Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

COPPE UFRJ

A SUPERVISORY CONTROL-BASED NAVIGATION ARCHITECTURE FOR AUTONOMOUS ROBOTS IN INDUSTRY 4.0 ENVIRONMENTS

Antonio Galiza Cerdeira Gonzalez

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientadores: João Carlos dos Santos Basilio
Lilian Kawakami Carvalho

Rio de Janeiro
Abril de 2019

A SUPERVISORY CONTROL-BASED NAVIGATION ARCHITECTURE FOR AUTONOMOUS ROBOTS IN INDUSTRY 4.0 ENVIRONMENTS

Antonio Galiza Cerdeira Gonzalez

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2019

Gonzalez, Antonio Galiza Cerdeira

A Supervisory Control-Based Navigation Architecture for Autonomous Robots in Industry 4.0 Environments/Antonio Galiza Cerdeira Gonzalez. – Rio de Janeiro: UFRJ/COPPE, 2019.

XVIII, 81 p.: il.; 29, 7cm.

Orientadores: João Carlos dos Santos Basilio
Lilian Kawakami Carvalho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2019.

Referências Bibliográficas: p. 74 – 77.

1. Sistemas a Eventos Discretos. 2. Autômatos. 3. Controle supervisório modular. 4. Robótica móvel. 5. Indústria 4.0. 6. Veículo guiado automaticamente. 7. Navegação de um robô. 8. Navegação de múltiplos robôs. 9. Discrete event systems. 10. Automaton. 11. Modular supervisory control. 12. Mobile robotics. 13. Industry 4.0. 14. Automated guided vehicle. 15. Single robot navigation. 16. Multiple robot navigation. I. Basilio, João Carlos dos Santos *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Non nobis Domine, non nobis,
sed nomini tuo da gloriam.

Acknowledgements

First, I thank God Almighty, who in His infinite love created me and enabled me to elaborate and complete this dissertation. I also thank Our Lady, Mother of God and Saint Joseph, her husband, for praying for me in the most difficult moments.

No words are enough to thank my parents, José and Elisiane, for all their love, care, patience, dedication and support throughout my life. Not only did they bring me into the world as they taught most of the things that make me who I am. Without all the support and their teachings, this dissertation would never exist.

To my brother Lucas, for all friendship and companionship throughout our lives; so many adventures lived together! Thanks for the patience in my moments of bad mood and for all the support throughout this Masters course.

To my grandparents, Maria (in memoriam), José, Elizabeth, and Antônio; for all love and life lessons given; and many good moments that I will never forget.

To José Gomes Pinto, my dear Dedé, whom I consider a third grandfather, for all the affection and dedication dedicated me and to our family.

To my uncles and aunts, Marcarmem, Gentill, Christine, Antônio and Vera, for all the affection and support throughout life; as well as to my cousins Felipe, Daniela, André, Lílian, Tiago and Bárbara for their friendship.

To my dog, Loup, for cheering me on when things got tough with his silent company and devotion.

To all my teachers, who taught me most of what I know; whose teachings were essential to the conception of this work.

To my advisers, João Carlos dos Santos Basilio and Lilian Kawakami Carvalho, who for so long have dedicated themselves to supporting me and motivating me with their attention and friendship; besides contributing enormously to the accomplishment of this work.

To my colleagues and friends of the Laboratory of Control and Automation, Marcos Vinícius and Gustavo Vianna; whose assistance was indispensable to the conclusion of this dissertation, Raphael Barcelos, Thiago Tuxi, Weslley Silva, Públio Macedo, José Villardi, Ricardo Macedo, Juliano Freire and Jéssica Vieira; whose friendship and knowledge helped me a lot.

To my colleagues of the masters course, Victor Hugo, Pamela Siekmann, Gabriel

Solino, Maria Zeneide, Ricardo Halfed, Ignácio Ricart, Paulo Padrão, Thales Silva, Marcel Mendes, Marcus Couto, Vinícius Ramos and Guilherme Avelino, for all friendship and companionship during the course.

To the great friends I met in college, all the companionship and support during my graduation and master's course; Henrique Maia, Rafael Prallon, Gabriel De Jong, Juliana Christina, Erick Gama, Yangye He and Karina Passos.

To my great friends Gustavo Amaral, Marcos Paulo, João Pedro, João Vitor, Vitor Silveira, Victoria Porto, Caio Julio, Eddie Nery and Ana Luiz; companions of so many adventures, trips and animated conversations.

Finally, I would like to thank all the brazilian people, whose collective efforts finance the institution of excellence that is UFRJ.

Agradecimentos

Agradeço primeiro a Deus Pai Todo Poderoso, que em Seu infinito amor, criou-me e me capacitou a elaborar e concluir esta dissertação. Agradeço também a Nossa Senhora, Mãe de Deus e a São José, seu esposo, por rogarem por mim nos momentos mais difíceis.

Faltam-me palavras para agradecer a meus pais, José e Elisiane, por todo amor, carinho, paciência, dedicação e apoio durante toda a minha vida. Não só me trouxeram ao mundo como ensinaram a maioria das coisas que me fazem ser quem sou. Sem todo o apoio e sem seus ensinamentos, esta dissertação jamais existiria.

A meu irmão Lucas, por toda amizade e todo companheirismo ao longo de nossas vidas; tantas aventuras vivemos juntos! Agradeço ainda pela paciência nos meus momentos de mau humor e por todo o apoio ao longo deste mestrado.

Aos meus avós, Maria (*in memoriam*), José, Elizabeth e Antônio; por todo carinho, lições de vida e tantos bons momentos que jamais hei de esquecer.

A José Gomes Pinto, nosso tão querido Dedé, a quem considero em grande estima por todo o carinho e dedicação dedicados à nossa família; que fazem os dias que passamos em Fortaleza mais felizes.

Aos meus tios e tias, Maricarmem, Gentil, Christine, Antônio e Vera, por todo carinho e apoio ao longo da vida; bem como aos meus primos e primas Felipe, Daniela, André, Lilian, Tiago e Bárbara por toda amizade.

Ao meu cachorro, Loup, por me animar quando as coisas ficavam difíceis, com sua companhia e sua devoção silenciosas.

A todos os meus professores, que ensinaram, cada um, uma parte de tu quanto sei; sem os quais esta dissertação jamais teria sido feita.

Aos meus orientadores, Lilian Kawakami Carvalho e João Carlos dos Santos Basilio, que por tanto tempo se dedicaram a me apoiar e a motivar com sua atenção e amizade; além de contribuírem enormemente para a realização deste trabalho.

Aos colegas e amigos do Laboratório de Controle e Automação, Marcos Vinícius e Gustavo Vianna; cuja ajuda foi indispensável à conclusão desta dissertação, Raphael Barcelos, Thiago Tuxi, Weslley Silva, Públio Macedo, José Villardi, Ricardo Macedo, Juliano Freire e Jéssica Vieira.

Aos colegas do mestrado, Victor Hugo, Pâmela Siekmann, Gabriel Solino, Maria

Zeneide, Ricardo Halfed, Ignácio Ricart, Paulo Padrão, Thales Silva, Marcel Mendes, Marcus Couto, Vinícius Ramos e Guilherme Avelino, por toda amizade e companheirismo durante o curso.

Aos grandes amigos que fiz na faculdade, por toda a companhia e apoio durante a graduação e o mestrado; Henrique Maia, Rafael Prallon, Gabriel De Jong, Juliana Christina, Erick Gama, Yangye He e Karina Passos.

Aos bons amigos Gustavo Amaral, Marcos Paulo, João Pedro, João Vitor, Vitor Silveira, Victória Porto, Caio Julio, Eddie Nery e Ana Luiz; companheiros de tantas aventuras, viagens e conversas animadas.

Por fim, gostaria de agradecer a todo o povo brasileiro, cujos esforços coletivos custeiam a instituição de excelência que é a UFRJ.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA ARQUITETURA DE NAVEGAÇÃO BASEADA EM CONTROLE SUPERVISÓRIO PARA ROBÔS EM AMBIENTES DA INDÚSTRIA 4.0

Antonio Galiza Cerdeira Gonzalez

Abril/2019

Orientadores: João Carlos dos Santos Basilio
Lilian Kawakami Carvalho

Programa: Engenharia Elétrica

Esta dissertação apresenta uma nova arquitetura geral para a navegação de robôs móveis em ambientes da Indústria 4.0, na qual o comportamento em malha aberta do robô e as especificações para o sistema são baseadas em autômatos. Um supervisor modular, que é a conjunção de dois supervisores é proposto: o primeiro, que garante que o robô siga o caminho definido por um planejador e o segundo, que assegura que as especificações, tais como prevenção de colisões, cumprimento de tarefas e gerenciamento de movimentos. A arquitetura de navegação proposta permite uma implementação descentralizada, na qual o supervisor modular é embarcado no robô móvel, enquanto o planejador pode ser executado em um agente externo. Tal característica torna mais fácil a adaptação da arquitetura proposta para diversos ambientes. O funcionamento da arquitetura proposta nesta dissertação é demonstrado por meio de uma simulação em um ambiente hipotético com as características de uma fábrica inteligente. Além disso, é também apresentada uma adaptação da arquitetura para a operação de múltiplos robôs, tendo sido realizadas também neste caso simulações.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A SUPERVISORY CONTROL-BASED NAVIGATION ARCHITECTURE FOR AUTONOMOUS ROBOTS IN INDUSTRY 4.0 ENVIRONMENTS

Antonio Galiza Cerdeira Gonzalez

April/2019

Advisors: João Carlos dos Santos Basilio
Lilian Kawakami Carvalho

Department: Electrical Engineering

In this dissertation, a general architecture for mobile robot navigation in Industry 4.0 environments in which the open-loop behavior of the robot and the specifications are based on automata is presented. A modular supervisory control structure, which is the conjunction of two supervisors is proposed: the first one, that enforces the robot to follow the path defined by a planner, and the second one that guarantees the satisfaction of the specifications such as prevention of collisions, task and movement management. The proposed navigation architecture allows decentralized implementation, in which the modular supervisor is embedded in the mobile robot, whereas the planner can run in external agent. Such a feature makes the adaptation of the proposed navigation architecture to different environments easy. The navigation architecture proposed in this dissertation is illustrated by means of a simulation in a hypothetical environment that resembles a smart factory. An adaptation for multiple robots is also presented, also validated by simulations.

Contents

List of Figures xiii

List of Tables xvi

List of Symbols xvii

1 Introduction 1

1.1 Objective 2

1.2 Related Works 2

1.3 Structure of the dissertation 3

2 Theoretical Background 4

2.1 Discrete events models 4

2.2 Languages 5

2.2.1 Language operations 6

2.2.2 Language representations 9

2.3 Automata 9

2.3.1 Languages represented by automata 10

2.4 Operations with automata 11

2.4.1 Unary operations 12

2.4.2 Composition operations 13

2.5 Supervisory Control of Discrete Event Systems with partial controllability and full observation 17

2.5.1 Supervisory control problem 19

2.5.2 Control with partial controllability and full observation 20

2.5.3 Obtaining a supervisor 23

2.6 Supervisory control with modular specifications 25

2.7 Dijkstra's algorithm 29

3 A New Supervisory-control-based-framework for Robot Navigation 34

3.1 Introduction 34

3.2 System models . . . 35
3.2.1 The environment automaton model G_e . . . 35
3.2.2 The robot model G_r . . . 36
3.3 A DES-based robot navigation architecture . . . 38
3.3.1 The navigation architecture . . . 39
3.3.2 Path planning procedure . . . 41
3.3.3 Design of modular supervisor $S_{r_1} \wedge S_{r_2}$ . . . 41
3.4 Performance analysis of the planner algorithm . . . 47
3.4.1 Scalability Analysis . . . 47
3.4.2 Time complexity analysis . . . 49
3.5 Experimental results for a single robot . . . 51
3.5.1 Simulation Results . . . 51
3.6 Navigation of multiple robots . . . 57
3.6.1 Motivating examples . . . 58
3.6.2 A new navigation architecture for multiple robots . . . 62

4 Conclusions and Future Works 70

Bibliographic References 74

A Tables of Example 2.12 Automata 78

List of Figures

2.1 State transition diagram of the automanton G. 10
2.2 State transition diagram of the automanton G. 11
2.3 State transition diagram of the automata G_1 e G_2. 14
2.4 State transition diagram of the automaton $G_1 \times G_2$. 15
2.5 State transition diagram of the automaton $G_1||G_2$. 17
2.6 Event set Σ of an automaton G, with its observable, unobservable, controllable and uncontrollable subsets Σ_o, Σ_{uo}, Σ_c and Σ_{uc}; respectively. 18
2.7 The feedback loop structure of supervisory control. 18
2.9 Automaton H_{spec} that models the given specifications. 20
2.10 Automaton H_a that models the desired behavior of the compensated system. 20
2.11 Automaton H_a that models the behavior of the compensated system, with the transition that supervisor S tries to disable displayed in red. 22
2.12 The standard realization automaton R for the supervisor S of example 2.7. 24
2.13 A possible non-reduced realization for supervisor S. 24
2.14 A reduced-state realization for the supervisor S 25
2.15 Modular control architecture with two supervisors. 26
2.16 A simple processing plant, consisting of a machine M1, two robot arms Arm 1 and Arm 2. 26
2.17 Automaton M_1 that models the behavior of processing machine $M1$. . 27
2.18 Automaton Arm_1 that models the behavior of robot arm Arm 1. . . . 27
2.19 Automaton Arm_2 that models the behavior of robot arm Arm 2. . . . 28
2.20 Automaton H_{spec} that models the specification that forbids collisions between the robot arms. 28
2.21 Automaton $Hspec_{Buffer}$ that models the buffer specifications for the system. 28
2.22 Graph G. 30

2.23 Step by step solution of a shortest path problem with Dijkstra's algorithm . . . 31

3.1 The robot movement module G_{r_m}. Dashed lines represent transitions labeled with uncontrollable events. . . . 37
3.2 The robot sensing module G_{r_s}. Dashed lines represent transitions labeled with uncontrollable events. . . . 37
3.3 The robot task manager module $G_{r_{tm}}$. Dashed lines represent transitions labeled with uncontrollable events. . . . 38
3.4 The proposed navigation architecture. . . . 40
3.5 Automaton $H_{spec,1}$ used to synthesize supervisor S_{r_1}. . . . 42
3.6 Automaton $H_{spec,2}$ used to synthesize supervisor S_{r_2}. Dashed lines represent transitions labeled with uncontrollable events. . . . 44
3.7 Automaton $H_{spec,3}$ used to synthesize supervisor S_{r_2}. Dashed lines represent transitions labeled with uncontrollable events. . . . 45
3.8 Closed-loop behavior for robot trajectory $\sigma_1\sigma_2\ldots\sigma_f \in L(G_e)$. Dashed lines represent transition labeled with uncontrollable events. . . 48
3.9 Times taken to perform the path planning versus the size of the environment state space for Groups 2 and 3 51
3.10 Times taken to perform the path planning versus the number of target states for Groups 4 and 5 52
3.11 Map of the environment. The arrows represent the possible robot poses (states of automaton G_e): the tail indicates the positional coordinate and the direction corresponds to the robot pose. . . . 53
3.12 Simulation results: Type 1 task from pose M_2 to $X_{goal} = \{M_5, M_6\}$ without obstacle (a), and with obstacle (b), and, Type 2 task from pose M_1 to $X_{goal} = \{M_3, M_6\}$ without obstacle (c), and with obstacle (d). . . . 56
3.13 Problems that may arise when automaton G_e that models the environment has states that are reached by a single transition: when a robot tries to reach $M1$ while another robot is using it, (a), and when a robot tries to reach $M1$ while another tries to leave it(b) 59
3.14 Map of the environment. The arrows represent the possible robot poses (states of automaton G_e): the tail indicates the positional coordinate and the direction corresponds to the robot pose. . . . 60
3.15 Another problem that may arise when the automaton G_e has corridors that allow robots to pass in both directions by its corridors. . . . 61
3.16 The modified architecture, adapted for the navigation of n robots. . . 63

3.17 The modified robot sensing module $G_{r_s,i}$. Dashed lines represent transitions labeled with uncontrollable events. 64
3.18 Automaton $H_{spec,3,i}$ used to synthesize supervisor S_{r_2} for the navigation of multiple robots. Dashed lines represent transitions labeled with uncontrollable events. 64
3.19 Results of the test with two robots.. 66
3.20 Results of the test with three robots. 67
3.21 Results of the test with four robots. 68

List of Tables

3.1 Robot movement module events. 37
3.2 Robot sensing module events. 38
3.3 Robot task manager module events. 39
3.4 Summary of the results for time complexity analysis. 50
3.5 Parameters of the test groups. 50
3.6 Environment automaton events Σ_e. 54
3.7 Active events of the states of G_e. 55
3.8 Environment automaton events Σ_e. 59
3.9 Active events of the states of G_e. 61
3.10 Events that are modified. 63

A.1 Transition function of G_{system} 78
A.2 Transition function of supervisor S_1 79
A.3 Transition function of supervisor S_2 80

List of Symbols

$(.)^*$	Kleene-closure operator, p. 6
$(.)^C$	Complement of a set operator, p. 6
$Ac(G)$	Accessible part operation over automaton G, p. 12
$CoAc(G)$	Coaccessible part operation over automaton G, p. 12
$G_1 \times G_2$	Product composition between automata G_1 and G_2, p. 13
$L(G)$	Generated language of automaton G, p. 11
$L_m(G)$	Marked language of automaton G, p. 11
P	Natural projection operation, p. 8
P^{-1}	Inverse projection operation, p. 8
S/G	Supervisor S controlling system G, p. 18
$Trim(G)$	Trim operation over automaton G, p. 13
X	Set of states of an automaton, p. 10
X_m	Set of marked states of an automaton, p. 10
$\Gamma(x)$	Active event function of an automaton, p. 10
Σ	Event set of a discrete event system, p. 5
Σ_c	Controllable event set of a DES, p. 17
Σ_o	Observable event set of a DES, p. 17
Σ_{uc}	Uncontrollable event set of a DES, p. 17
Σ_{uo}	Unbservable event set of a DES, p. 17
$\setminus$	Subtraction of sets operator, p. 6

$\cap$	Intersection of sets operator, p. 6
$\cup$	Union of sets operator, p. 6
$\dot{\cup}$	Partition, p. 17
$\overline{(.)}$	Prefix-closure operator, p. 7
ε	Empty string, p. 6
$\wedge$	Logic operator and, p. 26
$f(x, s)$	Transition function of an automaton, p. 10
x_0	Initial state of an automaton, p. 10

Chapter 1

Introduction

In recent years, new challenges to make production processes more efficient, autonomous and customizable have led to a new industrial revolution. A new concept of industry, called Industry 4.0 [1], has emerged and is currently adopted to denominate the current trend of automation and data exchange in manufacturing technologies by creating a “smart factory” [2]. The fundamentals of Industry 4.0 are Cyber-Physical systems [3], Internet of things [1, 4], cloud computing, big data [5], and mobile robots [6].

In this dissertation, we consider factories with smart machines that are part of a distributed production line that requires a mobile robot in order to carry the parts they produce either to store or to take them to another machine for further processing. The robot is connected remotely to computer systems. Our work focuses on providing intelligence to the mobile robot so it can correctly establish the connection between the machines. To this end, we propose a navigation architecture based on modular supervisory control that, besides establishing the connection between the machines, also performs the supervision of the robot navigation.

Autonomous mobile robot navigation consists of four stages: mapping, localization, planning and execution [7]. We consider industrial environments where the structure rarely undergoes major modification, and thus it is reasonable to assume that the environment map is known *a priori*. Since there exist several techniques to deal with the robot localization, we will not address this issue in this dissertation, focusing only on planning and execution. Finally, regarding the navigation architecture classification usually deployed in the literature [8], we mention that the architecture used in this dissertation can be regarded as hybrid, being predominantly deliberative with reactive elements to deal with obstacles and sudden changes in the environment where robot navigation takes place.

1.1 Objective

The objective of this dissertation is to propose a supervisory control approach for mobile robot navigation in industrial environments, such as warehouses and smart factories. We model the environment, the planning structure, and the robot as automata and use modular supervisory control theory [9] to develop a navigation system for mobile robots. The modular supervisory controller ensures the correct navigation of the robot in the presence of unpredictable obstacles and is obtained by the conjunction of two supervisors: a first one that enforces the robot to follow the path defined by the planner and a second one that imposes other specifications such as prevention of collisions, task and movement management, and distinction between permanent and intermittent obstacles. The idea is to develop a general approach that allows the implementation of specifications by means of modules that depend on the task the robot will perform and on the industrial environment. As will become clear, when the environment is changed, the only modifications needed to adapt the previous supervisor design are those related to the new event set associated with the new environment. The main results of this dissertation were published in [10].

1.2 Related Works

Discrete Event Systems (DES) are dynamical systems whose behaviors are determined by the asynchronous occurrence of certain events [11]. Several important practical problems have been addressed using DES theory, ranging from theoretical issues, such as fault diagnosis [12] and supervisory control theory [13], to real systems applications [14–16]. The DES formalism has also been proved suitable to deal with mobile robot navigation [17–23]. In [17], a DES-based supervisory controller that ensures collision and deadlock avoidance for a group of robots that work in order to concurrently accomplish their missions in a 2D space is proposed. However, it is not clear in [17] how to plan and execute the movements necessary to reach the goal location. In [18], the path planning problem to decentralized systems — not necessarily robot path planning — with action costs was addressed by using several weighted automata. Although such an approach could be adapted to compute robot trajectories for the case of several cooperative robots, it is not suitable when only one robot is being considered, since the parameters are assumed to be distributed. The proposed approach also does not apply to the adaptation of the single robot navigation architecture, as their actions are not cooperative. In [19], a formal method based on Linear Temporal Logic (LTL) has been employed to describe and model specifications in mobile robot navigation. However, the disadvantage of that approach is the computational effort to convert such logics into Büchi automata, which is not required in our work. In [20], the authors outlined an integration between graph theory, automata, and Z notation in order to propose a supervisory control design framework for robot navigation systems. Structures for representing the environment and some specifications were presented, but further investigation is still needed in order to use the proposed approach to trajectory planning and execution. In [21], automaton-based models for mobile robot navigation were used to show the viability of using DES theory in the modeling, analysis, and synthesis of behaviors applied to the navigation of a mobile robot using visually guided navigation in an unknown or partially known environment. Since the focus was on execution rather than on planning, the optimal path to reach the goal location was not computed. A formal synthesis of supervisory control software for multiple robot systems was developed in [22] and subsequently in [23], where scalability was improved. Since the main goal was to manage task planning, the problem considered in [22] and [23] is different from one addressed here.

More recently, an online supervisory control approach based on limited lookahead policy was presented for the control of multi-agent discrete-event systems. The proposed online control architecture is also applied to model and control a warehouse automation system served by multiple mobile robots; validating its effectiveness with a case study and with an experiment with multiple robots. This work is similar to ours regarding the use of automata models and supervisory control, although it is online and it is not modular. All path planing is done externally, *i.e*, in a server, instead of leaving it to the robots, and it is not cooperative. It is not clear which algorithm was used to calculate the shortest path, as it yields multiple best pathes, although it apparently does not take the robot turns into account. In addition, the possibility of unknown obstacles is not considered in [24]; and it also does not guarantee that the resulting behavior is nonblocking,

1.3 Structure of the dissertation

In Chapter 2, we review some basic concepts on DES, such as language, automata and supervisory control. Chapter 3 is the main part of this dissertation, where we present Dijkstra's algorithm for optimal pathfinding , automaton-based models for the environment, planning structures and robot motion. We present a formulation for the robot navigation problem, and propose a navigation architecture that is based on modular supervisory control, and on a planner algoritm. We carry a performance analysis of the proposed planner algorithm and the work results by means of a simulation. We also present an adaptation of the developed architecture to multiple robots navigation, and the result of a few simulations. Finally, in Chapter 4 we draw some conclusions and outline future research works.

Chapter 2

Theoretical Background

This chapter presents a brief review of the basic concepts of discrete event system theory. Section 2.1, presents the concept of discrete event systems. Next, Section 2.2 presents some concepts on language theory, with the objective of appplying that theory in the description of the logical behavior of discrete event systems. Sections 2.3 and 2.4 are dedicated to the description of automata theory, essential to the understanding of this dissertation. Section 2.5 presents the theory behind monolithic supervisory control, addressing various cases when all events are controllable and observable and when that is not the case. The procedure for obtaining an observer automaton is also described. Section 2.6 presents the theory of supervisory control, and Section 2.7 describes Dijkstra's Algorithm, which will be necessary to obtain the shortest path for the robot navigation.

2.1 Discrete events models

From the earliest days, humanity has developed several models to explain the most varied range of phenomena encountered along history, rangin from legends to complex mathematical system of equations. One of the most basic and essential concepts to describe phenomena is the concept of systems, in which the scope of the model is limited to a certain space of interest. The concept of system can be defined in several ways, like the intuitive definition given in [25], which describes systems as a combination of parts that act together and perform objectives, physical or not.

In Engineering, when studying systems, a model intends to capture the relationship between a set of measurable variables, which may be classified as input variables or output variables. Input variables are, in general, grouped in a column vector $u(t) = [u_1(t), u_2(t), \dots, u_n(t)]^T$, and may be directly manipulated. Output variables may also be grouped in a column vector $y(t) = [y_1(t), y_2(t), \dots, y_n(t)]^T$, but they can't be manipulated, only measured.

A system whose output variables $y(t)$ at an instant t depends only on input $u(t)$

is called static, while those systems whose outputs $y(t)$ depend both on input $u(t)$ and on input values before t are called dynamic systems. The information of the past behavior of a system necessary to determine its output at every instant is represented by the concept of state. A more precise definition of state of a dynamic system is the minimum set of variables $x_n(t)$, $n = 1, 2, \ldots$, whose knowledge, together with the knowledge of the input $u(t), \forall t \geq t_0$ is known, determine the behavior of the system for all $t \geq t_0$. The space of all possibles states $x(t) = [x_1(t), x_2(t), \ldots, x_n(t)]$ that a system can assume is denoted by X, and is called the state space of the system.

Another way to classify systems is based on the characteristics of the space state X. If X is a continuous n-dimensional vector of either real or complex numbers, the system is a continuous-state system. If X is a discrete set, than the system is a discrete-state system. In addition, the model of a system can be classified as continuous-time, if the state variables of the system are continuous-time functions, or as discrete-time, if the state variables of the system are discrete-time functions, meaning they are defined only at discrete instants in time.

Some discrete-state systems have their state transitions associated with events, which occur asynchronously in time. An event may be seen as an instantaneous occurrence that makes the system transition from a state value to another. They are associated with some sort of occurrence, be it an specific action, like the pressing of a switch, or some spontaneous occurrence, like the detection of an object by a sensor. That type of system, whose transitions are driven by the occurrence of events is called discrete event systems.

Definition 2.1 (Discrete Event System [11]) *A discrete event system (DES) is a dynamic system whose state space is discrete; and whose state evolution is driven by the occurrence of asynchronous events.*

2.2 Languages

The state evolution of discrete event systems is driven by the occurrence of events; so, every discrete event systems has a set Σ of events that may occur. The event set Σ may be thought of as an alphabet; thus, sequences of events are thought of as words. A collection of finite-length words is called a language. Thus, the behavior of a system may be described by the states visited by the system and the sequence of events that induced those transitions. The set of all words, also called strings, generated by a system defines its language, which is formally defined as

Definition 2.2 (Language [11]) *A language defined over a finite event set Σ is a set of finite-length strings formed with events in Σ.*

The basic operation on events is concatenation, which links events in a series, forming strings. It also joins strings, creating strings of larger length. Thus, concatenation is essential to the creation of languages. For example, the string $s = bat$ is formed by the concatenation of the string ba with event t; while string ba is formed by the concatenation of events b and a. The identity element of concatenation is ε, which is called the empty string; so, any string concatenated with ε is the string itself, *i.e*, $s\varepsilon = \varepsilon s = s$.

The set of all finite length strings that can be created from an event set Σ is denoted by Σ^*, including the empty string ε. The $(.)^*$ operation is called the Kleene-closure. For example, if $\Sigma = \{a, e\}$, then $\Sigma^* = \{\varepsilon, a, e, aa, ae, ee, ea, aae, aaa, aea, aee, eea, eae, eaa, \ldots\}$

In order to adequately present operations on languages, some concepts and terminologies about strings must be known. First, for a string $s = abc$, where $a, b, c \in \Sigma$, we say that a, b, c, ab and bc are substrings of s. Substrings a and ab are prefixes of s, while c and bc are sufixes of s. Also, ε and s itself are prefixes and suffixes of s. The length of a string s is denoted by $||s||$. It must be noted that the empty string ε has length equal to 0.

2.2.1 Language operations

Languages are sets of strings, and so, all set operations such as union, intersection, difference and complement can be applied to languages in a straightforward way, as in the following example.

Example 2.1 *Given an event set $\Sigma = \{a, e, i, o, u\}$, and the languages $L_1 = \{a, aaa, e, eio, o, au, eu, iu\}$ and $L_2 = \{\varepsilon, aaa, eio, iu, aeiou, aio\}$ defined over Σ, the union, the intersection, the difference and the complement with respect to Σ^* are:*

$$L_1 \cup L_2 = \{\varepsilon, a, aaa, e, eio, o, au, eu, iu, aeiou, aio\},$$

$$L_1 \cap L_2 = \{aaa, eio, iu\},$$

$$L_1 \backslash L_2 = \{a, e, o, au, eu\},$$

$$L_1^C = \Sigma^* \backslash L_1,$$

There are several other operations on languages, which are frequently used in DES theory. Those of utmost interest are: concatenation, Kleene-closure, prefix-closure, natural projection and inverse projection.

Concatenation

Concatenation is not only defined for events and strings, but also for languages. The idea is to concatenate every string of a language to every string of the other. A more formal definition of the concatenation between two languages L_1 and L_2 is:

$$L_1 L_2 = \{s \in \Sigma^* : (\exists (s_1, s_2) \in L_1 \times L_2)[s = s_1 s_2]\}. \tag{2.1}$$

Kleene-closure

Let $L \subseteq \Sigma^*$; the Kleene-closure of L is given by the language

$$L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \cup \ldots \tag{2.2}$$

Prefix-closure

Let $L \subseteq \Sigma^*$, then, the prefix-closure of L is defined as:

$$\bar{L} = \{s \in \Sigma^* : (\exists t \in \Sigma^*)[st \in L]\}. \tag{2.3}$$

Notice that $\varepsilon \in \bar{L}$, because for any string s in L, $\varepsilon s = s$.
A language L such that $L = \overline{L}$ is said to be prefix-closed.

Example 2.2 *Consider an event set $\Sigma = \{a, b, c\}$, and languages $L_1 = \{\varepsilon, a, b, c\}$, $L_2 = \{bc, aa, cb\}$ and $L_3 = \{a, bb, ca\}$ defined over Σ. The concatenation $L_1 L_2$ is:*

$$L_1 L_2 = \{bc, aa, cb, abc, aaa, acb, bbc, baa, bcb, cbc, caa, ccb\}.$$

The Kleene-closure of L_3 is given by:

$$L_3^* = \{\varepsilon, a, bb, ca, aa, abb, aca, bba, bbbb, bbca, caa, cabb, caca, \ldots\},$$

and the prefix-closure of L_2 and L_3 are

$$\overline{L_2} = \{\varepsilon, b, bc, a, aa, c, cb\},$$

$$\overline{L_3} = \{\varepsilon, a, b, bb, c, ca\}.$$

Natural Projection

Natural projection, sometimes referred just as projection, is an operation performed on strings and languages, projecting them from a set of events Σ_l into a smaller set of events Σ_s, where $\Sigma_s \subseteq \Sigma_l$. An intuitive definition of projection is that it is the operation through which we remove from a string events that are in Σ_l,but not in Σ_s. For languages, it is just a matter of applying the operation to every string in the language. A more precise definition, for strings, is given by:

$$P : \Sigma_l^* \to \Sigma_s^*.$$

where

$$P(\varepsilon) := \varepsilon,$$
$$P(\sigma) := \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_s, \\ \varepsilon, & \text{if } \sigma \in \Sigma_l \backslash \Sigma_s, \end{cases}$$
$$P(s\sigma) := P(s)P(\sigma) \text{ for } s \in \Sigma_l^*, \sigma \in \Sigma_l.$$

The natural projection, can be extended to languages by applying P to all strings in a language L, that is:

$$P(L) := \{t \in \Sigma_s^* : (\exists s \in L)[P(s) = t]\}.$$

Inverse Projection

The inverse projection is the inverse map defined by the natural projection:

$$P^{-1} : \Sigma_s^* \to 2^{\Sigma_l^*}.$$

The inverse projection is defined both for strings and languages; and can be intuitively described as the operation that fills a string with all possible strings that were possibly removed during a projection between the events of the string. For languages, it is just a matter of applying such operation to all strings in it. A more precise definition is given by:

for $L \subseteq \Sigma_s^*$

$$P^{-1}(L) := \{t \in \Sigma_l^* : (\exists t \in L)[P(s) = t]\}.$$

Example 2.3 *Given an event set $\Sigma_l = \{a, b, c\}$ and a smaller event set $\Sigma_s = \{a, b\}$, and the language $L_1 = \{a, bab, cab\}$ defined over Σ, the natural projection $P : \Sigma_l^* \to \Sigma_s^*$ when applied to L_1 is given by:*

$$P(L_1) = \{a, bab, ab\}$$

And the inverse projection of L_{1s} is given by:

$$P^{-1}(L_{1s}) = \{\{c\}^*a\{c\}^*, \{c\}^*b\{c\}^*a\{c\}^*b\{c\}^*, \{c\}^*a\{c\}^*b\{c\}^*\}.$$

From the above example, we can see that, for any given language L, $L \subseteq P^{-1}(P(L))$.

