsilon}_u \end{bmatrix} \qquad \mathcal{F}_u(t) = \begin{bmatrix} \bar{\mathcal{F}}_u(t) \\ \vdots \\ \bar{\mathcal{F}}_u(t+N-1) \\ \mathcal{F}_{slew}(t) \\ \vdots \\ \mathcal{F}_{slew}(t+N-1) \end{bmatrix}$$

$$\varepsilon_x = \begin{bmatrix} \bar{\varepsilon}_x & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \bar{\varepsilon}_x \end{bmatrix} \qquad \mathcal{F}_x(t) = \begin{bmatrix} \bar{\mathcal{F}}_x(t) \\ \vdots \\ \bar{\mathcal{F}}_x(t+N-1) \end{bmatrix}$$

In particular the matrices $\bar{\varepsilon}_x$, $\bar{\varepsilon}_x$ are defined as follows:

$$\bar{\varepsilon}_x = \begin{bmatrix} 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & \dots & -1 & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & 0 & -1 & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 \\ 0 & 0 & \dots & 0 & 0 & -1 \end{bmatrix} \qquad \bar{\varepsilon}_u = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix},$$

Note that from the construction of the $\bar{\varepsilon}_x$ matrix, the state constraints are applied only to the x_1, x_2 coordinates and to the angle x_3.

With regard to vectors $\mathcal{F}_x(t)$ and $\mathcal{F}_u(t)$:

$$\bar{\mathcal{F}}_x(t{+}k) = \begin{bmatrix} x_1^{max} - \hat{x}_1(t+k) \\ x_1^{max} + \hat{x}_1(t+k) \\ x_2^{max} - \hat{x}_2(t+k) \\ x_2^{max} + \hat{x}_2(t+k) \\ x_3^{max} - \hat{x}_3(t+k) \\ x_3^{max} + \hat{x}_3(t+k) \end{bmatrix} \qquad \bar{\mathcal{F}}_u(t{+}k) = \begin{bmatrix} u_{max} - \hat{u}_R(t+k) \\ u_{max} + \hat{u}_R(t+k) \\ u_{max} - \hat{u}_L(t+k) \\ u_{max} + \hat{u}_L(t+k) \end{bmatrix} \qquad \forall k \in \{0, \cdots, N{-}1\}$$

$$\mathcal{F}_{slew}(t+k) = \begin{bmatrix} \Delta u - \hat{u}_R(t+k) + u_R^o(t-1) \\ \Delta u + \hat{u}_R(t+k) - u_R^o(t-1) \\ \Delta u - \hat{u}_L(t+k) + u_L^o(t-1) \\ \Delta u + \hat{u}_L(t+k) - u_L^o(t-1) \end{bmatrix} \quad k = 0$$

$$\mathcal{F}_{slew}(t+k) = \begin{bmatrix} \Delta u - \hat{u}_R(t+k) + \hat{u}_R(t+k-1) \\ \Delta u + \hat{u}_R(t+k) - \hat{u}_R(t+k-1) \\ \Delta u - \hat{u}_L(t+k) + \hat{u}_L(t+k-1) \\ \Delta u + \hat{u}_R(t+k) - \hat{u}_R(t+k-1) \end{bmatrix} \quad \forall k \in \{1, \cdots, N-1\}$$

where: $x^1_{max}, x^2_{max}, x^3_{max}$ and u_{max} are the values that, added to or subtracted from the variables, determine the allowed range of variability; Δu is the maximum slew rate value (i.e. the maximum difference between the PWM signal at the step k and the one at the previous step $k-1$); $u^o_R(t-1)$ and $u^o_L(t-1)$ are the optimal values applied to the robot motors at the previous time instant, i.e. $t-1$.
Since it is supposed that the robot at time $t = 0$ has zero velocity, both values of $u^o_R(t-1)$ and $u^o_L(t-1)$ are set equal to zero for $t = 0$.
Note that MPC aims to control the robot motors in order to follow the predefined trajectory. To achieve this, the MPC uses the time variant matrix (i.e. $\hat{A}(t)$) obtained by linearizing the system around the predefined trajectory, i.e. $\hat{X}$ and $\hat{U}$. These matrices describe the dynamic of the robot with a good approximation only around the point of the trajectory used as an equilibrium point. Since the robot might not start on the trajectory, it is of crucial importance to select and use the correct matrices, which are the matrices that best approximate the robot dynamic, during the optimizations. Otherwise, the risk of obtaining wrong control actions would be greater. Thus, in order to try to improve the performance of the control, at each instant, the point ($p_{nearest}$) of the reference trajectory closest to the position of the robot is found by computing the norm of the distance between the current position of the robot and the points of the reference trajectory. Hence, each optimization considers the matrices starting from $p_{start} = p_{nearest}$. However, if the robot is exactly on the trajectory, the MPC finds a zero control action, which implies that the robot remains at the equilibrium point. For this reason, when the difference between the vehicle position and the closest reference point is less than 0.1%, the closest point is moved to the next equilibrium point, i.e $p_{\text{star}} = p_{\text{nearest}} + 1$. However, the robot may not be able to exactly reproduce the ideal trajectory, so it is the optimization itself that generates, in addition to the actual control actions, a

reference trajectory that is obviously as similar as possible to the desired one, but which, at the same time, is regular if the predefined trajectory is not. This reduces the risk of infeasibility due to the constraints imposed, and the new trajectory calculated by the optimization will consequently become the real reference that the vehicle must follow.

So, in order to improve the performance of the MPC in this work the robot model around the MPC open loop solution is linearized after each optimization. Thus, after each optimization, in accordance with the RH approach, only the first optimal control action $u^o(t) = \begin{bmatrix} \delta u^o_R(t|t) + \hat{u}_R(t|t) \\ \delta u^o_L(t|t) + \hat{u}_L(t|t) \end{bmatrix}$ is applied to the robot.

Then, in order to obtain the trajectory that the vehicle will follow in the next horizon (i.e. $\tilde{X}(t)$), it is necessary to virtually apply the inputs $\tilde{U}(t)$ to the robot model, where $\tilde{U}(t)$ is obtained as follows:

$$\tilde{U}(t) = \begin{bmatrix} \delta u^o_{t+1|t} \\ \vdots \\ \delta u^o_{t+N-1|t} \\ \delta u^o_{t+N-1|t} \end{bmatrix} = \begin{bmatrix} \hat{u}^o_{t+1|t} \\ \vdots \\ \hat{u}^o_{t+N-1|t} \\ \hat{u}^o_{t+N-1|t} \end{bmatrix}$$

Successively, the model is linearized, at each step, around $\tilde{X}(t)$ and $\tilde{U}(t)$ and no longer around the reference trajectory $\hat{X}(t)$ and $\hat{U}(t)$, obtaining $\tilde{\mathcal{A}}(t)$ and $\tilde{\mathcal{B}}$. Note that to enforce the robot to follow the reference trajectory $(\hat{X}, \hat{U})$ the new optimization is rewritten as follows:

$$\begin{aligned} \underset{\delta\tilde{X}(t),\delta\tilde{U}(t)}{\text{minimize}} \quad & J_x\left(\delta\tilde{X}(t) + \tilde{X}(t) - \hat{X}(t)\right) + J_u\left(\delta\tilde{U}(t) + \tilde{U}(t) - \hat{U}(t)\right) \\ \text{subject to} \quad & \varepsilon_z \begin{bmatrix} \delta\tilde{X}(t) \\ \delta\tilde{U}(t) \end{bmatrix} \leq \tilde{\mathcal{F}}_z(t) \\ & [I - \tilde{\mathcal{A}}(t), \, -\tilde{\mathcal{B}}] \begin{bmatrix} \delta\tilde{X}(t) \\ \delta\tilde{U}(t) \end{bmatrix} = \mathcal{H}\delta\tilde{x}_{t|t} \end{aligned}$$

where:

$$
\begin{aligned}
&J_x\left(\delta\tilde{X}(t)+\tilde{X}(t)-\hat{X}(t)\right)=\left(\delta\tilde{X}(t)+\tilde{X}(t)-\hat{X}(t)\right)^T\mathcal{Q}\left(\delta\tilde{X}(t)+\tilde{X}(t)-\hat{X}(t)\right)\\
&J_u\left(\delta\tilde{U}(t)+\tilde{U}(t)-\hat{U}(t)\right)=\left(\delta\tilde{U}(t)+\tilde{U}(t)-\hat{U}(t)\right)^T\mathcal{R}\left(\delta\tilde{U}(t)+\tilde{U}(t)-\hat{U}(t)\right)\\
&\delta\tilde{X}(t)=X(t)-\tilde{X}(t),\quad \delta\tilde{U}(t)=U(t)-\tilde{U}(t)
\end{aligned}
$$

and $\tilde{\mathcal{F}}_Z(t)$ is obtained as a $\mathcal{F}_Z(t)$ with the only difference that $\tilde{U}$ is used instead of $\hat{U}$.

MPC Application on Raspberry

After verifying the correct functioning of the MPC control in a simulation environment, a Real Time Model Predictive Control (RT-MPC) strategy is developed. For this purpose, the MPC algorithm is implemented on the Raspberry PI 3, through a code written in C ++.The optimization problem is converted into a quadratic programming problem (QP) and it is solved by the *quadprog* function. As can be seen from Figure 4.60 and Figure 4.59, the results obtained from the offline predictive control are satisfactory.
During these simulations the Real Time applicability is studied through a careful analysis of the calculation times of the quadratic programming problem. It should

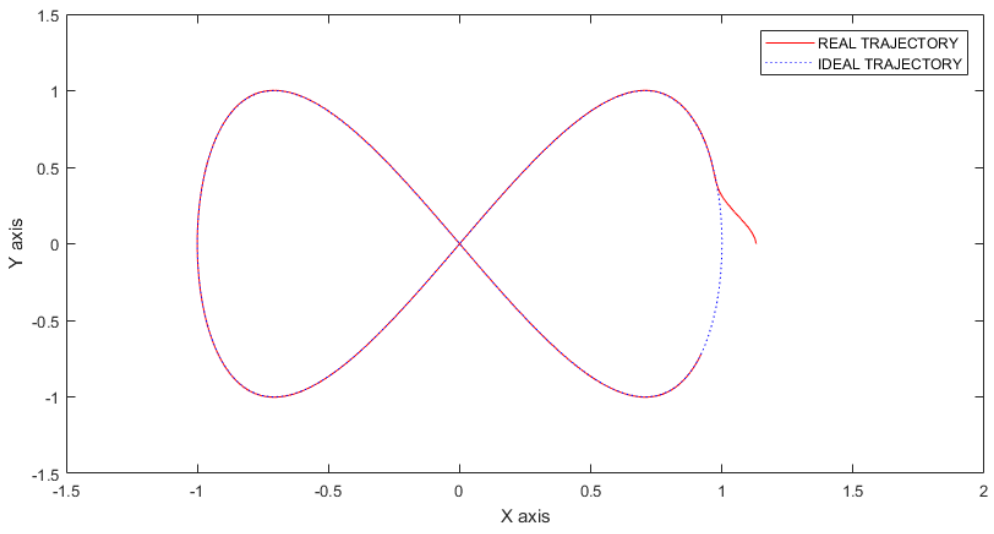


Figure 4.59: Simulation on Raspberry

be remembered that one of the disadvantages of predictive control is having to

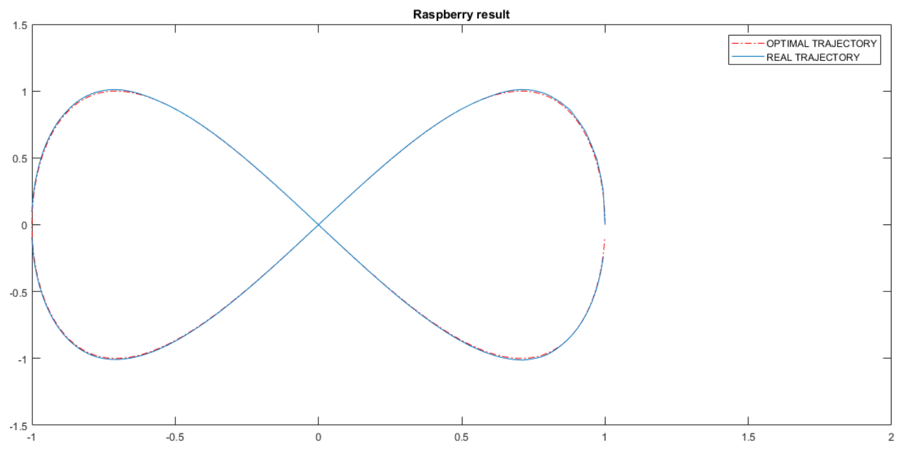


Figure 4.60: Simulation on Raspberry

work with a model that adequately describes the process but at the same time is sufficiently simple to allow real time control.

Despite the accuracy of the simulated trajectory, the results obtained from the application of the MPC control on the Raspberry are not the desirable ones.

The problem found is that using the ARX model 914 (i.e with $n_a = 9$ $n_b = 1$ and $n_k = 4$) the computational cost of the MPC algorithm is very high. In this way, the control loop cannot operate in real time (i.e. in a time less than the sampling time $t_s = 0.03$).

Thus, a new model of the system is identified with the same sampling time as before. During the identification process, the model orders are now set to $n_a = 4$ $n_b = 1$ and $n_k = 1$ in order to work with smaller matrices and then try to reduce the calculation times.

As appears from the validation process, the ARX 411 model accurately represents the collected data (see Figure 4.61).

Using this model now, the timing of the QP problem is reduced significantly. However, the shortest time obtained is equal to $t_{\text{quadprog execution}} = 0,08\ s$ which corresponds to $N = 5$. This calculation time is still too long compared to the sampling time.

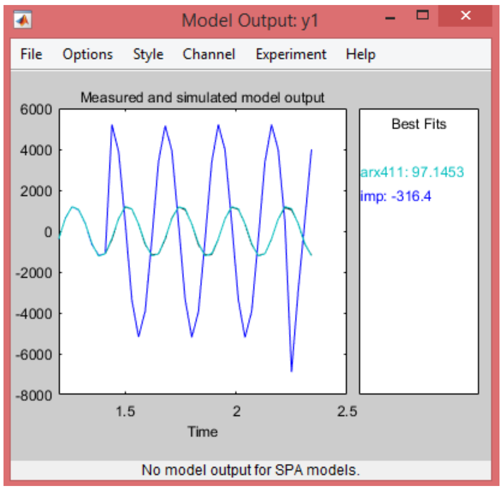


Figure 4.61: *ARX model validation*

Horizon	**ARX model 914**	**ARX model 411**
length	quadprog execution (sec)	
15	9	2
10	3	$0,45$
5	$0,5$	$0,08$

With these execution times of the MPC, the real coordinate values of the robot cannot be acquired and processed in time.
For this reason, in the next section the motor model and therefore the overall robot model are simplified in order to obtain a MPC suitable for the real time control of the mobile robot.

