



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



ESCUELA TÉCNICA
SUPERIOR INGENIEROS
INDUSTRIALES VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DEVELOPMENT AND PROGRAMMING OF ALGORITHMS FOR THE AUTOMATIC COLLISION AVOIDANCE. APPLICATION TO TERRESTRIAL MOBILE ROBOTS

AUTHORESS: PALOMA BAHILO ALPUENTE

SUPERVISOR: ÁNGEL VALERA FERNÁNDEZ

SUPERVISOR: VICENTE FERMÍN CASANOVA CALVO

Academic year: 2017-18

“A mis padres,
por ser mi símbolo de esfuerzo y constancia.”



ACKNOWLEDGEMENTS

This project is the result of a six-year journey across the industrial engineering universe, which was round out with an intense exchange term at the Hong Kong University of Science and Technology. It has been a worth taken journey that has driven me to the fascinating area of Control, Automation and Robotics. I gratefully thank my two advisors, Ángel Valera Fernández and Vicente Fermín Casanova Calvo, for their enthusiastic and wise guidance, dedication and encouragement. I would also like to thank my family for making everything easy and pushing me forward to achieve my goals.



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots



ABSTRACT

The goal of this Master Thesis is to design and implement a methodology and system to endow a mobile terrestrial robot with the capability to efficiently reach its goals in a bounded environment that can be constrained with unknown obstacles. To meet this goal, the robotic system first identifies the obstacles in the environment and then plans the best robot trajectory that efficiently avoids any possible collision, which is a necessary requirement for the robot to stay safe and completely autonomous. The technique proposed in this thesis supports planning and deploying of such complex robot maneuvers in a real environment and can be applied to any industrial environment. This is done by combining simple models, efficient implementations, and interactive simulations that leverage the agility and maneuverability of the robot. First, an optimal solution to the problem of obstacle avoidance is achieved by developing in Simulink a controller that relies on a Matlab implementation of the A* algorithm, a classical Artificial Intelligence algorithm that is able to compute minimal cost paths from a start point to a target point in a bounded area. The proposed navigation strategy is first tested with S-Functions to interactively validate the controller's behavior, and then in co-simulation of Simulink and the Simscape Multibody framework for a more real view that considers the robot's physical properties and friction. Starting from a matrix of input values that correspond to an image of the environment obtained by using a monocular camera, the image is segmented and processed in the Matlab software environment, and an optimal trajectory is computed which leads from the starting point to the target point without collision. The path trajectory is evaluated by using co-simulation to visually analyse the estimated path trajectory followed by the robot. Then, the technique is implemented in a real robotic system, the LEGO MINDSTORMS EV3, by following a methodology that also combines two different software systems following two complementary approaches: a more academic one, by using LEGO MINDSTORMS EV3 support library for Simulink, which allows the robot real-time behavior to be evaluated, and a more professional one, by using the C-based robotics programming language RobotC, which witnesses its applicability to any industrial system.

Keywords: Mobile robot, Optimal path, A* Algorithm, Unknown Obstacles, Obstacle Avoidance, Co-Simulation



RESUMEN

El objetivo de este Trabajo Fin de Máster es diseñar e implementar una metodología y sistema para dotar a un robot terrestre móvil con la capacidad de alcanzar eficientemente una posición objetivo dentro de un entorno limitado y que puede verse condicionado por obstáculos desconocidos. Para cumplir este objetivo, el sistema robótico identifica en primer lugar los obstáculos presentes en el entorno y, a continuación, planifica la mejor trayectoria que permita al robot evitar de forma eficiente cualquier posible colisión, lo cual es un requisito necesario para su seguridad y completa autonomía. La técnica propuesta aborda la planificación y el despliegue de tales maniobras complejas del robot en un entorno real y son aplicables a cualquier entorno industrial. Esto se logra combinando modelos simples, implementaciones eficientes y simulaciones interactivas que aprovechan la agilidad y la maniobrabilidad del robot. En primer lugar, se obtiene una solución óptima al problema de la evasión de obstáculos mediante el desarrollo de un controlador en Simulink que se basa en una implementación en Matlab del algoritmo A*, un algoritmo clásico de Inteligencia Artificial que permite calcular trayectorias de coste mínimo desde un punto inicial hasta un objetivo dado en un área delimitada. La estrategia de navegación propuesta se prueba primero con S-Functions para validar interactivamente el comportamiento del controlador, y luego en co-simulación de Simulink y el framework Simscape Multibody, para obtener una visión más real al considerar las propiedades físicas del robot y la fricción. A partir de una matriz de entradas que representa una imagen del entorno obtenida utilizando una cámara monocular, la imagen se segmenta y se procesa utilizando la herramienta software Matlab, y se calcula una trayectoria óptima para ir desde un punto inicial a un punto final evadiendo los obstáculos del entorno. La trayectoria calculada se evalúa entonces mediante co-simulación, lo que permite analizar visualmente la ruta estimada seguida por el robot. A continuación, la técnica se implementa en un sistema robótico real, el LEGO MINDSTORMS EV3, siguiendo una metodología que combina también dos sistemas software diferentes siguiendo dos enfoques complementarios: uno más académico, utilizando la biblioteca de soporte de LEGO MINDSTORMS EV3 para Simulink, que permite evaluar su comportamiento en tiempo real y otro más profesional, utilizando el lenguaje de programación basado en C para robótica RobotC, que atestigua su aplicabilidad a cualquier sistema industrial.