2.2.2 Language representations

As previously mentioned, languages are a formal way of describing the behavior of discrete event systems, *i.e.*, languages make possible to register all sequences of events a DES can generate. Nonetheless, languages are sometimes difficult to work with. For example, given $\Sigma = \{b, c, d\}$ and the languages $L_1 = \{\varepsilon, b, bcc\}$, $L_2 = \{$all possible strings of length 2 that begin with $b\}$ and $L_3 = \{$all finite length strings that start with $c\}$, it is possible to see that L_1 is quite easy to work with, since it contains only three strings. L_2, on the other hand, is described only to make it more concise, but is still possible to list all of its elements. However, L_3 is impossible to be fully enumerated, limiting its representation to a description.

Obtaining a concise and easy way to work with some representation of a language is a hard task and, sometimes, even impossible. So, to better work with languages, a set of more compact structures that define a language and which can be manipulated by well-defined mathematical operations is needed. Such structures and their operations already exist; and some of the most popular ones are Automata [11] and Petri Nets [26]. As for the scope of this dissertation, automata are the only formalism of interest. The next section will be dedicated to presenting some topics of automata theory.

2.3 Automata

Automata are devices capable of representing a language according to well defined rules, being capable of representing the behavior of a discrete event system. As an example, consider a DES whose event set is $\Sigma = \{a, b\}$ and whose state space is x_0 and x_1. State x_0 is also the initial state of the system, that is, the state where the system is after being turned on. State x_1 is the only marked state of the system, being, therefore a state of interest. When the state of the system is x_0, the occurrence of event a makes the system state change to x_1. When the state of the system is x_1, the occurrence of event b makes the system state change to x_0. A simple way to present all this information about the automaton is by considering its state transition diagram, as presented on Figure 2.1.

It is possible to see all information that is needed to create an automaton. Thus, it is possible to give a formal definition of deterministic automata, as follows.

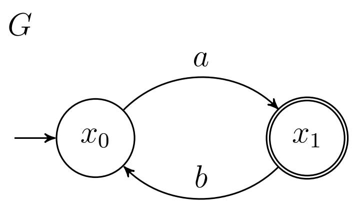


Figure 2.1: State transition diagram of the automanton G.

Definition 2.3 (Deterministic Automaton[11]) *A deterministic automaton, denoted by G, is a six-tuple*

$$G = (X, \Sigma, f, \Gamma, x_0, X_m),$$

where X is the set of states, Σ is the set of events, $f : X \times \Sigma \to X$ is the transition function, i.e., $f(x, \sigma) = y$, means that there exists a transition from state x to state y triggered by event σ, $\Gamma : X \to 2^E$ is the active event function, i.e., $\Gamma(x)$ is the set that contains all events σ for which $f(x, \sigma)$ is defined, x_0 is the initial state and $X_m \subseteq X$ is the set of marked states.

Some remarks must be made about Definition 2.3, so as to clarify a few points:

1. The deterministic automaton is said to be so because the transition function of the automaton is deterministic, in the sense that, for a certain x and a certain σ, $f(x, \sigma)$ is equal to a unique y. Automata where this is not the case, that is, $f(x, \sigma) \subseteq X$, are called non-deterministic,

2. The set of states X is, in general, finite. When this is the case, the deterministic automaton is called a deterministic finite-state automaton, DFA,

3. The inclusion of Γ in the definition of G is done because Γ makes it easier to define some operations, but it is redundant, i.e, all information about active events for a given state can be obtained directly from the transition function $f(x, \sigma)$,

4. The set of marked states X_m contains all states of some special interest for an automaton, but it may be empty, if all states are of same relevance. Also, it may be the case where $X_m = X$, which generally occur not to change the marking of another automaton.

2.3.1 Languages represented by automata

According to the previous subsection, automata are a more convenient way of representing language, as the languages they generate can be obtained directly from

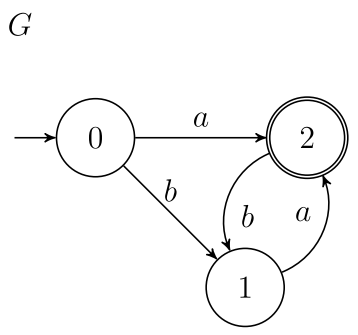


Figure 2.2: State transition diagram of the automanton G.

their state transition diagrams. The language generated by an automaton G, denoted as $L(G)$ can be obtained by verifying all directed paths that can be followed in the state transition diagram, beginning from the initial state. There is also another type of language generated by an automaton G: the marked language of G, denoted by $L_m(G)$. The generated language $L(G)$ contains all possible behaviors of the discrete event system modeled by G, whereas the marked language $L_m(G)$ represents all strings of $L(G)$ that leads the automaton from its initial state to marked states.

Definition 2.4 (Generated and marked languages [11]) *The language generated by an automaton $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ is:*

$$L(G) = \{s \in \Sigma^* : f(x_0, s) \text{ is defined } \},$$

The marked language of G is:

$$L_m(G) = \{s \in L(G) : f(x_0, s) \in X_m\}.$$

Example 2.4 *For automaton G, whose state transition diagram is represented on Figure 2.2, the generated language of G is $L(G) = \overline{\{a\}\{ba\}^* \cup \{b\}\{ab\}^*}$, and the marked language of G is $L_m(G) = \{a, ba\}\{ba\}^*$.*

2.4 Operations with automata

There are several operations on automata that modify their generated and marked languages and, accordingly, their state transition diagram. In addition, there are operations to combine automata, in such a way that the model of a complete system can be built from the models of its individual parts. This section is devoted to describing only the operations on automata that are of interest to this dissertation; more operations can be found on [11].

2.4.1 Unary operations

Unary operations are those operations that alter the state transition diagram of a single automaton, without changing its event set Σ.

Accessible Part

The Accessible part of an automaton is the set of states that can be reached from its initial state x_0 by some string in $L(G)$. The Accessible Part operation, denoted by $Ac(G)$, removes all unaccessible states from the automaton G, therefore, its generated and marked languages are not affected. When an unaccessible state is removed, all transitions attached to that state are also removed. Formally, the definition of $Ac(G)$ is as follows:

$$Ac(G) \quad := \quad (X_{ac}, \Sigma, f_{ac}, x_0, X_{ac,m}),$$

where,

$$X_{ac} = \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\},$$

$$X_{ac,m} = X_m \cap X_{ac},$$

$$f_{ac} = f|_{X_{ac} \times \Sigma \to X_{ac}}.$$

The notation $f|_{X_{ac} \times \Sigma \to X_{ac}}$ means the domain of f is being restricted to a smaller domain, the accessible states X_{ac}. If $Ac(G) = G$, the automaton G is said to be accessible.

Co-accessible Part

A state x of an automaton G is said to be co-accessible to X_m if there exists a path in the state transition diagram from the state x to any marked state. The operation which removes from G all states that are not co-accessible is the Co-accessible part, denoted by $CoAc(G)$. To obtain the co-accessible part of an automaton G it is necessary to build the following automaton:

$$CoAc(G) \quad := \quad (X_{coac}, \Sigma, f_{coac}, x_{0,coac}, X_m),$$

where

$$\begin{aligned} X_{coac} &= \{x \in X : (\exists s \in \Sigma^*)[f(x,s) \in X_m]\} \\ x_{0,coac} &= \begin{cases} x_0, & \text{if } x_0 \in X_{coac}, \\ \text{undefined}, & \text{otherwise}, \end{cases} \\ f_{coac} &= f|_{X_{coac} \times \Sigma \to X_{coac}}. \end{aligned}$$

It is worth noting that the co-accessible part operation can shrink the generated language of an automaton because an accessible state may not be co-accessible. Regarding the marked language, it is not affected by the co-accessible part, because all strings that belong to the marked language reach at least one marked state. If $CoAc(G) = G$, the automaton G is said to be co-accessible and has $L(G) = \overline{L_m(G)}$. Note that if $X_m = \emptyset$ for a given automaton G, $CoAc(G)$ will yield the empty automaton.

Trim Operation

A Trim automaton is both accessible and co-accessible; thus, the trim operation is:

$$Trim(G) = CoAc[Ac(G)] = Ac[CoAc(G)].$$

2.4.2 Composition operations

Composition operations involve two or more automata. For the scope of this work, the only ones that we need to define are the product and the parallel compositions.

Product

The product of two automata G_1 and G_2 is the automaton

$$G_1 \times G_2 := Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times2}, \Gamma_{1\times2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}),$$

where

$$f_{1\times2}((x_1,x_2),\sigma) := \begin{cases} (f_1(x_1,\sigma), f_2(x_2,\sigma)) & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ \text{undefined}, & \text{otherwise;} \end{cases}$$

thus,

$$\Gamma_{1\times2}(x_1,x_2) = \Gamma_1(x_1) \cap \Gamma_1(x_2)$$

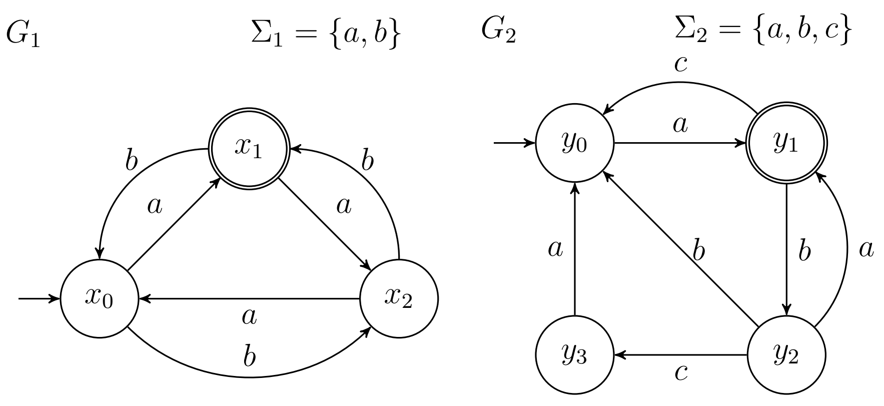


Figure 2.3: State transition diagram of the automata G_1 e G_2.

The Product composition completly synchronizes the behavior of both automata involved, whose state transitions are triggered by the events in $\Gamma_1(\Sigma_1) \cap \Gamma_2(\Sigma_2)$. For this reason, the product is also called the completely synchronous composition. The states of the resulting automata are denoted by tuples of the form (x_1, x_2), where x_n, $n = 1, 2$, is the current state of automaton G_n. The generated and marked languages of $G_1 \times G_2$ are given by:

$$\begin{aligned} L(G_1 \times G_2) &= L(G_1) \cap L(G_2), \\ L_m(G_1 \times G_2) &= L_m(G_1) \cap L_m(G_2). \end{aligned}$$

Notice that when $\Sigma_1 \cap \Sigma_2 = \emptyset$, $G_1 \times G_2$ yields the empty automaton. The product between two automata may be extended to several automata as follows:

$$G_1 \times G_2 \times \ldots \times G_n = G_1 \times (G_2 \times (\ldots \times (G_{n-1} \times G_n)));$$

because the product operation is associative. Also, the product operation is commutative, that is

$$G_1 \times G_2 \times G_3 = G_1 \times (G_2 \times G_3) = G_2 \times (G_1 \times G_3);$$

Example 2.5 *In order to compute the product between automata G_1 and G_2, shown in Figure 2.3,*

We start by computing the initial state of automaton $G_1 \times G_2$, which is done by joining the initial states of G_1 and G_2 in a tuple, giving the initial state (x_0, y_0). Next, we need to check which events are in $\Gamma_1(x_0) \cap \Gamma_2(y_0)$. As event a is the only event in $\Gamma_1(x_0) \cap \Gamma_2(y_0)$, the next state is given by $(f_1(x_0, a), f_2(y_0, a)) = (x_1, y_1)$. Continuing the process, as $\Gamma_1(x_1) \cap \Gamma_2(y_1) = \{b\}$, the next state we obtain is $(f_1(x_0, b), f_2(y_0, b)) = (x_0, y_2)$. Now, as $\Gamma_1(x_0) \cap \Gamma_2(y_2) = \{a, b\}$, states $(f_1(x_0, a), f_2(y_2, a)) = (x_1, y_1)$ and $(f_1(x_0, b), f_2(y_2, b)) = (x_2, y_0)$ are reached. We continue this procedure until no new states can be createded, leading to automaton $G_1 \times G_2$, shown in Figure 2.4.

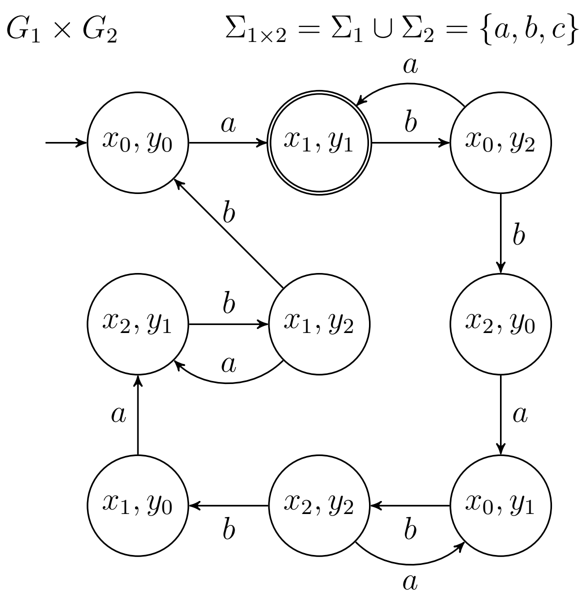


Figure 2.4: State transition diagram of the automaton $G_1 \times G_2$.

Parallel Composition

While the product between two automata is quite restrictive, *i.e.*, it blocks the occurrence of events that are not common to the set of current active events of the automata, the Parallel composition is less restrictive, since it allows the occurrence of private events, *i.e.*, events that belong exclusively to one automaton, and only permits the occurrence of common events when they are both active in the current states of all involved automata. Therefore, the parallel composition of two automata G_1 and G_2 is defined as follows:

$$G_1||G_2 := Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1||2}, \Gamma_{1||2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}),$$

where

$$f_{1||2}((x_1, x_2), \sigma) := \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ (f_1(x_1, \sigma), x_2), & \text{if } \sigma \in \Gamma_1(x_1) \backslash \Sigma_2, \\ (x_1, f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_2(x_2) \backslash \Sigma_1, \\ \text{undefined}, & \text{otherwise}, \end{cases}$$

thus,

$$\Gamma_{1||2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \backslash \Sigma_2] \cup [\Gamma_2(x_2) \backslash \Sigma_1].$$

.

Notice that, once again, an event $\sigma \in \Sigma_1 \cap \Sigma_2$ can be executed in the state (x_1, x_2) if and only if $\sigma \in \Gamma_1(x_1)$ and $\sigma \in \Gamma_2(x_2)$. Thus, the resulting automata is synchronized on the events that are common to the automata, while private events $\sigma \in (\Sigma_1 \backslash \Sigma_2) \cup (\Sigma_2 \backslash \Sigma_1)$ may occur if either $\sigma \in \Gamma_1(x_1)$ or $\sigma \in \Gamma_2(x_2)$. It is worth remarking that, if $\Sigma_1 = \Sigma_2$, the parallel composition yields the same result as the product, since there are no private events.

The generated language $L(G_1||G_2)$ and the marked language $L_m(G_1||G_2)$, are defined using projection from the larger set $\Sigma_1 \cup \Sigma_2$ into a smaller set Σ_1 or Σ_2. Thus, the two needed projections are

$$P_1 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_1^*;$$
$$P_2 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_2^*.$$

Using these projections, the generated and marked languages of $G_1||G_2$ are given by

$$L(G_1||G_2) = P_1^{-1}[L(G_1)] \cap P_2^{-1}[L(G_2)];$$
$$L_m(G_1||G_2) = P_1^{-1}[L_m(G_1)] \cap P_2^{-1}[L_m(G_2)].$$

As it was the case for the product, the parallel composition of more than two automata can be thought of as:

$$G_1||G_2|| \ldots ||G_{n-1}||G_n = G_1||(G_2||(\ldots||(G_{n-1}||G_n));$$

since the paralel composition is associative. It is also comutative, that is

$$G_1||G_2||G_3 = G_1||(G_2||G_3) = G_2||(G_1||G_3) = G_3||(G_1||G_2);$$

Example 2.6 *We want now to compute the parallel composition of automata G_1 and G_2, shown on Figure 2.3, we first compute the initial state of $G_1||G_2$, which is obtained by coupling the initial states of G_1 and G_2 in a tuple, giving us (x_0, y_0). Next, we need to find the events that are in $[\Gamma_1(x_0) \cap \Gamma_2(y_0)] \cup [\Gamma_1 \cap (\Sigma_1(x_0)/\Sigma_2)] \cup [\Gamma_2(y_0) \cap (\Sigma_2/\Sigma_1)]$, that is, which events are either active common events or active particular events. For (x_0, y_0), there is only one active event, a, and thus, the next state is $(f_1(x_0, a), f_2(y_0, a)) = (x_1, y_1)$.*

For (x_1, y_1), $[\Gamma_1(x_1) \cap \Gamma_2(y_1)] \cup [\Gamma_1 \cap (\Sigma_1(x_1) \backslash \Sigma_2)] \cup [\Gamma_2(y_1) \cap (\Sigma_2 \backslash \Sigma_1)] = \{b\}$. The next state is, then, (x_0, y_2), whose active event set is $[\Gamma_1(x_1) \cap \Gamma_2(y_1)] \cup [\Gamma_1 \cap (\Sigma_1(x_1)/\Sigma_2)] \cup [\Gamma_2(y_1) \cap (\Sigma_2 \backslash \Sigma_1)] = \{a, b, c\}$. The transitions triggered by these events reach reach states $(f_1(x_0, a), f_2(y_2, a)) = (x_1, y_1)$, $(f_1(x_0, b), f_2(y_2, b)) = (x_2, y_0)$ and $(f_1(x_0, c), f_2(y_2, c)) = (x_0, y_3)$.

Since (x_2, y_0) and (x_0, y_3) are new states, the procedure performed described must be done again, until no new states can be created. Proceeding in this way we obtain

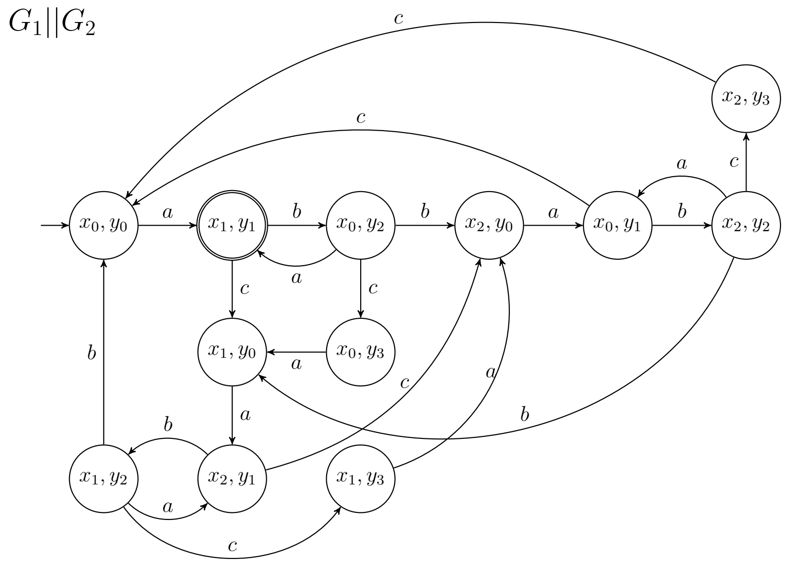


Figure 2.5: State transition diagram of the automaton $G_1||G_2$.

automaton $G_1||G_2$, shown in Figure 2.5.

With all previously described operations known, it is now possible to study and understand supervisory control theory.

2.5 Supervisory Control of Discrete Event Systems with partial controllability and full observation

Although some systems may behave exactly as desired just by themselves, that does not happen for most of them, so, feedback is needed to modify the behavior of systems. For example, a robot arm cannot move itself in a desired trajectory without a controller commanding the actuators of the arm.

Supervisory control has the same idea, *i.e*, to modify the behavior of a system modeled by an automaton G. To this end, the event set of G is partitioned as $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, where Σ_c and Σ_{uc} are the sets of controllable and of uncontrollable events, respectively. We say that a controllable event is an event whose occurrence may be disabled by the supervisor S, as opposed to uncontrollable events, which cannot be disabled.

The event set of G can also be partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o and Σ_{uo} are the sets of observable and unobservable events, respectively. Observable events are those events whose occurrences are detected by the supervisor S, while the occurrence of unobservable events are not.

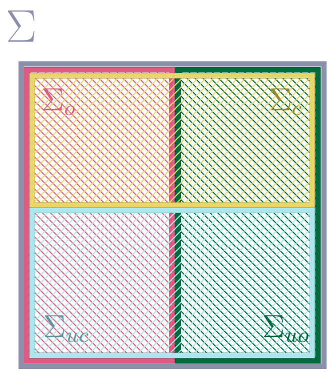


Figure 2.6: Event set Σ of an automaton G, with its observable, unobservable, controllable and uncontrollable subsets Σ_o, Σ_{uo}, Σ_c and Σ_{uc}; respectively.

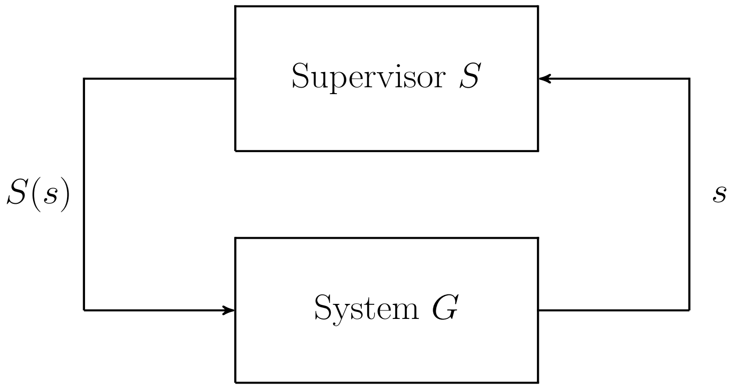
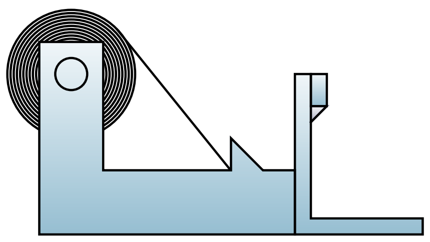


Figure 2.7: The feedback loop structure of supervisory control.

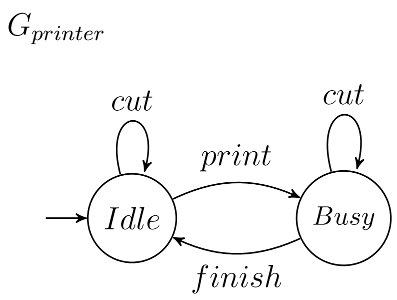
Controllable and Uncontrollable events may be either observable or unobservable. This way, the partition of the event set Σ of an automaton G can be performed as illustrated by Figure 2.6.

The specifications which the supervisor will try to enforce on the system are represented by an admissible language L_a. To conform the behavior of G, which is represented by the generated and marked languages of G, the supervisor S disables controllable events. Thus, the objective of the supervisor S is to make the language of the controlled system S/G (we say S controlling G) equal to L_a, or as closely as possible, in case the specifications cannot be met. Figure 2.7 presents the block diagram of a feedback DES.

In order to accomplish such task, supervisor S analyzes the sequences of observable events generated by G, represented by s and sends a control action $S(s)$, disabling the occurrence of certain controllable events in order to prevent unwanted behaviors, such as deadlocks, livelocks and unacceptable states, *e.g.*, the state that represents the overflow of a buffer. The configuration of S/G is the feedback control shown in Figure 2.7. It must be noted that $S(s)$ can only enable or disable feasible events, that is, events in $\Gamma(f(x_0, s))$.



(a) A printer with an automatic guillotine



(b) Automaton $G_{printer}$ that models the printer-guillotine system.

2.5.1 Supervisory control problem

Assume that L is the language generated by an automaton G, that is, $L = L(G)$. As roughly described on the previous section, the supervisory control problem consists of designing a supervisor S that interacts with G in a feedback control manner that is shown in Figure 2.7, and enforces the safety of the system S/G, that is, $L(S/G) = L_a \subseteq L(G)$.

Formally, a supervisor is a function $S : L(G) \rightarrow 2^{\Sigma}$ that relates the language generated by G to the power set of Σ. That way, the new set of active events of S/G, $\Gamma_N[f(x_0, s)]$ is equal to $\Gamma[f(x_0, s)] \cap S(s)$. Again, $\Gamma_N[f(x_0, s)] \subseteq \Gamma[f(x_0, s)]$, that is, S cannot enable events that do not belong to $\Gamma[f(x_0, s)]$.

The compensated system S/G is a DES, whose generated language can be recursively defined as:

1. $\varepsilon \in L(S/G)$,

2. $\forall s \in \Sigma^*$ and $\forall \sigma \in \Sigma$, $s\sigma \in L(S/G) \Leftrightarrow (s \in L(S/G)) \wedge (s\sigma \in L(G)) \wedge (\sigma \in S(s))$.

As it is the case that $L_m(G) \subseteq L(G)$, the marked language of S/G is defined as:

$$L_m(S/G) = L_m(G) \cap L(S/G).$$

Example 2.7 *Consider a printer with an automatic cutting system, displayed in Figure 2.8a; whose guillotine operates independently of the printer, cutting the paper whenever it receives a cut signal. The uncompensated behavior of the printer is modeled by the automaton $G_{printer}$, also displayed in Figure 2.8b. The event set of $G_{printer}$ is $\Sigma = \{print, cut, finish\}$, which can be partitiioned as $\Sigma_c = \{print, cut\}$ and $\Sigma_{uc} = \{finish\}$, corresponding to the controllable and uncontrollable event sets, respectively. Notice that all events of Σ are observable, thus $\Sigma = \Sigma_o$.*

Notice that the uncompensated behavior of the printer allows the guillotine to operate when either there is no sheet of paper to cut, risking to break the blade or to cut during a printing job, thus wasting material. In order to prevent such undesired

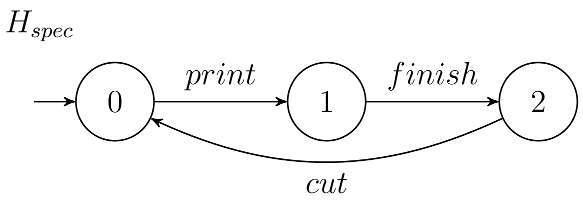


Figure 2.9: Automaton H_{spec} that models the given specifications.

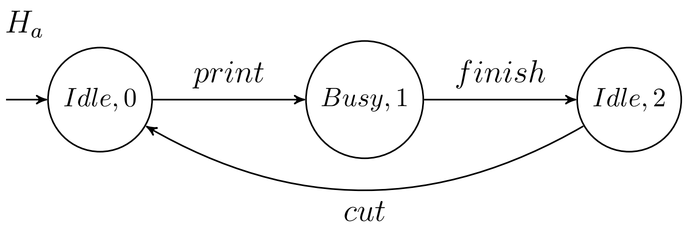


Figure 2.10: Automaton H_a that models the desired behavior of the compensated system.

behaviors, a supervisor must be designed so as to allow the guillotine to cut the paper only after a printing job is finished, and after the end of every single printing job. Such specifications can be modeled by the automaton H_{spec}, shown in Figure 2.9

The desired behavior of the controlled system is obtained by computing automaton $H_a = G_{printer}||H_{spec}$ which is displayed in Figure 2.10. Notice that event cut can only occur after event finish has happened, which shows that the specifications have been met.

The next section, will present a supervisory control problem that can deal with systems like the one addressed in Example 2.7.

2.5.2 Control with partial controllability and full observation

When the event set of an automaton G that models a DES is partitioned as Σ_c and Σ_{uc}, not all specifications modeled as a language K, may be enforced by a supervisor S, if it needs to disable an event that is uncontrollable in order to enforce the desired behavior.

Thus, it becomes necessary to know if the language K that represents the desired behavior can be achieved by S/G, *i.e.*, if it is possible to design a supervisor S such that $L(S/G) = K$. Such a desired behavior will be achieved if the language K is controllable with respect to $L(G) = M$ and event set Σ_{uc} of G, *i.e*, if an uncontrollable event must occur, it cannot take the controlled system outside K. This leads to the concept of language controllability, which is formally defined as follows.

Definition 2.5 (Controllability) *Let $L = \overline{L}$ and K be languages defined over $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, such that $K \subseteq L$. We say that K is controllable with respect to L and Σ_{uc} if*

$$\bar{K}\Sigma_{uc} \cap L \subseteq \bar{K} \tag{2.4}$$

.

Example 2.8 *Let us consider the same printer with an automatic cutting system and the same uncompensated model $G_{printer}$ of the Example 2.7, and assume that the same specifications must be enforced. Since $\Sigma_c = \{print, cut\}$ and $\Sigma_{uc} = \{finish\}$, we show that $L(H_a)$ is controllable with respect to $L(G)$ and Σ_{uc}.*

We have that:

$$L(G) = \overline{\{\{cut\}^*\{print\}\{cut\}^*\{finish\}\}^*}$$

and that

$$L(H_a) = \overline{\{print.finish.cut\}^*}$$

For every string $s \in \overline{L(H_a)}$, we have to check if $\{s\}\Sigma_{uc} \cap L(G) \subseteq \overline{L(H_a)}$. In the first case, we have that $s = \varepsilon$. As $\varepsilon\Sigma_{uc} = \Sigma_{uc}$; and no uncontrollable event may occur at state $Idle$ of G, the first case doesn't violate the controllability condition.

Now, with $s = print$, $s\Sigma_{uc} \cap L(G) = printfinish$, which is in $\overline{L(H_a)}$, thus, does not violate the controllability condition.

Next case is $s = print.finish$, which leaves us with $s\Sigma_{uc} = print.finish.finish$; and as $s\Sigma_{uc} \cap L(G) = \emptyset$ is contained by $L(H_a)$, the controllability condition holds.

Lastly, when $s = print.finish.cut$, the situation is the same when $s = \varepsilon$, because this string leads the automaton back to its initial state. Thus, we can see that H_a is admissible because no violation to the controllability condition can happen.

Thus, by studying Example 2.8, it becomes possible to intuitively understand what makes the language that models the specification controllable with respect to $L(G)$ and Σ_{uc};*i.e*, not disabling any uncontrollable events. The existence of a supervisor S that ensures that $L(S/G) = K$ when K is controllable is ensured by the followin theorem.

Theorem 2.1 (Controllability theorem) *Given a DES $G = (X, \Sigma, f, \Gamma, x_0)$ and a non-empty language K, such that $K \subseteq L(G)$, then, there exists a supervisor S such that $L(S/G) = \overline{K}$ if, and only if, K is controllable, i.e, if and only if*

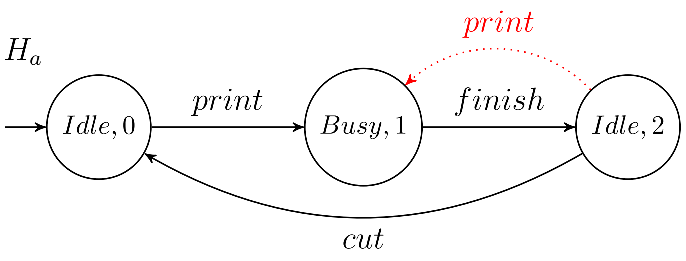


Figure 2.11: Automaton H_a that models the behavior of the compensated system, with the transition that supervisor S tries to disable displayed in red.

$$\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K} \tag{2.5}$$

This condition is called the controllability condition.

Example 2.9 *Let us consider the same printer with an automatic cutting system, whose uncompensated model is $G_{printer}$. Assume also the same specifications as in Examples 2.7 and 2.8, but with events print and finish both uncontrollable, that is, $\Sigma_c = \{cut\}$ and $\Sigma_{uc} = \{print, finish\}$.*

We now need to check if the desired behavior $L(H_a)$ of the compensated system is controllable with respect to the new uncontrollable events set Σ_{uc}. In order to do so, we need to verify if any string $s \in K$, violate the controllability condition. We begin examining the case when $s = \varepsilon$. Notice that $\varepsilon\Sigma_{uc} = \{print, finish\}$, and so, $\{print, finish\} \cap L(G) = \{print\}$, which is a subset of $L(H_a)$. Therefore, the controllability condition is not violated.

The next string to be checked is $s = print$, for which we have that $\{print\}\Sigma_{uc} \cap L(G) = \{printfinish\}$, which is a subset of $L(H_a)$, thus, not violating the controllability condition.

However, for $s = printfinish$, $\{printfinish\}\Sigma_{uc} \cap L(G) = \{printfinishprint\}$, which is not a subset of $L(H_a)$, thus, violating the Controllability Theorem. We can see that the supervisor we are trying to design tries to disable an uncontrollable event, as we can see in Figure 2.11.

In order to deal with such problems in supervisory control, when no admissible supervisor can be obtained for a desired behavior modeled by a language K, compromises must be made. One way of addressing this concern is by disabling controllable events before the system reaches states where the control condition is violated, in such a way that the largest sublanguage of K, called supremal sublanguage of K and denoted by $K^{\uparrow C}$ is achieved. It must be noted that $K^{\uparrow C} \subseteq K$, where $K^{\uparrow C} = K$ if language K is controllable. Another way is by allowing the least possible number of uncontrollable events to happen, in order to achieve admissibility. Such a solution, which extrapolates K is usually referred as the infimal prefix-closed controllable su-

perlanguage, that is, the smallest prefix-closed admissible language $K^{\downarrow C}$. It must be noted that $K^{\downarrow C} \supseteq K$, where $K^{\downarrow C} = K$ if language K is controllable.