MPC Real-Time

In order to make the MPC real-time implementation possible, only the part of the robot model that concerns the kinematics is used in the MPC. However, in this way the MPC considers the two motor angular speeds (w_R, w_L) as inputs to the system. Note that the angular speed cannot be set directly, but only through the PWMs sent to the motors. Therefore, a relationship that links the PWM signals to the motor angular velocity must be found. For this purpose, a piecewise linear relationship is estimated between input and output of the motor robot. To achieve this, tests are carried out which consist in supplying both motors with the same and constant PWM signal, so as to measure how long the robot takes to cover a trajectory of known length and to estimate both the average linear speed of the vehicle v_{robot} and the angular velocity of each wheel.

$$w_R = v_{robot}/r \quad w_L = v_{robot}/r \tag{4.48}$$

In Figure 4.62 the identified relationship between the PWM signal and the robot angular speed is shown, which is used for the real time MPC. With this simplification,

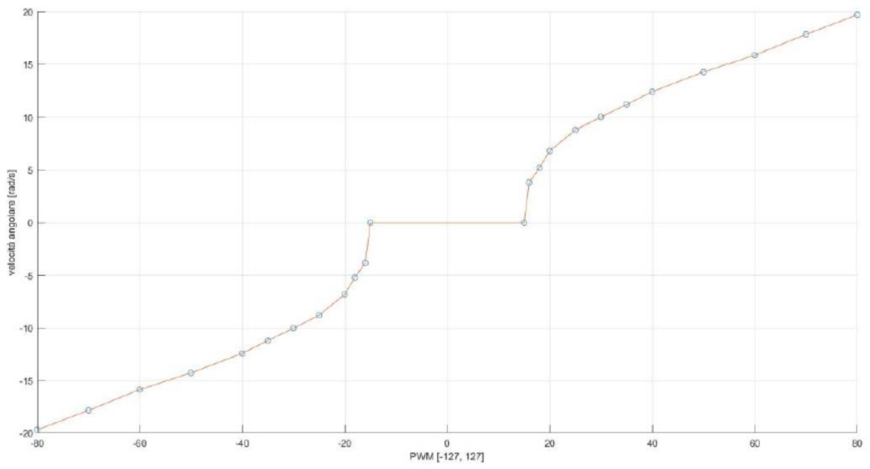


Figure 4.62: Piecewise linear motor model

a very fast MPC is obtained, which is suitable for the real time use. Indeed, Raspberry solves the MPC with $N = 10$ in only 0.01 s. However, this solution penalizes the quality of the control, as can be seen from Figure 4.63 where the red line is the reference trajectory, while the blue line is followed by the robot during the simulation performed in the testbed. The blu line is obtained by using the data collected by the infrared camera.

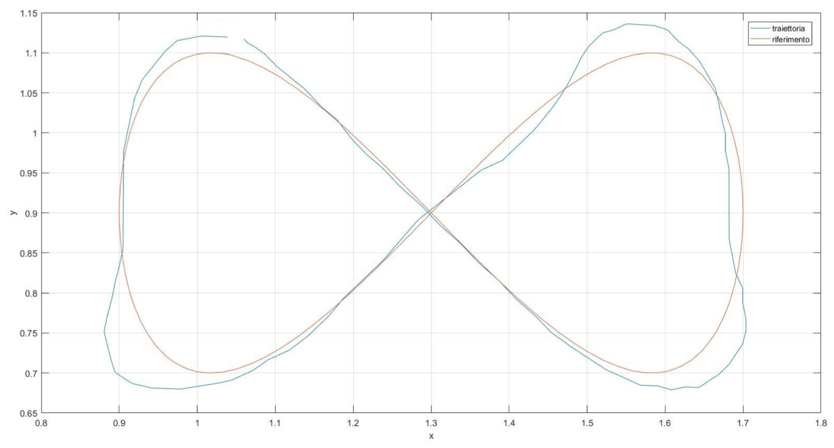


Figure 4.63: Comparison between the reference trajectory (red) and the one really followed by the robot (blue)

4.7 Results

In this section will be shown different operations performed by our testbed.

4.7.1 Path Following

Given a robot and a reference trajectory, in this case the Leminiscate, the problem to make the robot follow the trajectory.
The motion control problem is the *path following*, which is concerned with the design of control laws that force a vehicle to reach and follow a time parametrized reference. The desired trajectory does not need to be of a particular type and can be any sufficiently smooth bounded curve parameterized by time.
After the choice of the path, to complete the trajectory tracking it is necessary to define a motion law based on tasks to perform and optimal criteria to satisfy. In this case the mobile robot is required to proceed along the path at a constant angular speed w.
To deal with the path following first of all the path is defined in a continuous way

as follows:

$$\begin{cases} \hat{x}_1(t) = 2 \cdot a \cdot \cos(w \cdot t) \\ \hat{x}_2(t) = a \cdot \sin(2 \cdot w \cdot t) \\ \hat{x}_3(t) = tan^{-1}\Big(\dfrac{\hat{x}_2(t)}{\hat{x}_1(t)}\Big) \end{cases}$$

where $\hat{x}$ and $\hat{y}$ are the dependent variables and which indicate the coordinates (expressed in meters) of the robot on the reference system; t is the time indipendent variable (expressed in seconds); w is the angular velocity which is set constant along the entire path. Variable a is the distance between the central point and one of two focuses of the trajectory.
Once the parametric representation is defined, it's possible to calculate points describing the Lemniscata.
To this end, reference states are defined as:

$$\begin{cases} \dot{\hat{x}}_1(t) = \dfrac{d\hat{x}_1(t)}{dt} \\ \dot{\hat{x}}_2(t) = \dfrac{d\hat{x}_2(t)}{dt} \end{cases}$$

Regarding the derivative of $\hat{x}_3 = \hat{\theta}$ angle, formed with X axis, because of tangent function discontinuity a particular approach is necessary.
According to the literature, it is possible to calculate the derivative over the time, using partial derivatives with respect to $\dot{\hat{x}}$ and $\dot{\hat{y}}$:

$$\frac{d\hat{x}_3(t)}{d\dot{\hat{x}}_1(t)} = -\frac{\dot{\hat{x}}_2(t)}{(\dot{\hat{x}}_1^2(t) + \dot{\hat{x}}_2^2(t))}$$
$$\frac{d\hat{x}_3 t)}{d\dot{\hat{x}}_2(t)} = \frac{\dot{\hat{x}}_1(t)}{(\dot{\hat{x}}_1^2(t) + \dot{\hat{x}}_2^2(t))}$$

Based on these partial derivatives it is possible to calculate the derivative of $\hat{\theta}$ in relation to time as follows:

$$\dot{\hat{x}}_3(t) = \frac{d\hat{x}_3(t)}{dt} = \frac{d(tan^{-1}(\dot{\hat{x}}_2(t)/\dot{\hat{x}}_1(t)))}{dt} = \frac{d\hat{x}_3(t)}{d\dot{\hat{x}}_1(t)}\frac{d\dot{\hat{x}}_1(t)}{dt} + \frac{d\hat{x}_3(t)}{d\dot{\hat{x}}_2(t)}\frac{d\dot{\hat{x}}_2(t)}{dt}$$

In order to obtain the optimal trajectory, a Matlab code is implemented.
In this code, parameters related to sampling time, duration of the simulation and those specific to the system (i.e radius r of the wheels and the distance d between them) are evaluated.
Once the reference trajectory is defined, Equation 4.26 is solved as a function of w_L and w_R and the following inverse equations are obtained:

$$\begin{cases} w_\mathrm{R}(t) = \dfrac{\dot{\hat{x}}_1(t)}{r \cdot cos(\hat{x}_3(t))} + \dfrac{d}{2 \cdot r} \cdot \dot{\hat{x}}_3(t) \\ w_\mathrm{L}(t) = w_r(t) - \frac{d}{r} \cdot \dot{\hat{x}}_3(t) \end{cases} \tag{4.49}$$

The states, related to the points belonging to the Lemniscata, are replaced in these equations. Thus, the sequences of inputs necessary to follow the predefined path are obtained.
Then, the MPC Real Time controller is used in order to guarantee that the robot follows the reference trajectory. At the following link: `https://youtu.be/4I12eG2_WgM` it is possible to see the robot in action while following the Lemniscata. In particular, it is possible to note that the robot starts with a different angle than that of the reference trajectory. Therefore, after completing the operations of alignment with the reference trajectory, the robot follows it without any problems.

4.7.2 Path Following on Graph

Given a list of order to be transported, which are collected in a chronological way (FCFS list), see Table 4.4, and a graph $\mathcal{G}$ that abstracts all the available routes (see Figure 4.64), the robot has to satisfy transportation requests in order to minimize its travel path.

$\mathcal{D}$		
Number of order	Pickup Point	Delivery Point
1-st order	M_4	T_{OUT}
2-nd order	M_1	M_7

Table 4.4: FCFS list

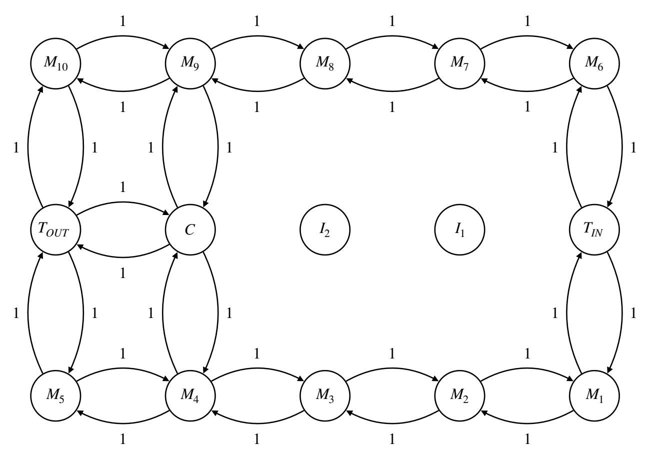


Figure 4.64: The environment $\mathcal{G}$

At the initial time, the robot is located at node 6. The transportation orders are listed chronologically : the first request is from node M_4 to T_{OUT}, while the second request is from node M_1 to M_7. By using Algorithm 3, it is possible to find the path that minimizes both waiting and transport time. According to Algorithm 3 the robot first satisfies the second call of $\mathcal{D}$ and then the first one. At the following link, `https://www.youtube.com/watch?v=6vqtz5xiQ8c` it is possible to see one of our robots in action. In particular, in this case the Pure Pursuit approach, previously explained, is used.

4.7.3 Normal Operation

As is previously described, in order to make the testbed more realistic, both the charging station and the pickup/delivery stations are created.
The first link shows `https://youtu.be/K5o2AonO_ZI` that first the robot follows the reference trajectory and subsequently reaches the charging station.
Instead, the link `https://youtu.be/OY-UUDoJXms` shows a classical robot operation, where the robot must pick an order from a machine (which is represented by a metal structure) and then it has to deliver the order in buffer out (wooden structure). In both video the MPC are used as lower controller in order to control the robot.

4.7.4 Multiple Robots

As explained before, in the UNIPV testbed it is possible to use use up to three mobile robots simultaneously.

Given list of orders $\mathcal{D}$, see Table 4.5, and a graph $\mathcal{G}$ as shown in Figure 4.65, the goal is to find a path for each available robot in order to satisfy with the shortest time all the requests of transport of $\mathcal{D}$. In the considered case, two robots (v_1, v_2) are available and their initial position are $M1$ and M_9. At the following link `https://youtu.be/DBmJCIEWNVk` it is possible to see how two robots works at the same time. In particular, by using Algorithm 4 the suboptimal assignments can

$\mathcal{D}$		
Number of order	Pickup Point	Delivery Point
1-st order	T_{IN}	C
2-nd order	I_2	M_2

Table 4.5: FCFS list

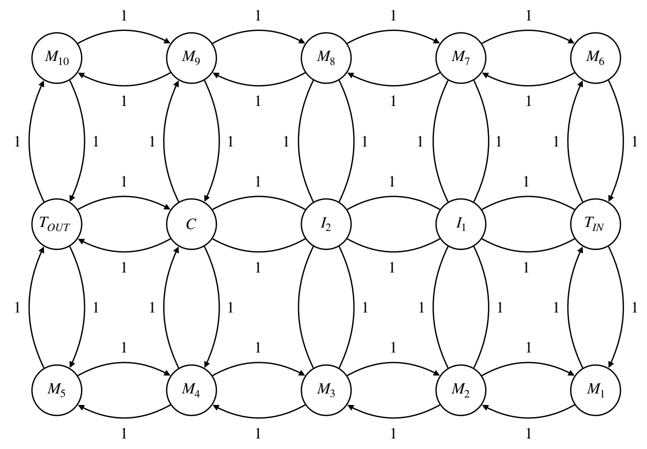


Figure 4.65: The environment $\mathcal{G}$

be found. Once for each robot both the MPS and the transport capacity C have been set, in our case MPS = 1 and C = 1, the suboptimal assignments are found as follows: robot v_1 satisfies the first request of transport while robot v_2 the second one.

4.8 Conclusion

In this chapter, the UNIPV test-bed realized under my co-supervision activity is described in detail. Furthermore, several different operations performed by the UNIPV test-bed are shown.

4.8.1 Future Development

As described in this chapter, in order to achieve a real-time MPC, the robot model is simplified by removing the ARXs, which describe the relationship between PWM signal and motor angular speed. For this reason, the results achieved by using the real-time MPC are not satisfactory in terms of path following, see Figure 4.63. This is mainly due to the use of a weak correlation between the PWM signal and the motor speed in the real-time MPC, see Figure 4.62. Indeed, it is more correct to describe this relation by using a dynamic model, such as ARX. Hence, possible future works are related to improve the real-time performance by using in the MPC the ARXs for describing the relationship between PWM and motor angular velocity. Note that the real-time trajectory control is a very high frequency problem, where calculations (i.e. solve the QP problem) are required to be performed in a time shorter than the sampling time. Therefore, in order to model the robot motors in the MPC via ARXs, a Raspberry with a computing power greater than the one available during this project (PI 3 model) is required. Alternatively, in order to reduce the computational time, an alternative QP formulation can be applied, where only $\delta U(t)$ is optimized instead of both $\delta X(t)$ and $\delta U(t)$.

Alternative QP Formulation

In this alternative quadratic problem formulation the prediction model is described by the following equation:

$$\delta X(t) = \hat{\mathcal{A}}(t)\delta x_{t|t} + \hat{\mathcal{B}}\delta U(t) \tag{4.50}$$

where

$$\hat{\mathcal{A}}(t) = \begin{bmatrix} I \\ \hat{A}(t) \\ \hat{A}(t+1)\hat{A}(t) \\ \vdots \\ \hat{A}(t+N-2)..\hat{A}(t) \end{bmatrix}$$

$$\hat{\mathcal{B}}(t) = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ \hat{B} & 0 & \dots & 0 & 0 \\ \hat{A}(t)\hat{B} & \hat{B} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \hat{A}(t+N-3)..\hat{A}(t)\hat{B} & \hat{A}(t+N-4)..\hat{A}(t)\hat{B} & \dots & \hat{B} & 0 \end{bmatrix}$$

The final optimization problem can be rewritten as:

$$\begin{aligned} \underset{\delta U(t)}{\text{minimize}} \quad & \delta X^T(t)\mathcal{Q}\delta X(t) + \delta U^T(t)\mathcal{R}\delta U(t) \\ \text{subject to} \quad & \varepsilon_u \delta U(t) \leq \mathcal{F}_u(t) \\ & \varepsilon_x \delta X(t) \leq \mathcal{F}_x(t) \end{aligned} \tag{4.51}$$

Considering the definition of $\delta X(t)$ it is possible to rewrite the problem as:

$$\begin{aligned} \underset{\delta U(t)}{\text{minimize}} \quad & \delta U^T(\hat{\mathcal{B}}(t)^T\mathcal{Q}\hat{\mathcal{B}}(t) + \mathcal{R})\delta U + 2\delta U^T\hat{\mathcal{B}}(t)^T\mathcal{Q}\hat{\mathcal{A}}(t)\delta x_{t|t} \\ \text{subject to} \quad & \varepsilon_u \delta U(t) \leq \mathcal{F}_u(t) \\ & \varepsilon_x \hat{\mathcal{B}}(t) \leq \mathcal{F}_x(t) - \varepsilon_x \hat{\mathcal{A}}(t)\delta x_{t|t} \end{aligned} \tag{4.52}$$

where the QP solver (i.e. *quadprog*) finds the optimum value of only the $\delta U(t)$. Thus, the computational times can be reduced.