Palabras clave: Robot Móvil, Camino Óptimo, Algoritmo A*, Evasión de obstáculos, Co-simulación



RESUM

L'objectiu d'aquest Treball Fi de Màster és dissenyar i implementar una metodologia i sistema per dotar un robot terrestre mòbil amb la capacitat d'arribar eficientment a una posició objectiu dins d'un entorn limitat i que es pot veure condicionat per obstacles desconeguts. Per complir aquest objectiu, el sistema robòtic identifica en primer lloc els obstacles presents en l'entorn i tot seguit planifica la millor trajectòria que permeti al robot evitar de manera eficient qualsevol possible col·lisió, el qual és un requisit necessari per a la seva seguretat i completa autonomia. La tècnica proposta aborda la planificació i el desplegament d'aquestes maniobres complexes del robot en un entorn real i són aplicables a qualsevol entorn industrial. Això s'aconsegueix combinant models simples, implementacions eficients i simulacions interactives que aprofiten l'agilitat i la maniobrabilitat del robot. En primer lloc, s'obté una solució òptima al problema de l'evasió d'obstacles mitjançant el desenvolupament d'un controlador en Simulink que es basa en una implementació Matlab de l'algoritme A*, un algoritme clàssic d'Intel·ligència Artificial que permet calcular trajectòries de cost mínim des d'un punt inicial fins a un objectiu donat en una àrea delimitada. L'estratègia de navegació proposta es prova primer amb S-Functions per provar visualment el comportament del controlador, i després en co-simulació de Simulink i el framework Simscape Multibody, per obtenir una visió més real en considerar les propietats físiques del robot i la fricció. A partir d'una matriu d'entrades que representa una imatge de l'entorn obtinguda utilitzant una càmera monocular, la imatge es segmenta i es processa utilitzant el primer sistema programari, l'eina Matlab, i es calcula una trajectòria òptima per anar des d'un punt inicial a un punt final evadint els obstacles de l'entorn. La trajectòria calculada s'avalua llavors mitjançant co-simulació, el que permet analitzar visualment la ruta estimada seguida pel robot. A continuació, la tècnica s'implementa en un sistema robòtic real, el LEGO MINDSTORMS EV3, seguint una metodologia que combina també dos sistemes programari diferents seguint dos enfocaments complementaris: un més acadèmic, utilitzant la biblioteca de suport de LEGO MINDSTORMS EV3 per Simulink, que permet avaluar el seu comportament en temps real i un altre més professional, utilitzant l'entorn de programació basat en C per a robòtica RobotC, que testimonia la seva possible aplicabilitat a qualsevol sistema industrial.

Paraules clau: Robot Mòbil, Camí Òptim, Algoritme A*, Evasió d'Obstacles, Co-Simulació



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots



GENERAL INDEX

CONTENTS OF THE TFM

- DOCUMENT N°1: Project Report
- DOCUMENT N°2: Project Budget

DOCUMENT N°1: PROJECT REPORT

CHAPTER 1: Introduction	1
1.1. Objectives of this project.....	1
1.2. Motivation. A layman's Introduction to Robotics.....	3
1.3. Background: Fundamentals of Path Finding and Obstacle Avoidance	4
CHAPTER 2: Fundamentals of Robotic Manipulation: Rigid Body Transformations	7
2.1. Rigid Body Transformations	7
2.2. Properties of Rotation Matrices	8
2.3. Rotation Representations.....	10
2.4. Rigid Body Motions.....	14
2.5. Rigid Body Velocities	15
CHAPTER 3: A* Algorithm in Artificial Intelligence	17
3.1. Algorithms and Artificial Intelligence	17
3.2. A-STAR: Combination of Uniform Cost Search and Heuristic Search.....	17
3.3. Tree Construction	18
CHAPTER 4: Backbone of the Proposed Methodology	21
4.1. The Proposed Methodology	21
CHAPTER 5: Implementation of the A* Algorithm for our target robot	25
5.1. The Proposed Technique	25
5.2. Implementation.....	26
5.3. Piecewise Cubic Hermite Interpolating Polynomial.....	31