2.5.3 Obtaining a supervisor

Lets assume that a DES is modeled by an automaton G, whose events are all controllable and observable, for the sake of simplicity. The uncompensated behavior of G may lead to undesirable states, such as failures, overflows, wrong operations and deadlocks. Assume that the uncompensated behavior and the desired behavior of the system can be modeled by languages $L(G)$ and K, respectively. In order to obtain a supervisor S such that $L(S/G) = K$, the first step is to construct an automaton that represents the specifications for the system, called H_{spec}. There are several techniques for representing various specifications as automata, some of them can be found on [11]. The second step is to verify the behavior of the system when the specifications are being enforced. In order to do so, automaton H_a must be obtained, which can be done either by a parallel composition between H_{spec} and G or by a product composition between the same automata; depending on how H_{spec} was structured. The generated language of H_a, $L(H_a) = L_a$, must be equal to K.

If there are multiple specifications, instead of building a single H_{spec}, sometimes it is more practical to build several automata, $H_{spec,1}$, $H_{spec,2}$, $\ldots$, $H_{spec,n}$, which must be all composed with G in order to obtain H_a, that is, $H_a = H_{spec,1}||H_{spec,2}||\ldots||H_{spec,n}||G$ or $H_a = H_{spec,1} \times H_{spec,2} \times \ldots \times H_{spec,n} \times G$, depending on how the specification automata were generated.

Finally, to obtain a realization of the supervisor S, it is just a matter of building an automaton R such that:

$$R := (Y, \Sigma, g, \Gamma_R, y_0, Y),$$

where R is trim and

$$L_m(R) = L(R) = \overline{L_a}.$$

Notice that language $L(H_a)$ must be maximally permissive, *i.e*, only strings associated with the specifications must be disabled. The following example illustrates how to obtain a realization automaton R.

Example 2.10 *Let us obtain a realization for the supervisor S of Example 2.7. Analyzing H_a, displaying in Figure 2.10, it is possible to see that cut events may only occur after a $finish$ event has happened, thus, the specifications given have been met. As there are neither uncontrollable nor unobservable events, the controlability and observability theorem cannot be violated. So, it becomes just a matter of obtaining a*

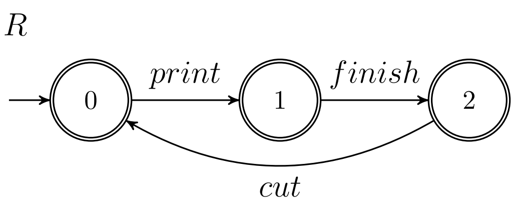


Figure 2.12: The standard realization automaton R for the supervisor S of example 2.7.

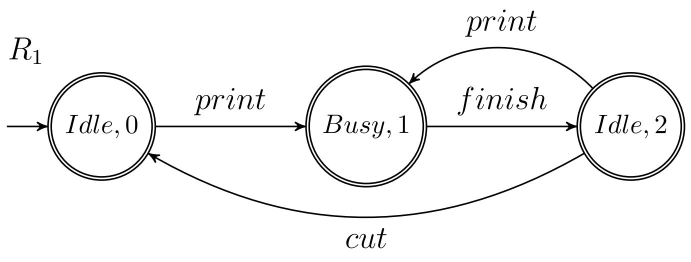


Figure 2.13: A possible non-reduced realization for supervisor S.

realization for the desired supervisor S. Automaton H_a itself, albeit with all states marked, could be a realization R for supervisor S:

If the event set Σ of G has uncontrollable events, but all of them are observable, an additional step is necessary to the procedure described before, as follows: after obtaining automaton H_a, we must check if no uncontrollable events were disabled. That is done by comparing each state of H_a, with the corresponding state of G. If no uncontrollable event was disabled, H_a is controllable with respect to L_a, $L(G)$ and Σ_{uc}.

Example 2.11 *We are now interested in obtaining a realization for supervisor S of Example 2.9. The analysis of the controlled behavior, modeled by automaton H_a, which is displayed in Figure 2.11 has shown that there is no violation of the controllability theorem and it satisfies all specifications. Thus, it is possible to obtain a realization R_1 from H_a, with all states marked, as shown in Figure 2.13.*

Realization automaton R_1 displayed in Figure 2.13 is a viable realization, as its generated language is L_a, but it is not the smallest realization possible. Sometimes, it is possible to obtain a reduced-state realization. If we relax the requirement that $L(R) = L_a$, to the weaker condition $L(R) \supseteq L_a$, i.e, at some states of R, extra events may be enabled. Although, if those events are not feasable in the corresponding states of G, the generated language $L(R||G)$ does not changes. A reduced-state realization R_2 is shown in Figure 2.14

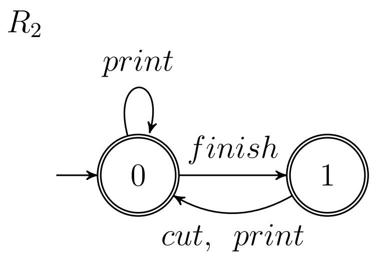


Figure 2.14: A reduced-state realization for the supervisor S

2.6 Supervisory control with modular specifications

The number of states and transitions increase exponentially with the complexity of the system and the control specifications. In example, if there are 10 specifications, each one modeled by an automaton with three states and three transitions, the resulting specification automaton $H_{spec} = H^1_{spec}||H^2_{spec}|| \dots ||H^{10}_{spec}$ may have up to 3^{10} states and up to 10×3^{10} transitions. It becomes easy to understand that for large and complex systems, a vast quantity of memory is needed to implement a monolithic supervisory control, making that implementation either too expensive or even impossible.

In order to deal with such a problem, a modular approach was sugested in [9], which can be seen as a in according to the “divide and conquer” approach. This form of modular supervisory control architecture exploits the structure of the specifications, instead of the architecture of the system. The safety specifications for the system are modeled by a language L_a that may be decomposed in prefix closed-languages, as follows:

$$L_a = L_{a,1} \cap L_{a,2} \cap \dots \cap L_{a,n}.$$

For example, fot the modular architecture of Figure 2.15, language L_a consists of the intersection of two languages $L_{a,1}$ and $L_{a,2}$, and supervisor S_1 is synthesized in order to enforce the specifications modeled by $L_{a,1}$, while supervisor S_2 is synthesized to enforce the specifications modeled by $L_{a,2}$. The joint supervisor, with admissible modular supervisors S_1 and S_2, is defined as:

$$S_{mod12}(s) := S_1(s) \cap S_2(s)$$

Thus, in order for an event to be enabled by S_{mod12}, it must be enabled by both supervisors S_1 and S_2. Thus, if a single supervisor disables an event, such an event will be disabled by S_{mod12}. So, the generated and the marked languages of the joint supervisor are given by

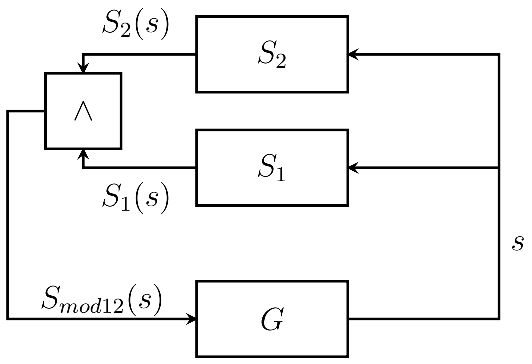


Figure 2.15: Modular control architecture with two supervisors.

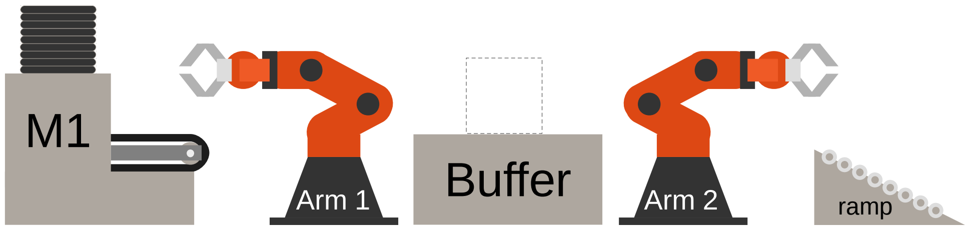


Figure 2.16: A simple processing plant, consisting of a machine M1, two robot arms Arm 1 and Arm 2.

$$L(S_{mod12}) := L(S_1) \cap L(S_2)$$

$$L_m(S_{mod12}) := L_m(S_1) \cap L_m(S_2)$$

In order to obtain an automaton representation of S_{mod12} for analysis, it is only necessary to compute the product between supervisors S_1 and S_2, given they are admissible, *i.e*, the automaton representation of S_{mod12}/G is $S_{mod12} = S_1 \times S_2||G$.

When the system model automaton G has marked states, *i.e*, $X_m \neq \emptyset$, the resulting closed-loop behavior may be blockin, that is, $\overline{L_m(S_{mod12})/G} \subset L(S_{mod12})/G$, even if supervisors S_1 and S_2 are indivudally non-blocking. This means that it is necessary to ensure that the resulting closed-loop behaviour $S_{mod12})/G$ is non-blocking. This happens if $L_m(S_1/G)$ and $L_m(S_2/G)$ are non-conflicting, *i.e*

$$\overline{L_m(S_1/G) \cap L_m(S_2/G)} = \overline{L_m(S_1/G)} \cap \overline{L_m(S_2/G)}.$$

The following example illustrates the modular supervisory control technique.

Example 2.12 *The manufacturing system shown in Figure 2.16 consists of a processing machine M1, two arms (Arm 1 and Arm 2), a buffer with capacity for a single processed part and a ramp, where processed parts are driven to the packaging process. The raw materials for M1 are considered infinite and the ramp has infinite capacity, that is, countless processed parts can be sent down the ramp.*

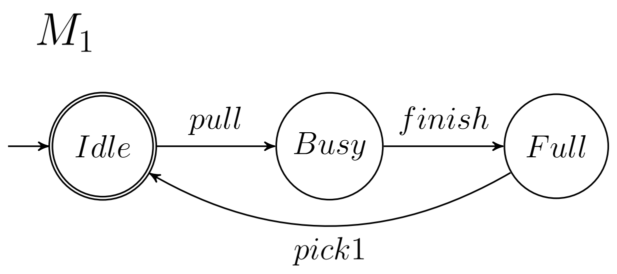


Figure 2.17: Automaton M_1 that models the behavior of processing machine $M1$.

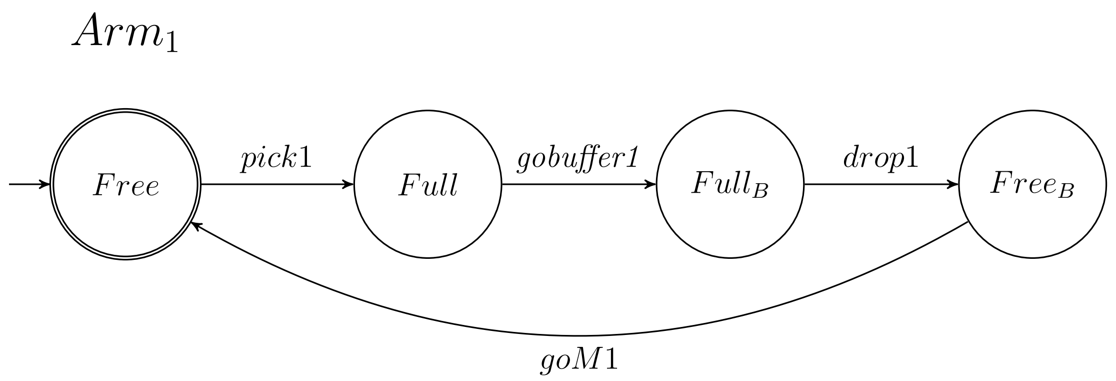


Figure 2.18: Automaton Arm_1 that models the behavior of robot arm *Arm* 1.

The Arms and processing machine $M1$ are represented by automata M_1, Arm_1 and Arm_2, shown in Figures 2.17, 2.18 and 2.19, respectively. The processing machine M1 has three states: Idle, where the machine is waiting for raw materials to be processed, Busy, when the machine is processing a raw material and the Full state, when the machine has finished processing the raw material and is awating for Arm 1 to pick the processed piece. The only uncontrollable event of $M1$ is $finish$.

The automaton model for Arm 1 has four states: Free, when the robot arm rests above $M1$, but has not picked a processed part from the machine; Full, that, represents the state when the robot arm has picked a processed part from $M1$, but is still above it, $Full_B$, that accounts for the fact that the arm has moved to the buffer, but is still holding the processed part, and $Free_B$, that, represents that the robot arm is still over the buffer, but has placed the processed part on the buffer. Automaton Arm_2 has the same states, with the difference that Free and Full states mean that Arm 2 rests above the ramp.

As for the specifications, it is possible to note that the uncontrolled behavior of the system allows lets robot arms access the buffer simultaneously, which may cause collisions. In addition, the buffer may receive processed parts indefinitely, which would cause an overflow; thus, the supervisory control system must prevent additional placement of parts when the Buffer is full.

To meet the first specification, we build automaton $Hspec_{Arms}$, displayed in Figure 2.20. After gotobuffer1 occurs, gotobuffer2 may only occur after the occurrence

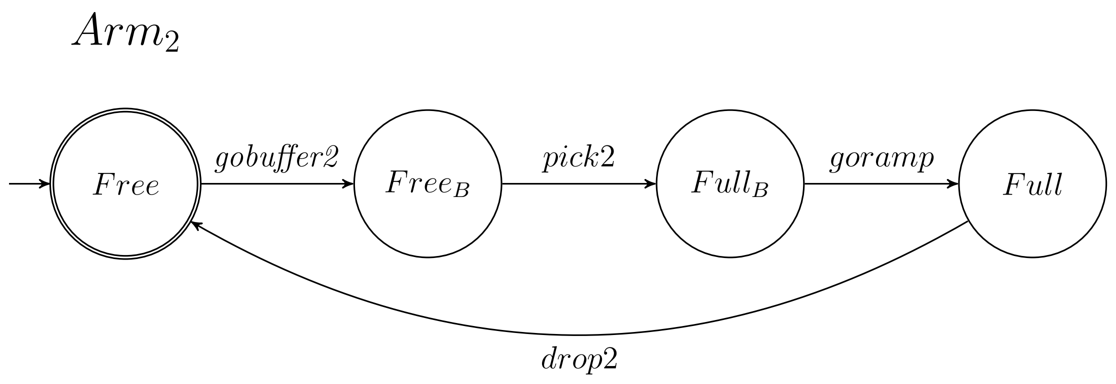


Figure 2.19: Automaton Arm_2 that models the behavior of robot arm *Arm* 2.

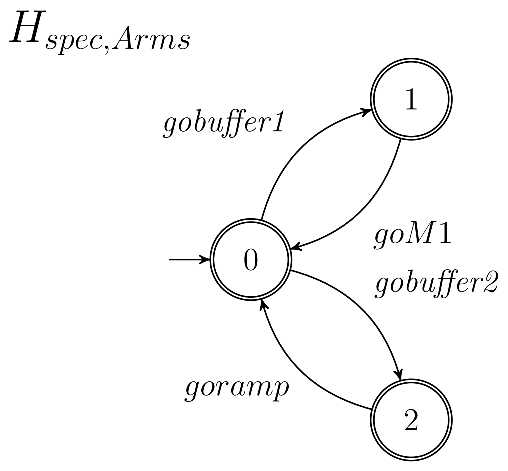


Figure 2.20: Automaton H_{spec} that models the specification that forbids collisions between the robot arms.

*of event goM*1 *and if event gotobuffer2 occurs, gotobuffer1 may only happen after event goramp occurs. This way, there will be no collisions over the buffer.*

In order to avoid both overflows and underflows in the buffer, we must create automaton $Hspec_{Buffer}$*, depicted in Figure 2.21, whose states represent the number of processed parts currently in the buffer. At state* 0*,* $Hspec_{Buffer}$ *forbids Arm 2 from picking up parts, and at state* 2*, the automaton forbids the placement of additional pieces by Arm 1, thus, preventing overflows.*

Now it becomes necessary to analyze the controlled behavior of the system. First,

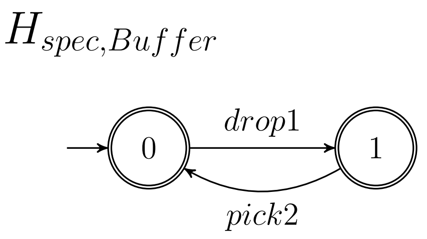


Figure 2.21: Automaton $Hspec_{Buffer}$ that models the buffer specifications for the system.

we must obtain automaton $G_{system} = M_1||Arm_1||Arm_2$, which models the complete system behavior and has 48 states and 120 transitions. Its state transition function is represented in Table A.1 in Appendix A.

After that, it is necessary to verify if the specifications violate the controllability theorem. So, we need to obtain the parallel composition between automata $Hspec_{Arms}$ and G_{system}, and between $Hspec_{Buffer}$ and G_{system}, the former has 36 states and 76 transitions, whereas the later has 96 states and 216 transitions Those compositions generate, respectively, automata H_{arms} and H_{buffer}. It can be verified with the assistance of DESLAB [27] that no uncontrollable events are disabled, thus, the specifications do not violate the Controllability Theorem. Now, we must obtain the realization for the supervisors, which can be done directly from H_{arms} and H_{buffer}. We, now, obtain supervisors S_1 and S_2, whose state transition functions are represented by tables A.2 and A.3 in Appendix A, respectively.

In addition, it can be verified with the assistance of DESLAB that languages $L_m(S_1/G)$ and $L_m(S_2/G)$ are non-conflicting; thus, the resulting behavior of S_{mod12}/G is non-blocking.

2.7 Dijkstra's algorithm

Dijkstra's algorithm solves the shortest path problem for graphs, oriented or not, whose edges have non-negative weights. It is a greedy algorithm, but, nevertheless, guarantees the shortest path from an initial node to every node in the graph. Being of easy implementation and of low computational cost, it is one of the most used path-finding algorithms. This algorithm always obtains the shortest path because it is based on the principle that the sum of the shortest pathes will give the shortest path possible [28].

Dijkstra's algorithm idea consists of computing the shortest distance between the nodes, one by one. The algorithm starts at a given initial node, and runs through every edges that originate from it, assigning every node those edges reach with the weight associated with those edges. At the end, the node with the smallest assigned weight is chosen. This same procedure is now repeated for the chosen node. It is worth noticing that for subsequent non-visited edges, the value assigned to the next achieved node is the sum of the weigh associated with the edge that reaches it with the weight assigned to the previous node. If a node has already been visited, therefore having an assigned weight, it must be checked if it is smaller than the value already assigned. If this is so, the previously assigned value must be replaced with the new and smaller weight; otherwise, nothing must be done. This process must be repeated until all nodes have been reached. The next example illustrates

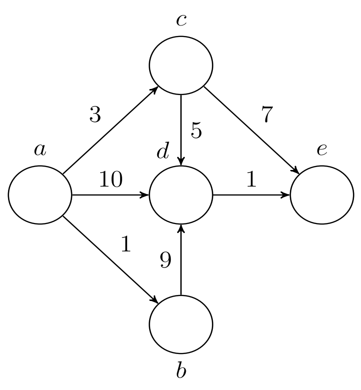


Figure 2.22: Graph G.

Dijkstra's algorithm.

Example 2.13

Let's compute the shortest path between nodes ***a*** *and* ***e*** *of graph G, represented in Figure 2.22, using Dijkstra's algorithm.*

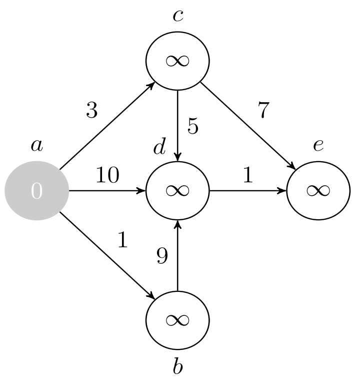
Initializing the algorithm, weight 0 is assigned to node ***a****, since it is the initial node, and atribute infinity weight to nodes that have not been reached yet by any edge.*

The algorithm iterates as follows. In the first step, nodes ***b****,* ***c*** *and* ***d*** *are reached, and to each one, it is assigned the weight of the edges that reached them, since the weight assigned to node* ***a*** *is 0. For the next iteration, node* ***b*** *is chosen, as it has the smallest assigned weight between* ***b****,* ***c*** *and* ***d****, which are 1, 3 and 10, respectively.*

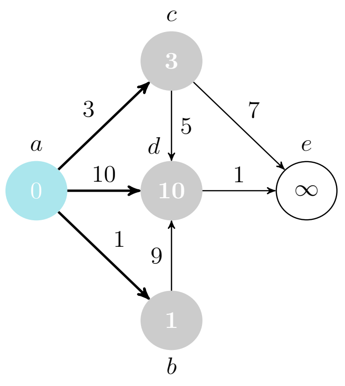
The second step follows the same principles as the first. Node ***b*** *is the origin of a single edge, of weight 9, that reaches node* ***d****. As node* ***d*** *already has an assigned weight of 10, it is necessary to check if the total weight from* ***b*** *is smaller. Since the result is equal to 10, the previous weight is kept.*

The third step has ***d*** *as the origin node. This node has a unique edge linking it to node* ***e****. Since this edge has weight 1 and the weight atributed to* ***d*** *is 10, a total weight of 11 is assigned to* ***e****.*

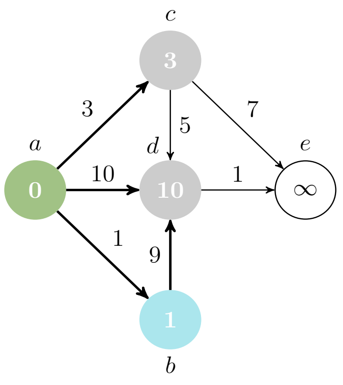
The fourth iteration returns to node ***a****, since not every possible path was obtained. Since the edge with the smallest weight has already benn chosen, the one with the smallest weight among the remaining edges must be selected. Since the edge with the second smallest weight is the one that links node* ***a*** *to node* ***c****, it iss necessary to go to node* ***c*** *and repeat the same procedure as the previous steps. Two edges originate from node* ***c****, one with weight 7, that leads to node* ***e****, and another that leads to node* ***d****, with weight* 5*. Computing the new distances, it is possiblle to see that the total weight to reach node* ***d*** *is 3+5=8* ***d****, which is smaller than the previous attributed weight of 10. Thus, the weight assigned to* ***d*** *must be changed. As for node* ***e****, the obtained weight is still 10, thus, keeping the old weight. Now, it is necessary to*



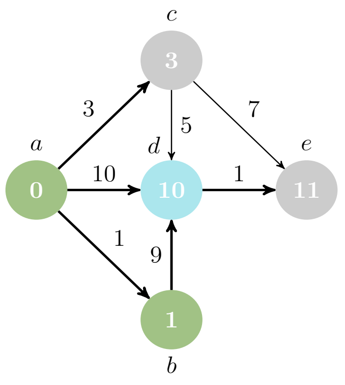
(a) Initialization.



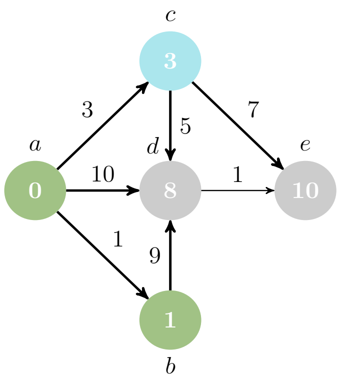
(b) First step.



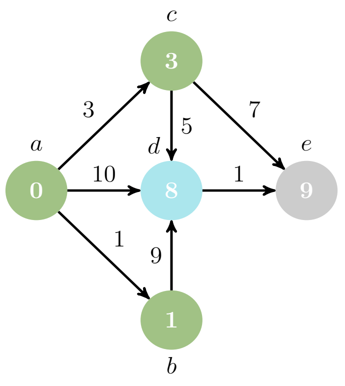
(c) Second step.



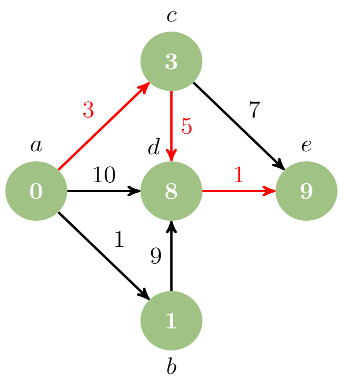
(d) Third step.



(e) Fourth step.



(f) Fifth step.



(g) Shortest path from node a to node e of graph G.

Figure 2.23: Step by step solution of a shortest path problem with Dijkstra's algorithm

follow the edge with the smallest weight, that is, into node ***d****.*

The fifth and final step will assign a total weight of 9 to node ***e****, because node* ***d*** *has a total weight of 8 and the edge that leads to* ***e*** *has weight 1. Since all nodes that have edges originating from them have been searched, the algorithm can finally be stop, yielding the result displayed in Figure 2.23g.*

Example 2.13 shows the efficiency of Dijkstra's Algorithm. The pseudo-code presented in Algorithm 1 summarizes the steps for the implementation of Dijkstra's Algorithm.

Algorithm 1: Dijkstra's Algorithm

```
input : Graph and initial node
output: Distances from every node to a designated initial node
begin
    Q=[]
    for every node n of Graph do
        dist[n] ← ∞ #unknown distances
        prev[n] ← UNDEFINED #previous node in the path
        adds n to Q #non-visited nodes
    end
    dist[initial node] ← 0
    while Q is non-empty do
        u ← node in Q with smallest dist[u] # begins with the initial
        node
        removes u from Q
        for every neighbor node n' of u do
            alt ← dist[u] + weight(u, n')
            if alt < dist[n'] then
                dist[n'] ← alt
                prev[n'] ← u
            end
        end
    end
    return dist[],prev[]
end
```

The simplest version of Dijkstra's Algorithm has a computational cost of order $O(N^2)$, where N is the number of nodes of a graph, beingm therefore, polinomial. A smarter way to implement Dijkstra's Algorithm is by arranging the list of nodes `Q` after receiving all nodes of the graph, in a binary heap[1]. With such a modification, the computation cost of the algorithm drops to $O(E + N\,log(N))$, where E is the number of edges of graph G [28]. It is worth mentioning that the obtained cost is for

[1]A binary heap is a data structure in which a list is represent by a binary tree. The value of a child node cannot be larger than the value of the parent, for max-heaps. For min-heaps, it is the oposite [28].

a single initial node, *i.e*, if we desire to obtain the shortest distance from all nodes to every other node, it is necessary to multiply that cost by N.

Chapter 3

A New Supervisory-control-based-framework for Robot Navigation

3.1 Introduction

This chapter presents the main part of this dissertation, the development of a general methodology for mobile robot navigation in industrial environments in which the open-loop behavior of the robot and the specifications are based on automata. This methodology consists of a modular supervisor which is the conjunction of two supervisors: the first one that enforces the robot to follow the path defined by a planner and the second one that guarantees the satisfaction of the specifications such as prevention of collisions, task completion and movement management.

The proposed navigation architecture allows decentralized implementation, in which the modular supervisor is embedded in the mobile robot, whereas the planner may run either locally or in an external agent. Such a feature makes the adaptation of the proposed navigation architecture to different environments easy.

We begin by describing the system models Section 3.2, which consists of the environment model G_e (Subsection 3.2.1), and the robot model G_r (Subsection 3.2.2). These models are essential for the technique developed in this dissertation, but are easily interchangeable with other automaton models in case is some need to represent changes in the environment or in the robot operation in the plant is required.

The navigation architecture itself is presented in Section, 3.3, which describes how the architecture is structured, in Subsection 3.3.1, the path planning procedure, in Subsection 3.3.2, and the process of designing the modular supervisor S_{r_1} and S_{r_2}, in Subsection 3.3.3.

The next section, Section 3.4, is dedicated to a performance analysis of the algorithms proposed in this dissertation, presenting first a scalability analysis in Subsection 3.4.1 and then, in Subsection 3.4.2, a time complexity analysis of the proposed algorithms.

After that, in Section 3.5.1, we present the results of some simulations in a virtual smart-factory-like environment for a single robot.

Finally, in Section 3.6, we adapt the architecture developed for the navigation of a single robot for multiple robots, while noting the difficulties of non-collaborative multi robot navigation in industrial environments.

3.2 System models

3.2.1 The environment automaton model G_e

Our work deals with the mobile robot navigation problem in industrial environments where the structure rarely undergoes major modifications and, thus, it is reasonable to assume *a priori* knowledge of the environment and, also, that the visitable places do not change; for example, in a warehouse, the visitable places correspond to those places associated with all possible shelves the robot must access. We leverage this feature to model the environment by an automaton $G_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e}, X_{m_e})$, where the states in X_e are all possible robot poses (visitable places together with robot orientation in the navigation environment), Σ_e is the set of command events that correspond to those movements that connect the poses in X_e, the transition function f_e and the set of active events Γ_e are defined according to the environment connectivity. Finally, in order to compute a path to be followed by the mobile robot, x_{0_e} is defined as the robot pose at the beginning of the task and X_{m_e} is defined as the set of states that represent the complete execution of the task.

Notice that, although automaton G_e models the environment, its states represent the possible robot poses (positional coordinates and orientation) in the navigation environment, *i.e.*, those poses the robot can visit when executes a string of command events formed from Σ_e^*. Notice that, since the robot must transport products, parts and raw materials around the plant, the positioning of the possible robot poses is dictated by both the environment structure and the places where machinery is laid in the plant. Automaton G_e can be constructed by using some roadmap construction technique, *e.g.*, vertical cell decomposition [29], reduced visibility graphs [30] and generalized Voronoi diagrams [31, 32].

Let $x_1, x_{n+1} \in X_e$. A path in automaton G_e that takes the robot from state x_1 to state x_{n+1} has the form $x_1\sigma_1x_2\sigma_2x_3\ldots\sigma_nx_{n+1}$, where, $\forall k \in \{1,\ldots,n\}$, $x_k \in X_e$, $\sigma_k \in \Sigma_e$ and $f_e(x_k, \sigma_k) = x_{k+1}$. Notice that there may exist several different paths that connect state x_1 to state x_{n+1}, and each of them is characterized by its corresponding string of command events $\sigma_1\sigma_2\ldots\sigma_n \in \Sigma_e^*$. In order to compare different paths, we define the weight function

$$\begin{aligned} w: \ \Sigma_e &\rightarrow \mathbb{R}, \\ \sigma &\mapsto w(\sigma) = c, \end{aligned} \tag{3.1}$$

where $c \in \mathbb{R}_+$ represents the cost of executing the robot movement corresponding to command event σ. We, then, define the cost of executing a string $s = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma_e^*$ as follows:

$$J(s) = \sum_{i=1}^{n} w(\sigma_i). \tag{3.2}$$

3.2.2 The robot model G_r

In order to construct a discrete event model for the robot, the features that are important for the correct planning and execution of the navigation task are separately modeled by using automata. In this dissertation, we propose a robot automaton model, denoted by G_r, which is obtained by performing the following parallel composition:

$$G_r = G_{r_m} \| G_{r_s} \| G_{r_{tm}}, \tag{3.3}$$

where G_{r_m}, G_{r_s} and $G_{r_{tm}}$ model the robot movement, sensing and task manager modules, respectively.

• *Robot movement module.* Automaton G_{r_m}, depicted in Figure 3.1, models the robot movement resources. The events of G_{r_m} are listed in Table 3.1. In order to navigate in a given industrial environment, the robot must be able to execute the events in Σ_e, which is the set of events of automaton G_e that models the industrial environment. In addition, it also requires other command events to deal with unpredictable obstacles. When an obstacle is detected, command event sr is used to stop the robot in order to prevent a collision. Command events ret and go are used to return the robot to the last visited state and, to complete the movement interrupted by the obstacle, respectively. Automaton G_{r_m} has also the uncontrollable event rs, which is due to sensor readings, and represents the transition from state M, where the robot is moving, to state S, where the robot is stopped.

It is assumed that the robot has low level controllers that are able to execute the movement commands presented in Table 3.1 and, if the robot is performing some movement and receives a new movement command, it cancels the current movement command and executes the new one. Nevertheless, the events belonging to Σ_e remain controllable since the supervisors are able to prevent their occurrences.

• *Robot sensing module.* Automaton G_{r_s}, depicted in Figure 3.2, models the

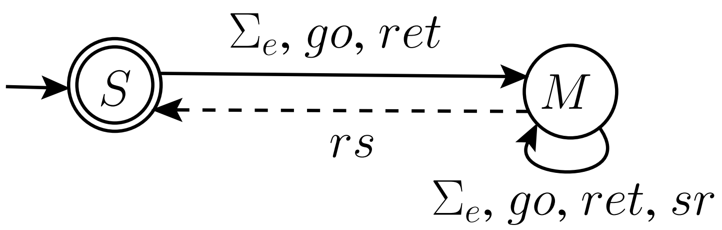


Figure 3.1: The robot movement module G_{r_m}. Dashed lines represent transitions labeled with uncontrollable events.

Table 3.1: Robot movement module events.