Chapter 5

Battery Ageing-Aware Stochastic Management of Power Networks in Islanded Mode

Contents

5.1 Notation and Network Description 187

5.1.1 Conventional Generators 188

5.1.2 Renewable Generators 188

5.1.3 Loads 189

5.1.4 Energy Storage Systems 190

5.1.5 Costs 191

5.1.6 Network Constraints 192

5.2 Deterministic Optimization 194

5.3 Stochastic Optimization 195

5.3.1 Scenario Optimization with Certificates 195

5.3.2 Stochastic OPF Formulation 197

5.4 Closed Loop Solution 201

5.5 Case Study 202

5.5.1 Simulation Results 204

5.6 Conclusion 206

In this work a battery ageing-aware SOPF approach suitable for networks in islanded mode is presented where DR policies and ESSs are considered. In particular, the ESSs are assumed to be lithium-ion batteries. Differently from standard approaches, where the objective is to minimize power production costs and curtailments, in this chapter an ESS ageing model is considered [117] and the optimization problem is modified so as to take into account also batteries degradation.

SOPFs rely on the solution of constrained optimizations. By adopting a multiperiod SOPF, i.e. an optimization over a prediction horizon rather than over a single step, it is possible to well exploit the ESSs taking into account loads and renewable forecasts. On the other side, the use of a too long horizon makes the problem computationally too expensive and, due to the stochastic nature of the forecasts, not that meaningful. For this reason, a receding horizon approach is here adopted where at each time step a new optimization problem is solved given the newly available measurements and forecasts. Still, it should be noticed that the time required for solving a finite horizon optimization is not negligible. Therefore the assumption of a controller at time k which depends on measurements up to time k included is not reasonable. This is a problem which is often underestimated. To deal with this issue an approach is proposed where the controller at time k depends on measurements up to time $k-1$ and predictions from $k-1$ to k using a previously computed controller.

The contributions of this work can be summarized as follows:

- the use of a degradation model of the ESSs;
- the use of a closed-loop multiperiod SOPF which takes into account the computational burden of the underlying optimizations.
- the use of a stochastic approach where the uncertainty of loads and renewables is managed through stochastic controllers. In particular, differently from e.g. [85], these determine all together the redistribution of conventional power, ESSs usage and curtailments to optimally satisfy the power requirements.

Note that the management of networks in islanded mode requires accurate voltage/frequency control. In this work, it is assumed that the control is structured in two levels where our SOPF operates at the highest level while a voltage/frequency control at the lowest. The voltage/frequency control is out of the scope of this work and the interested reader can refer to e.g. [118, 119]. The chapter is organized as follows. In Section 5.1 the main network elements and equations together with the battery degradation model are described. A deterministic SDP-based OPF approach is recalled in Section 5.2 while the Stochastic OPF adopted in this work is presented in Section 5.3. Section 5.4 introduces the closed-loop approach which takes into account the computational burden of the optimizations. In Section 5.5 the proposed control strategy is applied to the IEEE 14 buses. Section 5.6 provides the conclusions.

5.1 Notation and Network Description

Consider a power network defined by a graph $\mathbb{G} = \{\mathcal{N}, \mathcal{E}\}$, with $\mathcal{N} = \{1, 2, \cdots, n_b\}$ the set of network buses and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ the set of electrical lines. Denote with $\mathcal{G} \subseteq \mathcal{N}$ the set of buses where generators are present and indicate with n_g its cardinality. The network has conventional and renewable generators. $\mathcal{C} \subseteq \mathcal{G}$ indicates the set of n_c buses where the conventional generators are installed while $\mathcal{R} \subseteq \mathcal{G}$ the set of n_r buses where renewable sources are present. The set of network loads is denoted with $\mathcal{D} \subseteq \mathcal{N}$ and has cardinality n_d. The network includes also n_s Energy Storage Systems (ESSs). The set of buses with ESSs is denoted with $\mathcal{S} \subseteq \mathcal{N}$. The admittance matrix of the network is indicated with $Y \in \mathbb{C}^{n_b \times n_b}$. Its elements are defined as

$$Y_{n,m} = \begin{bmatrix} y_{n,n} + \sum_{r \neq n} y_{n,r} & n = m \\ -y_{n,m} & n \neq m \end{bmatrix}$$

where $y_{n,n}$ is the admittance-to-ground and $y_{n,m}$ represents the line admittance of connected buses n and m. Index r is used to indicate all buses directly connected to bus n. For each bus $n \in \mathcal{N}$, the power balance equation is

$$S_n(k) = V_n(k)[I_n(k)]^*$$

where $[\cdot]^*$ is the complex conjugate operator, $V_n(k)$ the complex voltage at the n-th node and $I_n(k)$ the complex current injection at bus n. For each $(n,m) \in \mathcal{E}$, the real power flowing from bus n to bus m is defined as

$$P_{n,m}(k) = Re[V_n(k)[V_n(k) - V_m(k)]^* Y_{n,m}^*]$$

where $Re[\cdot]$ returns the real part of a complex number.

5.1.1 Conventional Generators

For each bus $n \in \mathcal{C}$, and each discrete time instant k, with $k \geq 0$, it is possible to define the complex power produced by conventional generators as

$$S_n^c(k) = P_n^c(k) + \mathbf{j}Q_n^c(k)$$

with $P_n^c(k)$ and $Q_n^c(k)$ the active and reactive generated power respectively. For nodes in $\mathcal{N} \setminus \mathcal{C}$, $P_n^c = Q_n^c = 0$.

5.1.2 Renewable Generators

For each bus $n \in \mathcal{R}$, and each discrete time instant $k \geq 0$

$$S_n^r(k) = P_n^r(k)$$

indicates the power injected into the network by renewable generators. Note that the reactive power production by renewables is assumed null. Since transmission or operational constraints may be violated when excessive wind and solar penetration levels occur, renewable energy *curtailment* is considered. In the following, $P_n^r(k)$ indicates the curtailed active renewable power, while $\bar{P}_n^r(k)$ the one which could have been delivered. For each bus $n \in \mathcal{R}$, at each time instant k, it holds that

$$P_n^r(k) \leq \bar{P}_n^r(k) \tag{5.1}$$

This work assumes that the injected power can be chosen to be any fraction of the deliverable one

$$P_n^r(k) = \beta_r \bar{P}_n^r(k), \quad 0 \leq \beta_r \leq 1.$$

For nodes in $\mathcal{N} \setminus \mathcal{R}$, $P_n^r = 0$.

5.1.3 Loads

During periods of high loads (as for example during hot summers), power cuts and blackouts may occur. The discomfort of such events can be mitigated through *load curtailment (or demand response) programs*. These provide reduced electrical rates in exchange for an agreement to curtail energy use at the request of the utility. This is of particular interest for customers who have the ability to reduce loads by turning off equipment or using alternative sources of energy (e.g. commercial and industrial buildings). In the following it is assumed that a subset $\mathcal{D}_c \subset \mathcal{D}$ of the loads can be curtailed. For each bus $n \in \mathcal{D}$ and each time instant $k \geq 0$, let

$$S_n^d(k) = P_n^d(k) + \mathbf{j}Q_n^d(k)$$

denote the effective complex power delivered to customers (with $P_n^d(k)$, $Q_n^d(k)$ the effective active and reactive delivered power respectively). For nodes in $\mathcal{N} \setminus \mathcal{D}$, $P_n^d = Q_n^d = 0$. Denoting with $\bar{P}_n^d(k)$, $\bar{Q}_n^d(k)$ the originally required powers, it holds that

$$\begin{aligned}
P_n^d(k) &\leq \bar{P}_n^d(k) \quad \forall n \in \mathcal{D}_c \\
Q_n^d(k) &\leq \bar{Q}_n^d(k) \quad \forall n \in \mathcal{D}_c \\
P_n^d(k) &= \bar{P}_n^d(k) \quad \forall n \in \mathcal{D} \setminus \mathcal{D}_c \\
Q_n^d(k) &= \bar{Q}_n^d(k) \quad \forall n \in \mathcal{D} \setminus \mathcal{D}_c
\end{aligned} \tag{5.2}$$

For the buses where curtailment can be applied, the delivered power can be any fraction of the required one

$$P_n^d(k) = \beta_d \bar{P}_n^d(k), \quad Q_n^d(k) = \beta_d \bar{Q}_n^d(k), \quad 0 \leq \beta_d \leq 1.$$

As discussed in the next sections, curtailment has a cost for the provider which is assumed to be a function of the curtailed amount of power.

5.1.4 Energy Storage Systems

Each node $n \in \mathcal{S}$ is equipped with an ESS whose dynamics can be described as follows

$$c_n(k+1) = c_n(k) + \rho^c \cdot r_n^c(k) - \tfrac{r_n^d(k)}{\rho_d} \tag{5.3}$$

where c_n indicates the energy storage level, $r_n^c(k)$ the charging rate (i.e. the active power injected in the network), $r_n^d(k)$ the discharging rate (i.e. the active power absorbed), ρ_c and ρ_d the charging and discharging efficiency, respectively. In the following, the reactive power involved in the charging and discharging of ESSs is considered negligible. For nodes in $\mathcal{N} \setminus \mathcal{S}$, $c_n = r_n^c = r_n^d = 0$. In order to obtain a detailed representation of ESSs, some degradation effects have to be considered. Among them, the loss of capacity during charging and discharging cycles is the most important. In this work, the *bucket model* is considered for the ageing effects, as described in [117]. In particular, the loss of capacity due to ageing is given by

$$c_n^l(k+1) = c_n^l(k) + \theta_1 \max_{i=0,\cdots k} \left(r_n^c(i),\, r_n^d(i) \right) + \theta_2 \left(r_n^c(k) + r_n^d(k) \right) \tag{5.4}$$

where $c_n^l(k)$ is the total loss capacity at a time k, while the initial condition is given by $c_n^l(0) = 0$. The parameters $\theta_1 = 2.15 \cdot 10^{-4}$ and $\theta_2 = 1.25 \cdot 10^{-5}$ are taken from [117]. The dynamics in (5.4) presents a nonlinearity due to the maximization operator. To deal with a convex optimization problem, the model is reformulated as follows

$$c_n^l(k) = c_n^l(k-1) + \theta_1 \mu_n(k) + \theta_2 \left(r_n^c(k-1) + r_n^d(k-1) \right) \tag{5.5}$$

where the variables $\mu_n(k)$ has to be penalized in the objective function of the optimization problem (see next section) in order to be as close as possible to the maximum values of r_n^d and r_n^c. In order to achieve this goal, the following inequalities are considered

$$\mu_n(k) \geq r_n^c(j), \quad \mu_n(k) \geq r_n^d(j), \quad j = 0, \cdots, k$$

Note that for nodes in $\mathcal{N} \setminus \mathcal{S}$, $\mu_n(k) = 0$.

5.1.5 Costs

The network management has to take into account several operating costs:

- the cost of production through conventional generators, which is modeled as

  $$C_n^c(k) = \sigma_n^c P_n^c(k), \quad \forall n \in \mathcal{C}$$

  where $\sigma_n^c h \geq 0$ indicates the unitary production cost, which varies depending on the power generation type. The cost of power production from renewables is assumed negligible;

- the cost related to the use of storage devices, given by

  $$C_n^s(k) = \sigma_n^{ch} r_n^c(k) + \sigma_n^d r_n^d(k) + c_n^l(k), \quad \forall n \in \mathcal{S}$$

  with $\sigma_n^{ch}, \sigma_n^d \geq 0$, while c_n^l is the cost related to battery ageing. Note that the use of σ_n^{ch} and σ_n^d allows, without introducing binary variables, to avoid situations where ESSs are charged and discharged simultaneously;

- the cost due to loads curtailment, which is proportional to the power cut

  $$C_n^{d,curt}(k) = \sigma_n^{d,curt}(\bar{P}_n^d(k) - P_n^d(k)), \quad \forall n \in \mathcal{D}_c$$

  with $\sigma_n^{d,curt} \geq 0$. In order to encourage the use of green energy sources and to exploit the presence of ESSs, also the curtailment of the renewables is penalized

$$C_n^{r,curt}(k) = \sigma_n^{r,curt}(\bar{P}_n^r(k) - P_n^r(k)), \quad \forall n \in \mathcal{R}$$

with $\sigma_n^{r,curt} \geq 0$.

5.1.6 Network Constraints

The proper functioning of a power network depends on several constraints. Consider $W(k) = V(k)V(k)^*$. First of all, the following balance equations need to hold for each bus $n \in \mathcal{N}$

$$\begin{aligned} &P_n^c(k) + P_n^r(k) - P_n^d(k) - r_n^c(k) + r_n^d(k) \\ &= \sum_{m \in \mathcal{N}_n^{\mathcal{E}}} Re[(W_{n,n}(k) - W_{n,m}(k))Y_{n,m}^*] \end{aligned} \tag{5.6}$$

$$Q_n^c(k) + Q_n^r(k) - Q_n^d(k) = \sum_{m \in \mathcal{N}_n^{\mathcal{E}}} Im[(W_{n,n}(k) - W_{n,m}(k))Y_{n,m}^*]$$

where $\mathcal{N}_n^{\mathcal{E}} = \{m : (n,m) \in \mathcal{E}\}$ indicates the set of buses that are directly reachable from bus n. Furthermore, the network is subject to a set of inequality constraints, such as:

- the power production constraints on conventional generators, given by

$$\begin{aligned} P_n^{c,min} &\leq P_n^c(k) \leq P_n^{c,max} \\ Q_n^{c,min} &\leq Q_n^c(k) \leq Q_n^{c,max} \end{aligned} \tag{5.7}$$

  for each $n \in \mathcal{C}$ and each time $k \geq 0$

- the ramping rate constraints on conventional generators, which bind the power variation to a time instant

$$-\Delta P_n^{low} \leq P_n^c(k+1) - P_n^c(k) \leq \Delta P_n^{up}, \quad \forall n \in \mathcal{C} \tag{5.8}$$

  with $\Delta P_n^{low}, \Delta P_n^{up} \geq 0$.