CHAPTER 6: Forward and Inverse Kinematics and Robot Configuration	33
6.1. Introduction to Robot Kinematics	33
6.2. Introduction to Forward Kinematics.....	33
6.3. Introduction to Inverse Kinematics.	34
6.4. Robot Configuration.	34
6.5. Forward Kinematics with Differential Configuration Model.	35
6.6. Inverse Kinematics with Differential Configuration Model.	36
CHAPTER 7: Position Estimation and Control of Mobile Robots.....	37
7.1. Position Estimation.....	37
7.2. Control of Mobile Robots	38
CHAPTER 8: Simulink Visualization using S-Functions and Simscape Multibody Simulation	41
8.1. S-Functions	41
8.2. Simscape Multibody	45
CHAPTER 9: Implementation Platforms for EV3	55
9.1. LEGO MINDSTORMS EV3 Support for Simulink	56
9.2. RobotC Programming Language and Platform	58
CHAPTER 10: Computer Vision System for Obtention of the Environment Configuration ..	61
10.1. Modeling the Environment Configuration by means of a Monocular Camera	61
10.2. Perspective Modeling	62
10.3. Camera Calibration	64
CHAPTER 11: Application of the Developed Methodology and System to a Representative Case Study	69
11.1. Obstacle Detection by Camera Vision System and Transference to Matlab	70
11.2. Computing the Optimal Path Trajectory using A* Algorithm	71
11.3. Simulation of the Optimal Path	74
11.4. Simulink Implementation for EV3.....	77
11.5. RobotC Implementation for EV3	82
CHAPTER 12: Conclusion and Evaluation	83
12.1. Conclusion	83
12.2. Evaluation	84
CHAPTER 13: References	87



DOCUMENT Nº2: PROJECT BUDGET

CHAPTER 1. Unit Prices	89
1.1. Analysis of the Path Planning and Obstacle Avoidance Techniques	89
1.2. Design of a Navigation Methodology	89
1.3. Simulation of the Methodology in two different environments	89
1.3. Application of the obstacle avoidance in the EV3	89
1.4. Food expenses and Meetings	90
1.5. Additional tasks and Activities.....	90
1.6. Other Concepts.....	90
 CHAPTER 2: Measurements	 91
2.1. Analysis of the Path Planning and Obstacle Avoidance Techniques	91
2.2. Design of a Navigation Methodology	91
2.3. Simulation of the Methodology in two different environments	91
2.4. Application of the obstacle avoidance in the EV3	92
2.5. Food expenses and Meetings	92
2.6. Additional tasks and Activities.....	92
2.7. Other Concepts.....	92
 CHAPTER 3: Detailed Budget.....	 93
3.1. Analysis of the Path Planning and Obstacle Avoidance Techniques.....	93
3.2. Simulation of the Methodology in two different environments	93
3.3. Simulation of the obstacle avoidance in two different environments	93
3.4. Application of the obstacle avoidance in the EV3	94
3.5. Food expenses and Meetings	94
3.6. Additional tasks and Activities.....	94
3.7. Other Concepts.....	94
 CHAPTER 4: Total Budget of the Project	 95



INDEX OF FIGURES

Figure 1. Rigid Body Displacement.....	8
Figure 2. Rotation axes.....	9
Figure 3. Composition Rule	9
Figure 4. Examples of differential manifolds locally resembled to R^2	10
Figure 5. Exponential Coordinates	13
Figure 6. Transformation between different coordinate frames.....	14
Figure 7. Composition of rigid body motion	14
Figure 8. A* Algorithm Pseudocode.....	19
Figure 9. Intuitive Trajectory from Start Cell (0.4, 0.4) to Target Cell (1.8, 1.8).....	20
Figure 10. Path generated to avoid one obstacle	20
Figure 11. Path generated to avoid two obstacles.....	20
Figure 12. The proposed methodology.....	21
Figure 13. Example 1, A* Trajectory for Map 1.....	28
Figure 14. Example 2, A* Trajectory for Map 2.....	28
Figure 15. Example 3, A* Trajectory for Map 3.....	29
Figure 16. Example 4, A* Trajectory for Map 4.....	29
Figure 17. Example 5, A* Trajectory for Map 5.....	30
Figure 18. Example 6, A* Trajectory for Map 6.....	30
Figure 19. Pchip vs Spline.....	31
Figure 20. Pchip (red) vs Spline (blue) for our target robot	32
Figure 21. Forward Kinematics	33
Figure 22. Inverse Kinematics	34
Figure 23. Differential Configuration	35
Figure 24. Simulink Forward Kinematics.....	36
Figure 26. Mobile Robot Control.....	38