Event	Description	Controllable
Σ_e	set of environment automaton events	✓
sr	stop the robot	✓
ret	return to the last visited state	✓
go	complete the last movement	✓
rs	robot stopped	×

robot sensing resources. It is assumed that the robot has wheel encoders and at least one sensing system that is able to detect obstacles, such as sonars, laser rangefinders or vision-based systems. The wheel encoders are used to determine when event rs occurs, that is, when the robot stops after finishing the last movement command. The obstacle detection sensors are used to monitor the presence of unpredictable obstacles, that block the robot path, by means of command events msr and ssr. Command event msr is used when the robot is moving to request the execution of a sensing routine, which, then, returns event od, $\overline{od}$ or rs (active event set of state S_m). Event od (resp $\overline{od}$) indicates that an obstacle is detected (resp. no obstacle detected), and event rs indicates that the robot stopped and, consequently, the sensor reading is no longer necessary. Command event ssr is used, when the robot is stopped, to start a sensing routine that keeps G_{r_s} in state S_s until either a timeout event t or no obstacle detection represented by event $\overline{od}$ occurs. The complete list of events of G_{r_2} is presented in Table 3.2. Notice that events msr and ssr are controllable, whereas events rs, od, $\overline{od}$ and t are uncontrollable.

- *Robot task manager module.* Automaton $G_{r_{tm}}$, depicted in Figure 3.3, mod-

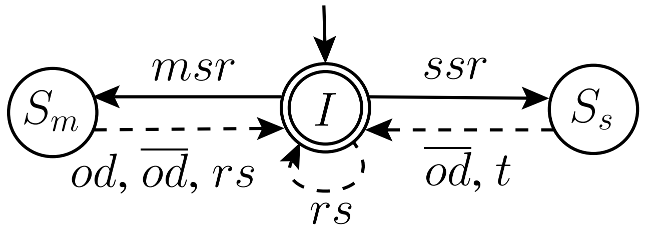


Figure 3.2: The robot sensing module G_{r_s}. Dashed lines represent transitions labeled with uncontrollable events.

Table 3.2: Robot sensing module events.

Event	Description	Controllable
rs	robot stopped	×
msr	obstacle sensor information request while the robot is moving	✓
ssr	obstacle sensor information request when the robot has stopped	✓
od	obstacle detected	×
$\overline{od}$	no obstacle detected	×
t	timeout	×

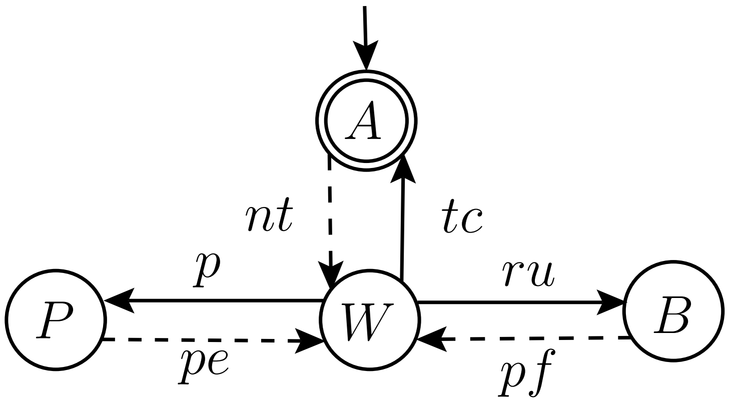


Figure 3.3: The robot task manager module $G_{r_{tm}}$. Dashed lines represent transitions labeled with uncontrollable events.

els the robot resources associated with the management of the robot task. It has four states which represent the current robot status regarding task management, as follows: *(i)* robot idle (I), *(ii)* planning the robot trajectory (P), *(iii)* executing the task (W), and *(iv)* waiting for the removal of an obstacle (B). The events of $G_{r_{tm}}$ are listed in Table 3.3. Event nt is issued by an external agent requesting the execution of a new task by the robot. Command event p is used to start the path planning procedure. After that, the planner starts to execute the path planning and when it is completed, the planner sends to the robot a signal, which corresponds to the occurrence of event pe. Command event ru (request for unblocking the path) is a robot request to an external agent to remove the obstacle between the current robot position and the last visited state. When the external agent removes the obstacle, it issues event pf, meaning that the path is free. Thus, events tc, p and ru are controllable, and events nt, pe and pf are uncontrollable.

3.3 A DES-based robot navigation architecture

A Robot Navigation Architecture is an structure composed by the modules that constitute a mobile robot navigation system (*e.g.*, path planning, obstacle avoidance,

Table 3.3: Robot task manager module events.

Event	Description	Controllable
nt	new task received	×
tc	robot reports task completion	✓
p	execute the planning	✓
pe	planning concluded	×
ru	request for unblocking the last path	✓
pf	the last path is free	×

etc.) and the framework used to combine them [7]. In this dissertation, we address the problem of the navigation of a mobile robot modeled by an automaton G_r, that navigates in an industrial environment modeled by an automaton G_e together with a cost function (function J, defined in Equation (3.2)). We assume that there may exist *a priori* unknown permanent or intermittent obstacles in the environment, so that some transitions of the environment automaton are not allowed to fire either temporarily or definitely. We refer to such transitions as blocked transitions.

The following robot tasks are considered:

- Task 1. This task is completed when the robot reaches some state (pose) belonging to a set $X_{goal} \subseteq X_e$;
- Task 2. This task is completed after the robot visits all states (pose) belonging to a set $X_{goal} \subseteq X_e$.

Since the robot is required to visit only one state in X_{goal} when it executes Task 1, it is necessary to determine the path starting at the current state of the robot and ending at one of the states in X_{goal} that minimizes the cost function J. On the other hand, when the robot executes Task 2, it needs to visit all states in X_{goal}, regardless of the ordering, by following a path that minimizes the cost function J.

3.3.1 The navigation architecture

The navigation architecture proposed in this dissertation is formed by a planner and a modular supervisory control structure, as shown in Figure 3.4. An advantage of this structure is that it allows a decentralized implementation, where the modular supervisor is embedded in the mobile robot, whereas the planner runs in an external agent, which suits very well to the design problem addressed here suitable, since it makes easier to adapt the proposed navigation architecture to other environments.

The navigation process starts when the robot is available and an external agent assigns a new task to the robot. This assignment is modeled by the occurrence of event nt of automaton $G_{r_{mt}}$ of Figure 3.3. Then, robot G_r generates event p (execute planning), which carries the following information:

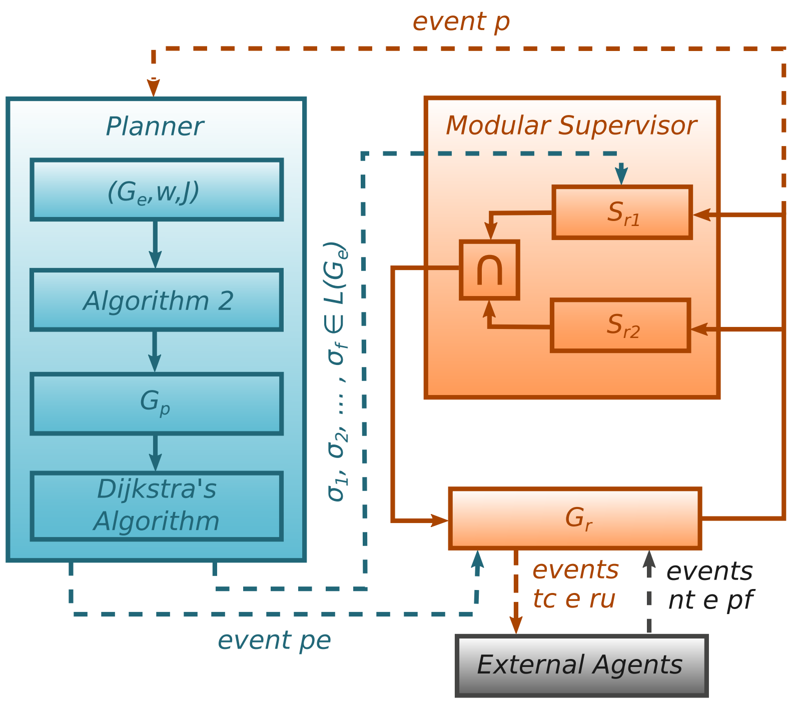


Figure 3.4: The proposed navigation architecture.

- The robot current state;
- The last task assigned to the robot and its respective set of target states X_{goal};
- Set T_b, which is formed with those transitions identified as blocked, being initialized as an empty set and modified by the robot when event t occurs in G_{r_s}, *i.e.*, the blocked transition of G_e is added to T_b when the robot detects a permanent obstacle. The blocked transition is determined from the last state of G_e visited by the robot and the command event in Σ_e whose execution was interrupted by the obstacle detection. It is worth remarking that we can limit the time interval in which a transition stays in T_b with a view to checking if this transition is still blocked during the execution of a future robot task.

According to the diagram depicted in Figure 3.4, when the planner receives event p, it runs a computer application to determine the path to be followed by the robot. Notice that the planner is composed by:

- Environment automaton G_e;
- Weight function w and cost function J, defined in accordance with Equations (3.1) and (3.2), respectively.

After the planning is finished, the planner sends event pe to automaton G_r to inform that the planning has been concluded. The planner also sends the string of command events that corresponds to the computed path to be used to design supervisor S_{r_1}; thus, a new supervisor S_{r_1} is computed after each new path planning completion.

The modular supervisory control $S_{r_1} \wedge S_{r_2}$ ensures the correct navigation of the robot in the presence of unpredictable permanent or intermittent obstacles; S_{r_1} acts so as to enforce the robot to follow the path computed by the planner, whereas S_{r_2} ensures that design specifications SP_1–SP_7 (to be presented in Section 3.3.3), are achieved.

3.3.2 Path planning procedure

The first step in the path planning procedure is to compute, using Algorithm 2, a refined automaton G_p so that the language marked by G_p is formed by those strings of command events that can be executed by the robot with a view to completing the robot task. Algorithm 2 starts by setting up as initial and marked states of G_e the robot current pose and X_{goal}, respectively, and, after that, G_e is assigned to G_p. If $T_b \neq \emptyset$, then we remove from G_p the transitions in T_b, and set G_p as $CoAc[Ac(G_p)]$. When Task 1 is assigned, automaton G_p must be nonempty. In this case, $L_m(G_p)$ will be formed by strings of command events that can be used to complete Task 1. When Task 2 is assigned, automaton G_p must be modified to form a new automaton whose marked language is formed by those strings in $L_m(G_p)$ that correspond to paths containing all of the states in X_{goal}. This can be done as follows. Let us assume that $X_{goal} = \{x_1, \ldots, x_n\}$. Then, for each $x_i \in X_{goal}$, automaton G_{x_i}, whose marked language contains all strings of $L_m(G_p)$ that correspond to paths that visit state x_i, is constructed. Subsequently, automaton G_p is redefined as $G_p \leftarrow G_p \| G_{x_1} \| \ldots \| G_{x_n}$. As a consequence, the language marked by the new automaton G_p will be formed by those strings that correspond to the paths that contain all of the states in X_{goal}, since it is equal to the intersection of the languages marked by the initial G_p and by automata G_{x_i}, for every $x_i \in X_{goal}$.

Finally, by applying Dijkstra's algorithm [28] using, as input, G_p and its initial state, we determine the marked state of G_p that is nearest the initial state, and the string $\sigma_1 \ldots \sigma_f \in \Sigma_e^*$ that corresponds to the feasible path that minimizes the cost function J, being, therefore, the solution to the path planning for the robot navigation problem.

3.3.3 Design of modular supervisor $S_{r_1} \wedge S_{r_2}$

In order to design supervisors S_{r_1} and S_{r_2}, we initially construct simple automata that capture the essence of the specifications we want to ensure by using these supervisors. We, then, combine these automata with G_r using the parallel composition to obtain the system desired behavior.

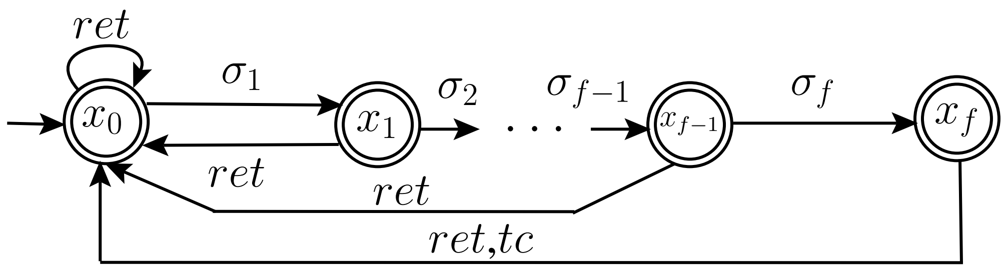


Figure 3.5: Automaton $H_{spec,1}$ used to synthesize supervisor S_{r_1}.

Let us first consider the design of S_{r_1}. According to the diagram of Figure 3.4, we intend to synthesize a supervisor S_{r_1} that enforces the robot to follow the path computed by the planner. Let $s = \sigma_1 \ldots \sigma_i \sigma_{i+1} \ldots \sigma_f \in L(G_e)$ be the string of command events computed by the planner. Notice that $\sigma_i \in \Sigma_e \subset \Sigma_r$, for $i = 1, \ldots, f$, and thus, the behavior of G_r must be restricted to ensure that sequence s is executed. This can be done by creating the specification automaton $H_{spec,1}$ depicted in Figure 3.5, which is formally defined as $H_{spec,1} = (X_1, \Sigma_1, f_1, \Gamma_1, x_0, X_1)$, where $X_1 = \{x_0, x_1, \ldots, x_f\}$, $\Sigma_1 = \{ret, tc\} \cup \Sigma_e$, and f_1 is defined, as follows:

$$f_1(x_i, \sigma) = \begin{cases} x_{i+1}, \text{ if } \sigma = \sigma_{i+1} \\ x_0, \text{ if } (\sigma = ret) \vee ((\sigma = tc) \wedge (x_i = x_f)) \\ \text{undefined, otherwise.} \end{cases}$$

Notice that events ret and tc have been included in $H_{spec,1}$ in order to account for possible obstacle detection, which makes the robot abort the execution of the planned trajectory, and to report that the task has been completed, respectively.

Automaton H_1 that marks the applicable language requirement K_1 is computed by performing the parallel composition between automaton G_r, obtained in accordance with Equation (3.3), and $H_{spec,1}$, as follows:

$$H_1 = G_r \| H_{spec,1}.$$

It is worth remarking that the set of events of automaton G_r is $\Sigma_r = \Sigma_e \cup \{sr, ret, go, rs, msr, ssr, od, \overline{od}, t, nt, tc, p, pe, ru, pf\}$. Thus, if $P_1 : \Sigma_r^* \to \Sigma_1^*$, we can state that:

$$K_1 = P_1^{-1}[L_m(H_{spec,1})] \cap L_m(G_r). \tag{3.4}$$

Notice that, to achieve the requirement imposed by language K_1, only events in Σ_1 may be disabled. Since all events in Σ_1 are controllable, it is not difficult to conclude that K_1 is controllable. In addition, because all states of $H_{spec,1}$ are marked, K_1 is, by construction, $L_m(G_r)$-closed. Then, an automaton realization of a nonblocking supervisor S_{r_1} such that $L_m(S_{r_1}/G_r) = K_1$ can be obtained from $H_{spec,1}$ by adding self-loops labeled by the events in $\Sigma_r \setminus \Sigma_1$ to all of its states.

Let us now consider the synthesis of supervisor S_{r_2}, which deals, among other requirements, with permanent and intermittent obstacles. In practice, the robot classifies a previously detected obstacle as permanent or intermittent by using the sensing routine started by command event ssr, that is, when this sensing routine returns timeout event t, the obstacle is said to be permanent, and, when it returns event $\overline{od}$, the obstacle is said to be intermittent. In order to achieve the desired behavior, the following specifications are enforced:

- SP_1. The robot can perform the command events in $\Sigma_e \cup \{ret, go\}$ only after it receives a new task and the trajectory has already been planned.

- SP_2. After the execution of command event ret, which, according to specification SP_6, will only be executed after a permanent obstacle is detected, the robot cannot perform movement commands in $\Sigma_e \cup \{go\}$ before a new trajectory is computed by the planner.

- SP_3. In order to prevent the wrong functioning of low level controllers that execute the robot movements, a command event in $\Sigma_e \cup \{ret, go\}$ cannot be sent before the execution of the previous movement has been either completed or aborted by command sr (stop the robot).

- SP_4. In order to prevent collisions, the robot continuously checks the existence of obstacles in the trajectory, and, when an obstacle is detected, it must stop.

- SP_5. After the robot stops due to an obstacle detection, it must distinguish between intermittent and permanent obstacles in order to avoid unnecessary computations of new trajectories. In addition, if the obstacle is intermittent, the robot must try to complete the interrupted movement when the obstacle is no longer detected.

- SP_6. When the robot detects a permanent obstacle, it must return to the last visited state in G_e, which corresponds to the last pose visited in the environment, by using command event ret.

- SP_7. When the robot detects a permanent obstacle while it executes the movements associated with command event ret, it must request an external agent to remove this obstacle by means of event ru.

We will now construct specification automata that capture the essence of specifications SP_i, $i = 1, \ldots, 7$. We first construct automaton $H_{spec,2}$ depicted in Figure 3.6, that accounts for specifications SP_1 and SP_2 whose set of events is $\Sigma_2 = \Sigma_e \cup \{nt, p, pe, tc, ret, go\}$. From Figure 3.6, we can see that: *(i)* events in $\Sigma_e \cup \{ret, go\}$ can only be executed at state 3 of $H_{spec,2}$, which is reached only after

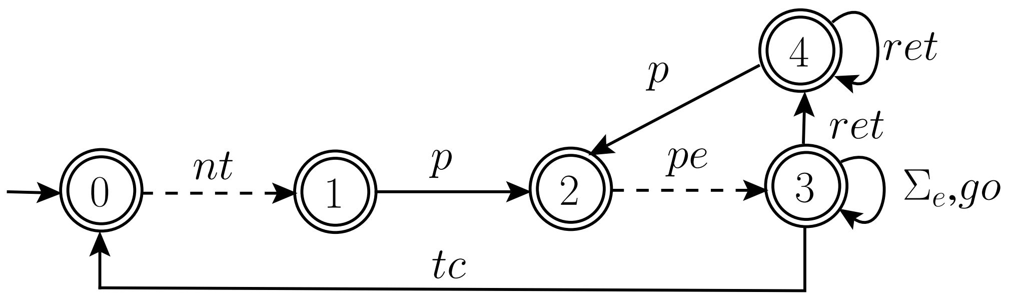


Figure 3.6: Automaton $H_{spec,2}$ used to synthesize supervisor S_{r_2}. Dashed lines represent transitions labeled with uncontrollable events.

the occurrence of pe (planning executed), and; *(ii)* after the occurrence of event ret, all events in $\Sigma_e \cup \{go\}$ remain disabled until the conclusion of a new path planning. We added a self-loop labeled by event ret at state 4, because it may be necessary to perform several occurrences of this event until the robot reaches the last visited state after the detection of a permanent obstacle. This is so because a new obstacle can be detected while the robot is returning to the last visited state. This issue is addressed in the next specification automaton.

Automaton $H_{spec,3}$, depicted in Figure 3.7, accounts for specifications SP_3—SP_7. Its set of events is $\Sigma_3 = \Sigma_r \setminus \{nt, pe\}$. The states of $H_{spec,3}$ correspond to the following situations: *(i)* state 0 represents the case when the robot has stopped without detecting obstacles; *(ii)* states 1 and 2 correspond to the case when the robot is executing the movements associated with the command events in Σ_e; and *(iii)* states 3 to 16 are associated with the procedure to handle obstacles.

After the robot executes an event in Σ_e, leading back to state 1 of $H_{spec,3}$, it must request obstacle sensor information by means of event msr, therefore moving to state 2. If no obstacle is detected, event $\overline{od}$ occurs; this procedure is, then, repeated until the completion of the current movement, which is indicated by the occurrence of event rs, therefore leading to state 0. If an obstacle is detected, $H_{spec,3}$ evolves to state 3 through the transition labeled by event od. The transition from state 3 to 0 labeled by event rs models the case when, after an obstacle detection, the robot stops before the command event sr is issued. In this case, we assume that the detected obstacle did not prevent the execution of the previous movement, and thus, it can be disregarded. On the other hand, the transition from state 3 to 4 labeled by event sr represents the control action that enforces the robot to stop, whose completion is confirmed by the occurrence of event rs (robot stopped) at state 4. At state 5 of $H_{spec,3}$, the robot requests obstacle sensor information by means of event ssr in order to determine whether the detected obstacle is permanent or intermittent. Then, $H_{spec,3}$ remains in state 6 until the occurrence of either $\overline{od}$ or t. The occurrence of $\overline{od}$ means that the previously detected obstacle is intermittent and no longer blocks the path; thus, the robot can complete the movement stopped

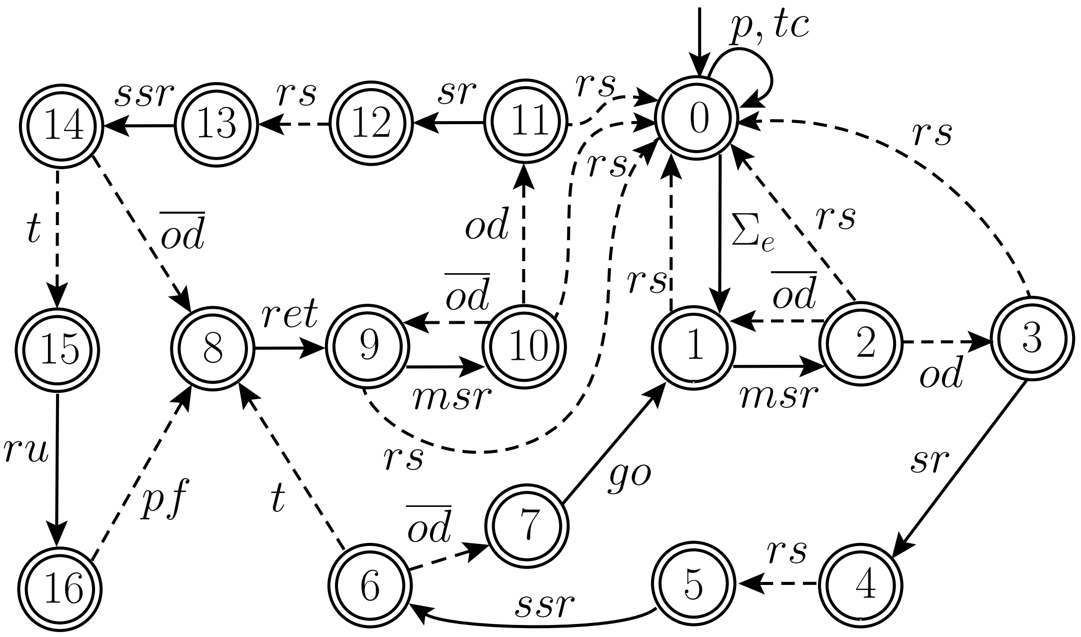


Figure 3.7: Automaton $H_{spec,3}$ used to synthesize supervisor S_{r_2}. Dashed lines represent transitions labeled with uncontrollable events.

due to the obstacle, which is done by means of command event *go*. On the other hand, if the timeout event t occurs, it can be inferred that the previously detected obstacle is permanent, and, so, the robot needs to return to the last visited state in G_e in order to compute a new trajectory, which is done by executing event *ret*. If the robot detects a permanent obstacle while it is returning to the last visited state, that is, if event t occurs at state 15 of $H_{spec,3}$, then, it executes event *ru* by sending a request to unblock the path, and remains at state 16 until the occurrence of event *pf*, which means that the path has been unblocked. After the occurrence of event *pf*, the robot executes event *ret* again to return to the last visited state of G_e.

Automaton H_2 that marks the applicable language requirement K_2 is computed by performing the parallel composition between G_r, $H_{spec,2}$ and $H_{spec,3}$, as follows:

$$H_2 = G_r \| H_{spec,2} \| H_{spec,3}.$$

If $P_2 : \Sigma_r^* \to \Sigma_2^*$ and $P_3 : \Sigma_r^* \to \Sigma_3^*$ denote projections, then,

$$K_2 = L_m(G_r) \cap P_2^{-1}[L_m(H_{spec,2})] \cap P_3^{-1}[L_m(H_{spec,3})]. \tag{3.5}$$

It can be check that K_2 is controllable and $L_m(G_r)$-closed.

An automaton realization of a nonblocking supervisor S_{r_2} such that $L_m(S_{r_2}/G_r) = K_2$ can be obtained by calculating the parallel composition $H_{spec,2} \| H_{spec,3}$. Notice that the set of events of $H_{spec,2} \| H_{spec,3}$ is equal to Σ_r, and, thus, we do not need to add self-loops to it in order to obtain a realization

Algorithm 2: Computation of automaton G_p

```
Inputs:
• $G_e = (X_e, \Sigma_e, f_e, \Gamma_e, x_{0_e}, X_{m_e})$: environment automaton
• The robot current pose
• $X_{goal}$: the set of target states
• task_type $\in \{\text{Task 1}, \text{Task 2}\}$;
• $T_b$: set of blocked transitions of $G_e$.
Output: Automaton $G_p = (X_p, \Sigma_p, f_p, \Gamma_p, x_{0_p}, X_{m_p})$.
begin
    Set the robot current pose and $X_{goal}$ as the initial and marked states of $G_e$,
    respectively;
    $G_p \leftarrow G_e$;
    if $T_b \neq \emptyset$ then
        for every transition $x \xrightarrow{\sigma} y \in T_b$ do
            $f_p(x, \sigma) \leftarrow$ undefined;
            $\Gamma_p(x) \leftarrow \Gamma_p(x) \setminus \{\sigma\}$;
        end
    end
    $G_p \leftarrow CoAc[Ac(G_p)]$;
    if task_type = Task 1 then
        if the set of states of $G_p$ is empty then
            return "Error: impossible task, request path unblocking"
        else
            return $G_p$;
        end
    else if task_type = Task 2 then
        if the set of states of $G_p$ does not contain $X_{goal}$ then
            return "Error: impossible task, request path unblocking"
        else
            $G_{temp} \leftarrow G_p$;
            for every state $x \in X_{goal}$ do
                $G_x = (X_x, \Sigma_x, f_x, \Gamma_x, x_{0_x}, X_{m_x}) \leftarrow G_p$;
                for every event $\sigma \in \Sigma_e$ do
                    $f_x(x, \sigma) \leftarrow x$;
                end
                $\Gamma_x(x) \leftarrow \Sigma_e$;
                Redefine the set of marked states of $G_x$ as $\{x\}$;
                $G_{temp} \leftarrow G_{temp} \| G_x$
                if the set of marked states of $G_{temp}$ is empty then
                    return "Error: impossible task, request path unblocking"
                end
            end
            $G_p \leftarrow CoAc(G_{temp})$;
            return $G_p$
        end
    end
end
```

of supervisor S_{r_2}.

According to modular supervisory control theory presented in Section 2.6, the modular supervisory architecture constructed by the conjunction of S_{r_1} and S_{r_2} is such that $L_m(S_{r_1} \wedge S_{r_2}/G_r) = K_1 \cap K_2$, where K_1 and K_2 are defined in Equations (3.4) and (3.5), respectively. In addition, it can be verified that the admissible languages K_1 and K_2 are nonconflicting, which ensures that the conjunctive modular supervisor $S_{r_1} \wedge S_{r_2}$ is nonblocking, *i.e.*, $L(S_{r_1} \wedge S_{r_2}/G) = \overline{K_1 \cap K_2}$.

Figure 3.8 shows automaton $S_{r_1} \wedge S_{r_2}/G_r$ that models the closed-loop behavior in the case that the robot trajectory computed by the planner is equal to $\sigma_1\sigma_2 \ldots \sigma_f \in L(G_e)$. Such an automaton can be obtained by performing the following parallel composition:

$$S_{r_1} \wedge S_{r_2}/G_r = G_r \| H_{spec,1} \| H_{spec,2} \| H_{spec,3}. \tag{3.6}$$

Notice that, although the automaton shown in Figure 3.8 grows linearly with the length of the robot trajectory (the colored part), the parallel computation presented in Equation (3.6) is not required to implement the modular architecture proposed here, since the designs of S_{r_1} and S_{r_2} are based on different automaton specifications, $H_{spec,1}$ and $H_{spec,2} \| H_{spec,3}$, respectively.

3.4 Performance analysis of the planner algorithm

We present, in this section, a performance analysis of the algorithm proposed here. In the planner, we firstly present the space complexity analysis and, in the sequel, a time complexity analysis.

3.4.1 Scalability Analysis

In the DES-based robot navigation architecture proposed here, the number of transitions of the environment automaton model G_e is $O(|X_e|\,|\Sigma_e|)$, where $|X_e|$ and $|\Sigma_e|$ are the number of the robot poses of interest and command events that are necessary to the robot navigation, respectively. On the other hand, the robot model G_r, obtained in accordance with Equation (3.3), has 24 states and $132 + |\Sigma_e|$ transitions, and, thus, the robot model is $O(|\Sigma_e|)$.

The computational complexity of the planning procedure presented in Section 3.3.2 depends on the type of the robot task. When the robot performs Task 1, Algorithm 2 is executed in time $O(|T_b| + |X_e|\,|\Sigma_e| + |X_{goal}|)$ and generates an automaton G_p whose number of states $|X_p|$ is $O(|X_e|)$. Assuming that the priority queue used in Dijkstra's algorithm is implemented as a Fibonacci heap,

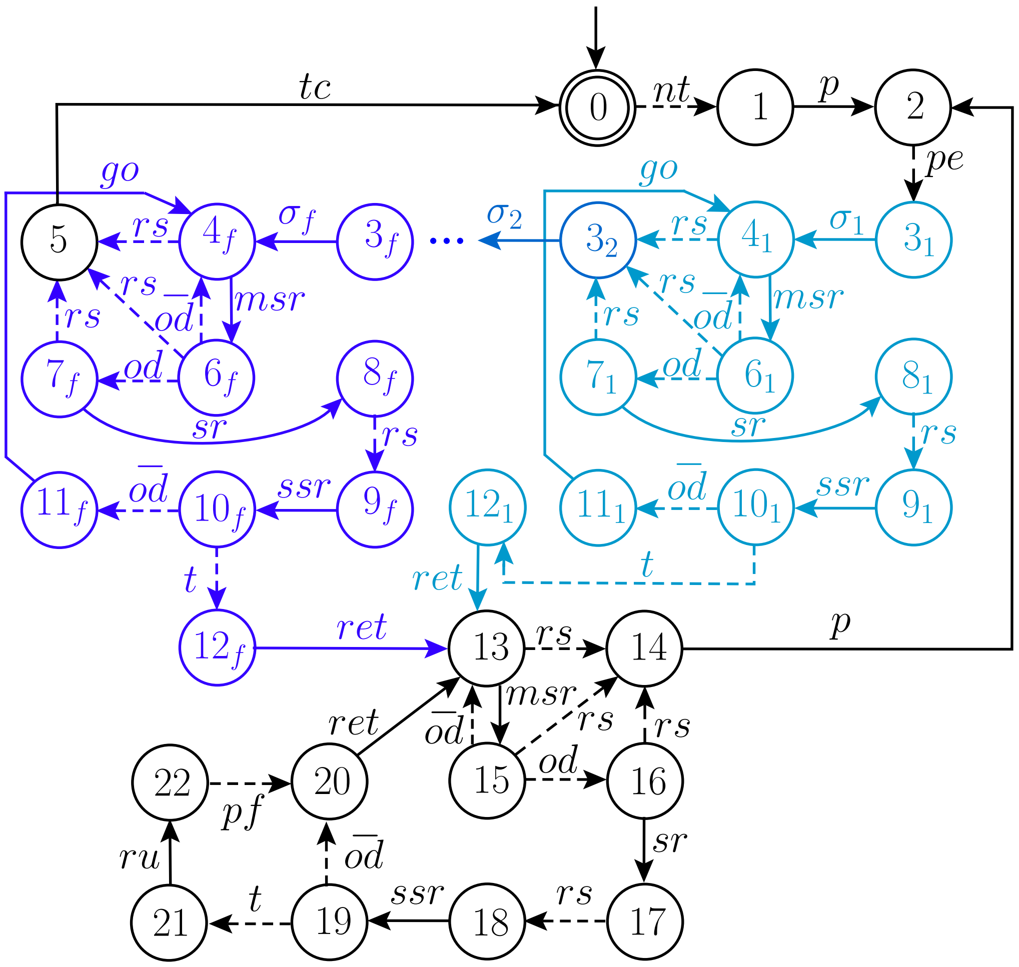


Figure 3.8: Closed-loop behavior for robot trajectory $\sigma_1\sigma_2 \dots \sigma_f \in L(G_e)$. Dashed lines represent transition labeled with uncontrollable events.

the shortest path in automaton G_p is found in time $O(|X_p|\,log|X_p| + |X_p|\,|\Sigma_e|)$. Therefore, the path planning procedure in the case of Task 1 is executed in time $O(|T_b| + |X_e|(|\Sigma_e| + log|X_e|) + |X_{goal}|)$. When the robot performs Task 2, Algorithm 2 is executed in time $O(|T_b| + 2^{|X_{goal}|}\,|X_e|\,|\Sigma_e|)$ and the number of states of automaton G_p, $|X_p|$, is $O(2^{|X_{goal}|}\,|X_e|)$. As a consequence, in the case of Task 2, the path planning procedure is $O(|T_b| + 2^{|X_{goal}|}\,|X_e|(|\Sigma_e| + log|X_e| + |X_{goal}|))$.

Regarding the complexity of the design of modular supervisor $S_{r_1} \wedge S_{r_2}$, it can be seen that the number of states and transitions of automaton $H_{spec,1}$ are both $O(\|path\|)$, where $\|path\|$ denotes the number of command events whose robot must execute to complete the task and was obtained in the path planning procedure. As a consequence, supervisor S_{r_1} is $O(\|path\|)$. In addition, it can be seen that automaton $H_{spec,2}\|H_{spec,3}$ has 22 states and $32 + |\Sigma_e|$ transitions and, consequently, supervisor S_{r_2} is $O(|\Sigma_e|)$.