- the voltage constraints, which require that

$$(V_n^{min})^2 \leq W_{n,n}(k) \leq (V_n^{max})^2 \quad \forall n \in \mathcal{N}$$

$$W_{n,n}(k) + W_{m,m}(k) - W_{n,m}(k) - W_{m,n}(k) \quad \leq (\Delta V_{n,m}^{max})^2 \qquad \forall (n,m) \in \mathcal{E} \tag{5.9}$$

- the ESSs constraints due to the maximum capacity (which decreases with ageing) and the maximum ramping rates, which limit the speed of charging and discharging

$$\begin{aligned} 0 \leq c_n(k) + c_n^l(k) \leq c_n^{max} \\ 0 \leq r_n^c(k) \leq \Delta r_n^{c,max} \\ 0 \leq r_n^d(k) \leq \Delta r_n^{d,max} \end{aligned} \tag{5.10}$$

- the constraints which ensure the complete equivalence between the standard formulation in $V(k)$ and the one here adopted in $W(k)$

$$W(k) \succeq 0 \tag{5.11}$$

$$rank(W(k)) = 1 \tag{5.12}$$

Note that, the formulation above is non-convex due to (5.12). A semidefinite relaxation can be obtained by dropping the rank constraint. By doing so, there is no guarantee that the solution of the relaxed problem will be feasible and optimal for the original one. As discussed in [81], in case of weakly-cyclic networks with cycles of size 3, a rank 1 near-optimal solution can be obtained by adding to the cost function a penalization term γ related to the reactive power of the generators.

5.2 Deterministic Optimization

The traditional OPF problem consists in an optimal choice of power production and voltage of generators. In this work, the choice of curtailment level of renewables and loads is addressed, as well as the exploitation of ESSs. In particular, the choice of the above variables is considered optimal since they minimize the summation of the costs listed in Section 5.1.5 over a prediction horizon N_h, while fulfilling the network constraints in Section 5.1.6. The conventional active power production at all nodes at time k is denoted with $P^c(k) = \left[P_1^c(k), \cdots, P_{n_b}^c(k)\right]$. Similarly,it is possible to define $P^r(k)$, $P^d(k)$, $Q^c(k)$, $Q^r(k)$, $Q^d(k)$, $c(k)$, $r^c(k), r^d(k)$. With $\mathbf{P}^c = [P^c(k_0), \cdots, P^c(k_0+N_h)]$ the overall active power production over the horizon N_h is indicated, with k_0 the initial time instant. Similarly, one has $\mathbf{P}^r$, $\mathbf{P}^d$, $\mathbf{Q}^c$, $\mathbf{Q}^r$, $\mathbf{Q}^d$, $\mathbf{c}$. Furthermore, $\mathbf{r}^c = [r^c(k_0), \cdots, r^c(k_0+N_h-1)]$, $\mathbf{r}^d = \left[r^d(k_0), \cdots, r^d(k_0+N_h-1)\right]$, $\mathbf{c}^{loss} = \left[c^l(k_0), \cdots, \ c^l(k_0+N_h-1)\right]$, $\mu = [\mu(k_0), \cdots, \mu(k_0+N_h-1)]$ are considered. Moreover, $\mathbf{W^u} = [W^u(k_0), \cdots, W^u(k_0+N_h)]$, where each $W^u(k)$, $k = k_0, \cdots, k_0+N_h$ is a diagonal matrix with diagonal elements $W^u_{n,n}(k) = |V_n|^2 \ \forall n \in \mathcal{G}$ and $W^u_{n,n}(k) = 0 \ \forall n \in \mathcal{N} \setminus \mathcal{G}$. Finally, $\mathbf{W^x} = [W^x(k_0), \cdots, W^x(k_0+N_h)]$ is defined, with $W^x(k) = W(k) - W^u(k)$. As discussed in [120], the variables of an optimal power flow problem are divided into control ($\mathbf{u}$) and state ($\mathbf{x}$) variables. The former are variables which can be chosen by the network operator while the latter depend on the control variables and network constraints. According to this formulation,

$$\begin{aligned} \mathbf{u} &= \left[\mathbf{P}^c, \mathbf{P}^r, \mathbf{P}^d, \mathbf{r}^c, \mathbf{r}^d, \mathbf{W^u}\right] \\ \mathbf{x} &= \left[\mathbf{Q}^c, \mathbf{Q}^r, \mathbf{Q}^d, \mathbf{W^x}, \mathbf{c}, \mathbf{c}^l, \mu\right] \end{aligned} \tag{5.13}$$

The distinction between $\mathbf{W^x}$ and $\mathbf{W^u}$ (and their use instead of $\mathbf{W}$) is necessary since only the voltage at the generating nodes is controllable. In order for $\mathbf{u}$ and $\mathbf{x}$ to be vectors, $\mathbf{W^u}$ and $\mathbf{W^x}$ in (5.13) are reshaped into vectors. Finally, the optimization problem can be formulated as follows:

$$\begin{aligned}
\min_{\mathbf{u},\mathbf{x}} \quad & \textstyle\sum_{k=1}^{N_h} \sum_{n=1}^{n_b} C_n^c(k) + C_n^s(k) + C_n^{d,curt}(k) + C_n^{r,curt}(k) \\
& +\gamma(Q_n^c(k) + Q_n^r(k)) \\
\text{subj. to} \quad & \text{ESSs and balance equations (5.3), (5.5), (5.6)} \\
& \text{Constraints (5.1), (5.2), (5.7), (5.8), (5.9), (5.10), (5.11)}
\end{aligned}$$

The optimization above allows to solve a deterministic optimal power flow, in which the load demands and production of renewables need to be known precisely. Unfortunately, especially for networks in islanded mode, considering the increasing penetration of renewables and the inaccurate predictability of renewables and loads, a deterministic OPF may lead to lines overloading and the exceeding of operational limits.

5.3 Stochastic Optimization

Before introducing the proposed stochastic OPF, the *scenario optimization with certificates* [88] problem is recalled.

5.3.1 Scenario Optimization with Certificates

Consider an optimization problem where the variables can be distinguished in design variables $\mathbf{u}$ and certificate variables $\mathbf{x}$ and where $\boldsymbol{\delta}$ represents the problem uncertainty. Let the constraints be represented by the function $f(\mathbf{u}, \mathbf{x}, \boldsymbol{\delta})$ which is assumed jointly convex in $\mathbf{u}$ and $\mathbf{x}$, for any given $\boldsymbol{\delta} \in \boldsymbol{\Delta}$. Consider the *robust optimization problem with certificates*

$$\begin{aligned}
& \min_{\mathbf{u}} c^T \mathbf{u} && (5.14) \\
& \text{subj. to } \forall \boldsymbol{\delta} \in \boldsymbol{\Delta},\ \exists \mathbf{x} = \mathbf{x}(\boldsymbol{\delta}) \text{ s.t. } f(\mathbf{u}, \mathbf{x}, \boldsymbol{\delta}) \geq 0
\end{aligned}$$

The problem above is very hard to solve since, as the uncertainty varies in its admissible domain, a possibly infinite number of constraints can be generated. An approximation of (5.14) can be obtained by solving a *scenario optimization with certificates*

$$\begin{aligned} \min_{\mathbf{u},\mathbf{x}_1,\cdots,\mathbf{x}_{N_\Delta}} \quad & c^T\mathbf{u} \\ \text{subj. to} \quad & f(\mathbf{u},\mathbf{x}_i,\boldsymbol{\delta}_i) \geq 0, \\ & i = 1,\cdots,N_\Delta \end{aligned} \tag{5.15}$$

which is based on the extraction of a number N_Δ of i.i.d. (independent and identically distributed) random samples of the uncertainty $\boldsymbol{\delta}_1,\cdots,\boldsymbol{\delta}_{N_\Delta}$ and the creation of a certificate variable $\mathbf{x}_i$ for each $i \in \{1,\cdots N_\Delta\}$. The following theorem can now be recalled [88].

Theorem 5.3.1. Assume that, for any uncertainty sample extraction, problem (5.15) is feasible and has a unique optimal solution $\mathbf{u}_{SwC}$. Then, given an accuracy level $\varepsilon \in (0,1)$ and a confidence level $\beta \in (0,1)$, if N_Δ is chosen as

$$N_\Delta \geq \frac{e}{\varepsilon(e-1)}\left[ln\left(\frac{1}{\beta}\right) + n_u - 1\right] \tag{5.16}$$

with n_u the dimension of $\mathbf{u}$ and e the Euler number, then, with probability at least 1-β

$$Pr\left\{\exists\boldsymbol{\delta} \in \boldsymbol{\Delta} \mid \nexists\mathbf{x} \text{ s.t.} f\left(\mathbf{u}_{SwC},\mathbf{x},\boldsymbol{\delta}\right) \geq 0\right\} \leq \varepsilon \tag{5.17}$$

i.e., if N_Δ is chosen according to (5.16), the probability of $\mathbf{u}_{SwC}$ violating the constraints of the robust problem in (5.14) is less than a predefined level ϵ (with confidence level (1-β)). Note that, as discussed in [121] a less conservative bound on the number of samples to be extracted can be obtained by finding the minimum N_Δ satisfying

$$\sum_{i=0}^{n_u-1}\binom{N_\Delta}{i}\varepsilon^i(1-\varepsilon)^{N_\Delta-i} \leq \beta \tag{5.18}$$

5.3.2 Stochastic OPF Formulation

In the following, a scenario-based stochastic OPF is formulated. The described approach extends what proposed in [85] in case of multiple time instants, the presence of ESSs, and the possibility of curtailment of loads and renewables. Let

$$\delta^{d,[i]}(k) = \left[\delta_1^{d,[i]}(k), \cdots, \delta_{n_b}^{d,[i]}(k)\right] \in \Delta_{\mathcal{N}}^{d}$$

$$\delta^{r,[i]}(k) = \left[\delta_1^{r,[i]}(k), \cdots, \delta_{n_b}^{r,[i]}(k)\right] \in \Delta_{\mathcal{N}}^{r}$$

indicate respectively the vector of complex load and renewable uncertainties at time k for any extracted sample i. Note that $\delta_n^{d,[i]}(k) = 0, \forall n \notin \mathcal{D}$ and $\delta_n^{r,[i]}(k) = 0, \forall n \notin \mathcal{R}$. At each time step, for each $n \in \mathcal{N}$, the corresponding uncertain load power demand and the uncertain renewable power production before the application of any curtailment are

$$\begin{aligned}
\bar{P}_n^d(\delta_n^{d,[i]}, k) &= \bar{P}_n^d(k) + Re\left\{\delta_n^{d,[i]}(k)\right\}, \\
\bar{Q}_n^d(\delta_n^{d,[i]}, k) &= \bar{Q}_n^d(k) + Im\left\{\delta_n^{d,[i]}(k)\right\}, \\
\bar{P}_n^r(\delta_n^{r,[i]}, k) &= \bar{P}_n^r(k) + Re\left\{\delta_n^{r,[i]}(k)\right\}, \\
\bar{Q}_n^r(\delta_n^{r,[i]}, k) &= \bar{Q}_n^r(k) + Im\left\{\delta_n^{r,[i]}(k)\right\},
\end{aligned}$$

where $\bar{P}_n^d(k)$, $\bar{P}_n^r(k)$ and $\bar{Q}_n^d(k)$, $\bar{Q}_n^r(k)$ here indicate the nominal (forecasted) active and reactive powers. In order for the network constraints to be satisfied for the possible uncertainties, especially for networks in islanded mode, uncertainty dependent curtailments are necessary. The same is required for the power produced by conventional generators and for the ESSs usage. To this end, a solution similar to the one proposed in [69], [85] is adopted. Introducing the vectors

$$\begin{aligned}
&\delta^{[i]}(k) = \left[\delta^{d,[i]}(k)\ \delta^{r,[i]}(k)\right], \\
&s = [1_{n_b}\ \ -1_{n_b}], \quad \alpha^c = \left[\alpha_1^c, \cdots, \alpha_{n_b}^c\right], \\
&\alpha^d = \left[\alpha_1^d, \cdots, \alpha_{n_b}^d\right], \quad \alpha^r = \left[\alpha_1^r, \cdots, \alpha_{n_b}^r\right], \\
&\alpha^{ESS,c} = \left[\alpha_1^{ESS,c}, \cdots, \alpha_{n_s}^{ESS,c}\right], \ \alpha^{ESS,d} = \left[\alpha_1^{ESS,d}, \cdots, \alpha_{n_s}^{ESS,d}\right],
\end{aligned}$$

with 1_{n_b} denoting an 1-by-n_b vector of ones, $\alpha_n^c = 0$, $\forall n \in \mathcal{N} \setminus \mathcal{C}$, $\alpha_n^d = 1$, $\forall n \in \mathcal{D}_c$, $\alpha_n^d = 0$, $\forall n \in \mathcal{N} \setminus \mathcal{D}$, $\alpha_n^r = 0$, $\forall n \in \mathcal{N} \setminus \mathcal{R}$ and $\alpha_n^{ESS,c} = \alpha_n^{ESS,d} = 0$, $\forall n \in \mathcal{N} \setminus \mathcal{S}$. Define

$$\begin{aligned}
P_n^c(k, \delta^{[i]}) &= P_n^c(k) + \alpha_n^c Re\left\{\delta^{[i]}(k)\right\} s^T, && (5.19)\\
P_n^d(k, \delta^{[i]}) &= P_n^d(k) + \alpha_n^d Re\{\delta^{[i]}(k)\} s^T, && (5.20)\\
P_n^r(k, \delta^{[i]}) &= P_n^r(k) + \alpha_n^r Re\left\{\delta^{[i]}(k)\right\} s^T, && (5.21)\\
r_n^c(k, \delta^{[i]}) &= r_n^c(k) + \alpha_n^{ESS,c} Re\left\{\delta^{[i]}(k)\right\} s^T, && (5.22)\\
r_n^d(k, \delta^{[i]}) &= r_n^d(k) + \alpha_n^{ESS,d} Re\left\{\delta^{[i]}(k)\right\} s^T, && (5.23)
\end{aligned}$$

the *uncertainty dependent* generation of conventional power, curtailed loads, curtailed renewables, charging and discharging rates. The role of the different alphas is to cope with the uncertainties by *i)* increasing the production of conventional generators, *ii)* absorbing or releasing more power from ESSs, *iii)* suitably curtailing loads and renewables. It is worth to mention that, while in [85] the deployment vector α is used only to redistribute the net difference between the uncertainty in the loads and the uncertainty in the renewables among the several conventional generators, the approach here presented is more complex. Indeed, due to the possible curtailments of loads and renewables, only a part of such net difference should be taken into account. Moreover, the presence of storage devices allows the redistribution not only on conventional generators but also on ESSs.

Remarks 5.3.2. Note that different alphas could be used for each $k = k_0, \cdots, k_0+N_h$. However, in this chapter, the same vector is used over the entire horizon, so to reduce the overall computational burden.

Due to the uncertainties, it is necessary to re-write the balance equations $\forall n \in \mathcal{N}$

$$\begin{aligned}
&P_n^c(k,\delta^{[i]}) - P_n^d(k,\delta^{[i]}) + r_n^c(k,\delta^{[i]}) - r_n^d(k,\delta^{[i]}) + P_n^r(k,\delta^{[i]}) = \\
&\sum_{m\in\mathcal{N}_n^{\mathcal{E}}} Re\left[(W_{n,n}(k,\delta^{[i]}) - W_{n,m}(k,\delta^{[i]}))\cdot Y_{n,m}^*\right] \\
&Q_n^c(k,\delta^{[i]}) + Q_n^r(k,\delta^{[i]}) + Q_n^d(k,\delta^{[i]}) = \\
&\sum_{m\in\mathcal{N}_n^{\mathcal{E}}} Im\left[(W_{n,n}(k,\delta^{[i]}) - W_{n,m}(k,\delta^{[i]}))\cdot Y_{n,m}^*\right]
\end{aligned} \tag{5.24}$$

and the power constraints $\forall n \in \mathcal{N}$

$$\begin{aligned}
P_{n_{min}}^c \leq P_n^c(k,\delta^{[i]}) &\leq P_{n_{max}}^c \\
Q_{n_{min}}^c \leq Q_n^c(k,\delta^{[i]}) &\leq Q_{n_{max}}^c \\
-\Delta P_n^{low} \leq P_n^c(k+1,\delta^{[i]}) - P_n^c(k,\delta^{[i]}) &\leq \Delta P_n^{up}
\end{aligned} \tag{5.25}$$

Moreover, the ESSs dynamics is updated as follows

$$\begin{aligned}
c_n^l(k,\delta^{[i]}) =& c_n^l(k-1,\delta^{[i]}) + \theta_1\mu_n(k,\delta^{[i]}) + \\
&+ \theta_2\left(r_n^c(k-1,\delta^{[i]}) + r_n^d(k-1,\delta^{[i]})\right)
\end{aligned} \tag{5.26}$$

while the variables $\mu_n(k,\delta^{[i]})$ are subject to

$$\begin{aligned}
\mu_n(k,\delta^{[i]}) \geq r_n^c(j,\delta^{[i]}), \quad \mu_n(k,\delta^{[i]}) \geq r_n^d(j,\delta^{[i]}) \\
j = 0,\cdots,k
\end{aligned}$$

and their constraints become

$$\begin{aligned}
0 \leq c_n(k,\delta^{[i]}) + c_n^l(k,\delta^{[i]}) &\leq c_n^{max} \\
0 \leq r_n^c(k,\delta^{[i]}) &\leq \Delta r_n^{c,max} \\
0 \leq r_n^d(k,\delta^{[i]}) &\leq \Delta r_n^{d,max}
\end{aligned} \tag{5.27}$$

The following constraints are also required

$$\begin{aligned} 0 \leq P_n^d(k,\delta^{[i]}) &\leq \bar{P}_n^d(k,\delta^{[i]}) \quad \forall n \in \mathcal{D}_c \\ Q_n^d(k,\delta^{[i]}) &\leq \bar{Q}_n^d(k,\delta^{[i]}) \quad \forall n \in \mathcal{D}_c \\ 0 \leq P_n^d(k,\delta^{[i]}) &= \bar{P}_n^d(k,\delta^{[i]}) \quad \forall n \in \mathcal{D} \setminus \mathcal{D}_c \\ Q_n^d(k,\delta^{[i]}) &= \bar{Q}_n^d(k,\delta^{[i]}) \quad \forall n \in \mathcal{D} \setminus \mathcal{D}_c \\ 0 \leq P_n^r(k,\delta^{[i]}) &\leq \bar{P}_n^r(k,\delta^{[i]}) \quad \forall n \in \mathcal{R} \\ Q_n^r(k,\delta^{[i]}) &\leq \bar{Q}_n^r(k,\delta^{[i]}) \quad \forall n \in \mathcal{R} \end{aligned} \tag{5.28}$$

Note that these are required to guarantee that the uncertainty dependent curtailed loads and renewables do not get negative and do not exceed the uncertain loads and renewables respectively.
Also the formal division between control and state variables needs to be updated as follows

$$\begin{aligned} &\mathbf{u} = \left[\mathbf{P}^c, \mathbf{P}^r, \mathbf{P}^d, \mathbf{r}^c, \mathbf{r}^d, \mathbf{W}^u, \boldsymbol{\alpha}^c, \boldsymbol{\alpha}^r, \boldsymbol{\alpha}^d, \boldsymbol{\alpha}^{ESS,c}, \boldsymbol{\alpha}^{ESS,d}\right] \\ &\mathbf{x}(\boldsymbol{\delta}^{[i]}) = \left[\mathbf{Q}^c(\boldsymbol{\delta}^{[i]}), \mathbf{Q}^r(\boldsymbol{\delta}^{[i]}), \mathbf{Q}^d(\boldsymbol{\delta}^{[i]}), \mathbf{W}^x(\boldsymbol{\delta}^{[i]}), \mathbf{c}(\boldsymbol{\delta}^{[i]}), \mathbf{c}^l(\boldsymbol{\delta}^{[i]}), \mu(\boldsymbol{\delta}^{[i]})\right] \end{aligned} \tag{5.29}$$

where $\mathbf{W}^x = \mathbf{W}(\boldsymbol{\delta}^{[i]}) - \mathbf{W}^u$. Note that the uncertain dependent generation of conventional power $\mathbf{P}^c(\delta^{[i]})$ can be obtained from $\mathbf{P}^c$ and $\alpha^{\mathbf{c}}$ by applying (5.19) (the same holds for (5.20)-(5.23)). It is possible now formulate the stochastic optimal power flow problem using a scenario optimization with certificates where $\mathbf{u}$ is kept fixed while a different $\mathbf{x}(\boldsymbol{\delta}^{[i]})$ is used for each different uncertain sample $\boldsymbol{\delta}^{[i]}$

$$\min_{\mathbf{u},\mathbf{x}(\boldsymbol{\delta}^{[1]}),\cdots\mathbf{x}(\boldsymbol{\delta}^{[N_\Delta]})} \quad \phi \tag{5.30}$$

that is subject to

$$\begin{aligned} &\textstyle\sum_{k=1}^{N_h}(\sum_{n=1}^{n_b} C_n^c(k,\delta^{[i]}) + C_n^{d,curt}(k,\delta^{[i]}) + C_n^{r,curt}(k,\delta^{[i]}) + \\ &+ C_n^s(k,\delta^{[i]}) + \gamma(Q_n^c(k,\delta^{[i]}) + Q_n^r(k,\delta^{[i]})) \leq \phi \\ &\text{ESSs and balance equations (5.24), (5.26), (5.27)} \\ &\text{Constraints (5.19), (5.20), (5.21), (5.22), (5.23), (5.25), (5.28)} \\ &W(k,\delta^{[i]}) \succeq 0 \quad i = 1, ..., N_\Delta \end{aligned} \tag{5.31}$$

The terms $\gamma(Q_n^c(k)+Q_n^r(k))+\gamma_l\sum_{(l,m)\in L^{prob}} L_{lm}(k,\delta^{[i]})$ are necessary for recovering a rank 1 near optimal solution when an SDP relaxation is used. Note that, in the problem above, the objective function is the maximum among the costs obtained for the different $\delta^{[i]}$, $i=1,\cdots,N_\Delta$.

5.4 Closed Loop Solution

The solution of the stochastic OPF problem provides values for the control variables $\mathbf{u}$ over the horizon $k\cdots,k+N_h$. However, rather than using the full sequence, according to the receding horizon paradigm, only the first input is applied. Then, at time $k+1$, measurements are collected and, if possible, the forecasts of power demands and renewables production updated, thus allowing to cope with possible unexpected variations or better weather predictions. Note that the complexity of a scenario-based optimization depends on the number of control variables and grows significantly with the length of the prediction horizon. In presence of storage devices and curtailments, the use of a long N_h results in better performance (reduced curtailment of loads and renewables, wiser use of ESSs, reduced use of conventional generators). On the other side, the time required by the optimization needs to be compatible with the sampling time used for controlling the grid. In any case, it is unrealistic to assume that the time required to solve the optimization is negligible. For this reason, at time k, an optimization where $u(k)$ is fixed (given by the previous optimization) is solved and only the variables on the window $k+1,\cdots,k+N_h$ are optimized. In practice the procedure can be summarized as follows:

1. at time k, the uncertainty $\delta(k)$ is measured. The uncertainty dependent generation of conventional power, curtailed loads, curtailed renewables, charging and discharging rates is computed according to $\delta(k)$ and the $u(k)$ which was previously computed at time $k-1$.

2. In case $u(k)$ satisfies all the OPF constraints at time k, proceed with the next step. Otherwise, further actions may be required;

3. At time k, predict the states of the grid (i.e. the SOC of the ESSs) for time $k+1$,

using $u(k)$ and solve the stochastic OPF over the window $k+1, \cdots, k+N_h$;

4. Increase counter $k = k + 1$;

5. Go back to 1) .

5.5 Case Study

In this section, the proposed approach is applied on a modified version of IEEE 14 bus which is considered in islanded mode. In particular, conventional generators are considered at buses 2 and 6, both the renewable resource and the ESS at bus 8. The curtailable loads are placed at buses 3 and 9, while the not curtailable ones are situated at buses 4, 5 and 10. The production costs for the conventional generators

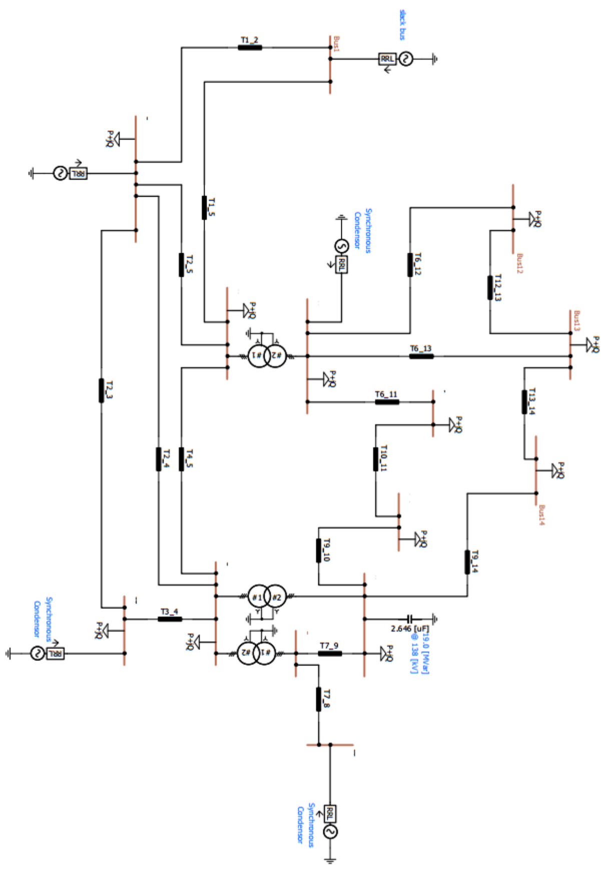


Figure 5.1: IEEE 14 bus

are chosen as in [122]:

$$\sigma_2^c = 2710\frac{\$}{p.u\frac{1}{4}h} \quad \sigma_6^c = 1677\frac{\$}{p.u\frac{1}{4}h} \tag{5.32}$$

The cost of load curtailments is chosen in accordance with [91] and is equal to 0.15\$ per kWh detached, which, transformed in p.u., becomes $\sigma_n^{curt} = 15000\,\frac{\$}{p.u}$. Since this work aims to maximize the use of renewable sources, the curtailment of renewables is weighted with $\sigma_n^r = 10\,\frac{\$}{p.u}$. On the other side, the charging and discharging of the batteries are penalized with $\sigma_n^{ch} = \sigma_n^d = 1$. The upper and lower limits for the power production of the conventional generators are:

$$\begin{aligned} &P_2^{c,min} = 0.1p.u. \quad P_6^{c,min} = 0.1\,p.u. \\ &P_2^{c,max} = 1p.u. \quad\;\; P_6^{c,max} = 2.3\,p.u. \end{aligned} \tag{5.33}$$

while the ramping rate constraints are:

$$\begin{aligned} &\Delta P_2^{low} = 0.8\,p.u. \quad \Delta P_6^{low} = 1\,p.u. \\ &\Delta P_2^{up} = 0.15p.u. \quad \Delta P_6^{up} = 0.75\,p.u. \end{aligned} \tag{5.34}$$

The time varying profile of a real renewable source is taken from [123], while the load profiles are obtained from real data of active power demand. All the profiles have length of 24 hours and are sampled every 15 minutes. The capacity of the battery is set equal to $c_n^{max}(k) = 2.6\,p.u.$ and the $\Delta r_n^{c,max} = \Delta r_n^{d,max} = 0.1\ p.u.$ (these values are the same as in [122]). As far as the parameters of the SwC, a prediction horizon $N_h = 5$ is chosen where each step has the duration of 15 minutes are concerned. Moreover, the values of confidence and accuracy level are chosen respectively as $\beta_{SwC} = 1e^{-6}$ and $\epsilon = 0.1$. The values of renewable and loads uncertainty are extracted from a leptokurtic distribution in accordance with [124]. In particular, the Pearson system is adopted in which the values of the parameters are chosen as $\sigma = 0.2\cdot$ (predicted values), γ= 0, and $k = 3.5$. This particular distribution helps to take into account large forecast errors better than the Gaussian one. The number of control variables results equal to $\mathbf{u} = 103$. By applying (5.18), the minimum number of samples needed to guarantee the desired probabilities is

$N_\Delta = 1558$. The resulting stochastic SDP problem is solved using MATLAB. In particular, YALMILP [125] and the commercial solver MOSEK are used. The PC used for simulations is equipped with an i7 @3.5 Ghz 64bit CPU sytem, 16 Gbytes of RAM and O.S Windows 10.