Based on the computational complexities presented above, it can be concluded that the proposed DES-based navigation architecture scales well with respect to the size of the environment since the aforementioned computational complexities increase with a factor less than $|X_e|^2$ and $|\Sigma_e|$ when either $|X_e|$ or $|\Sigma_e|$ increases, and $|X_e|(|X_e| + |\Sigma_e|)$, when both $|X_e|$ and $|\Sigma_e|$ increase. On the other hand, when the robot performs Task 2, the approach may not scale well with respect to the number of target states since the computational effort to compute the robot path increases exponentially with the number of target states ($|X_{goal}|$).

3.4.2 Time complexity analysis

The results obtained from a series of numerical experiments carried out on a laptop with an Intel Core i5-4210U Processor, 8Gb DDR3 RAM, are shown in Table 3.4. It is divided in five groups, as displayed in Table 3.5, where Group 1, consists in running Tasks 1 cases in an environment modeled by an automaton G_e with 273 states and 1000 transitions, all having the same target state but different initial robot positions; Groups 2 and 3 that show the influence of changes in the environment size in the execution times of Tasks 1 and 2, respectively; and Groups 4 and 5, that show the performance results for Tasks 1 and 2, respectively, as the number of target states in X_{goal} increases. Notice that the times taken to perform the path planning are approximately the same for all simulations of Group 1. This result has already been expected since, as shown in Section 3.4.1, the computational effort of the path planning procedure for cases of Task 1 is predominantly determined by input parameters $|X_e|$, $|\Sigma_e|$, $|T_b|$ and $|X_{goal}|$. As far as Groups 2 and 3 are concerned, the time taken to perform the path planning grows almost linearly with the increase in the environment state space, as we can seen from the plots depicted

Table 3.4: Summary of the results for time complexity analysis.

Group	G_e	Parameters				Planning
	$\|X_e\|/\|T_e\|$	Task	x_{0_e}	$\|X_{goal}\|$	$\|obst\|$	(seconds)
1	273/1000	1	1	1	0	0.024
	273/1000	1	M_1	1	0	0.016
	273/1000	1	29	1	0	0.025
	273/1000	1	M_6	1	0	0.016
	273/1000	1	M_4	1	0	0.016
	273/1000	1	M_2	1	0	0.016
2	39/140	1	1	1	0	0.003
	78/284	1	1	1	0	0.007
	117/428	1	1	1	0	0.008
	156/57	1	1	1	0	0.017
	195/714	1	1	1	0	0.018
	234/857	1	1	1	0	0.020
	273/1000	1	1	1	0	0.023
3	39/140	2	1	2	0	0.364
	78/284	2	1	2	0	0.424
	117/428	2	1	2	0	1.085
	156/57	2	1	2	0	1.401
	195/714	2	1	2	0	1.922
	234/857	2	1	2	0	2.001
	273/1000	2	1	2	0	2.564
4	273/1000	1	1	2	0	0.022
	273/1000	1	1	3	0	0.024
	273/1000	1	1	4	0	0.024
	273/1000	1	1	5	0	0.023
	273/1000	1	1	6	0	0.024
	273/1000	1	1	7	0	0.024
5	39/140	2	1	2	0	0.364
	39/140	2	1	3	0	0.572
	39/140	2	1	4	0	1.148
	39/140	2	1	5	0	2.024
	39/140	2	1	6	0	5.440
	39/140	2	1	7	0	19.473

Table 3.5: Parameters of the test groups.

Group	Task type	Modified parameter
1	1	x_0
2	1	$\|\|X_e\|\|$, transições
3	2	$\|\|X_e\|\|$, transições
4	1	$\|\|X_{goal}\|\|$
5	2	$\|\|X_{goal}\|\|$

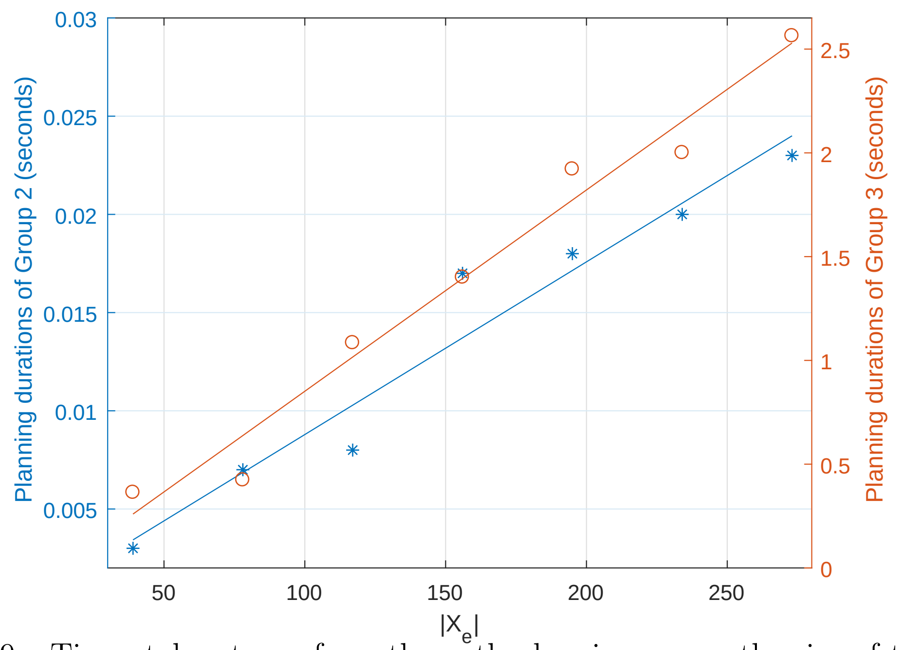


Figure 3.9: Times taken to perform the path planning versus the size of the environment state space for Groups 2 and 3 .

in Figure 3.9. Regarding Groups 4 and 5, as shown in Figure 3.10, the time for path planning is approximately the same in all cases of Group 4 whereas the state size of G_p grows exponentially with the number of target states in tasks of type 2. Notice that these results are in accordance with the the theoretical ones of Section 3.4.1 since automaton G_p is as large as G_e in type 1 tasks but grows exponentially with the number of target states for type 2 tasks.

3.5 Experimental results for a single robot

3.5.1 Simulation Results

Consider the hypothetical industrial environment depicted in Figure 3.11, which resembles a smart factory composed by a conveyor belt ①, which is a loading and unloading terminal, shelves ⑤ and ⑥ that are used to store raw materials and parts processed either by the computer numerically controlled (CNC) milling machine ⑦ or the painting machines ② and ③. The robot arm ④ is used either to transfer parts from one paint machine to the other, or to reject those painted parts that do not meet some quality standard to the orange rectangle. Notice that mobile robot ⓪ must be at pose M_i to interact with element ⓘ, $i = 1, \ldots, 7$.

The automaton that models this environment is $G_e = (X_e, \Sigma_e, \Gamma_e, x_{0_e}, X_{m_e})$, where *(i)* $X_e = X_{int} \dot\cup X_{ps}$ with $X_{int} = \{1, 2, 3, \ldots, 32\}$ and $X_{ps} = \{M_1, M_2, M_3, \ldots, M_7\}$ are formed with those poses that correspond to the corridor intersections and those poses that allow the robot to pick and store parts, respec-

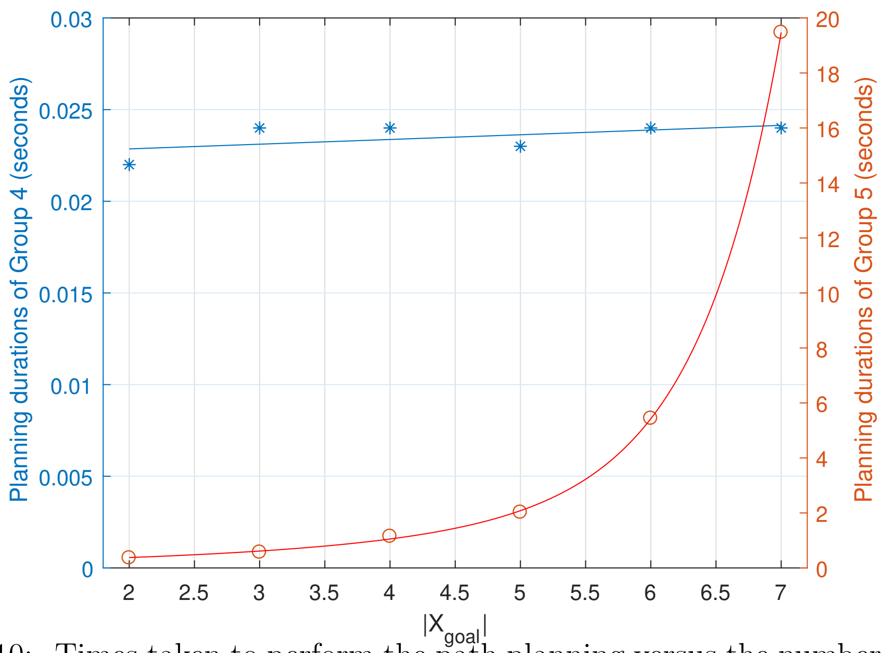


Figure 3.10: Times taken to perform the path planning versus the number of target states for Groups 4 and 5 .

tively, *(ii)* Σ_e, formed by the events listed in Table 3.6, *(iii)* $\Gamma_e : X_e \to 2^{\Sigma_e}$, presented in Table 3.7 for all $x \in X_e$; *(iv)* f_e, defined, for each state $x \in X_e$, according to the active events presented in Table 3.7 — for example, for $x = 1$, transition function f_e is defined for events $m4.5$, $t180$, $t90$ and $t90^-$, and, as it can be seen, with the help of Figure 3.11, $f_e(1, m4.5) = 9$, since command event $m4.5$ models a 4.5m forward movement, $f_e(1, t180) = 3$, since command event $t180$ corresponds to a 180° rotational movement, and $f_e(1, t90) = 2$ (resp. $f_e(1, t90^-) = 4$) since command event $t90$ (resp. $t90^-$) is associated with a 90° counterclockwise (resp. clockwise) rotational movement, and; *(v)* x_{0_e} and X_{m_e}, respectively, which are defined by the planner at each task assigned to the robot, as described in Section 3.3.2. Notice that events smi, $i = 1, 2, \ldots, 13$ are composed by a sequence of translational and rotational movements, where the translational movements do not necessarily correspond to some event md. The composed events were created in order to reduce the number of states of G_e. For instance, if event $sm5$ was not created, 5 states would be added to X_e, 4 between states 9 and 21, just at left of the position of state M_5 and 1 at the position of M_5, in order to allow for the robot to turn 180°. The same would have to be done to states M_2, M_3, M_5 and M_6, adding 20 more states. For states M_1, M_4 and M_7, a single state would have to be added, adding 23 new statesin total.

The values of the weight function w are listed in the third column of Table 3.6, being defined as follows: *(i)* $w(md) = d + 0.01$, where $d \in \{0.75, 4.5, 9.0\}$ is the distance to be traveled; *(ii)* $w(t\theta) = |\theta|/200 + 0.01$, where $\theta \in \{90, 180\}$ is the rotation angle, and; *(iii)* $w(smk) = \sum_i(|\theta_i|/200 + 0.01) + \sum_j(d_j + 0.01)$, $k =$

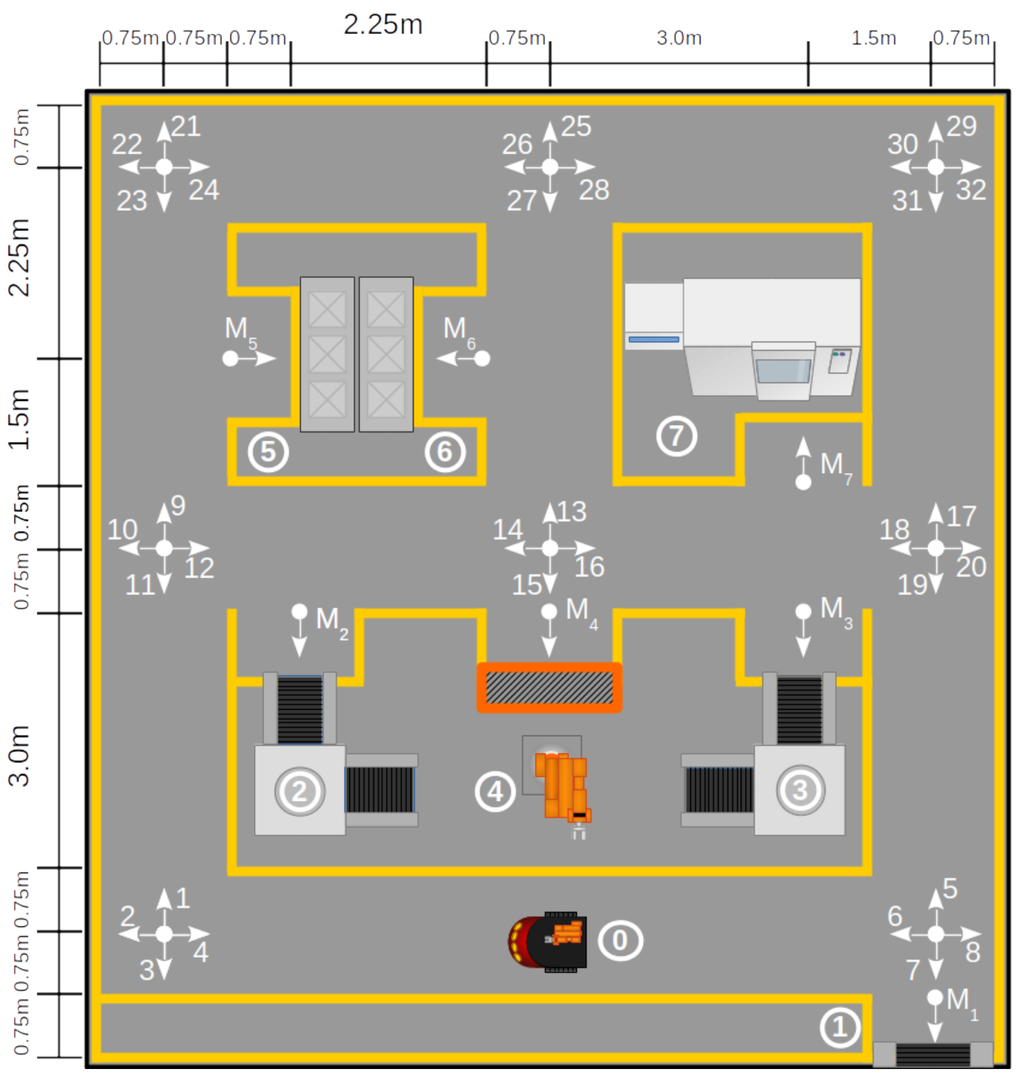


Figure 3.11: Map of the environment. The arrows represent the possible robot poses (states of automaton G_e): the tail indicates the positional coordinate and the direction corresponds to the robot pose.

Table 3.6: Environment automaton events Σ_e.

Event	Description	$w(.)$
$m0.75$	move robot $0.75m$	0.76
$m4.5$	move robot $4.5m$	4.51
$m9.0$	move robot $9.0m$	9.01
$t90$	turn robot 90° (counterclockwise)	0.46
$t90^-$	turn robot −90° (clockwise)	0.46
$t180$	turn robot 180°	0.91
$sm1$	turn robot 180° and move $0.75m$	1.67
$sm2$	move $1.5m$, turn 90° and move $0.75m$	2.73
$sm3$	move $1.5m$, turn −90° and move $0.75m$	2.73
$sm4$	move $2.25m$, turn 90° and move $0.75m$	3.48
$sm5$	move $2.25m$, turn −90° and move $0.75m$	3.48
$sm6$	move $3.00m$, turn 90° and move $0.75m$	4.23
$sm7$	move $3.00m$, turn −90° and move $0.75m$	4.23
$sm8$	turn 180°, move $0.75m$, turn 90° and move $1.5m$	3.64
$sm9$	turn 180°, move $0.75m$, turn −90° and move $1.5m$	3.64
$sm10$	turn 180°, move $0.75m$, turn 90° and move $2.25m$	4.39
$sm11$	turn 180°, move $0.75m$, turn −90° and move $2.25m$	4.39
$sm12$	turn 180°, move $0.75m$, turn 90° and move $3.00m$	5.14
$sm13$	turn 180°, move $0.75m$, turn −90° and move $3.00m$	5.14

$1, 2, \ldots, 13$, where θ_i (resp. d_j) are all rotational (resp. translational) movements present in smk.

In order to illustrate the results presented in this section, we will perform two pairs of simulation, corresponding to two different tasks assigned to the robot, assuming, initially, no obstacle and, then, with some obstacle in the robot trajectory. The simulations were performed using MobileSim 0.7.5 software for a virtual Pioneer P3DX mobile robot.

For the first simulation, we assume that the robot is initially at pose M_2 and has just picked up a processed part in painting machine ② when it receives a request to store the part in shelf ⑤ or ⑥, which is the closest. Since this is a Type 1 task, the mobile robot must determine which shelf is the nearest, compute the shortest path to there, and perform the computed string of command movements. According to Figure 3.8, after event nt is issued, the robot sends event p to the planner, which starts the computation of the optimal path taking into account automaton G_e, the set of target states $X_{goal} = \{M_5, M_6\}$, and the set of blocked transitions $T_b = \emptyset$ (no obstacle is initially assumed to exist). As a consequence, automaton G_p, computed by applying Algorithm 2, is equal to G_e with initial and marked states defined as M_2 and $\{M_5, M_6\}$, respectively. After automaton G_p is computed, Dijkstra's algorithm is applied to find the path from the initial state to one of the marked states of G_p that minimizes cost function J given in Equation (3.2), yielding string

Table 3.7: Active events of the states of G_e.

X_e	Γ_e	X_e	Γ_e
1	$\{t90, t90^-, t180, m4.5\}$	21	$\{t90, t90^-, t180\}$
2	$\{t90, t90^-, t180\}$	22	$\{t90, t90^-, t180\}$
3	$\{t90, t90^-, t180\}$	23	$\{t90, t90^-, t180, m4.5,$
4	$\{t90, t90^-, t180, m9.0\}$		$sm4\}$
5	$\{t90, t90^-, t180, m4.5\}$	24	$\{t90, t90^-, t180, m4.5\}$
6	$\{t90, t90^-, t180, m9.0\}$	25	$\{t90, t90^-, t180\}$
7	$\{t90, t90^-, t180, m0.75\}$	26	$\{t90, t90^-, t180, m4.5\}$
8	$\{t90, t90^-, t180\}$	27	$\{t90, t90^-, t180, m4.5,$
9	$\{t90, t90^-, t180, m4.5, sm5\}$		$sm5\}$
10	$\{t90, t90^-, t180\}$	28	$\{t90, t90^-, t180, m4.5\}$
11	$\{t90, t90^-, t180, m4.5\}$	29	$\{t90, t90^-, t180\}$
12	$\{t90, t90^-, t180, m4.5, sm3\}$	30	$\{t90, t90^-, t180, m4.5\}$
13	$\{t90, t90^-, t180, m4.5, sm4\}$	31	$\{t90, t90^-, t180, m4.5\}$
14	$\{t90, t90^-, t180, m4.5, sm6\}$	32	$\{t90, t90^-, t180\}$
15	$\{t90, t90^-, t180, m0.75\}$	M_1	$\{sm1\}$
16	$\{t90, t90^-, t180, m4.5,$	M_2	$\{sm8, sm11\}$
	$sm6, sm7\}$	M_3	$\{sm1\}$
17	$\{t90, t90^-, t180, m4.5\}$	M_4	$\{sm9, sm12\}$
18	$\{t90, t90^-, t180, m4.5,$	M_5	$\{sm10, sm11\}$
	$sm2, sm3\}$	M_6	$\{sm10, sm11\}$
19	$\{t90, t90^-, t180, m4.5\}$	M_7	$\{sm8, sm13\}$
20	$\{t90, t90^-, t180\}$		

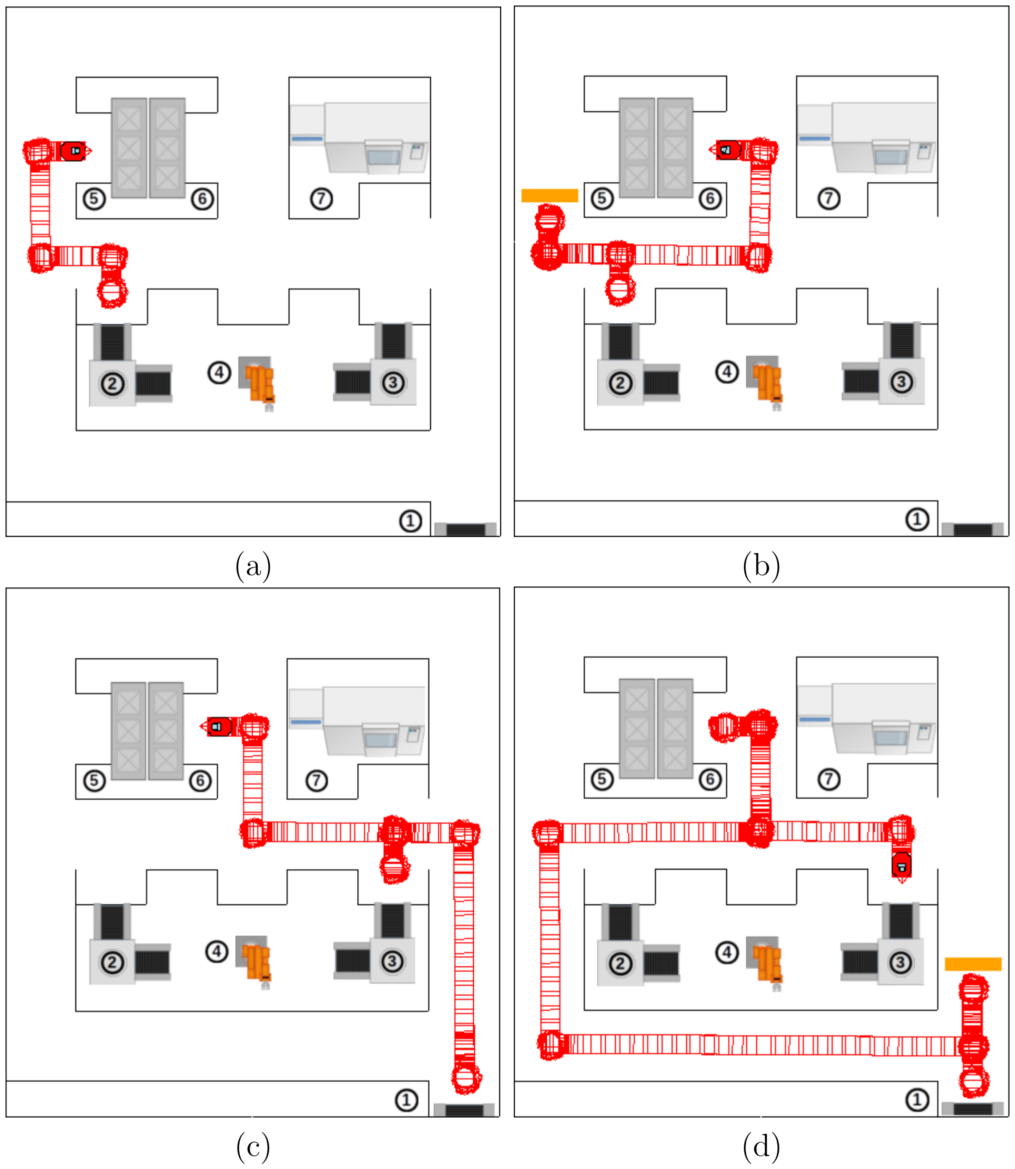


Figure 3.12: Simulation results: Type 1 task from pose M_2 to $X_{goal} = \{M_5, M_6\}$ without obstacle (a), and with obstacle (b), and, Type 2 task from pose M_1 to $X_{goal} = \{M_3, M_6\}$ without obstacle (c), and with obstacle (d).

$s_1 = sm8\,t90^{-}sm5$. Once the planning has been concluded, event pe is issued, and so, the robot executes string s_1 under the control action of supervisor $S_{r_1} \wedge S_{r_2}$, designed according to Section 3.3.3, performing the path shown in Figure 3.12(a).

For the second simulation, we assume that the robot must execute the same task as before but now we added obstacle between the poses of states 9 and M_5, as highlighted in orange in Figure 3.12(b). Since, initially, the robot does not know the obstacle, the string of command events obtained by the path planning procedure is equal to string $s_1 = sm8\,t90^{-}sm5$ computed in the first simulation. However, while the robot is executing the movements associated with command $sm5$, it executes command event msr and, thus, its sonars detects the obstacle that is blocking the path, which makes event od occur. As a consequence, the robot stops immediately after that (event sr). Thus, since the obstacle is permanent, the robot waits until the occurrence of timeout t (defined, empirically, as five seconds), and, in the sequel, executes event ret to reach state 9, which is the last visited state before command event $sm5$ has been executed. The planner then computes a new path adding the blocked transitions to T_b, *i.e*, $T_b = \{(9, sm5, M_5)\}$, where (x, σ, y) denotes a transition of G_e from state x to state y labeled by σ. The new string computed by the planner is equal to $s_2 = t90^{-}m4.5\,t90\,sm4$, which corresponds to the path depicted in Figure 3.12(c). Since this new trajectory is free of obstacles, the robot is then able to arrive at state M_6.

For the third and fourth simulations, we assume that the robot is initially at pose M_1 when it receives a request to pick up some amount of paint at the conveyor belt ① and to deliver part of this paint to machine ③ and to store the remainder in shelf ⑥, which corresponds to a Type 2 task. When there is no obstacle in the environment, the robot performs the path depicted in Figure 3.12(c)by executing the string of command events $s_3 = t180\,m4.5\,t90\,sm2\,sm12\,t90^{-}sm4$ obtained by the planner for $T_b = \emptyset$, $x_{0_e} = M_1$, and $X_{m_e} = \{M_3, M_6\}$. When an obstacle is added to the environment between the poses of states 5 and 17, the robot starts executing s_3, as before, because it is not, at first, aware of the existence of the obstacle, but detects the obstacle while executing the movement corresponding to event $m4.5$. As a consequence, the robot returns to pose 5 and new command string $s_4 = t90\,m9.0\,t90^{-}m4.5\,t90^{-}m4.5\,t90\,sm4\,sm11\,t90\,sm13$ is computed by the planner for $T_b = \{(5, m4.5, 17)\}$, $x_{0_e} = 5$, and $X_{m_e} = \{M_3, M_6\}$, which allows the robot to successfully complete the task.

3.6 Navigation of multiple robots

The new navigation architecture presented in this chapter has been developed for the operation of a single robot. Nonetheless, it may be used to control the navigation of several robots in the same environment, with some restrictions.

3.6.1 Motivating examples

Since a robot does not know the position of other robots, it must see the other robots as mobile obstacles. This solution may work smoothly for some environments, but may lead to problems in others, such as deadlocks when, for example, two robots have to acces the same resource, or when two robots meet facing each other, *etc.* We will now illustrate these two situations.

Let us consider the same hypothetical industrial environment of Figure 3.11. Notice that there is a single transition that reaches state M_1. If one robot (robot R_1) is at the position M_1, and another robot (robot R_2) tries to move to position M_1 to also access the conveyor belt, either one of the following two situations may occur(Figures 3.13(a, b))

1. If robot R_1 is still finishing its task at M_1, then robot R_2 will assume that the access to M_1 is blocked and will not be able to complete its task, as illustrated by Figure 3.13(a), and will request "path unblock"

2. If robot R_1 is trying to leave while robot R_2 is trying to access M_1, robot R_1 will assume that it is trapped and request the unblocking of the path, whereas robot R_2 will assume the access to M_1 is blocked and will treat the task as impossible, as illustrated by Figure 3.13(b).

Both situations described above may happen in environments whose automaton models have states that are reached by a single transition. Without some modifications, the architecture developed in this dissertation may fail when multiple robots are being used simultaneously.

Another problem that may occur when multiple robots are used in this environment is due to each corridor being traveled in both directions. In this regard, if two robots meet facing each other, they will, then, wait for the obstacle to move and, since both are waiting, they will consider each other as static obstacles and will try a different route. It is possible that this sort of encounter may repeat itself in all corridors, turning feasible tasks into impossible ones. To illustrate this situation, let us consider the hypothetical industrial warehouse depicted in Figure 3.14, which is modeled by automaton $G_e = (X_e, \Sigma_e, \Gamma_e, x_{0_e}, X_{m_e})$, where $X_e = \{1, 2, 3, \ldots, 38\}$, Σ_e is formed by the events listed in Table 3.8, $\Gamma_e : X_e \to 2^{\Sigma_e}$, is presented in Table 3.9 for all $x \in X_e$, f_e is defined for each state $x \in X_e$, according to the active events presented in Table 3.9 — for example, for $x = 1$, transition function f_e is defined for events $m2.0$ and $t180$, and, as it can be seen, with the help of Figure 3.14, $f_e(2, m2.0) = 5$, since command event $m2.0$ models a 2.0m forward movement and

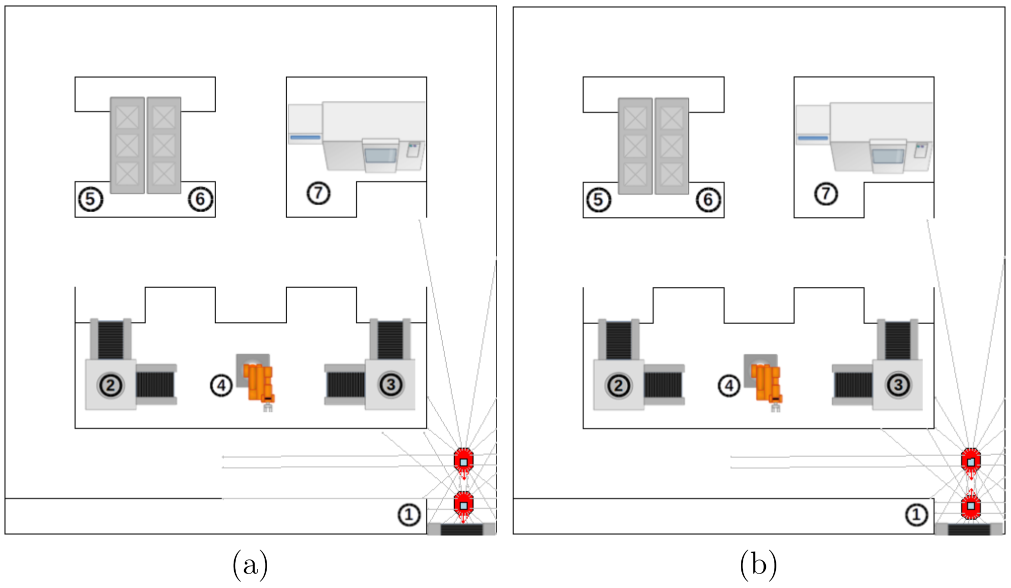


Figure 3.13: Problems that may arise when automaton G_e that models the environment has states that are reached by a single transition: when a robot tries to reach $M1$ while another robot is using it, (a), and when a robot tries to reach $M1$ while another tries to leave it(b) .

Table 3.8: Environment automaton events Σ_e.

Event	Description	$w(.)$
$m2.0$	move robot $2.0m$	2.01
$m2.5$	move robot $2.5m$	2.51
$m3.0$	move robot $3.0m$	3.01
$t90$	turn robot 90° (counterclockwise)	0.46
$t90^{-}$	turn robot $-90°$ (clockwise)	0.46
$t180$	turn robot 180°	0.91

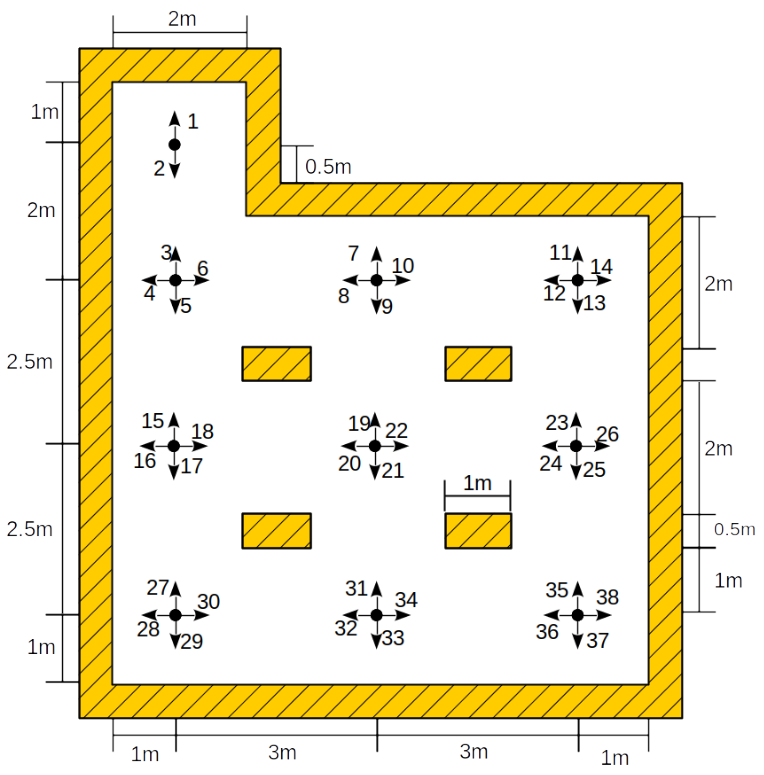


Figure 3.14: Map of the environment. The arrows represent the possible robot poses (states of automaton G_e): the tail indicates the positional coordinate and the direction corresponds to the robot pose.