5.5.1 Simulation Results

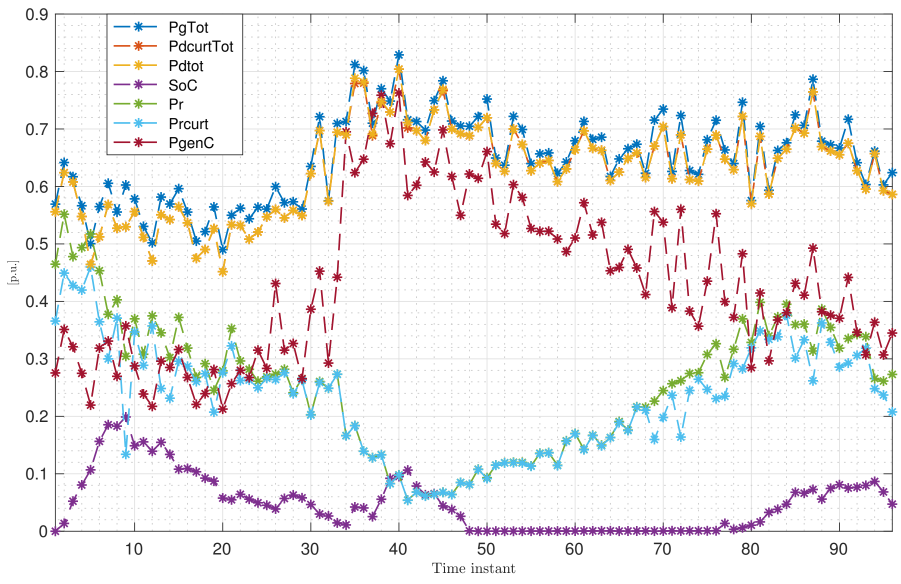


Figure 5.2: Profiles with ESSs

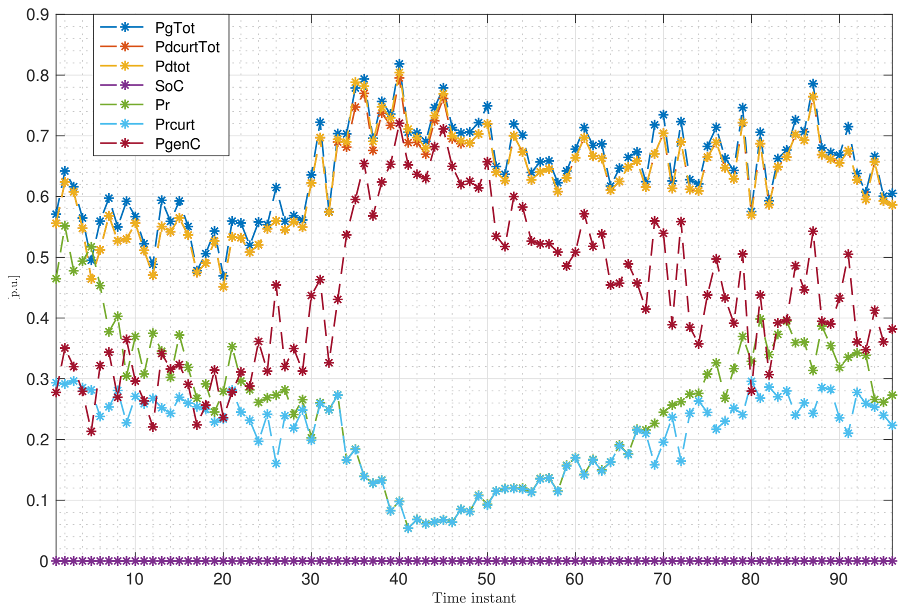


Figure 5.3: Profiles without ESSs

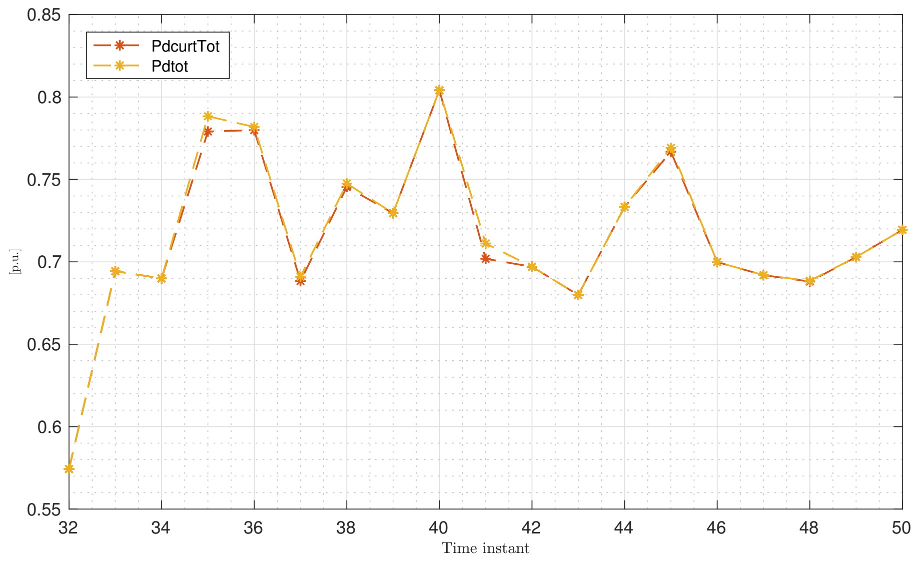


Figure 5.4: Detail of demand profiles with ESSs

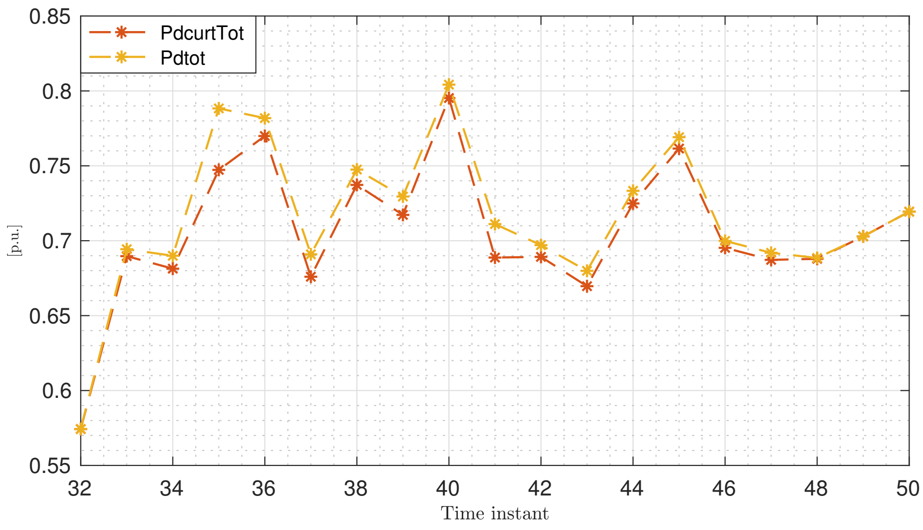


Figure 5.5: Detail of demand profiles without ESSs

In this section, the optimization results are analyzed. Fig. 5.2 reports the profiles of the production from a conventional generator and renewable source, the power demand and the battery SoC. As can be noticed, the total power generation (blue) is always greater than the total power demand (red) and the renewable source is exploited almost to its full capacity. The values of SoC increases when the total generated power exceeds the demand and the energy stored in the battery is then used in order to keep the optimization cost as low as possible, allowing only a 0.04 % load curtailment (Fig. 5.4). Note that the load curtailment value is almost negligible. In order to show the battery importance, the same optimization is conducted without the ESS. Fig. 5.3 shows that the conventional production is increased by 2.08 %, while the renewable source is not completely exploited. Moreover, the total load curtailment is 0.30 % (Fig.5.5), which, although still negligible, becomes 7.5 times worse than the previous case.

5.6 Conclusion

In this chapter a closed loop battery ageing-aware stochastic OPF for networks in islanded mode is proposed. In particular, in this work, the possibility to use DR and ESSs is considered to better cope with uncertainties and a degradation battery model is used to account for the ageing of the ESSs and their cost of replacement. Simulation test performed on IEEE 14 bus benchmark, in which the profile of renewable production and power demands and costs are taken from real data, show the effectiveness of the proposed approach.

Chapter 6

Conclusion and future work

In this thesis, novel and interesting solutions for optimal dispatching in the context of Industry 4.0 are proposed. In particular, two different dispatching problems are addressed. The main problem is the dispatching problem in a semiconductor production site while the minor problem is the optimal dispatching of electric power in power networks operating in islanded mode.

Chapter 2 and Chapter 3 deal with the optimal dispatching in a semiconductor production site, which is a very complex problem, at least NP-hard. In particular, in Chapter 2 several approaches are proposed in order to solve the dispatching problem in a semiconductor manufacturing process. However, as highlighted in the chapter, even if the problem is NP-hard, it is possible to obtain suboptimal solutions that can be both practical and effective. Indeed, our approach proposed in Algorithm 4 has been tested on a real application in Infineon Dresden.
Chapter 3 addresses the problem of planning and scheduling for a fleet of robots with travel time uncertainty operating in manufacturing systems where some demands require to be satisfied at the same time. Inspired by model checking techniques, a general nominal solution is proposed, where the motions of the robots are considered without uncertainty. Successively, with the aim to guarantee the synchronization demands even in the presence of travel time uncertainty, an online controller is proposed.

In Chapter 4, all the details about the hardware and the experimental setting used to validate the approaches proposed in the Chapter 2 and Chapter 3 are provided. Furthermore, several different operations performed by the UNIPV test-bed are shown, see the following YouTube links:

- Path Following - Leminiscate: `https://youtu.be/4I12eG2_WgM`

- Path Following on Graph : `https://youtu.be/K5o2AonO_ZI`, `https://youtu.be/OY-UUDoJXms`

- Multiple robots: `https://youtu.be/DBmJCIEWNVk`

In Chapter 5 a closed loop battery ageing-aware stochastic OPF for networks operating in islanded mode is proposed. In particular, in this chapter, the possibility to use DR and ESSs is considered to better cope with uncertainties. Moreover, a degradation battery model is used to account for the ageing of the ESSs and their cost of replacement.

In the next future, diverse extensions and applications can be studied.
First, the approach proposed in Section 2.7 can be extended to a more generic class of vehicles in order to have an approach as close as possible to the needs of the production site. Indeed, it will be useful to take into account during the

optimization the possibility to have vehicles with different transport capacities.
Secondly, the approach proposed in Chapter 3 could be modified in order to develop an approach in which the uncertainties are considered directly on the Product Automata (PA). In this way, it would be possible to find directly on the PA the robot trajectories that allow to ensure the synchronization requests without using the on-line controller.
Thirdly, as for Chapter 4, possible future works can deal with improving the real-time performance by using the ARXs in the MPC for describing the relationship between PWM and motor angular velocity. Thus, in order to model the robot motors in the MPC via ARXs, a Raspberry with a computing power greater than the one available for this project (PI 3 model) will be required. Alternatively, in order to reduce the computational time, an alternative QP formulation can be applied.
Lastly, as concerns Chapter 5, there are a few possible future works. Chance constraints optimization could be used in order to reduce the computational burden. Besides, it will be interesting to include different programs of demand response, such as load-shifting demand response.

From an application point of view, since the topics of Chapter 2 and Chapter 3 are related to different methods for coordinating a set of robots that must perform a dynamically changing set of tasks, it will be interesting to apply our methods also to similar challenges arising in other areas, e.g., the coordination of autonomous robots in logistics.

Bibliography

[1] seekmomentum. (2019) The evolution of industry 1.0 to 4.0. [Online]. Available: https://www.seekmomentum.com/blog/manufacturing/the-evolution-of-industry-from-1-to-4

[2] S. Wang, J. Wan, D. Li, and C. Zhang, "Implementing smart factory of industrie 4.0: an outlook," *International Journal of Distributed Sensor Networks*, vol. 12, no. 1, p. 3159805, 2016.

[3] V. Alcácer and V. Cruz-Machado, "Scanning the industry 4.0: a literature review on technologies for manufacturing systems," *Engineering Science and Technology, an International Journal*, 2019.

[4] Samsung SDI CO.,LTD, "Wafer," https://www.samsungsdi.com/column/technology/detail/55446.html?listType=gallery, 2019, [Online; accessed 5-September-2019].

[5] F. Thiesse and E. Fleisch, "On the value of location information to lot scheduling in complex manufacturing processes," *International Journal of Production Economics*, vol. 112, no. 2, pp. 532–547, 2008.

[6] MathWorks, "Pinhole camera model," 2019, [Online; accessed September 01, 2019]. [Online]. Available: https://it.mathworks.com/help/vision/ug/camera-calibration.html

[7] Kornia, "Pinhole camer," https://torchgeometry.readthedocs.io/en/latest/pinhole.html, 2019, [Online; accessed 5-September-2019].

[8] T. R. P. Foundation, “Raspberry pi 3 model b,” 2019, [Online: accessed September 01, 2019]. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[9] Dfrobot, “Arduino bluno nano,” 2019, [Online: accessed September 01, 2019]. [Online]. Available: https://www.dfrobot.com/product-1122.html

[10] K. Schwab, *The fourth industrial revolution.* Currency, 2017.

[11] Elisa Convertini, Osservatorio Industria 4.0 , “Le smart technologies alla base della quarta rivoluzione industriale,” https://blog.osservatori.net/it_it/smart-technologies-quarta-rivoluzione-industriale, 2018, [Online; accessed 5-September-2019].

[12] Boston Consulting Group, “Embracing industry 4.0 and rediscovering growth,” https://www.bcg.com/capabilities/operations/embracing-industry-4.0-rediscovering-growth.aspx, [Online; accessed 5-September-2019].

[13] Y. Lu, “Industry 4.0: A survey on technologies, applications and open research issues,” *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, 2017.

[14] H. Kagermann, W. Wahlster, and J. Helbig, “Securing the future of german manufacturing industry,” *Recommendations for implementing the strategic initiative INDUSTRIE*, vol. 4, no. 199, p. 14, 2013.

[15] T. Bauernhansl, B. Diegner, J. Diemer, M. Dümmler, C. Eckert, W. Herfs, H. Heyn, C. Hilger, M. Ten Hompel, J. Kalhoff *et al.*, “Industrie 4.0-whitepaper fue-themen,” *Berlin: Bundesministerium für Wirtschaft und Energie-Plattform Industrie*, vol. 4, 2014.

[16] V. Bitkom, “Zvei.(2015). umsetzungsstrategie industrie 4.0 ergebnisbericht der plattform industrie 4.0,” *Acatech. Munich.*

[17] T. Stock and G. Seliger, “Opportunities of sustainable manufacturing in industry 4.0,” *Procedia Cirp*, vol. 40, pp. 536–541, 2016.

[18] H. Foidl and M. Felderer, “Research challenges of industry 4.0 for quality management,” in *International Conference on Enterprise Resource Planning Systems.* Springer, 2015, pp. 121–137.

[19] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg, “How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective,” *International journal of mechanical, industrial science and engineering*, vol. 8, no. 1, pp. 37–44, 2014.

[20] S. Immediato. (2018) Industry 4.0 e digital twin: grado di conoscenza e analisi di applicabilita alle pmi. [Online]. Available: https://webthesis.biblio.polito.it/8815/

[21] K. Zhou, T. Liu, and L. Zhou, “Industry 4.0: Towards future industrial opportunities and challenges,” in *2015 12th International conference on fuzzy systems and knowledge discovery (FSKD).* IEEE, 2015, pp. 2147–2152.

[22] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster, *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group.* Forschungsunion, 2013.

[23] M. Hermann, T. Pentek, and B. Otto, “Design principles for industrie 4.0 scenarios: A literature review,” 01 2015.

[24] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[25] E. A. Lee, “Cyber physical systems: Design challenges,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC).* IEEE, 2008, pp. 363–369.

[26] J. W. Strandhagen, E. Alfnes, J. O. Strandhagen, and L. R. Vallandingham, “The fit of industry 4.0 applications in manufacturing logistics: a multiple case study,” *Advances in Manufacturing*, vol. 5, no. 4, pp. 344–358, 2017.

[27] N. Boulila, “Cyber-physical systems and industry 4.0: Properties, structure, communication, and behavior,” 04 2019.

[28] S. Keil, “Design of a cyber-physical production system for semiconductor manufacturing,” in *Proceedings of the Hamburg International Conference of Logistics (HICL).* epubli, 2017, pp. 319–340.

[29] D. Giusto, A. Iera, G. Morabito, and L. Atzori, *The internet of things: 20th Tyrrhenian workshop on digital communications.* Springer Science & Business Media, 2010.

[30] V. Mittelstädt, P. Brauner, M. Blum, and M. Ziefle, “On the visual design of erp systems the–role of information complexity, presentation and human factors,” *Procedia Manufacturing*, vol. 3, pp. 448–455, 2015.

[31] M. Singh, I. Khan, and S. Grover, “Tools and techniques for quality management in manufacturing industries (pp. 853–859),” in *Faridabad, Haryana: Proceedings of the National Conference on Trends and Advances in Mechanical Engineering, YMCA University of Science & Technology*, 2012.

[32] SemI40 Consortium, “Power semiconductor and electronics manufacturing 4.0,” http://www.semi40.eu/, 2018, [Online; accessed 5-September-2019].

[33] T. Chao, *Introduction to semiconductor manufacturing technology.* SPIE PRESS, 2001.

[34] Y. Nishi and R. Doering, *Handbook of semiconductor manufacturing technology.* CRC Press, 2007.

[35] M. Quirk and J. Serda, *Semiconductor manufacturing technology.* Prentice Hall Upper Saddle River, NJ, 2001, vol. 1.

[36] F. Thiesse and E. Fleisch, “On the value of location information to lot scheduling in complex manufacturing processes,” *International Journal of Production Economics*, vol. 112, no. 2, pp. 532–547, 2008.

[37] S. C. Sarin, A. Varadarajan, and L. Wang, "A survey of dispatching rules for operational control in wafer fabrication," *Production Planning and Control*, vol. 22, no. 1, pp. 4–24, 2011.

[38] I. F. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 170, no. 3, pp. 677–709, 2006.

[39] T. Ganesharajah, N. G. Hall, and C. Sriskandarajah, "Design and operational issues in agv-served manufacturing systems," *Annals of Operations Research*, vol. 76, pp. 109–154, 1998.

[40] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[41] P.-H. Koo, J. Jang, and J. Suh, "Vehicle dispatching for highly loaded semiconductor production considering bottleneck machines first," *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 1, pp. 23–38, 2005.

[42] O. Z. Maimon and S. B. Gershwin, "Dynamic scheduling and routing for flexible manufacturing systems that have unreliable machines," *Operations Research*, vol. 36, no. 2, pp. 279–292, 1988.

[43] A. Seidmann, "On-line scheduling of a robotic manufacturing cell with stochastic sequence-dependent processing rates," *International journal of production research*, vol. 25, no. 6, pp. 907–924, 1987.

[44] A. Tenenbaum and A. Seidmann, "Dynamic load control policies for a flexible manufacturing system with stochastic processing rates," *International Journal of Flexible Manufacturing Systems*, vol. 2, no. 2, pp. 93–120, 1989.

[45] Y. Yih and A. Thesen, "Semi-markov decision models for real-time scheduling," *The International Journal of Production Research*, vol. 29, no. 11, pp. 2331–2346, 1991.

[46] J. Kimemia and S. B. Gershwin, "An algorithm for the computer control of a flexible manufacturing system," *AIIE Transactions*, vol. 15, no. 4, pp. 353–362, 1983.

[47] O. Z. Maimon, "Real-time operational control of flexible manufacturing systems," *Journal of Manufacturing Systems*, vol. 6, no. 2, pp. 125–136, 1987.

[48] L. M. Wein, "Scheduling semiconductor wafer fabrication," *IEEE Transactions on semiconductor manufacturing*, vol. 1, no. 3, pp. 115–130, 1988.

[49] S. X. Lou and P. W. Kager, "A robust production control policy for vlsi wafer fabrication," *IEEE Transactions on semiconductor manufacturing*, vol. 2, no. 4, pp. 159–164, 1989.

[50] Y.-F. Hung and I.-R. Chen, "A simulation study of dispatch rules for reducing flow times in semiconductor wafer fabrication," *Production Planning & Control*, vol. 9, no. 7, pp. 714–722, 1998.

[51] R. Uzsoy, L. A. Martin-Vega, C.-Y. Lee, and P. A. Leonard, "Production scheduling algorithms for a semiconductor test facility," *IEEE Transactions on Semiconductor Manufacturing*, vol. 4, no. 4, pp. 270–280, 1991.

[52] C. I. Vasile and C. Belta, "An automata-theoretic approach to the vehicle routing problem." in *Robotics: Science and Systems*, 2014.

[53] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[54] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," in *2008 47th IEEE Conference on Decision and Control*. IEEE, 2008, pp. 3953–3958.

[55] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[56] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems.* Springer, 2004, pp. 152–166.

[57] L. Mönch, J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose, "A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations," *Journal of scheduling*, vol. 14, no. 6, 2011.

[58] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-uav mission planning," *International Journal of Robust and Nonlinear Control*, vol. 21, no. 12, pp. 1372–1395, 2011.

[59] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *International Conference on Hybrid systems: computation and control.* ACM, 2013, pp. 1–10.

[60] J. Tumova, S. Karaman, C. Belta, and D. Rus, "Least-violating planning in road networks from temporal logic specifications," in *Procs. of the 7th International Conference on Cyber-Physical Systems.* IEEE, 2016, p. 17.

[61] D. Aksaray, C.-I. Vasile, and C. Belta, "Dynamic routing of energy-aware vehicles with Temporal Logic Constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3141–3146.

[62] Y. Meng, Y. Yang, H. Chung, P.-H. Lee, and C. Shao, "Enhancing sustainability and energy efficiency in smart factories: A review," *Sustainability*, vol. 10, no. 12, p. 4779, 2018.

[63] A. Rojko, "Industry 4.0 concept: background and overview," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 11, no. 5, pp. 77–90, 2017.

[64] F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm," in *2014 IEEE international*

conference on industrial engineering and engineering management. IEEE, 2014, pp. 697–701.

[65] M. Lom, O. Pribyl, and M. Svitek, “Industry 4.0 as a part of smart cities,” in *2016 Smart Cities Symposium Prague (SCSP).* IEEE, 2016, pp. 1–6.

[66] E. Commission, “Strategic energy technology plan (set-plan,” European Commission, Tech. Rep., 2016.

[67] E. E. D. of Energy (DOE) and R. Energy, “20Increasing wind energy’s contribution to u.s. electricity supply,” Energy Efficiency Department of Energy (DOE) and Renewable Energy, Tech. Rep., 2008.

[68] E. position paper., “2050: Facilitating 50transmission infrastructure, system operation and electricity market integration.” EWEA European Wind Energy Association, Tech. Rep., 2009.

[69] D. Bienstock, M. Chertkov, and S. Harnett, “Chance-constrained optimal power flow: Risk-aware network control under uncertainty,” *SIAM Review*, vol. 56, no. 3, pp. 461–495, 2014.

[70] J. Carpentier, “Contribution a l’étude du dispatching économique,” *Bulletin de la Société Française des Électriciens*, vol. vol. III, pp. 431–437, 1962.

[71] R. P. O. M. B. Cain and A. Castillo, “History of optimal power flow and formulations (opf paper 1),” US Federal Energy Resource Commission (FERC), Tech. Rep., 2012.

[72] E. Dall’Anese and A. Simonetto, “Optimal power flow pursuit,” *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 942–952, March 2018.

[73] S. Paudyal, C. A. Canizares, and K. Bhattacharya, “Optimal operation of distribution feeders in smart grids,” *IEEE Transactions on Industrial Electronics*, vol. 58, no. 10, pp. 4495–4503, Oct 2011.

[74] H. M. Khodr, M. A. Matos, and J. Pereira, “Distribution optimal power flow,” in *2007 IEEE Lausanne Power Tech*, July 2007, pp. 1441–1446.

[75] R. Tonkoski, L. A. C. Lopes, and T. H. M. El-Fouly, "Coordinated active power curtailment of grid connected pv inverters for overvoltage prevention," *IEEE Transactions on Sustainable Energy*, vol. 2, no. 2, pp. 139–147, April 2011.

[76] J. von Appen, T. Stetz, M. Braun, and A. Schmiegel, "Local voltage control strategies for pv storage systems in distribution grids," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, pp. 1002–1009, March 2014.

[77] X. Bai, H. Wei, K. Fujisawa, and Y. Wang, "Semidefinite programming for optimal power flow problems," *International Journal of Electrical Power & Energy Systems*, vol. 30, no. 6, pp. 383 – 392, 2008.

[78] O. Alsac, J. Bright, M. Prais, and B. Stott, "Further developments in lp-based optimal power flow," *IEEE Transactions on Power Systems*, vol. 5, no. 3, pp. 697–711, 1990.

[79] B. Stott, J. Jardim, and O. Alsac, "Dc power flow revisited," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1290–1300, 2009.

[80] J. Lavaei and S. H. Low, "Zero duality gap in optimal power flow problem," *IEEE Transactions on Power Systems*, vol. 27, no. 1, pp. 92–107, 2012.

[81] R. Madani, S. Sojoudi, and J. Lavaei, "Convex relaxation for optimal power flow problem: Mesh networks," *IEEE Transactions on Power Systems*, vol. 30, no. 1, pp. 199–211, 2015.

[82] R. Madani, M. Ashraphijuo, and J. Lavaei, "Promises of conic relaxation for contingency-constrained optimal power flow problem," *IEEE Transactions on Power Systems*, vol. 31, no. 2, pp. 1297–1307, 2016.

[83] H. Zhang and P. Li, "Probabilistic analysis for optimal power flow under uncertainty," *IET Generation, Transmission & Distribution*, vol. 4, no. 5, pp. 553–561, 2010.

[84] T. Mühlpfordt, T. Faulwasser, and V. Hagenmeyer, "Solving stochastic ac power flow via polynomial chaos expansion," in *Control Applications (CCA), 2016 IEEE Conference on.* IEEE, 2016, pp. 70–76.

[85] M. Chamanbaz, F. Dabbene, and C. Lagoa, "Ac optimal power flow in the presence of renewable sources and uncertain loads," *arXiv preprint arXiv:1702.02967*, 2017.

[86] M. Vrakopoulou, M. Katsampani, K. Margellos, J. Lygeros, and G. Andersson, "Probabilistic security-constrained ac optimal power flow," in *PowerTech, 2013 IEEE Grenoble.* IEEE, 2013, pp. 1–6.

[87] M. Vrakopoulou, J. L. Mathieu, and G. Andersson, "Stochastic optimal power flow with uncertain reserves from demand response," in *System Sciences (HICSS), 2014 47th Hawaii International Conference on.* IEEE, 2014, pp. 2353–2362.

[88] S. Formentin, F. Dabbene, R. Tempo, L. Zaccarian, and S. M. Savaresi, "Scenario optimization with certificates and applications to anti-windup design," in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 2810–2815.

[89] A. Papavasiliou and S. S. Oren, "A stochastic unit commitment model for integrating renewable supply and demand response," in *Power and Energy Society General Meeting, 2012 IEEE.* IEEE, 2012, pp. 1–6.

[90] W. A. Bukhsh, C. Zhang, and P. Pinson, "An integrated multiperiod opf model with demand response and renewable generation uncertainty," *IEEE Transactions on Smart Grid*, vol. 7, no. 3, pp. 1495–1503, 2016.

[91] R. M. Institute, "Demand response: an introduction-overview of lessons, technologies, and lesson learned," Rocky Mountain Institute, Tech. Rep., 2006.

[92] G. Coppez, S. Chowdhury, and S. P. Chowdhury, "The importance of energy storage in renewable power generation: A review," in *45th International Universities Power Engineering Conference UPEC2010*, 2010, pp. 1–5.

[93] X. Luo, J. Wang, M. Dooner, and J. Clarke, “Overview of current development in electrical energy storage technologies and the application potential in power system operation,” *Applied Energy*, vol. 137, no. Supplement C, pp. 511 – 536, 2015.

[94] H. Chen, T. N. Cong, W. Yang, C. Tan, Y. Li, and Y. Ding, “Progress in electrical energy storage system: A critical review,” *Progress in Natural Science*, vol. 19, no. 3, pp. 291 – 312, 2009.

[95] T. Summers, J. Warrington, M. Morari, and J. Lygeros, “Stochastic optimal power flow based on conditional value at risk and distributional robustness,” *International Journal of Electrical Power & Energy Systems*, vol. 72, pp. 116–125, 2015.

[96] R. Bellman, “s dynamic programming ,Äûprinceton university press,” 1957.

[97] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control.* Athena scientific Belmont, MA, 1995, vol. 1, no. 2.

[98] H. N. Psaraftis, “A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem,” *Transportation Science*, vol. 14, no. 2, pp. 130–154, 1980.

[99] H. He, R. Xiong, and J. Fan, “Evaluation of lithium-ion battery equivalent circuit models for state of charge estimation by an experimental approach,” *energies*, vol. 4, no. 4, pp. 582–598, 2011.

[100] P. Cicconi, L. Postacchini, E. Pallotta, A. Monteriù, M. Prist, M. Bevilacqua, and M. Germani, “A life cycle costing of compacted lithium titanium oxide batteries for industrial applications,” *Journal of Power Sources*, vol. 436, p. 226837, 2019.

[101] A. Pozzi, M. Zambelli, A. Ferrara, and D. M. Raimondo, “Balancing-aware charging strategy for series-connected lithium-ion cells: A nonlinear model predictive control approach,” *arXiv preprint arXiv:1902.02122*, 2019.

[102] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “Casadi: a software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[103] F. Bullo, E. Frazzoli, M. Pavone, K. Savla, and S. L. Smith, “Dynamic vehicle routing for robotic systems,” *Procs. of the IEEE*, vol. 99, no. 9, pp. 1482–1504, 2011.

[104] P. Toth and D. Vigo, *The Vehicle Routing Problem.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001.

[105] C. Baier and J.-P. Katoen, *Principles of model checking.* MIT press, 2008.

[106] C.-I. Vasile, D. Aksaray, and C. Belta, “Time window temporal logic,” *Theoretical Computer Science*, vol. 691, pp. 27–54, 2017.

[107] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems.* Springer, 2017, vol. 89.

[108] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach.* Prentice Hall Professional Technical Reference, 2002.

[109] P.Kool, *Robotica.*

[110] R. Ilardo. Elettronica e altro. [Online]. Available: http://www.raffaeleilardo.it/pwmsp1.htm

[111] Wikipedia contributors, “Bluetooth — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=Bluetooth&oldid=913570834, 2019, [Online; accessed 5-September-2019].

[112] S. M. LaValle, *Planning Algorithms.* Cambridge University Press, 2006.

[113] B. Huyck, J. Van Impe, and B. De Moor, *Identification and Modeling of Dynamical System*, 2008.

[114] S. Rachad, B. Bensassi, and B. Nsiri, “System identification of inventory system using arx and armax models,” *International journal of Control and Automation*, 2015.

[115] Y.Chetouani, *Using ARX approach for modelling and prediction of the dynamics of a reactor-exchanger.*

[116] L. Magni and R. Scattolini, *Advanced and multivariable control.* Pitagora, 2019.

[117] J. M. Reniers, G. Mulder, S. Ober-Blobaum, and D. A. Howey, "Improving optimal control of grid-connected lithium-ion batteries through more accurate battery and degradation modelling," *arXiv preprint arXiv:1710.04552*, 2017.

[118] D. V. Kumar, "Intelligent controllers for automatic generation control," in *TENCON'98. 1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control*, vol. 2. IEEE, 1998, pp. 557–574.

[119] A. Soundarrajan, S. Sumathi, and G. Sivamurugan, "Voltage and frequency control in power generating system using hybrid evolutionary algorithms," *Journal of Vibration and Control*, vol. 18, no. 2, pp. 214–227, 2012.

[120] J. Carpentier, "Optimal power flows," *International Journal of Electrical Power & Energy Systems*, vol. 1, no. 1, pp. 3–15, 1979.

[121] M. Campi and S. Garatti, "The eaxct feasibility of randomized solutions of robust convex programs," *SIAM Journal on Control and Oprimization*, vol. 19, no. 3, pp. 1211–1230, 2007.

[122] M. Torchio, L. Magni, and D. M. Raimondo, "A mixed integer sdp approach for the optimal placement of energy storage devices in power grids with renewable penetration," in *2015 American Control Conference (ACC)*, July 2015, pp. 3892–3897.

[123] [Online]. Available: http://www.transparency.eex.com/

[124] B.-M. Hodge, D. Lew, M. Milligan, H. Holttinen, S. Sillanpää, E. Gómez-Lázaro, R. Scharff, L. Söder, X. G. Larsén, G. Giebel *et al.*, "Wind power forecasting error distributions: An international comparison," in *11th Annual*

International Workshop on Large-Scale Integration of Wind Power into Power Systems as well as on Transmission Networks for Offshore Wind Power Plants Conference, 2012.