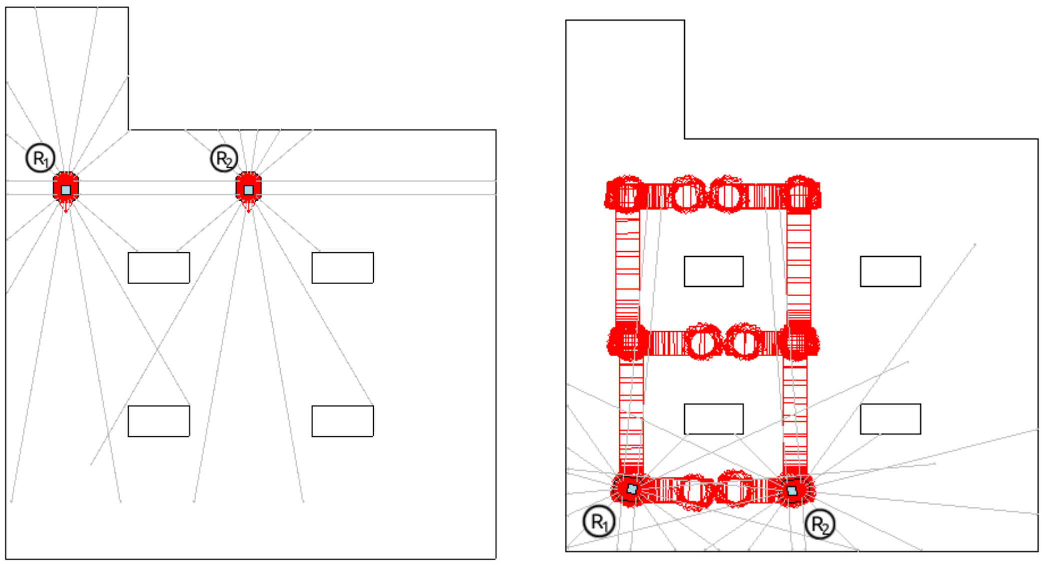
$f_e(1, t180) = 2$, since command event $t180$ corresponds to a 180° rotational movement, and, finally, x_{0_e} and X_{m_e}, are respectively defined by the planner at each task assigned to the robot, as described in Section 3.3.2.

Assume that there are two robots R_1 and R_2 operating in the environment of Figure 3.15a, whose initial poses are at the positions of states 5 and state 9, respectively, as shown in Figure 3.15a. Now, suppose that robot R_1 receives a task to go to the position corresponding to state 9 and robot R_2 receives a task that demands it to go to the position corresponding to state 5. Without modifications to the architecture presented before, the planner for robot R_1 calculate sthe shortest path and obtains $t90\, m3.0 t90^-$, while the planner for robot R_2 obtains $t90^-\ m3.0\ t90$.

Both robots will try to follow the planned shortest paths, and will meet facing each other somewhere in the top horizontal corridor, detecting each other and stopping. As both robots wait for each other to move away, they both consider the other robot as a static obstacle. The robots will return to their previous states, state 6 for R_1 and state 8 for robot R_2. They will then remove the blocked transitions and will recalculate the shortest path; which will lead again and again to similar encounters between the two robots. Thus, both tasks will be deemed impossible to finish, when, in fact, both were feasible. The final result of such a conundrum can be seen in Figure 3.15b.

Table 3.9: Active events of the states of G_e.

X_e	Γ_e	X_e	Γ_e
1	$\{t180\}$	20	$\{t90, t90^-, t180, m3.0\}$
2	$\{t180, m2.0\}$	21	$\{t90, t90^-, t180\, m2.5\}$
3	$\{t90, t90^-, t180\, m2.0\}$	22	$\{t90, t90^-, t180, m3.0\}$
4	$\{t90, t90^-, t180\}$	23	$\{t90, t90^-, t180, m2.5\}$
5	$\{t90, t90^-, t180, m2.5\}$	24	$\{t90, t90^-, t180, m3.0\}$
6	$\{t90, t90^-, t180, m3.0\}$	25	$\{t90, t90^-, t180, m2.5\}$
7	$\{t90, t90^-, t180\}$	26	$\{t90, t90^-, t180\}$
8	$\{t90, t90^-, t180, m3.0\}$	27	$\{t90, t90^-, t180, m2.5\}$
9	$\{t90, t90^-, t180, m2.5\}$	28	$\{t90, t90^-, t180\}$
10	$\{t90, t90^-, t180, m3.0\}$	29	$\{t90, t90^-, t180\}$
11	$\{t90, t90^-, t180\}$	30	$\{t90, t90^-, t180, m3.0\}$
12	$\{t90, t90^-, t180, m3.0\}$	31	$\{t90, t90^-, t180, m2.5\}$
13	$\{t90, t90^-, t180, m2.5\}$	32	$\{t90, t90^-, t180, m3.0\}$
14	$\{t90, t90^-, t180\}$	33	$\{t90, t90^-, t180\}$
15	$\{t90, t90^-, t180, m2.5\}$	34	$\{t90, t90^-, t180, m3.0\}$
16	$\{t90, t90^-, t180\}$	35	$\{t90, t90^-, t180, m2.5\}$
17	$\{t90, t90^-, t180, m2.5\}$	36	$\{t90, t90^-, t180, m3.0\}$
18	$\{t90, t90^-, t180, m3.0\}$	37	$\{t90, t90^-, t180\}$
19	$\{t90, t90^-, t180, m2.5\}$	38	$\{t90, t90^-, t180\}$



(a) (b)

Figure 3.15: Another problem that may arise when the automaton G_e has corridors that allow robots to pass in both directions by its corridors.

We will now suggest how the problem of two robots that met facing each other can be solved . Notice that without a way for the robots to infer that they encountered another robot and a way to decide collectively which robot will follow the already calculated path and which will take another route, the solution may incur into other problems. A first approach would be to distribute the tasks in such a way to not ask multiple robots to move through the same places at the same time. This solution would require greater changes, as it would require either collective trajectory planning or a collective scheduling method.

A second and more straightforward approach is to create a method to decide which robot must recalculate its trajectory and which one should wait until the path is free to resume its previously calculated path. This can be done either by a local decision made between the robots themselves or by another agent, such as a master robot or a server. Since it is already assumed that there are external agents that assign tasks to the robots, we will assume that a server receives obstacle detection signals from all robots together with the information of their current positions and their distances to their task goals. If they are at states whose poses imply they have met head-on, the server will identify that the robots have met each other and will ask for the robot that is further away from its goal to recalculate its path, while the other robot will be allowed to continue following the previously planned trajectory. This solution is easier in terms of adapting the architecture developed in previous sections, since we only need to change a few states, events and transitions of automata G_{r_s} and $H_{spec,3}$. The new G_{r_s} and $H_{spec,3}$, now named $G_{r_s,i}$ and $H_{spec,3,i}$, are shown in Figures 3.17 and 3.18, respectively.

3.6.2 A new navigation architecture for multiple robots

The architecture proposed in previous sections, displayed in Figure 3.4, does not reflects the changes necessary for the navigation of multiple robots. We, then, propose the adapted architecture for the navigation of n robots shown in Figure 3.16. Notice that the robots share the external agents, but the Planner modules are particular to each robot, even if they are run externally. The new and updated events are shown in Table 3.10.

Comparing Figures 3.2 and 3.17, we can see that events rs, msr and ssr remain, but event t has been removed, since it is now an event issued by the server. Event od_i means that i-th robot has detected an obstacle and sent a message to the server, together with its current pose and its distance to its objective. Event $\overline{od_i}$ is a signal sent by the i-th robot to the server after it stops detecting obstacles. Comparing now Figures 3.7 and 3.18, we see that the event set Σ_e of the environment automaton event set has not changed, and that event t has been split in two different events,

Table 3.10: Events that are modified.

Event	Description	Controllable
Σ_e	set of environment automaton events	✓
sr	stop robot signal	✓
nt_i	new task received by the i-th robot	$\times$
tc_i	the i-th robot reports task completion	✓
ru_i	the i-th robot requests for unblocking the last path	✓
pf_i	the last path of the i-th robot is free	$\times$
rs	robot stopped	$\times$
msr	obstacle sensor information request while the robot is moving	✓
ssr	obstacle sensor information request when the robot has stopped	✓
od_i	The i-th robot detects an obstacle and sends a message to the server with the its pose and its distance from the task objective	$\times$
$\overline{od}_i$	message sent by the i-th robot to the server, to inform it does not detect obstacles	$\times$
$t_{w,i}$	signal sent by the server, allowing the i-th robot to return to its previous state pose	$\times$
$t_{d,i}$	signal sent by the server to the i-th robot, telling it to resume the previously calculated trajectory	$\times$

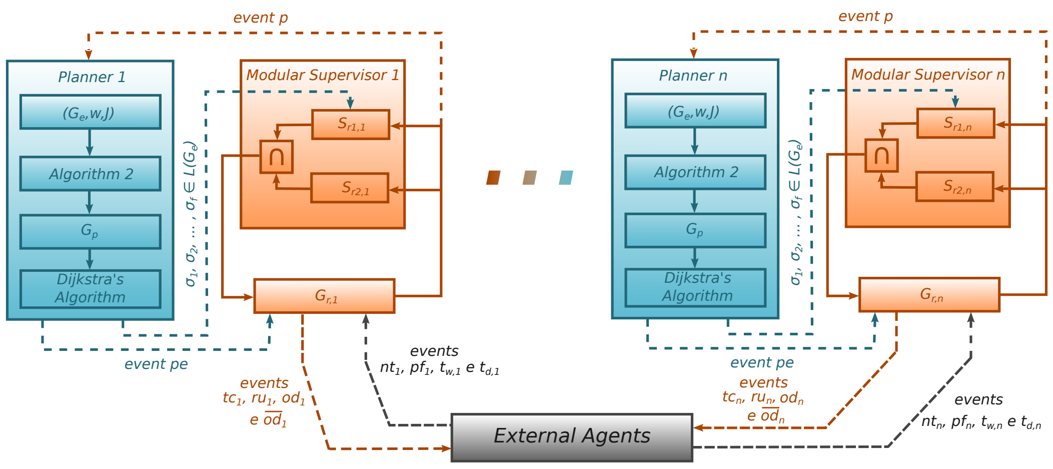

event p
Planner 1
(G_e,w,J)
Algorithm 2
G_p
Dijkstra's Algorithm
Modular Supervisor 1
$S_{r1,1}$
$\cap$
$S_{r2,1}$
$\sigma_1, \sigma_2, \dots, \sigma_f \in L(G_e)$
$G_{r,1}$
event pe
events tc_1, ru_1, od_1 e $\overline{od}_1$
events nt_1, pf_1, $t_{w,1}$ e $t_{d,1}$
External Agents
event p
Planner n
(G_e,w,J)
Algorithm 2
G_p
Dijkstra's Algorithm
Modular Supervisor n
$S_{r1,n}$
$\cap$
$S_{r2,n}$
$\sigma_1, \sigma_2, \dots, \sigma_f \in L(G_e)$
$G_{r,n}$
event pe
events tc_n, ru_n, od_n e $\overline{od}_n$
events nt_n, pf_n, $t_{w,n}$ e $t_{d,n}$

Figure 3.16: The modified architecture, adapted for the navigation of n robots.

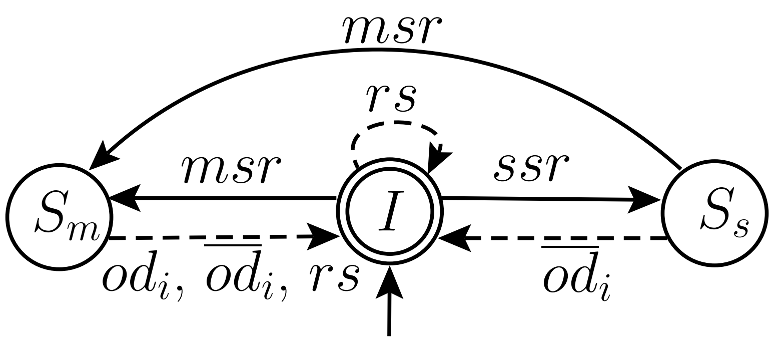


Figure 3.17: The modified robot sensing module $G_{r_s,i}$. Dashed lines represent transitions labeled with uncontrollable events.

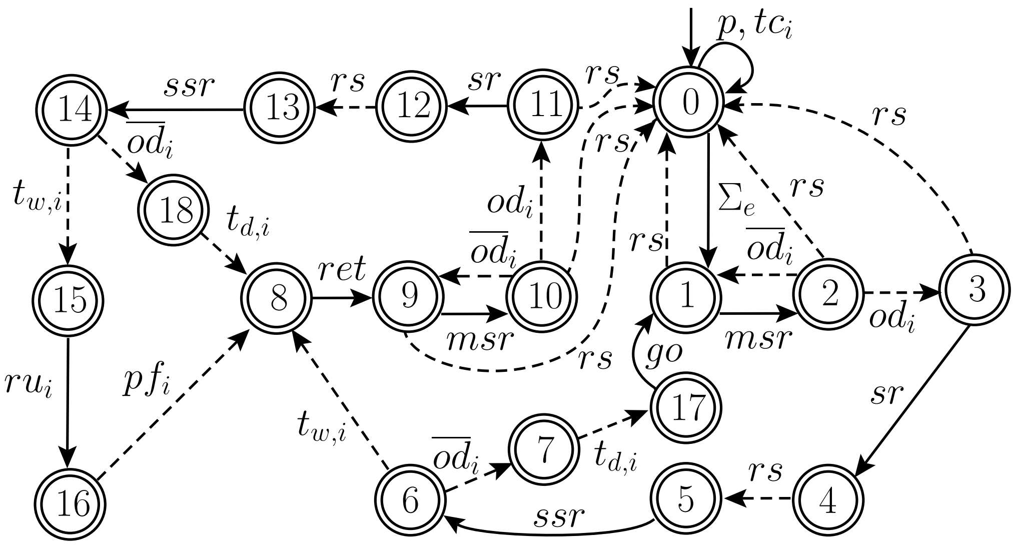


Figure 3.18: Automaton $H_{spec,3,i}$ used to synthesize supervisor S_{r_2} for the navigation of multiple robots. Dashed lines represent transitions labeled with uncontrollable events.

$t_{w,i}$ and $t_{d,i}$; event $t_{w,i}$ is a signal sent by the server, allowing the i-th robot to return to its previous state pose since it is no longer necessary to wait for the obstacle in front of it to move away, and event $t_{d,i}$ is a signal sent by the server to the i-th robot, telling it to resume the previously calculated trajectory. Events nt_i, tc_i, ru_i and pf_i are now labeled to indicate the event either was sent by the i-th robot or it is destined to it.

Notice that since events $t_{w,i}$ and $t_{d,i}$, which replace event t, are issued by the server and so, the removal of event t from the event set of G_{r_s} with all associated transitions may cause a deadlock in state S_s. Thus, in order to avoid such an issue, a transition from state S_s to state S_m, labelled by ssr, is added, as one can see in Figure 3.17.

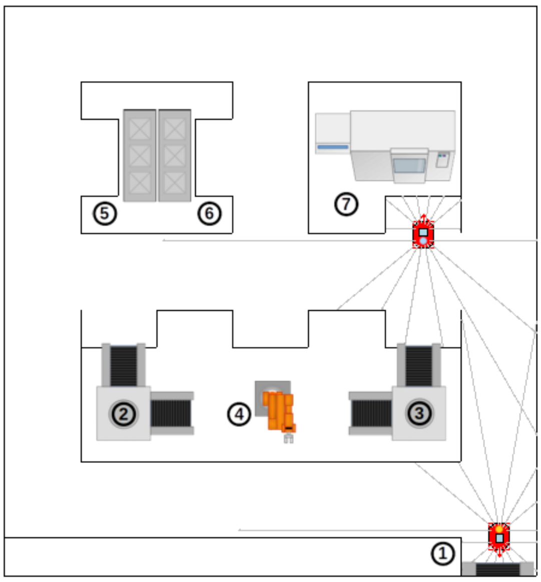
That way, every robot will have its own Modular Supervisor. Regarding the Planning, it can be done locally or by the server; in the former, each robot must have its own Planner Module, an so, for clarity, although automata G_{r_m} $G_{r_{tm}}$ and

$H_{spec,2}$ do not change, they will be renamed as $G_{r_m,i}$ $G_{r_{tm},i}$ and $H_{spec,2,i}$. Together with automata $G_{r_s,i}$ and $H_{spec,3,i}$, they will from the modular supervisor. It is worth mentioning that $H_{spec,1,i}$ defines the best trajectory that will be computed by the planner.

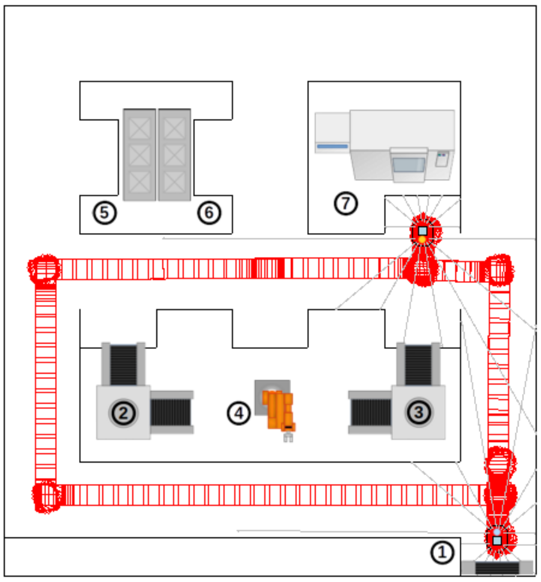
Notice that two new states, 17 and 18 have been added to $H_{spec,3,i}$. To understand the idea behind their introduction, assume now that robots, i and j, have detected each other, sending events od_i and od_j to the server. Their respective automata $H_{spec,3,i}$ and $H_{spec,3,j}$ will be both at state 3. Then, after the occurrence of events sr and rs, both robots will have stopped and, with the occurrence of event ssr, will read their sonars, reaching state 6. Assume that the j-th robot is further away from its task objective than the i-th robot. Then, the server will send a $t_{w,j}$ signal, telling the j-th robot to stop waiting, and thus, $H_{spec,3,j}$ will reach state 8, from which it will return to its previous pose. The i-th robot, on the other hand, will wait for the j-th to move away and, when this happens, it will receive an event $t_{d,i}$ from the server, and $H_{spec,3,i}$ moves to state 17, from which it resumes the navigation of the previously calculated trajectory.

There are several possible methods for the server to know when the j-th robot is at a safe distance from the i-th robot in order to send a $t_{d,i}$ event. A first solution would be the server estimating the largest time for the j-th robot to move away from the path of the i-th robot based on the information sent by the j-th robot when it detected the i-th robot and on the weight of the active events of previously reached state of the environment automaton. Another solution, which is the one adopted in this work, is to assume that the server always knows the position of every robot. This way, the server just needs to keep a set of "unsafe" states for the j-th robot for a given state x of the i-th robot. At least a single state of G_e must be "safe", so the maximum number of "unsafe" states must be equal to $||X_e|| - 2$ in order for the i-th to be able to resume its previously calculated path. Since a non-redundant path may consist of at most of $||X_e|| - 1$ steps, the total cost of checking if the j-th robot has left the set of "unsafe" states is, at most, of $(||X_e|| - 1) * (||X_e|| - 2)$, thus having a complexity of $O(||X_e||^2)$.

Now assume that, while the i-th robot awaits, the j-th robot tries to return to the previously reached environment pose, but it detects another robot: say the k-th robot. The j-th and the k-th robots will send, respectively, od_j and od_k signals to the server, which will infer they encountered each other head-on. State 11 is the current state of $H_{spec,3,j}$ after it detects the k-th and after the occurrence of sr and rs, it will reach state 13, meaning that it has stopped. The current state of $H_{spec,3,k}$ after the occurrence of od_k sr and rs is state 5. Thus, both robots will activate their sonars to detect if the other has moved away. Since the j-th robot is already trying to return to its previously reached pose in the environment, it cannot return and



(a) Initial configuration for the first test, with robot R_2 at state M_7 and robot R_1 at M_1.



(b) Final configuration of the experiment, with the robots trajectories highlighted in red.

Figure 3.19: Results of the test with two robots..

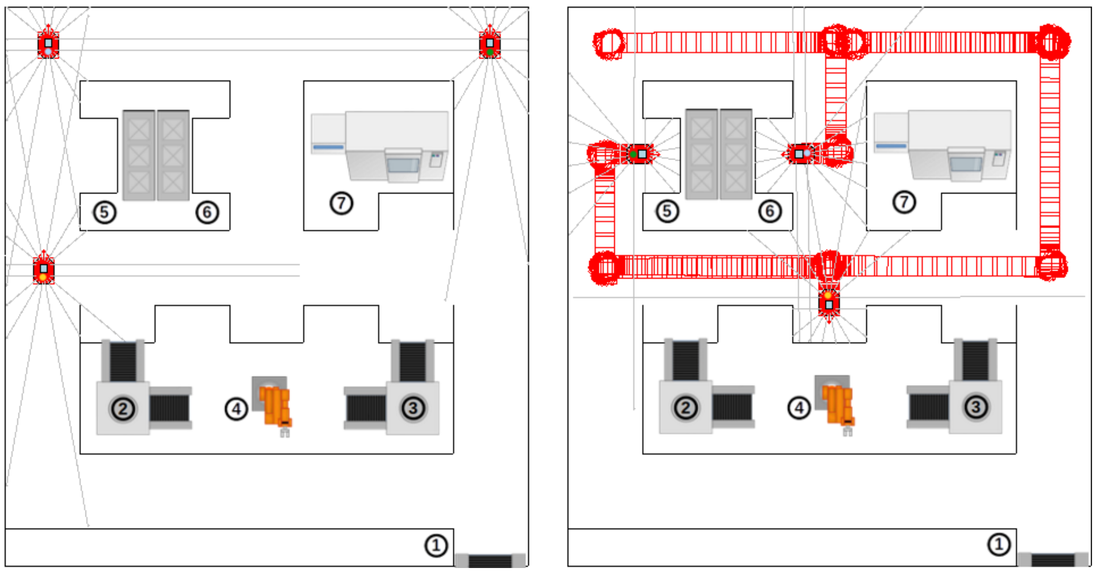
recalculate is trajectory, so, regardless of the proximity of the k-th of its objective, the server will send a $t_{w,k}$ signal, ordering the k-th robot ot stop waiting and to recalculate its trajectory. The j-th robot, on the other hand will wait for the k-th robot to move away, and, when its sonars stop detecting it, will send a $\overline{od}_j$ signal to the server, reaching state 18. When the k-th robot has moved away from the j-th robot, the server will send a $t_{d,j}$ signal, so that the j-th robot stops waiting and returns to its previously reached environment pose.

In order to validate this new architecture, we will run in the sequel, tests for multiple robots.

Test with two robots

The first test consists of two robots R_1, marked with a yellow dot, and R_2, marked with a blue dot. Robot R_1 task is to receive raw materials from the conveyor belt at M_1, that need to be processed by the CNC machine at M_7, as shown in Figure 3.19(a). Robot R_2 task, on its turn, is to receive a processed part from the CNC machine at M_7 and to deliver it to the conveyor belt for packaging, as shown in Figure 3.19(b).

Immediately after receiving the task, robot R_2 begins the planning for the shortest path from M_7 to M_1, obtaining string $s_2 = sm8\ t90^{-}\ m4.5\ m0.75$. It starts following the calculated trajectory, but robot R_1 finishes receivig the raw materials and calculates the shortest path from M_1 to M_7, which is given by the string $s_1 = sm1\ m4.5\ t90\ sm3$.



(a) Initial configuration for the second test, with robot R_1 at state 9, robot R_2 at 21 and robot R_3 at state 29.

(b) Final configuration of the experiment, with the robots trajectories highlighted in red.

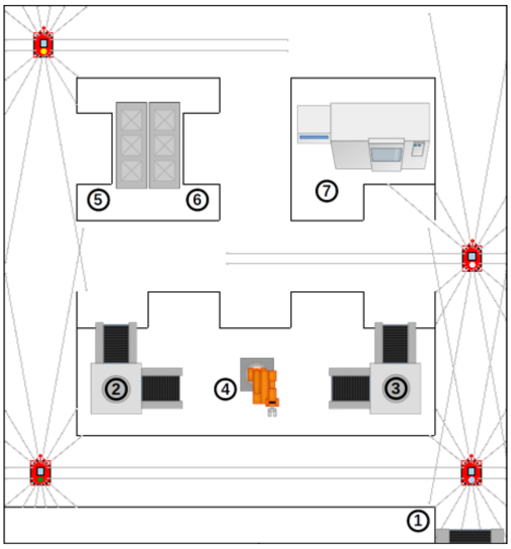
Figure 3.20: Results of the test with three robots.

As shown in Figure 3.19(b), robot R_2 executes the movements associated with events $sm8\ t90^-$ without any problems, but when it is about to finish executing the movement associated with $m4.5$, it encounters R_1. The server, then, determines that R_1 and R_2 have encountered each other, and, since R_2 is closer to its goal, it will wait for a *timeout* signal from the server, for it to move away. Meanwhile, robot R_1 will, turn 180°, move to state 5, turn 180° again and recalculate the shortest path, now considering the previous path blocked by R_2. This way, the new trajectory obtained is given by string $s_3 = t90\ m9.0\ t90^-\ m4.5\ t90^-\ m4.5\ sm2$.

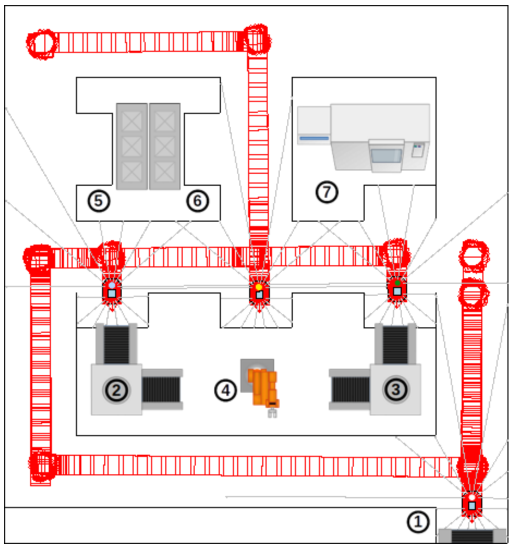
After robot R_1 turns 90° and moves $9m$, robot R_2 receives a *timeout* signal from the server and finishes the previously calculated path, moving the remainder of the $4.5m$ and, then, the $0.75m$ to reach M_1. Since no other robots or obstacles are not present in this test, R_1 finishes the second calculated trajectory and reaches M_7 without further problems, as shown in Figure 3.19(b).

Test with three robots

The second test consists of three robots: R_1, marked with a yellow dot, R_2, marked with a blue dot and R_3, marked with a green dot; all of them are shown at their initial positions in Figure 3.20(a), states 9, 21 and 29, respectively. Robot R_1 must go to the disposal area of the robot arm at M_4 to clear it, while robot R_2 must go to the shelf at M_6 and pick its stored materials. Robot R_3 too must pick its stored materials, but at M_5. After receiving their tasks, they all calculate the shortest paths, which are $t90\ m4.5\ t180\ m0,75$ for R_1, $t90^-\ m4.5\ t90^- sm7$ for R_2,



(a) Initial configuration for the second test, with robot R_1 at state 21, robot R_2 at 5, robot R_3 at state 1 and robot R_4 at state 17.



(b) Final configuration of the experiment, with the robots trajectories highlighted in red.

Figure 3.21: Results of the test with four robots.

and $t90$ $m4.5$ $m4.5$ $t90$ $sm6$ for R_3.

All robots start to follow the calculated paths. Robot R_1 reaches M_4 without problem, but robots R_2 and R_3 encounter each other while trying to reach the position of states 28 and 26, respectively. They detect each other and the server infers that the two robots have met head-on. Since robot R_2 is closer to its objective, robot R_3 has to recalculate its path while R_2 waits for it to move away. R_3 return to the position of state 30 and recalculates the shortest path, obtaining $t90^{-}1$ $m4.5$ $t90^{-}$ $m4.5$ $m4.5$ $t90^{-}$ $sm7$. Robot R_2 moves after robot R_3 executes $t90$ and $m4.5$, reaching M_6 without further problems. Since all the other robots are at their objectives and there are no static obstacles, robot R_3 reaches M_5, as shown in the left image of Figure 3.20.

Test with four robots

The third and final test consists of robots R_1, R_2, R_3 and R_4, marked with yellow, blue, green and white dots, respectively. The initial positions are those of states 21, 5, 1 and 17, respectively, for robots R_1, R_2, R_3 and R_4, as shown in Figure 3.21(a). Robot R_1 must reach state M_4 , robot R_2 must move to state M_2, robot R_3 must reach state M_3 and robot 4 must go to M_1. After receiving the tasks, each robot calculates the shortest path to its objective, which are $t90^{-}$ $m4.5$ $t90^{-}$ $m4.5$ $m0.75$, $m4.5$ $t90$ $m4.5$ $sm6$, $m4.5$ $t90^{-}$ $m4.5$ $sm7$ and $t180$ $m4.5$ $m0.75$, for robots R_1, R_2, R_3 and R_4, respectively. After that, all robots begin to move, trying to reach their destinations, but robots R_2 and R_4 will meet facing each other, while trying to reach states 17 and 7, respectively. Since robot R_4 is closer to its objective, R_2 returns to state 5 and recalculates the best path, obtaining $t90$ $m9.0$ $t90^-$ $m4.5$ $t90^-$ $sm3$. While R_4 and R_2 encountered each other, robots R_3 detects robot R_1, but since robot R_1 was fast enough, moving into M_4 before the timeout happened, R_3 detects a free path and resumes its path, reaching M_3 without further encounters. Robots R_2 and R_4 too reach their goals without further hassles.

Chapter 4

Conclusions and Future Works

In this dissertation, a new automaton-based method for mobile robot navigation in Industry 4.0 environments was proposed. All components of the architecture were made to be modular, making easier to modify them when needed. The free behavior of robots, the environment and all specifications were modeled by automata, and the theory of supervisory control of discrete event systems was used to obtain a modular supervisory controller that ensures the correct navigation of the robot in the presence of unpredictable obstacles, including other robots.The proposed approach provides a general modeling framework that allows the implementation of several specifications by means of modules that depend solely on the type of task the robots will perform and on the environment; thus, making easier to modify the specification automata, the Planner Algorithm and the environment automaton model, without changing all other components. The results obtained for the navigation of a single robot have shown the efficiency of the presented architecture, always obtaining the shortest path not only for tasks with a single objective, but also for tasks with multiple objectives even in the presence of unknown obstacles.

Comparing the results obtained in this dissertation with [17, 19–23], we can see that none of the later present a thorough computational analysis of the algorithms and architectures proposed; only [23] provides a computing time analysis. This dissertation, on the other hand, provides both analyses: Big O asymptotic growth order and computing time. In addition, none of those works addresses the problem of obtaining the shortest path for multiple objectives with no specific order, which is addressed in this dissertation as Type 2 tasks, where a Big O asymptotic growth order analysis has been performed for the proposed algorithm. It is worth mentioning that HILL and LAFORTUNE [23] addressed a similar problem to that considered here, but it has an automaton that acts like a specification for the tasks precedence, and uses Dijkstra's algorithm to obtain the smallest total cost.

The results of the experiments with multiple robots navigation demonstrated that the architecture developed for the navigation of a single robot could be successfully adapted for the navigation of multiple robots, with just minor changes. It also scales very well, although additional compositions must be made in order to obtain a trajectory for all robots. Nonetheless, this computational efficiency comes at the cost of robots not cooperating, which may frequently meet other robots head-on and having to solve that problem. When that happens, one robot returns to its previously reached pose in the environment, recalculates a new trajectory and executes a probably longer route, while the other robot keeps waiting for its path to be free, thus wasting time and battery life. In this regard, the results of the navigation itself are somewhat similar to those results obtained in [33], where robots individually plan their own trajectories and, when they eventually encounter other robots, they decide, based on a dynamic priority attribution, which route the conflicting robots shall take. The main differences between this solution and the one proposed in this dissertation, are that there are no servers involved, the robots decide themselves their priorities and what routes they will take in order to solve the conflict. In addition, they consider the possibility of a conflict between several robots, while this dissertation considers encounters between two robots only; if more robots detect the two conflicting robots, they will treat them as unknown obstacles.

The architecture proposed in [22] and subsequently in [23], differs a lot from the adaptation of the single to multiple robot navigation architecture proposed in this dissertation, since the planning is performed in two different levels: (i) a global one, carried out offline, that determines the order at which the tasks should be performed using an abstraction of the global model and (ii) a local high-level controller that tracks the state of all modular supervisor as the robot progresses through the chosen global plan, updating the states each time a command is given or a task finished event is observed. If all supervisors allow the next start event in high-level plan, it becomes the next command. If that is no the case, the local planner has to generate a limited horizon plan that reaches the next start event for every robot. Since the plant model itself does not allow robot to try entering a region occupied by another robot, such conflicts do not happen, and the main focus is not to violate the safety specifications while reaching vivacity, *i.e*, always obtaining at least one path to goal states.

The architecture presented in [24] is somewhat simpler than the one presented in [23] and also employs a global online supervisory control with a limited horizon, whereas our work generate all models, supervisors and trajectories offline, only changing the environment automaton and the calculated trajectory in the face of unknown obstacles. It also tries to avoid deadlocks, which in [24] are defined as robot encounters, but it does not provide a formal proof for the minimal number of steps for the truncation of the automata of the robots necessary to ensure the liveness of the system; it is assumed that deadlocks are avoided for a large enough number of steps, which holds in the experiments, since no robot encounters another robot head-on. Nonetheless, although effective, the architecture proposed does not take unknown obstacles into account and does not scale well as the number of robots in the environment increases, since the complexity of the proposed architecture is $O(k^{2n})$, where k is the largest number of states of the robot models automata and n is the number of robots.

Analyzing all present results, one can see that the adaptation of the single robot navigation framework will not yield the best results regarding the time of task completion, since robots can interrupt other robots trajectories. This problem only gets worse as the quantity of robots operating in the environment increase, making clear that some type of coordination is necessary. On the other hand, the results of the other works have shown that global supervisory control architectures are more computationally expensive than decentralized ones, which may become prohibitive as the number of robots increases. Thus, future research must seek to develop an architecture that incorporates the advantages from centralized and decentralized approaches, while reaching a compromise with the disadvantages, such as the solution developed in [34]. Although, in [34], neither supervisory control nor automata models are developed, it incorporates the best of decentralized and centralized approaches. First, robots calculate the optimal trajectories from a given starting points to their respective objective points. After all optimal trajectories are computed, a global scheduler use CPSM (Critical Path Schedule Method) to schedule the robot actions, creating dummy tasks so that a robot can wait in order to not interfere with other robot navigations. This approach seems quite promising, although it does not allow modification on the calculated schedule, *i.e*, it does not account for new tasks arriving after a schedule is obtained, namely, if we employed this method without modifications, robots that finished their tasks earlier would have to wait for the completion of all scheduled tasks before being able to execute new ones.

Another future research direction is to deal with real robot cooperation, since in this dissertation robots simply operate concurrently in the same environment competing for a limited resource – the poses of the states of the environment automaton – while carrying out isolated tasks. This way, adding cooperation between the robots to carry out a task would make the architecture developed here more flexible and powerful. The theory developed in [18] treats the planning problem as a reachability problem in a network of automata, where agents are modeled by one automaton, whose events are subsets of all possible action event set, representing which action an agent can perform. The work then uses formal language theory, distributed constraint solving, distributed optimization and weighted automata calculus to solve the best plan problem. However, the most important contributions that could be added to the architecture developed here are the distributed planning and the interaction graph, that ensure that the local plans are compatible by synchronizing actions shared by the robots.

So, everything considered, the single robot navigation architecture developed in this dissertation is very efficient, although could be better extended to multiple robots, having a lot of room to improve.

Bibliographic References

[1] GILCHRIST, A. *Industry 4.0: the industrial internet of things.* 1st ed. Berkeley, Apress, 2016.

[2] WANG, S., WAN, J., ZHANG, D., LI, D., ZHANG, C. "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination", *Computer Networks*, v. 101, pp. 158–168, 2016.

[3] JIRKOVSKY, V., OBITKO, M., MARIK, V. "Understanding Data Heterogeneity in the Context of Cyber-Physical Systems Integration", *IEEE Transactions on Industrial Informatics*, v. 13, n. 2, pp. 660–667, 2017.

[4] DA XU, L., HE, W., LI, S. "Internet of things in industries: A survey", *IEEE Transactions on Industrial Informatics*, v. 10, n. 4, pp. 2233–2243, 2014.

[5] WAN, J., TANG, S., LI, D., WANG, S., LIU, C., ABBAS, H., VASILAKOS, A. V. "A Manufacturing Big Data Solution for Active Preventive Maintenance", *IEEE Transactions on Industrial Informatics*, v. 13, n. 4, pp. 2039–2047, 2017.

[6] KEHOE, B., PATIL, S., ABBEEL, P., GOLDBERG, K. "A survey of research on cloud robotics and automation", *IEEE Transactions on Automation Science and Engineering*, v. 12, n. 2, pp. 398–409, 2015.

[7] SIEGWART, R., NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots.* London, England, Bradford Book, 2004. ISBN: 026219502X.

[8] NAKHAEINIA, D., TANG, S., NOOR, S. M., MOTLAGH, O. "A review of control architectures for autonomous navigation of mobile robots", *International Journal of Physical Sciences*, v. 6, n. 2, pp. 169–174, 2011.

[9] WONHAM, W. M., RAMADGE, P. J. "Modular supervisory control of discrete-event systems", *Mathematics of Control, Signals, and Systems (MCSS)*, v. 1, n. 1, pp. 13–30, 1988.

[10] GONZALEZ, A. G. C., ALVES, M. V. S., VIANA, G. S., CARVALHO, L. K., BASILIO, J. C. "Supervisory Control-Based Navigation Architecture: A New Framework for Autonomous Robots in Industry 4.0 Environments", *IEEE Transactions on Industrial Informatics*, v. 14, n. 4, pp. 1732–1743, April 2018.

[11] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems.* 2nd ed. New York, Springer, 2008.

[12] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., SINNAMOHIDEEN, K., TENEKETZIS, D. "Diagnosability of discrete-event systems", *IEEE Transactions on Automatic Control*, v. 40, n. 9, pp. 1555–1575, 1995.

[13] RAMADGE, P. J. G., WONHAM, W. M. "The control of discrete event systems", *Proceedings of the IEEE*, v. 77, n. 1, pp. 81–98, 1989.

[14] MOREIRA, M. V., BASILIO, J. C. "Bridging the gap between design and implementation of discrete-event controllers", *IEEE Transactions on Automation Science and Engineering*, v. 11, n. 1, pp. 48–65, 2014.

[15] ANTUNES, I., CARVALHO, L. K., BASILIO, J. C. "A stochastic Petri net model for simulation-based performance analysis of public bicycle sharing systems". In: *Proc. International Conference on Automation Science and Engineering (CASE),* Fort Worth, TX, USA, pp. 433–439, 2016.

[16] LIU, B., GHAZEL, M., TOGUYÉNI, A. "Model-Based Diagnosis of Multi-Track Level Crossing Plants", *IEEE Transactions on Intelligent Transportation Systems*, v. 17, n. 2, pp. 546–556, 2016.

[17] ROSZKOWSKA, E. "Supervisory control for multiple mobile robots in 2D space". In: *Proc. of the International Workshop on Robot Motion and Control, (RoMoCo),* Bukowy Dworek, Poland, pp. 187–192, 2002.

[18] FABRE, E., JEZEQUEL, L. "Distributed optimal planning: an approach by weighted automata calculus". In: *Proc. of the 48th IEEE Conference on Decision and Control held jointly with the 28th Chinese Control Conference (CDC/CCC),* Shanghai, China, pp. 211–216, 2009.

[19] KLOETZER, M., MAHULEA, C. "Multi-robot path planning for syntactically co-safe LTL specifications". In: *Proc. of the International Workshop on Discrete Event Systems (WODES),* Xi'an, China, pp. 452–458, 2016.

[20] IQBAL, J., KHAN, S., ZAFAR, N., AHMAD, F. “Modeling Supervisory Control of Autonomous Mobile Robots using Graph Theory, Automata and Z Notation”, *Journal of American Science*, v. 8, n. 12, pp. 799–804, 2012.

[21] KOŠECKÁ, J., BAJCSY, R. “Discrete Event Systems for autonomous mobile agents”, *Robotics and Autonomous Systems*, v. 12, pp. 187–198, 1994.

[22] GORYCA, J., HILL, R. “Formal synthesis of supervisory control software for multiple robot systems”. In: *Proc. of the American Control Conference (ACC),* Washington, DC, USA, pp. 125–131, 2013.

[23] HILL, R., LAFORTUNE, S. “Scaling the formal synthesis of supervisory control software for multiple robot systems”. In: *Proc. of the American Control Conference (ACC),* Seattle, WA, USA, pp. 3840–3847, 2017.

[24] TATSUMOTO, Y., SHIRAISHI, M., CAI, K., LIN, Z. “Application of online supervisory control of discrete-event systems to multi-robot warehouse automation”, *Control Engineering Practice*, v. 81, pp. 97 – 104, 2018.

[25] OGATA, K. *Engenharia de Controle Moderno.* São Paulo, Brasil, Pearson Prentice Hall, 2010.

[26] MURATA, T. “Petri nets: Properties, analysis and applications”, *Proceedings of the IEEE*, v. 77, n. 4, pp. 541–580, Apr 1989.

[27] CLAVIJO, L. B., BASILIO, J. C., CARVALHO, L. K. “DESLAB: A scientific computing program for analysis and synthesis of discrete-event systems”, *IFAC Proceedings Volumes*, v. 45, n. 29, pp. 349–355, 2012.

[28] CORMEN, T. H., STEIN, C., RIVEST, R. L., LEISERSON, C. E. *Introduction to Algorithms.* 2nd ed. Massachusetts, MIT press, 2001.

[29] CHAZELLE, B. “Approximation and Decomposition of Shapes”. In: Schwartz, J. T., Yap, C. K. (Eds.), *Algorithmic and Geometric Aspects of Robotics*, Lawrence Erlbaum Associates, pp. 145–185, Hillsdale, NJ, 1987.

[30] GHOSH, S. K., MOUNT, D. M. “An Output Sensitive Algorithm for Computing Visibility Graphs”, *SIAM Journal on Computing*, v. 20, pp. 888–910, 1991.

[31] O’DUNLAING, C., YAP, C. K. “A retraction method for planning the motion of a disc”, *Journal of Algorithms*, v. 6, pp. 104–111, 1982.

[32] SHARIR, M. “Algorithmic Motion Planning”. In: Goodman, J. E., O’Rourke, J. (Eds.), *Handbook of Discrete and Computational Geometry, 2nd Ed.*, Chapman and Hall/CRC Press, pp. 1037–1064, New York, 2004.

[33] AZARM, K., SCHMIDT, G. “Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation”. In: *Proceedings of International Conference on Robotics and Automation*, v. 4, pp. 3526–3533 vol.4, April 1997.

[34] HAN, S., ZHOU, X., CHEN, C. “Path planning for multi-robot systems using PSO and Critical Path Schedule Method”. In: *2016 IEEE 13th International Conference on Networking, Sensing, and Control (ICNSC)*, pp. 1–6, April 2016.

Appendix A

Tables of Example 2.12 Automata

Table A.1: Transition function of G_{system}

Origin state	Event	Destination state	Origin state	Event	Destination state
$\{\{Idle, Free\}, Free\}$	$pull$	$\{\{Busy, Free\}, Free\}$	$\{\{Idle, Free\}, Free\}$	$gobuffer2$	$\{\{Idle, Free\}, Free_B\}$
$\{\{Idle, Free\}, Free_B\}$	$pull$	$\{\{Busy, Free\}, Free_B\}$	$\{\{Idle, Free\}, Free_B\}$	$pick2$	$\{\{Idle, Free\}, Full_B\}$
$\{\{Idle, Free\}, Full_B\}$	$goramp$	$\{\{Idle, Free\}, Full\}$	$\{\{Idle, Free\}, Full_B\}$	$pull$	$\{\{Busy, Free\}, Full_B\}$
$\{\{Busy, Free\}, Full_B\}$	$goramp$	$\{\{Busy, Free\}, Full\}$	$\{\{Busy, Free\}, Full_B\}$	$finish$	$\{\{Full, Free\}, Full_B\}$
$\{\{Full, Free\}, Full_B\}$	$goramp$	$\{\{Full, Free\}, Full\}$	$\{\{Full, Free\}, Full_B\}$	$pick1$	$\{\{Idle, Full\}, Full_B\}$
$\{\{Idle, Full\}, Full_B\}$	$gobuffer1$	$\{\{Idle, Full_B\}, Full_B\}$	$\{\{Idle, Full\}, Full_B\}$	$pull$	$\{\{Busy, Full\}, Full_B\}$
$\{\{Idle, Full\}, Full_B\}$	$goramp$	$\{\{Idle, Full\}, Full\}$	$\{\{Idle, Full\}, Full\}$	$gobuffer1$	$\{\{Idle, Full_B\}, Full\}$
$\{\{Idle, Full\}, Full\}$	$pull$	$\{\{Busy, Full\}, Full\}$	$\{\{Idle, Full\}, Full\}$	$drop2$	$\{\{Idle, Full\}, Free\}$
$\{\{Idle, Full\}, Free\}$	$gobuffer1$	$\{\{Idle, Full_B\}, Free\}$	$\{\{Idle, Full\}, Free\}$	$pull$	$\{\{Busy, Full\}, Free\}$
$\{\{Idle, Full\}, Free\}$	$gobuffer2$	$\{\{Idle, Full\}, Free_B\}$	$\{\{Idle, Full\}, Free_B\}$	$gobuffer1$	$\{\{Idle, Full_B\}, Free_B\}$
$\{\{Idle, Full\}, Free_B\}$	$pull$	$\{\{Busy, Full\}, Free_B\}$	$\{\{Idle, Full\}, Free_B\}$	$pick2$	$\{\{Idle, Full\}, Full_B\}$
$\{\{Busy, Full\}, Free_B\}$	$gobuffer1$	$\{\{Busy, Full_B\}, Free_B\}$	$\{\{Busy, Full\}, Free_B\}$	$finish$	$\{\{Full, Full\}, Free_B\}$
$\{\{Busy, Full\}, Free_B\}$	$pick2$	$\{\{Busy, Full\}, Full_B\}$	$\{\{Full, Full\}, Free_B\}$	$gobuffer1$	$\{\{Full, Full_B\}, Free_B\}$
$\{\{Full, Full\}, Free_B\}$	$pick2$	$\{\{Full, Full\}, Full_B\}$	$\{\{Full, Full\}, Full_B\}$	$gobuffer1$	$\{\{Full, Full_B\}, Full_B\}$
$\{\{Full, Full\}, Full_B\}$	$goramp$	$\{\{Full, Full\}, Full\}$	$\{\{Full, Full\}, Full\}$	$gobuffer1$	$\{\{Full, Full_B\}, Full\}$
$\{\{Full, Full\}, Full\}$	$drop2$	$\{\{Full, Full\}, Free\}$	$\{\{Full, Full\}, Free\}$	$gobuffer1$	$\{\{Full, Full_B\}, Free\}$
$\{\{Full, Full\}, Free\}$	$gobuffer2$	$\{\{Full, Full\}, Free_B\}$	$\{\{Full, Full_B\}, Free\}$	$drop1$	$\{\{Full, Free_B\}, Free\}$
$\{\{Full, Full_B\}, Free\}$	$gobuffer2$	$\{\{Full, Full_B\}, Free_B\}$	$\{\{Full, Free_B\}, Free\}$	$gobuffer2$	$\{\{Full, Free_B\}, Free_B\}$
$\{\{Full, Free_B\}, Free\}$	$goM1$	$\{\{Full, Free\}, Free\}$	$\{\{Full, Free\}, Free\}$	$gobuffer2$	$\{\{Full, Free\}, Free_B\}$
$\{\{Full, Free\}, Free\}$	$pick1$	$\{\{Idle, Full\}, Free\}$	$\{\{Full, Free\}, Free_B\}$	$pick1$	$\{\{Idle, Full\}, Free_B\}$
$\{\{Full, Free\}, Free_B\}$	$pick2$	$\{\{Full, Free\}, Full_B\}$	$\{\{Full, Free_B\}, Free_B\}$	$goM1$	$\{\{Full, Free\}, Free_B\}$
$\{\{Full, Free_B\}, Free_B\}$	$pick2$	$\{\{Full, Free_B\}, Full_B\}$	$\{\{Full, Free_B\}, Full_B\}$	$goramp$	$\{\{Full, Free_B\}, Full\}$
$\{\{Full, Free_B\}, Full_B\}$	$goM1$	$\{\{Full, Free\}, Full_B\}$	$\{\{Full, Free_B\}, Full\}$	$drop2$	$\{\{Full, Free_B\}, Free\}$
$\{\{Full, Free_B\}, Full\}$	$goM1$	$\{\{Full, Free\}, Full\}$	$\{\{Full, Full_B\}, Full\}$	$drop1$	$\{\{Full, Free_B\}, Full\}$
$\{\{Full, Full_B\}, Full\}$	$drop2$	$\{\{Full, Full_B\}, Free\}$	$\{\{Full, Full_B\}, Full_B\}$	$goramp$	$\{\{Full, Full_B\}, Full\}$
$\{\{Full, Full_B\}, Full_B\}$	$drop1$	$\{\{Full, Free_B\}, Full_B\}$	$\{\{Full, Full_B\}, Free_B\}$	$drop1$	$\{\{Full, Free_B\}, Free_B\}$
$\{\{Full, Full_B\}, Free_B\}$	$pick2$	$\{\{Full, Full_B\}, Full_B\}$	$\{\{Busy, Full_B\}, Free_B\}$	$drop1$	$\{\{Busy, Free_B\}, Free_B\}$
$\{\{Busy, Full_B\}, Free_B\}$	$finish$	$\{\{Full, Full_B\}, Free_B\}$	$\{\{Busy, Full_B\}, Free_B\}$	$pick2$	$\{\{Busy, Full_B\}, Full_B\}$
$\{\{Busy, Full_B\}, Full_B\}$	$goramp$	$\{\{Busy, Full_B\}, Full\}$	$\{\{Busy, Full_B\}, Full_B\}$	$drop1$	$\{\{Busy, Free_B\}, Full_B\}$
$\{\{Busy, Full_B\}, Full_B\}$	$finish$	$\{\{Full, Full_B\}, Full_B\}$	$\{\{Busy, Free_B\}, Full_B\}$	$goramp$	$\{\{Busy, Free_B\}, Full\}$
$\{\{Busy, Free_B\}, Full_B\}$	$finish$	$\{\{Full, Free_B\}, Full_B\}$	$\{\{Busy, Free_B\}, Full_B\}$	$goM1$	$\{\{Busy, Free\}, Full_B\}$

$\{\{Busy, Free_B\}, Full\}$	$drop2$	$\{\{Busy, Free_B\}, Free\}$	$\{\{Busy, Free_B\}, Full\}$	$finish$	$\{\{Full, Free_B\}, Full\}$
$\{\{Busy, Free_B\}, Full\}$	$goM1$	$\{\{Busy, Free\}, Full\}$	$\{\{Busy, Free_B\}, Free\}$	$finish$	$\{\{Full, Free_B\}, Free\}$
$\{\{Busy, Free_B\}, Free\}$	$goM1$	$\{\{Busy, Free\}, Free\}$	$\{\{Busy, Free_B\}, Free\}$	$gobuffer2$	$\{\{Busy, Free_B\}, Free_B\}$
$\{\{Busy, Full_B\}, Full\}$	$drop1$	$\{\{Busy, Free_B\}, Full\}$	$\{\{Busy, Full_B\}, Full\}$	$drop2$	$\{\{Busy, Full_B\}, Free\}$
$\{\{Busy, Full_B\}, Full\}$	$finish$	$\{\{Full, Full_B\}, Full\}$	$\{\{Busy, Full_B\}, Free\}$	$drop1$	$\{\{Busy, Free_B\}, Free\}$
$\{\{Busy, Full_B\}, Free\}$	$finish$	$\{\{Full, Full_B\}, Free\}$	$\{\{Busy, Full_B\}, Free\}$	$gobuffer2$	$\{\{Busy, Full_B\}, Free_B\}$
$\{\{Busy, Free_B\}, Free_B\}$	$finish$	$\{\{Full, Free_B\}, Free_B\}$	$\{\{Busy, Free_B\}, Free_B\}$	$goM1$	$\{\{Busy, Free\}, Free_B\}$
$\{\{Busy, Free_B\}, Free_B\}$	$pick2$	$\{\{Busy, Free_B\}, Full_B\}$	$\{\{Idle, Full_B\}, Free_B\}$	$drop1$	$\{\{Idle, Free_B\}, Free_B\}$
$\{\{Idle, Full_B\}, Free_B\}$	$pick2$	$\{\{Idle, Full_B\}, Full_B\}$	$\{\{Idle, Full_B\}, Free_B\}$	$pull$	$\{\{Busy, Full_B\}, Free_B\}$
$\{\{Idle, Free_B\}, Free_B\}$	$pull$	$\{\{Busy, Free_B\}, Free_B\}$	$\{\{Idle, Free_B\}, Free_B\}$	$goM1$	$\{\{Idle, Free\}, Free_B\}$
$\{\{Idle, Free_B\}, Free_B\}$	$pick2$	$\{\{Idle, Free_B\}, Full_B\}$	$\{\{Idle, Free_B\}, Full_B\}$	$goramp$	$\{\{Idle, Free_B\}, Full\}$
$\{\{Idle, Free_B\}, Full_B\}$	$pull$	$\{\{Busy, Free_B\}, Full_B\}$	$\{\{Idle, Free_B\}, Full_B\}$	$goM1$	$\{\{Idle, Free\}, Full_B\}$
$\{\{Idle, Free_B\}, Full\}$	$pull$	$\{\{Busy, Free_B\}, Full\}$	$\{\{Idle, Free_B\}, Full\}$	$drop2$	$\{\{Idle, Free_B\}, Free\}$
$\{\{Idle, Free_B\}, Full\}$	$goM1$	$\{\{Idle, Free\}, Full\}$	$\{\{Idle, Free_B\}, Free\}$	$pull$	$\{\{Busy, Free_B\}, Free\}$
$\{\{Idle, Free_B\}, Free\}$	$gobuffer2$	$\{\{Idle, Free_B\}, Free_B\}$	$\{\{Idle, Free_B\}, Free\}$	$goM1$	$\{\{Idle, Free\}, Free\}$
$\{\{Busy, Full\}, Free\}$	$gobuffer1$	$\{\{Busy, Full_B\}, Free\}$	$\{\{Busy, Full\}, Free\}$	$finish$	$\{\{Full, Full\}, Free\}$
$\{\{Busy, Full\}, Free\}$	$gobuffer2$	$\{\{Busy, Full\}, Free_B\}$	$\{\{Idle, Full_B\}, Free\}$	$drop1$	$\{\{Idle, Free_B\}, Free\}$
$\{\{Idle, Full_B\}, Free\}$	$gobuffer2$	$\{\{Idle, Full_B\}, Free_B\}$	$\{\{Idle, Full_B\}, Free\}$	$pull$	$\{\{Busy, Full_B\}, Free\}$
$\{\{Busy, Full\}, Full\}$	$gobuffer1$	$\{\{Busy, Full_B\}, Full\}$	$\{\{Busy, Full\}, Full\}$	$drop2$	$\{\{Busy, Full\}, Free\}$
$\{\{Busy, Full\}, Full\}$	$finish$	$\{\{Full, Full\}, Full\}$	$\{\{Idle, Full_B\}, Full\}$	$drop1$	$\{\{Idle, Free_B\}, Full\}$
$\{\{Idle, Full_B\}, Full\}$	$drop2$	$\{\{Idle, Full_B\}, Free\}$	$\{\{Idle, Full_B\}, Full\}$	$pull$	$\{\{Busy, Full_B\}, Full\}$
$\{\{Busy, Full\}, Full_B\}$	$gobuffer1$	$\{\{Busy, Full_B\}, Full_B\}$	$\{\{Busy, Full\}, Full_B\}$	$goramp$	$\{\{Busy, Full\}, Full\}$
$\{\{Busy, Full\}, Full_B\}$	$finish$	$\{\{Full, Full\}, Full_B\}$	$\{\{Idle, Full_B\}, Full_B\}$	$goramp$	$\{\{Idle, Full_B\}, Full\}$
$\{\{Idle, Full_B\}, Full_B\}$	$drop1$	$\{\{Idle, Free_B\}, Full_B\}$	$\{\{Idle, Full_B\}, Full_B\}$	$pull$	$\{\{Busy, Full_B\}, Full_B\}$
$\{\{Full, Free\}, Full\}$	$drop2$	$\{\{Full, Free\}, Free\}$	$\{\{Full, Free\}, Full\}$	$pick1$	$\{\{Idle, Full\}, Full\}$
$\{\{Busy, Free\}, Full\}$	$drop2$	$\{\{Busy, Free\}, Free\}$	$\{\{Busy, Free\}, Full\}$	$finish$	$\{\{Full, Free\}, Full\}$
$\{\{Idle, Free\}, Full\}$	$pull$	$\{\{Busy, Free\}, Full\}$	$\{\{Idle, Free\}, Full\}$	$drop2$	$\{\{Idle, Free\}, Free\}$
$\{\{Busy, Free\}, Free_B\}$	$finish$	$\{\{Full, Free\}, Free_B\}$	$\{\{Busy, Free\}, Free_B\}$	$pick2$	$\{\{Busy, Free\}, Full_B\}$
$\{\{Busy, Free\}, Free\}$	$finish$	$\{\{Full, Free\}, Free\}$	$\{\{Busy, Free\}, Free\}$	$gobuffer2$	$\{\{Busy, Free\}, Free_B\}$

Table A.2: Transition function of supervisor S_1

Origin state	Event	Destination state	Origin state	Event	Destination state
$\{0, \{\{Idle, Free\}, Free\}\}$	$pull$	$\{0, \{\{Busy, Free\}, Free\}\}$	$\{0, \{\{Idle, Free\}, Free\}\}$	$gobuffer2$	$\{2, \{\{Idle, Free\}, Free_B\}\}$
$\{2, \{\{Idle, Free\}, Free_B\}\}$	$pull$	$\{2, \{\{Busy, Free\}, Free_B\}\}$	$\{2, \{\{Idle, Free\}, Free_B\}\}$	$pick2$	$\{2, \{\{Idle, Free\}, Full_B\}\}$
$\{2, \{\{Idle, Free\}, Full_B\}\}$	$goramp$	$\{0, \{\{Idle, Free\}, Full\}\}$	$\{2, \{\{Idle, Free\}, Full_B\}\}$	$pull$	$\{2, \{\{Busy, Free\}, Full_B\}\}$
$\{2, \{\{Busy, Free\}, Full_B\}\}$	$goramp$	$\{0, \{\{Busy, Free\}, Full\}\}$	$\{2, \{\{Busy, Free\}, Full_B\}\}$	$finish$	$\{2, \{\{Full, Free\}, Full_B\}\}$
$\{2, \{\{Full, Free\}, Full_B\}\}$	$goramp$	$\{0, \{\{Full, Free\}, Full\}\}$	$\{2, \{\{Full, Free\}, Full_B\}\}$	$pick1$	$\{2, \{\{Idle, Full\}, Full_B\}\}$
$\{2, \{\{Idle, Full\}, Full_B\}\}$	$goramp$	$\{0, \{\{Idle, Full\}, Full\}\}$	$\{2, \{\{Idle, Full\}, Full_B\}\}$	$pull$	$\{2, \{\{Busy, Full\}, Full_B\}\}$
$\{2, \{\{Busy, Full\}, Full_B\}\}$	$goramp$	$\{0, \{\{Busy, Full\}, Full\}\}$	$\{2, \{\{Busy, Full\}, Full_B\}\}$	$finish$	$\{2, \{\{Full, Full\}, Full_B\}\}$
$\{2, \{\{Full, Full\}, Full_B\}\}$	$goramp$	$\{0, \{\{Full, Full\}, Full\}\}$	$\{0, \{\{Full, Full\}, Full\}\}$	$gobuffer1$	$\{1, \{\{Full, Full_B\}, Full\}\}$
$\{0, \{\{Full, Full\}, Full\}\}$	$drop2$	$\{0, \{\{Full, Full\}, Free\}\}$	$\{0, \{\{Full, Full\}, Free\}\}$	$gobuffer1$	$\{1, \{\{Full, Full_B\}, Free\}\}$
$\{0, \{\{Full, Full\}, Free\}\}$	$gobuffer2$	$\{2, \{\{Full, Full\}, Free_B\}\}$	$\{2, \{\{Full, Full\}, Free_B\}\}$	$pick2$	$\{2, \{\{Full, Full\}, Full_B\}\}$
$\{1, \{\{Full, Full_B\}, Free\}\}$	$drop1$	$\{1, \{\{Full, Free_B\}, Free\}\}$	$\{1, \{\{Full, Free_B\}, Free\}\}$	$goM1$	$\{0, \{\{Full, Free\}, Free\}\}$
$\{0, \{\{Full, Free\}, Free\}\}$	$gobuffer2$	$\{2, \{\{Full, Free\}, Free_B\}\}$	$\{0, \{\{Full, Free\}, Free\}\}$	$pick1$	$\{0, \{\{Idle, Full\}, Free\}\}$
$\{0, \{\{Idle, Full\}, Free\}\}$	$gobuffer1$	$\{1, \{\{Idle, Full_B\}, Free\}\}$	$\{0, \{\{Idle, Full\}, Free\}\}$	$pull$	$\{0, \{\{Busy, Full\}, Free\}\}$
$\{0, \{\{Idle, Full\}, Free\}\}$	$gobuffer2$	$\{2, \{\{Idle, Full\}, Free_B\}\}$	$\{2, \{\{Idle, Full\}, Free_B\}\}$	$pull$	$\{2, \{\{Busy, Full\}, Free_B\}\}$
$\{2, \{\{Idle, Full\}, Free_B\}\}$	$pick2$	$\{2, \{\{Idle, Full\}, Full_B\}\}$	$\{2, \{\{Busy, Full\}, Free_B\}\}$	$finish$	$\{2, \{\{Full, Full\}, Free_B\}\}$
$\{2, \{\{Busy, Full\}, Free_B\}\}$	$pick2$	$\{2, \{\{Busy, Full\}, Full_B\}\}$	$\{0, \{\{Busy, Full\}, Free\}\}$	$gobuffer1$	$\{1, \{\{Busy, Full_B\}, Free\}\}$
$\{0, \{\{Busy, Full\}, Free\}\}$	$gobuffer2$	$\{2, \{\{Busy, Full\}, Free_B\}\}$	$\{0, \{\{Busy, Full\}, Free\}\}$	$finish$	$\{0, \{\{Full, Full\}, Free\}\}$
$\{1, \{\{Busy, Full_B\}, Free\}\}$	$drop1$	$\{1, \{\{Busy, Free_B\}, Free\}\}$	$\{1, \{\{Busy, Full_B\}, Free\}\}$	$finish$	$\{1, \{\{Full, Full_B\}, Free\}\}$
$\{1, \{\{Busy, Free_B\}, Free\}\}$	$finish$	$\{1, \{\{Full, Free_B\}, Free\}\}$	$\{1, \{\{Busy, Free_B\}, Free\}\}$	$goM1$	$\{0, \{\{Busy, Free\}, Free\}\}$
$\{1, \{\{Idle, Full_B\}, Free\}\}$	$drop1$	$\{1, \{\{Idle, Free_B\}, Free\}\}$	$\{1, \{\{Idle, Full_B\}, Free\}\}$	$pull$	$\{1, \{\{Busy, Full_B\}, Free\}\}$
$\{1, \{\{Idle, Free_B\}, Free\}\}$	$pull$	$\{1, \{\{Busy, Free_B\}, Free\}\}$	$\{1, \{\{Idle, Free_B\}, Free\}\}$	$goM1$	$\{0, \{\{Idle, Free\}, Free\}\}$
$\{2, \{\{Full, Free\}, Free_B\}\}$	$pick1$	$\{2, \{\{Idle, Full\}, Free_B\}\}$	$\{2, \{\{Full, Free\}, Free_B\}\}$	$pick2$	$\{2, \{\{Full, Free\}, Full_B\}\}$
$\{1, \{\{Full, Full_B\}, Full\}\}$	$drop1$	$\{1, \{\{Full, Free_B\}, Full\}\}$	$\{1, \{\{Full, Full_B\}, Full\}\}$	$drop2$	$\{1, \{\{Full, Full_B\}, Free\}\}$
$\{1, \{\{Full, Free_B\}, Full\}\}$	$drop2$	$\{1, \{\{Full, Free_B\}, Free\}\}$	$\{1, \{\{Full, Free_B\}, Full\}\}$	$goM1$	$\{0, \{\{Full, Free\}, Full\}\}$
$\{0, \{\{Busy, Full\}, Full\}\}$	$gobuffer1$	$\{1, \{\{Busy, Full_B\}, Full\}\}$	$\{0, \{\{Busy, Full\}, Full\}\}$	$drop2$	$\{0, \{\{Busy, Full\}, Free\}\}$
$\{0, \{\{Busy, Full\}, Full\}\}$	$finish$	$\{0, \{\{Full, Full\}, Full\}\}$	$\{1, \{\{Busy, Full_B\}, Full\}\}$	$drop1$	$\{1, \{\{Busy, Free_B\}, Full\}\}$
$\{1, \{\{Busy, Full_B\}, Full\}\}$	$drop2$	$\{1, \{\{Busy, Full_B\}, Free\}\}$	$\{1, \{\{Busy, Full_B\}, Full\}\}$	$finish$	$\{1, \{\{Full, Full_B\}, Full\}\}$
$\{1, \{\{Busy, Free_B\}, Full\}\}$	$drop2$	$\{1, \{\{Busy, Free_B\}, Free\}\}$	$\{1, \{\{Busy, Free_B\}, Full\}\}$	$finish$	$\{1, \{\{Full, Free_B\}, Full\}\}$
$\{1, \{\{Busy, Free_B\}, Full\}\}$	$goM1$	$\{0, \{\{Busy, Free\}, Full\}\}$	$\{0, \{\{Idle, Full\}, Full\}\}$	$gobuffer1$	$\{1, \{\{Idle, Full_B\}, Full\}\}$
$\{0, \{\{Idle, Full\}, Full\}\}$	$pull$	$\{0, \{\{Busy, Full\}, Full\}\}$	$\{0, \{\{Idle, Full\}, Full\}\}$	$drop2$	$\{0, \{\{Idle, Full\}, Free\}\}$
$\{1, \{\{Idle, Full_B\}, Full\}\}$	$drop1$	$\{1, \{\{Idle, Free_B\}, Full\}\}$	$\{1, \{\{Idle, Full_B\}, Full\}\}$	$drop2$	$\{1, \{\{Idle, Full_B\}, Free\}\}$
$\{1, \{\{Idle, Full_B\}, Full\}\}$	$pull$	$\{1, \{\{Busy, Full_B\}, Full\}\}$	$\{1, \{\{Idle, Free_B\}, Full\}\}$	$pull$	$\{1, \{\{Busy, Free_B\}, Full\}\}$
$\{1, \{\{Idle, Free_B\}, Full\}\}$	$drop2$	$\{1, \{\{Idle, Free_B\}, Free\}\}$	$\{1, \{\{Idle, Free_B\}, Full\}\}$	$goM1$	$\{0, \{\{Idle, Free\}, Full\}\}$
$\{0, \{\{Full, Free\}, Full\}\}$	$drop2$	$\{0, \{\{Full, Free\}, Free\}\}$	$\{0, \{\{Full, Free\}, Full\}\}$	$pick1$	$\{0, \{\{Idle, Full\}, Full\}\}$
$\{0, \{\{Busy, Free\}, Full\}\}$	$drop2$	$\{0, \{\{Busy, Free\}, Free\}\}$	$\{0, \{\{Busy, Free\}, Full\}\}$	$finish$	$\{0, \{\{Full, Free\}, Full\}\}$
$\{0, \{\{Idle, Free\}, Full\}\}$	$pull$	$\{0, \{\{Busy, Free\}, Full\}\}$	$\{0, \{\{Idle, Free\}, Full\}\}$	$drop2$	$\{0, \{\{Idle, Free\}, Free\}\}$
$\{2, \{\{Busy, Free\}, Free_B\}\}$	$finish$	$\{2, \{\{Full, Free\}, Free_B\}\}$	$\{2, \{\{Busy, Free\}, Free_B\}\}$	$pick2$	$\{2, \{\{Busy, Free\}, Full_B\}\}$
$\{0, \{\{Busy, Free\}, Free\}\}$	$gobuffer2$	$\{2, \{\{Busy, Free\}, Free_B\}\}$	$\{0, \{\{Busy, Free\}, Free\}\}$	$finish$	$\{0, \{\{Full, Free\}, Free\}\}$