[125] J. Lofberg, "Yalmip : a toolbox for modeling and optimization in matlab," in *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, Sept 2004, pp. 284–289.

[126] D. Phan and S. Ghosh, "Two-stage stochastic optimization for optimal power flow under renewable generation uncertainty," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 24, no. 1, p. 2, 2014.

[127] S. Gill, I. Kockar, and G. W. Ault, "Dynamic optimal power flow for active distribution networks," *IEEE Transactions on Power Systems*, vol. 29, no. 1, pp. 121–131, 2014.

[128] A. Giannitrapani, S. Paoletti, A. Vicino, and D. Zarrilli, "Optimal allocation of energy storage systems for voltage control in lv distribution networks," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2859–2870, 2017.

[129] D. Gayme and U. Topcu, "Optimal power flow with distributed energy storage dynamics," in *Proceedings of the 2011 American Control Conference*, June 2011, pp. 1536–1542.

[130] Y. M. Atwa and E. F. El-Saadany, "Optimal allocation of ess in distribution systems with a high penetration of wind energy," *IEEE Transactions on Power Systems*, vol. 25, no. 4, pp. 1815–1822, Nov 2010.

[131] X. Bai, H. Wei, K. Fujisawa, and Y. Wang, "Semidefinite programming for optimal power flow problems," *International Journal of Electrical Power & Energy Systems*, vol. 30, no. 6, pp. 383 – 392, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0142061507001378

[132] M. Farivar and S. H. Low, "Branch flow model: Relaxations and convexification—part i," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 2554–2564, 2013.

[133] Z. Hu, X. Wang, and G. Taylor, "Stochastic optimal reactive power dispatch: Formulation and solution method," *International Journal of Electrical Power & Energy Systems*, vol. 32, no. 6, pp. 615 – 621, 2010.

[134] H. Zhang and P. Li, "Chance constrained programming for optimal power flow under uncertainty," *IEEE Transactions on Power Systems*, vol. 26, no. 4, pp. 2417–2424, Nov 2011.

[135] M. El-Hawary and G. Mbamalu, "Stochastic optimal load flow using a combined quasi-newton and conjugate gradient technique," *International Journal of Electrical Power & Energy Systems*, vol. 11, no. 2, pp. 85 – 93, 1989.

[136] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski, "Adjustable robust solutions of uncertain linear programs," *Mathematical Programming*, vol. 99, no. 2, pp. 351–376, 2004.

[137] M. Madrigal, K. Ponnambalam, and V. H. Quintana, "Probabilistic optimal power flow," in *Conference Proceedings. IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.98TH8341)*, vol. 1, May 1998, pp. 385–388 vol.1.

[138] U. D. of Energy, "Benefit of demand response in electricity market and recommendations for achieving them," United States Congress Pursuant to Section 1252 of the Energy Policy Act of 2005, Tech. Rep., Feb. 2006.

[139] M. Negnevitsky, T. D. Nguyen, and M. de Groot, "Novel business models for demand response exchange," in *IEEE PES General Meeting*, July 2010, pp. 1–7.

[140] X. Bai and H. Wei, "A semidefinite programming method with graph partitioning technique for optimal power flow problems," *International Journal of Electrical Power & Energy Systems*, vol. 33, no. 7, pp. 1309 – 1314, 2011.

[141] M. Farivar and S. H. Low, "Branch flow model: Relaxations and convexification;part ii," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 2565–2572, 2013.

[142] S. H. Low, “Convex relaxation of optimal power flow;part ii: Exactness,” *IEEE Transactions on Control of Network Systems*, vol. 1, no. 2, pp. 177–189, 2014.

[143] ——, “Convex relaxation of optimal power flow;part i: Formulations and equivalence,” *IEEE Transactions on Control of Network Systems*, vol. 1, no. 1, pp. 15–27, 2014.

[144] C. Coffrin and P. V. Hentenryck, “A linear-programming approximation of ac power flows,” *INFORMS Journal on Computing*, vol. 26, no. 4, pp. 718–734, 2014. [Online]. Available: https://doi.org/10.1287/ijoc.2014.0594

[145] B. Stott, J. Jardim, and O. Alsac, “Dc power flow revisited,” *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1290–1300, 2009.

[146] M. Abido, “Optimal power flow using particle swarm optimization,” *International Journal of Electrical Power and Energy Systems*, vol. 24, no. 7, pp. 563 – 571, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0142061501000679

[147] L. Lai, J. Ma, R. Yokoyama, and M. Zhao, “Improved genetic algorithms for optimal power flow under both normal and contingent operation states,” *International Journal of Electrical Power and Energy Systems*, vol. 19, no. 5, pp. 287 – 292, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0142061596000518

[148] X. Yan and V. H. Quintana, “Improving an interior-point-based opf by dynamic adjustments of step sizes and tolerances,” *IEEE Transactions on Power Systems*, vol. 14, no. 2, pp. 709–717, May 1999.

[149] J. A. Momoh and J. Z. Zhu, “Improved interior point method for opf problems,” *IEEE Transactions on Power Systems*, vol. 14, no. 3, pp. 1114–1120, Aug 1999.

[150] H. W. Dommel and W. F. Tinney, “Optimal power flow solutions,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-87, no. 10, pp. 1866–1876, Oct 1968.

[151] ——, “Optimal power flow solutions,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-87, no. 10, pp. 1866–1876, Oct 1968.

[152] M. Huneault and F. D. Galiana, “A survey of the optimal power flow literature,” *IEEE Transactions on Power Systems*, vol. 6, no. 2, pp. 762–770, 1991.

[153] J. Lavaei, “Zero duality gap for classical opf problem convexifies fundamental nonlinear power problems,” in *Proceedings of the 2011 American Control Conference*, 2011, pp. 4566–4573.

[154] J. Tumova and D. V. Dimarogonas, “Multi-agent planning under local ltl specifications and event-based synchronization,” *Automatica*, vol. 70, pp. 239–248, 2016.

[155] D. Aksaray, K. Leahy, and C. Belta, “Distributed multi-agent persistent surveillance under temporal logic constraints,” *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 174–179, 2015.

[156] J. M. Smith and A. B. Jones, *Book Title*, 7th ed. Publisher, 2012.

[157] R. Uzsoy, L. A. Martin-Vega, C.-Y. Lee, and P. A. Leonard, “Production scheduling algorithms for a semiconductor test facility,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 4, no. 4, pp. 270–280, 1991.

[158] S. C. Sarin, A. Varadarajan, and L. Wang, “A survey of dispatching rules for operational control in wafer fabrication,” *Production Planning and Control*, vol. 22, no. 1, pp. 4–24, 2011.

[159] A. B. Jones and J. M. Smith, “Article Title,” *Journal Title*, vol. 13, no. 52, pp. 123–456, March 2013.

[160] S. Karaman and E. Frazzoli, “Complex mission optimization for multiple-uavs using linear temporal logic,” in *2008 american control conference*. IEEE, 2008, pp. 2003–2009.

[161] C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scltl motion planning for mobility-on-demand," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on.* IEEE, 2017, pp. 1481–1488.

[162] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-uav mission planning," *International Journal of Robust and Nonlinear Control*, vol. 21, no. 12, pp. 1372–1395, 2011.

[163] M. Pavone, S. L. Smith, E. Frazzoli, and D. Rus, "Robotic load balancing for mobility-on-demand systems," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 839–854, 2012.

[164] M. Faied, A. Mostafa, and A. Girard, "Dynamic optimal control of multiple depot vehicle routing problem with metric temporal logic," in *American Control Conference, 2009. ACC'09.* IEEE, 2009, pp. 3268–3273.

[165] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *Proceedings of the 16th international conference on Hybrid systems: computation and control.* ACM, 2013, pp. 353–362.

[166] L. Mönch and I. Habenicht, "Factory scheduling and dispatching: simulation-based assessment of batching heuristics in semiconductor manufacturing," in *Proceedings of the 35th conference on Winter simulation: driving innovation.* Winter Simulation Conference, 2003, pp. 1338–1345.

[167] L. F. Atherton and R. W. Atherton, *Wafer fabrication: Factory performance and analysis.* Springer Science & Business Media, 1995, vol. 339.

[168] S. M. Sze, *Semiconductor devices: physics and technology.* John Wiley & Sons, 2008.

[169] R. Uzsoy, C.-Y. Lee, and L. A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry part i: system charac-

teristics, performance evaluation and production planning," *IIE transactions*, vol. 24, no. 4, pp. 47–60, 1992.

[170] K. Kim and G. E. Fainekos, "Approximate solutions for the minimal revision problem of specification automata," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2012, pp. 265–271.

[171] M. M. Srinivasan, Y. A. Bozer, and M. Cho, "Trip-based material handling systems: throughput capacity analysis," *IIE transactions*, vol. 26, no. 1, pp. 70–89, 1994.

[172] C. G. Co and J. M. A. Tanchoco, "A review of research on agvs vehicle management," *Engineering Costs and Production Economics*, vol. 21, no. 1, pp. 35–42, 1991.

[173] I. Sabuncuoglu, "A study of scheduling rules of flexible manufacturing systems: a simulation approach," *International Journal of Production Research*, vol. 36, no. 2, pp. 527–546, 1998.

[174] T. Le-Anh, *Intelligent control of vehicle-based internal transport systems*, 2005, no. 51.

[175] T. Le-Anh and M. De Koster, "A review of design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 171, no. 1, pp. 1–23, 2006.

[176] R. de Koster and J. R. van der Meer, "Centralized versus decentralized control of internal transport, a case study," in *Advances in distribution logistics.* Springer, 1998, pp. 403–420.

[177] J. J. Bartholdi I and L. K. Platzman, "Decentralized control of automated guided vehicles on a simple loop," *IIE transactions*, vol. 21, no. 1, pp. 76–81, 1989.

[178] C. Klei and J. Kim, "Agv dispatching," *International Journal of Production Research*, vol. 34, no. 1, pp. 95–110, 1996.

[179] P. J. Egbelu and J. M. Tanchoco, “Characterization of automatic guided vehicle dispatching rules,” *The International Journal of Production Research*, vol. 22, no. 3, pp. 359–374, 1984.

[180] T. J. Hodgson, R. E. King, S. K. Monteith, and S. R. Schultz, “Developing control rules for an agv using markov decision processes,” in *1985 24th IEEE Conference on Decision and Control*. IEEE, 1985, pp. 1817–1821.

[181] J. T. Lin, F.-K. Wang, and P.-Y. Yen, “Simulation analysis of dispatching rules for an automated interbay material handling system in wafer fab,” *International Journal of Production Research*, vol. 39, no. 6, pp. 1221–1238, 2001.

[182] Z. Bian, Y. Yang, W. Mi, and C. Mi, “Dispatching electric agvs in automated container terminals with long travelling distance,” *Journal of coastal research*, vol. 73, no. sp1, pp. 75–82, 2015.

[183] T. Kawakami and S. Takata, “Battery life cycle management for automatic guided vehicle systems,” in *Design for Innovative Value Towards a Sustainable Society*. Springer, 2012, pp. 403–408.

[184] X. Zhan, L. Xu, J. Zhang, and A. Li, “Study on agvs battery charging strategy for improving utilization,” *Procedia CIRP*, vol. 81, pp. 558–563, 2019.

[185] Q. S. Kabir and Y. Suzuki, “Increasing manufacturing flexibility through battery management of automated guided vehicles,” *Computers & Industrial Engineering*, vol. 117, pp. 225–236, 2018.

[186] R. McHaney, “Modelling battery constraints in discrete event automated guided vehicle simulations,” *International journal of production research*, vol. 33, no. 11, pp. 3023–3040, 1995.

[187] M. Ebben, *Logistic control in automated transportation networks*. Twente University Press Enschede, The Netherlands, 2001.

[188] I. Sabuncuoglu and D. L. Hommertzheim, “Dynamic dispatching algorithm for scheduling machines and automated guided vehicles in a flexible manufacturing

system," *The International Journal of Production Research*, vol. 30, no. 5, pp. 1059–1079, 1992.

[189] C. Kim, J. Tanchoco, and P.-H. Koo, "Agv dispatching based on workload balancing," *International Journal of Production Research*, vol. 37, no. 17, pp. 4053–4066, 1999.

[190] B. H. Jeong and S. U. Randhawa, "A multi-attribute dispatching rule for automated guided vehicle systems," *International Journal of Production Research*, vol. 39, no. 13, pp. 2817–2832, 2001.

[191] A. Pozzi, M. Torchio, and D. M. Raimondo, "Assessing the performance of model-based energy saving charging strategies in li-ion cells," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2018, pp. 806–811.

[192] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, no. 6, pp. 503–515, 11 1954. [Online]. Available: https://projecteuclid.org:443/euclid.bams/1183519147

[193] A. Fusiello, "Visione computazionale," *Appunti delle lezioni. Pubblicato a cura dell,Äôautore*, 2008.

[194] O. Faugeras, Q.-T. Luong, and T. Papadopoulo, *The geometry of multiple images: the laws that govern the formation of multiple images of a scene and some of their applications*. MIT press, 2001.

[195] Q. Zhang and R. Pless, "Extrinsic calibration of a camera and laser range finder (improves camera calibration)," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2301–2306.

[196] OpenCv, "Pinhole camera model," 2019, [Online: accessed September 01, 2019]. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[197] Y. Ma, S. Sastry, and S. Soatto, *An Invitation to 3-D Vision.* Springer-Verlag New York, 2004.

[198] P. Aguiar and J. P. Hespanha, *Trajectory-Tracking and Path-Following of Underactuated Autonomous Vehicles with Parametric Modeling Uncertainty*, 2007.

[199] T. Tashiro, "Vehicle steering control with mpc for target trajectory tracking of autonomous reverse parking," *International Conference on Control Applications (CCA)*, 2013.

[200] J. K. Huusom, N. K. Poulsen, and S. B. Jorgensen, "Arx-model based model predictive control with offset-free tracking," *Computer Aided Chemical Engineering*, 2010.

[201] D. M. Raimondo. (2009) Nonlinear model predictive control stability, robustness and applications. [Online]. Available: http://sisdin.unipv.it/labsisdin/raimondo/vienna.php

[202] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[203] L. Research, "Building smarter manufacturing with the internet of things (iot)," 2014.

[204] C. Bartodziej, *The Concept Industry 4.0*, 01 2017.

[205] J. H. Ang, C. Goh, A. A. F. Saldivar, and Y. Li, "Energy-efficient through-life smart design, manufacturing and operation of ships in an industry 4.0 environment," *Energies*, vol. 10, no. 5, p. 610, 2017.

[206] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *Cirp Annals*, vol. 65, no. 2, pp. 621–641, 2016.

[207] A. Pereira and F. Romero, "A review of the meanings and the implications of the industry 4.0 concept," *Procedia Manufacturing*, vol. 13, pp. 1206–1214, 2017.

[208] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, May 2014, pp. 1–4.

[209] K. Suri, A. Cuccuru, J. Cadavid, S. Gerard, W. Gaaloul, and S. Tata, "Model-based development of modular complex systems for accomplishing system integration for industry 4.0," 02 2017, pp. 487–495.