Table A.3: Transition function of supervisor S_2

Origin state	Event	Destination state	Origin state	Event	Destination state
$\{0,\{\{Idle,Free\},Free\}\}$	$pull$	$\{0,\{\{Busy,Free\},Free\}\}$	$\{0,\{\{Idle,Free\},Free\}\}$	$gobuffer2$	$\{0,\{\{Idle,Free\},Free_B\}\}$
$\{0,\{\{Idle,Free\},Free_B\}\}$	$pull$	$\{0,\{\{Busy,Free\},Free_B\}\}$	$\{0,\{\{Busy,Free\},Free_B\}\}$	$finish$	$\{0,\{\{Full,Free\},Free_B\}\}$
$\{0,\{\{Full,Free\},Free_B\}\}$	$pick1$	$\{0,\{\{Idle,Full\},Free_B\}\}$	$\{0,\{\{Idle,Full\},Free_B\}\}$	$gobuffer1$	$\{0,\{\{Idle,Full_B\},Free_B\}\}$
$\{0,\{\{Idle,Full\},Free_B\}\}$	$pull$	$\{0,\{\{Busy,Full\},Free_B\}\}$	$\{0,\{\{Busy,Full\},Free_B\}\}$	$gobuffer1$	$\{0,\{\{Busy,Full_B\},Free_B\}\}$
$\{0,\{\{Busy,Full\},Free_B\}\}$	$finish$	$\{0,\{\{Full,Full\},Free_B\}\}$	$\{0,\{\{Full,Full\},Free_B\}\}$	$gobuffer1$	$\{0,\{\{Full,Full_B\},Free_B\}\}$
$\{0,\{\{Full,Full_B\},Free_B\}\}$	$drop1$	$\{1,\{\{Full,Free_B\},Free_B\}\}$	$\{1,\{\{Full,Free_B\},Free_B\}\}$	$goM1$	$\{1,\{\{Full,Free\},Free_B\}\}$
$\{1,\{\{Full,Free_B\},Free_B\}\}$	$pick2$	$\{0,\{\{Full,Free_B\},Full_B\}\}$	$\{0,\{\{Full,Free_B\},Full_B\}\}$	$goramp$	$\{0,\{\{Full,Free_B\},Full\}\}$
$\{0,\{\{Full,Free_B\},Full_B\}\}$	$goM1$	$\{0,\{\{Full,Free\},Full_B\}\}$	$\{0,\{\{Full,Free\},Full_B\}\}$	$goramp$	$\{0,\{\{Full,Free\},Full\}\}$
$\{0,\{\{Full,Free\},Full_B\}\}$	$pick1$	$\{0,\{\{Idle,Full\},Full_B\}\}$	$\{0,\{\{Idle,Full\},Full_B\}\}$	$goramp$	$\{0,\{\{Idle,Full\},Full\}\}$
$\{0,\{\{Idle,Full\},Full_B\}\}$	$pull$	$\{0,\{\{Busy,Full\},Full_B\}\}$	$\{0,\{\{Idle,Full\},Full_B\}\}$	$gobuffer1$	$\{0,\{\{Idle,Full_B\},Full_B\}\}$
$\{0,\{\{Idle,Full_B\},Full_B\}\}$	$goramp$	$\{0,\{\{Idle,Full_B\},Full\}\}$	$\{0,\{\{Idle,Full_B\},Full_B\}\}$	$drop1$	$\{1,\{\{Idle,Free_B\},Full_B\}\}$
$\{0,\{\{Idle,Full_B\},Full_B\}\}$	$pull$	$\{0,\{\{Busy,Full_B\},Full_B\}\}$	$\{0,\{\{Busy,Full_B\},Full_B\}\}$	$goramp$	$\{0,\{\{Busy,Full_B\},Full\}\}$
$\{0,\{\{Busy,Full_B\},Full_B\}\}$	$drop1$	$\{1,\{\{Busy,Free_B\},Full_B\}\}$	$\{0,\{\{Busy,Full_B\},Full_B\}\}$	$finish$	$\{0,\{\{Full,Full_B\},Full_B\}\}$
$\{0,\{\{Full,Full_B\},Full_B\}\}$	$goramp$	$\{0,\{\{Full,Full_B\},Full\}\}$	$\{0,\{\{Full,Full_B\},Full_B\}\}$	$drop1$	$\{1,\{\{Full,Free_B\},Full_B\}\}$
$\{1,\{\{Full,Free_B\},Full_B\}\}$	$goramp$	$\{1,\{\{Full,Free_B\},Full\}\}$	$\{1,\{\{Full,Free_B\},Full_B\}\}$	$goM1$	$\{1,\{\{Full,Free\},Full_B\}\}$
$\{1,\{\{Full,Free\},Full_B\}\}$	$goramp$	$\{1,\{\{Full,Free\},Full\}\}$	$\{1,\{\{Full,Free\},Full_B\}\}$	$pick1$	$\{1,\{\{Idle,Full\},Full_B\}\}$
$\{1,\{\{Idle,Full\},Full_B\}\}$	$goramp$	$\{1,\{\{Idle,Full\},Full\}\}$	$\{1,\{\{Idle,Full\},Full_B\}\}$	$pull$	$\{1,\{\{Busy,Full\},Full_B\}\}$
$\{1,\{\{Idle,Full\},Full_B\}\}$	$gobuffer1$	$\{1,\{\{Idle,Full_B\},Full_B\}\}$	$\{1,\{\{Idle,Full_B\},Full_B\}\}$	$goramp$	$\{1,\{\{Idle,Full_B\},Full\}\}$
$\{1,\{\{Idle,Full_B\},Full_B\}\}$	$pull$	$\{1,\{\{Busy,Full_B\},Full_B\}\}$	$\{1,\{\{Busy,Full_B\},Full_B\}\}$	$goramp$	$\{1,\{\{Busy,Full_B\},Full\}\}$
$\{1,\{\{Busy,Full_B\},Full_B\}\}$	$finish$	$\{1,\{\{Full,Full_B\},Full_B\}\}$	$\{1,\{\{Full,Full_B\},Full_B\}\}$	$goramp$	$\{1,\{\{Full,Full_B\},Full\}\}$
$\{1,\{\{Full,Full_B\},Full\}\}$	$drop2$	$\{1,\{\{Full,Full_B\},Free\}\}$	$\{1,\{\{Full,Full_B\},Free\}\}$	$gobuffer2$	$\{1,\{\{Full,Full_B\},Free_B\}\}$
$\{1,\{\{Full,Full_B\},Free_B\}\}$	$pick2$	$\{0,\{\{Full,Full_B\},Full_B\}\}$	$\{1,\{\{Busy,Full_B\},Full\}\}$	$drop2$	$\{1,\{\{Busy,Full_B\},Free\}\}$
$\{1,\{\{Busy,Full_B\},Full\}\}$	$finish$	$\{1,\{\{Full,Full_B\},Full\}\}$	$\{1,\{\{Busy,Full_B\},Free\}\}$	$gobuffer2$	$\{1,\{\{Busy,Full_B\},Free_B\}\}$
$\{1,\{\{Busy,Full_B\},Free\}\}$	$finish$	$\{1,\{\{Full,Full_B\},Free\}\}$	$\{1,\{\{Busy,Full_B\},Free_B\}\}$	$finish$	$\{1,\{\{Full,Full_B\},Free_B\}\}$
$\{1,\{\{Busy,Full_B\},Free_B\}\}$	$pick2$	$\{0,\{\{Busy,Full_B\},Full_B\}\}$	$\{1,\{\{Idle,Full_B\},Full\}\}$	$pull$	$\{1,\{\{Busy,Full_B\},Full\}\}$
$\{1,\{\{Idle,Full_B\},Full\}\}$	$drop2$	$\{1,\{\{Idle,Full_B\},Free\}\}$	$\{1,\{\{Idle,Full_B\},Free\}\}$	$pull$	$\{1,\{\{Busy,Full_B\},Free\}\}$
$\{1,\{\{Idle,Full_B\},Free\}\}$	$gobuffer2$	$\{1,\{\{Idle,Full_B\},Free_B\}\}$	$\{1,\{\{Idle,Full_B\},Free_B\}\}$	$pull$	$\{1,\{\{Busy,Full_B\},Free_B\}\}$
$\{1,\{\{Idle,Full_B\},Free_B\}\}$	$pick2$	$\{0,\{\{Idle,Full_B\},Full_B\}\}$	$\{1,\{\{Busy,Full\},Full_B\}\}$	$goramp$	$\{1,\{\{Busy,Full\},Full\}\}$
$\{1,\{\{Busy,Full\},Full_B\}\}$	$gobuffer1$	$\{1,\{\{Busy,Full_B\},Full_B\}\}$	$\{1,\{\{Busy,Full\},Full_B\}\}$	$finish$	$\{1,\{\{Full,Full\},Full_B\}\}$
$\{1,\{\{Full,Full\},Full_B\}\}$	$goramp$	$\{1,\{\{Full,Full\},Full\}\}$	$\{1,\{\{Full,Full\},Full_B\}\}$	$gobuffer1$	$\{1,\{\{Full,Full_B\},Full_B\}\}$
$\{1,\{\{Full,Full\},Full\}\}$	$gobuffer1$	$\{1,\{\{Full,Full_B\},Full\}\}$	$\{1,\{\{Full,Full\},Full\}\}$	$drop2$	$\{1,\{\{Full,Full\},Free\}\}$
$\{1,\{\{Full,Full\},Free\}\}$	$gobuffer1$	$\{1,\{\{Full,Full_B\},Free\}\}$	$\{1,\{\{Full,Full\},Free\}\}$	$gobuffer2$	$\{1,\{\{Full,Full\},Free_B\}\}$
$\{1,\{\{Full,Full\},Free_B\}\}$	$gobuffer1$	$\{1,\{\{Full,Full_B\},Free_B\}\}$	$\{1,\{\{Full,Full\},Free_B\}\}$	$pick2$	$\{0,\{\{Full,Full\},Full_B\}\}$
$\{0,\{\{Full,Full\},Full_B\}\}$	$goramp$	$\{0,\{\{Full,Full\},Full\}\}$	$\{0,\{\{Full,Full\},Full_B\}\}$	$gobuffer1$	$\{0,\{\{Full,Full_B\},Full_B\}\}$
$\{0,\{\{Full,Full\},Full\}\}$	$gobuffer1$	$\{0,\{\{Full,Full_B\},Full\}\}$	$\{0,\{\{Full,Full\},Full\}\}$	$drop2$	$\{0,\{\{Full,Full\},Free\}\}$
$\{0,\{\{Full,Full\},Free\}\}$	$gobuffer1$	$\{0,\{\{Full,Full_B\},Free\}\}$	$\{0,\{\{Full,Full\},Free\}\}$	$gobuffer2$	$\{0,\{\{Full,Full\},Free_B\}\}$
$\{0,\{\{Full,Full_B\},Free\}\}$	$drop1$	$\{1,\{\{Full,Free_B\},Free\}\}$	$\{0,\{\{Full,Full_B\},Free\}\}$	$gobuffer2$	$\{0,\{\{Full,Full_B\},Free_B\}\}$
$\{1,\{\{Full,Free_B\},Free\}\}$	$gobuffer2$	$\{1,\{\{Full,Free_B\},Free_B\}\}$	$\{1,\{\{Full,Free_B\},Free\}\}$	$goM1$	$\{1,\{\{Full,Free\},Free\}\}$
$\{1,\{\{Full,Free\},Free\}\}$	$gobuffer2$	$\{1,\{\{Full,Free\},Free_B\}\}$	$\{1,\{\{Full,Free\},Free\}\}$	$pick1$	$\{1,\{\{Idle,Full\},Free\}\}$
$\{1,\{\{Idle,Full\},Free\}\}$	$gobuffer1$	$\{1,\{\{Idle,Full_B\},Free\}\}$	$\{1,\{\{Idle,Full\},Free\}\}$	$pull$	$\{1,\{\{Busy,Full\},Free\}\}$
$\{1,\{\{Idle,Full\},Free\}\}$	$gobuffer2$	$\{1,\{\{Idle,Full\},Free_B\}\}$	$\{1,\{\{Idle,Full\},Free_B\}\}$	$gobuffer1$	$\{1,\{\{Idle,Full_B\},Free_B\}\}$
$\{1,\{\{Idle,Full\},Free_B\}\}$	$pull$	$\{1,\{\{Busy,Full\},Free_B\}\}$	$\{1,\{\{Idle,Full\},Free_B\}\}$	$pick2$	$\{0,\{\{Idle,Full\},Full_B\}\}$
$\{1,\{\{Busy,Full\},Free_B\}\}$	$gobuffer1$	$\{1,\{\{Busy,Full_B\},Free_B\}\}$	$\{1,\{\{Busy,Full\},Free_B\}\}$	$finish$	$\{1,\{\{Full,Full\},Free_B\}\}$
$\{1,\{\{Busy,Full\},Free_B\}\}$	$pick2$	$\{0,\{\{Busy,Full\},Full_B\}\}$	$\{1,\{\{Busy,Full\},Free\}\}$	$gobuffer1$	$\{1,\{\{Busy,Full_B\},Free\}\}$
$\{1,\{\{Busy,Full\},Free\}\}$	$finish$	$\{1,\{\{Full,Full\},Free\}\}$	$\{1,\{\{Busy,Full\},Free\}\}$	$gobuffer2$	$\{1,\{\{Busy,Full\},Free_B\}\}$
$\{1,\{\{Busy,Full\},Full\}\}$	$gobuffer1$	$\{1,\{\{Busy,Full_B\},Full\}\}$	$\{1,\{\{Busy,Full\},Full\}\}$	$drop2$	$\{1,\{\{Busy,Full\},Free\}\}$
$\{1,\{\{Busy,Full\},Full\}\}$	$finish$	$\{1,\{\{Full,Full\},Full\}\}$	$\{1,\{\{Idle,Full\},Full\}\}$	$gobuffer1$	$\{1,\{\{Idle,Full_B\},Full\}\}$
$\{1,\{\{Idle,Full\},Full\}\}$	$pull$	$\{1,\{\{Busy,Full\},Full\}\}$	$\{1,\{\{Idle,Full\},Full\}\}$	$drop2$	$\{1,\{\{Idle,Full\},Free\}\}$
$\{1,\{\{Full,Free\},Full\}\}$	$drop2$	$\{1,\{\{Full,Free\},Free\}\}$	$\{1,\{\{Full,Free\},Full\}\}$	$pick1$	$\{1,\{\{Idle,Full\},Full\}\}$
$\{1,\{\{Full,Free_B\},Full\}\}$	$drop2$	$\{1,\{\{Full,Free_B\},Free\}\}$	$\{1,\{\{Full,Free_B\},Full\}\}$	$goM1$	$\{1,\{\{Full,Free\},Full\}\}$
$\{0,\{\{Full,Full_B\},Full\}\}$	$drop1$	$\{1,\{\{Full,Free_B\},Full\}\}$	$\{0,\{\{Full,Full_B\},Full\}\}$	$drop2$	$\{0,\{\{Full,Full_B\},Free\}\}$
$\{1,\{\{Busy,Free_B\},Full_B\}\}$	$goramp$	$\{1,\{\{Busy,Free_B\},Full\}\}$	$\{1,\{\{Busy,Free_B\},Full_B\}\}$	$finish$	$\{1,\{\{Full,Free_B\},Full_B\}\}$
$\{1,\{\{Busy,Free_B\},Full_B\}\}$	$goM1$	$\{1,\{\{Busy,Free\},Full_B\}\}$	$\{1,\{\{Busy,Free\},Full_B\}\}$	$goramp$	$\{1,\{\{Busy,Free\},Full\}\}$
$\{1,\{\{Busy,Free\},Full_B\}\}$	$finish$	$\{1,\{\{Full,Free\},Full_B\}\}$	$\{1,\{\{Busy,Free\},Full\}\}$	$drop2$	$\{1,\{\{Busy,Free\},Free\}\}$
$\{1,\{\{Busy,Free\},Full\}\}$	$finish$	$\{1,\{\{Full,Free\},Full\}\}$	$\{1,\{\{Busy,Free\},Free\}\}$	$gobuffer2$	$\{1,\{\{Busy,Free\},Free_B\}\}$
$\{1,\{\{Busy,Free\},Free\}\}$	$finish$	$\{1,\{\{Full,Free\},Free\}\}$	$\{1,\{\{Busy,Free\},Free_B\}\}$	$finish$	$\{1,\{\{Full,Free\},Free_B\}\}$
$\{1,\{\{Busy,Free\},Free_B\}\}$	$pick2$	$\{0,\{\{Busy,Free\},Full_B\}\}$	$\{0,\{\{Busy,Free\},Full_B\}\}$	$goramp$	$\{0,\{\{Busy,Free\},Full\}\}$
$\{0,\{\{Busy,Free\},Full_B\}\}$	$finish$	$\{0,\{\{Full,Free\},Full_B\}\}$	$\{0,\{\{Busy,Free\},Full\}\}$	$drop2$	$\{0,\{\{Busy,Free\},Free\}\}$
$\{0,\{\{Busy,Free\},Full\}\}$	$finish$	$\{0,\{\{Full,Free\},Full\}\}$	$\{1,\{\{Busy,Free_B\},Full\}\}$	$drop2$	$\{1,\{\{Busy,Free_B\},Free\}\}$
$\{1,\{\{Busy,Free_B\},Full\}\}$	$finish$	$\{1,\{\{Full,Free_B\},Full\}\}$	$\{1,\{\{Busy,Free_B\},Full\}\}$	$goM1$	$\{1,\{\{Busy,Free\},Full\}\}$
$\{1,\{\{Busy,Free_B\},Free\}\}$	$gobuffer2$	$\{1,\{\{Busy,Free_B\},Free_B\}\}$	$\{1,\{\{Busy,Free_B\},Free\}\}$	$goM1$	$\{1,\{\{Busy,Free\},Free\}\}$
$\{1,\{\{Busy,Free_B\},Free\}\}$	$finish$	$\{1,\{\{Full,Free_B\},Free\}\}$	$\{1,\{\{Busy,Free_B\},Free_B\}\}$	$finish$	$\{1,\{\{Full,Free_B\},Free_B\}\}$
$\{1,\{\{Busy,Free_B\},Free_B\}\}$	$goM1$	$\{1,\{\{Busy,Free\},Free_B\}\}$	$\{1,\{\{Busy,Free_B\},Free_B\}\}$	$pick2$	$\{0,\{\{Busy,Free_B\},Full_B\}\}$
$\{0,\{\{Busy,Free_B\},Full_B\}\}$	$goramp$	$\{0,\{\{Busy,Free_B\},Full\}\}$	$\{0,\{\{Busy,Free_B\},Full_B\}\}$	$finish$	$\{0,\{\{Full,Free_B\},Full_B\}\}$
$\{0,\{\{Busy,Free_B\},Full_B\}\}$	$goM1$	$\{0,\{\{Busy,Free\},Full_B\}\}$	$\{0,\{\{Busy,Free_B\},Full\}\}$	$drop2$	$\{0,\{\{Busy,Free_B\},Free\}\}$
$\{0,\{\{Busy,Free_B\},Full\}\}$	$finish$	$\{0,\{\{Full,Free_B\},Full\}\}$	$\{0,\{\{Busy,Free_B\},Full\}\}$	$goM1$	$\{0,\{\{Busy,Free\},Full\}\}$
$\{0,\{\{Busy,Free_B\},Free\}\}$	$gobuffer2$	$\{0,\{\{Busy,Free_B\},Free_B\}\}$	$\{0,\{\{Busy,Free_B\},Free\}\}$	$goM1$	$\{0,\{\{Busy,Free\},Free\}\}$
$\{0,\{\{Busy,Free_B\},Free\}\}$	$finish$	$\{0,\{\{Full,Free_B\},Free\}\}$	$\{0,\{\{Full,Free_B\},Free\}\}$	$gobuffer2$	$\{0,\{\{Full,Free_B\},Free_B\}\}$

$\{0, \{\{Full, Free_B\}, Free\}\}$	$goM1$	$\{0, \{\{Full, Free\}, Free\}\}$	$\{0, \{\{Full, Free\}, Free\}\}$	$gobuffer2$	$\{0, \{\{Full, Free\}, Free_B\}\}$
$\{0, \{\{Full, Free\}, Free\}\}$	$pick1$	$\{0, \{\{Idle, Full\}, Free\}\}$	$\{0, \{\{Idle, Full\}, Free\}\}$	$gobuffer1$	$\{0, \{\{Idle, Full_B\}, Free\}\}$
$\{0, \{\{Idle, Full\}, Free\}\}$	$pull$	$\{0, \{\{Busy, Full\}, Free\}\}$	$\{0, \{\{Idle, Full\}, Free\}\}$	$gobuffer2$	$\{0, \{\{Idle, Full\}, Free_B\}\}$
$\{0, \{\{Busy, Full\}, Free\}\}$	$gobuffer1$	$\{0, \{\{Busy, Full_B\}, Free\}\}$	$\{0, \{\{Busy, Full\}, Free\}\}$	$finish$	$\{0, \{\{Full, Full\}, Free\}\}$
$\{0, \{\{Busy, Full\}, Free\}\}$	$gobuffer2$	$\{0, \{\{Busy, Full\}, Free_B\}\}$	$\{0, \{\{Busy, Full_B\}, Free\}\}$	$drop1$	$\{1, \{\{Busy, Free_B\}, Free\}\}$
$\{0, \{\{Busy, Full_B\}, Free\}\}$	$gobuffer2$	$\{0, \{\{Busy, Full_B\}, Free_B\}\}$	$\{0, \{\{Busy, Full_B\}, Free\}\}$	$finish$	$\{0, \{\{Full, Full_B\}, Free\}\}$
$\{0, \{\{Idle, Full_B\}, Free\}\}$	$drop1$	$\{1, \{\{Idle, Free_B\}, Free\}\}$	$\{0, \{\{Idle, Full_B\}, Free\}\}$	$gobuffer2$	$\{0, \{\{Idle, Full_B\}, Free_B\}\}$
$\{0, \{\{Idle, Full_B\}, Free\}\}$	$pull$	$\{0, \{\{Busy, Full_B\}, Free\}\}$	$\{1, \{\{Idle, Free_B\}, Free\}\}$	$pull$	$\{1, \{\{Busy, Free_B\}, Free\}\}$
$\{1, \{\{Idle, Free_B\}, Free\}\}$	$gobuffer2$	$\{1, \{\{Idle, Free_B\}, Free_B\}\}$	$\{1, \{\{Idle, Free_B\}, Free\}\}$	$goM1$	$\{1, \{\{Idle, Free\}, Free\}\}$
$\{1, \{\{Idle, Free\}, Free\}\}$	$pull$	$\{1, \{\{Busy, Free\}, Free\}\}$	$\{1, \{\{Idle, Free\}, Free\}\}$	$gobuffer2$	$\{1, \{\{Idle, Free\}, Free_B\}\}$
$\{1, \{\{Idle, Free\}, Free_B\}\}$	$pull$	$\{1, \{\{Busy, Free\}, Free_B\}\}$	$\{1, \{\{Idle, Free\}, Free_B\}\}$	$pick2$	$\{0, \{\{Idle, Free\}, Full_B\}\}$
$\{0, \{\{Idle, Free\}, Full_B\}\}$	$goramp$	$\{0, \{\{Idle, Free\}, Full\}\}$	$\{0, \{\{Idle, Free\}, Full_B\}\}$	$pull$	$\{0, \{\{Busy, Free\}, Full_B\}\}$
$\{0, \{\{Idle, Free\}, Full\}\}$	$pull$	$\{0, \{\{Busy, Free\}, Full\}\}$	$\{0, \{\{Idle, Free\}, Full\}\}$	$drop2$	$\{0, \{\{Idle, Free\}, Free\}\}$
$\{1, \{\{Idle, Free_B\}, Free_B\}\}$	$pull$	$\{1, \{\{Busy, Free_B\}, Free_B\}\}$	$\{1, \{\{Idle, Free_B\}, Free_B\}\}$	$goM1$	$\{1, \{\{Idle, Free\}, Free_B\}\}$
$\{1, \{\{Idle, Free_B\}, Free_B\}\}$	$pick2$	$\{0, \{\{Idle, Free_B\}, Full_B\}\}$	$\{0, \{\{Idle, Free_B\}, Full_B\}\}$	$goramp$	$\{0, \{\{Idle, Free_B\}, Full\}\}$
$\{0, \{\{Idle, Free_B\}, Full_B\}\}$	$pull$	$\{0, \{\{Busy, Free_B\}, Full_B\}\}$	$\{0, \{\{Idle, Free_B\}, Full_B\}\}$	$goM1$	$\{0, \{\{Idle, Free\}, Full_B\}\}$
$\{0, \{\{Idle, Free_B\}, Full\}\}$	$pull$	$\{0, \{\{Busy, Free_B\}, Full\}\}$	$\{0, \{\{Idle, Free_B\}, Full\}\}$	$drop2$	$\{0, \{\{Idle, Free_B\}, Free\}\}$
$\{0, \{\{Idle, Free_B\}, Full\}\}$	$goM1$	$\{0, \{\{Idle, Free\}, Full\}\}$	$\{0, \{\{Idle, Free_B\}, Free\}\}$	$pull$	$\{0, \{\{Busy, Free_B\}, Free\}\}$
$\{0, \{\{Idle, Free_B\}, Free\}\}$	$gobuffer2$	$\{0, \{\{Idle, Free_B\}, Free_B\}\}$	$\{0, \{\{Idle, Free_B\}, Free\}\}$	$goM1$	$\{0, \{\{Idle, Free\}, Free\}\}$
$\{0, \{\{Idle, Free_B\}, Free_B\}\}$	$pull$	$\{0, \{\{Busy, Free_B\}, Free_B\}\}$	$\{0, \{\{Idle, Free_B\}, Free_B\}\}$	$goM1$	$\{0, \{\{Idle, Free\}, Free_B\}\}$
$\{0, \{\{Full, Free_B\}, Free_B\}\}$	$goM1$	$\{0, \{\{Full, Free\}, Free_B\}\}$	$\{0, \{\{Busy, Free_B\}, Free_B\}\}$	$finish$	$\{0, \{\{Full, Free_B\}, Free_B\}\}$
$\{0, \{\{Busy, Free_B\}, Free_B\}\}$	$goM1$	$\{0, \{\{Busy, Free\}, Free_B\}\}$	$\{0, \{\{Busy, Full_B\}, Full\}\}$	$drop1$	$\{1, \{\{Busy, Free_B\}, Full\}\}$
$\{0, \{\{Busy, Full_B\}, Full\}\}$	$drop2$	$\{0, \{\{Busy, Full_B\}, Free\}\}$	$\{0, \{\{Busy, Full_B\}, Full\}\}$	$finish$	$\{0, \{\{Full, Full_B\}, Full\}\}$
$\{1, \{\{Idle, Free_B\}, Full_B\}\}$	$goramp$	$\{1, \{\{Idle, Free_B\}, Full\}\}$	$\{1, \{\{Idle, Free_B\}, Full_B\}\}$	$pull$	$\{1, \{\{Busy, Free_B\}, Full_B\}\}$
$\{1, \{\{Idle, Free_B\}, Full_B\}\}$	$goM1$	$\{1, \{\{Idle, Free\}, Full_B\}\}$	$\{1, \{\{Idle, Free\}, Full_B\}\}$	$goramp$	$\{1, \{\{Idle, Free\}, Full\}\}$
$\{1, \{\{Idle, Free\}, Full_B\}\}$	$pull$	$\{1, \{\{Busy, Free\}, Full_B\}\}$	$\{1, \{\{Idle, Free\}, Full\}\}$	$pull$	$\{1, \{\{Busy, Free\}, Full\}\}$
$\{1, \{\{Idle, Free\}, Full\}\}$	$drop2$	$\{1, \{\{Idle, Free\}, Free\}\}$	$\{1, \{\{Idle, Free_B\}, Full\}\}$	$pull$	$\{1, \{\{Busy, Free_B\}, Full\}\}$
$\{1, \{\{Idle, Free_B\}, Full\}\}$	$drop2$	$\{1, \{\{Idle, Free_B\}, Free\}\}$	$\{1, \{\{Idle, Free_B\}, Full\}\}$	$goM1$	$\{1, \{\{Idle, Free\}, Full\}\}$
$\{0, \{\{Idle, Full_B\}, Full\}\}$	$drop1$	$\{1, \{\{Idle, Free_B\}, Full\}\}$	$\{0, \{\{Idle, Full_B\}, Full\}\}$	$drop2$	$\{0, \{\{Idle, Full_B\}, Free\}\}$
$\{0, \{\{Idle, Full_B\}, Full\}\}$	$pull$	$\{0, \{\{Busy, Full_B\}, Full\}\}$	$\{0, \{\{Busy, Full\}, Full_B\}\}$	$goramp$	$\{0, \{\{Busy, Full\}, Full\}\}$
$\{0, \{\{Busy, Full\}, Full_B\}\}$	$gobuffer1$	$\{0, \{\{Busy, Full_B\}, Full_B\}\}$	$\{0, \{\{Busy, Full\}, Full_B\}\}$	$finish$	$\{0, \{\{Full, Full\}, Full_B\}\}$
$\{0, \{\{Busy, Full\}, Full\}\}$	$gobuffer1$	$\{0, \{\{Busy, Full_B\}, Full\}\}$	$\{0, \{\{Busy, Full\}, Full\}\}$	$drop2$	$\{0, \{\{Busy, Full\}, Free\}\}$
$\{0, \{\{Busy, Full\}, Full\}\}$	$finish$	$\{0, \{\{Full, Full\}, Full\}\}$	$\{0, \{\{Idle, Full\}, Full\}\}$	$gobuffer1$	$\{0, \{\{Idle, Full_B\}, Full\}\}$
$\{0, \{\{Idle, Full\}, Full\}\}$	$pull$	$\{0, \{\{Busy, Full\}, Full\}\}$	$\{0, \{\{Idle, Full\}, Full\}\}$	$drop2$	$\{0, \{\{Idle, Full\}, Free\}\}$
$\{0, \{\{Full, Free\}, Full\}\}$	$drop2$	$\{0, \{\{Full, Free\}, Free\}\}$	$\{0, \{\{Full, Free\}, Full\}\}$	$pick1$	$\{0, \{\{Idle, Full\}, Full\}\}$
$\{0, \{\{Full, Free_B\}, Full\}\}$	$drop2$	$\{0, \{\{Full, Free_B\}, Free\}\}$	$\{0, \{\{Full, Free_B\}, Full\}\}$	$goM1$	$\{0, \{\{Full, Free\}, Full\}\}$
$\{1, \{\{Full, Free\}, Free_B\}\}$	$pick1$	$\{1, \{\{Idle, Full\}, Free_B\}\}$	$\{1, \{\{Full, Free\}, Free_B\}\}$	$pick2$	$\{0, \{\{Full, Free\}, Full_B\}\}$
$\{0, \{\{Busy, Full_B\}, Free_B\}\}$	$drop1$	$\{1, \{\{Busy, Free_B\}, Free_B\}\}$	$\{0, \{\{Busy, Full_B\}, Free_B\}\}$	$finish$	$\{0, \{\{Full, Full_B\}, Free_B\}\}$
$\{0, \{\{Idle, Full_B\}, Free_B\}\}$	$drop1$	$\{1, \{\{Idle, Free_B\}, Free_B\}\}$	$\{0, \{\{Idle, Full_B\}, Free_B\}\}$	$pull$	$\{0, \{\{Busy, Full_B\}, Free_B\}\}$
$\{0, \{\{Busy, Free\}, Free\}\}$	$gobuffer2$	$\{0, \{\{Busy, Free\}, Free_B\}\}$	$\{0, \{\{Busy, Free\}, Free\}\}$	$finish$	$\{0, \{\{Full, Free\}, Free\}\}$