Figure 27. Dynamic Control Simulink Structure	38
Figure 28. Dynamic Control Simulink Structure	39
Figure 29. Position Control by Decentralized Point	40
Figure 30. Simulink Global Structure for S-Functions	41
Figure 31. Example 1, S-Functions Map 1	42
Figure 32. Example 2, S-Functions Map 2	43
Figure 33. Example 3, S-Functions Map 3	43
Figure 34. Example 4, S-Functions Map 4	44
Figure 35. Example 5, S-Functions Map 5	44
Figure 36. Example 6, S-Functions Map 6	45
Figure 37. EV3 Real Model	46
Figure 38. Body Structure of the Robot in .stl.....	46
Figure 39. The Robot following the Path Simulation	47
Figure 40. The wheel's properties.....	47
Figure 41. General Simulink Structure	48
Figure 42. Joints used in the body, considering gravity	49
Figure 43. General Floor structure with third hierarchical level subsystem	50
Figure 44. General EV3 structure	51
Figure 45. Each of the wheel's structure model	52
Figure 46. The Wheel's Behavior	52
Figure 47. SM-Contact Forces for each wheel	53
Figure 48. The Resulting Simulation Side View	53
Figure 49. The Resulting Upper Simulation.....	54
Figure 50. Simulation Behaviour: Desired, Estimated and Real Trajectory.....	54
Figure 51. Simulink Support Package for LEGO MINSTORMS EV3 Hardware	56
Figure 52. Global Structure	57
Figure 53. Wheel's Structure	57
Figure 54. System Structure, considering Kinematics and Inverse Kinematics control	57
Figure 55. Simulation Behaviour vs Real Behaviour.....	58
Figure 56. Thin lens perspective projection	61
Figure 57. Pin -hole Camera Model.....	62



Figure 58. Perspective Projection	63
Figure 59. Pin-hole Camera Model	64
Figure 60. Calibration object.....	64
Figure 62. Case Study: Environment Configuration 1	70
Figure 63. Case Study: Environment Configuration 2	71
Figure 64. The Initial Position (left) and the Eroded and Segmented obstacles (right) of Configuration 2	71
Figure 65. The found path in pixels for Environment Configuration 1.....	72
Figure 66. The found path in pixels for Environment Configuration 2.....	72
Figure 67. Obstacle avoidance Path for Environment Configuration 1.....	73
Figure 68. Obstacle avoidance Path for Environment Configuration 2.....	73
Figure 69. Visualization with S-Functions for Configuration 1	74
Figure 70. Visualization with S-Functions for Configuration 2	75
Figure 71. Simulink Structure requiring changes for a new environment configuration.....	75
Figure 72. Simulated behavior of the robot with Simscape.....	76
Figure 73. Wheel Velocity	76
Figure 74. Simulation with Simscape Multibody Side View	77
Figure 75. Simulation with Simscape Multibody Upper View	77
Figure 76. Simulink Model for Host computer.....	78
Figure 77. Model for LEGO MINDSTORMS EV3.....	78
Figure 78. EV3 Robot following the path trajectory computed by the Simulink implementation	79
Figure 79. Path followed by the robot under the Simulink control implementation.....	79
Figure 80. X Values with Simulink Implementation	80
Figure 81. Y Values with Simulink Implementation.....	80
Figure 82. Motor C Control Behaviour	81
Figure 83. Motor B Control Behaviour	81
Figure 84. EV3 Robot following the path trajectory computed by the RobotC implementation	82



INDEX OF TABLES

Table 1. Rotation Matrix	10
Table 2. Euler Angles	11
Table 3. Euler Angle Conversions	12
Table 4. Extraction of Environment Configuration using Monocular Camera.....	66



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots

**MASTER DEGREE IN
INDUSTRIAL ENGINEERING
FINAL PROJECT**

**DOCUMENT Nº1:
PROJECT REPORT**



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots



DOCUMENT Nº1: PROJECT REPORT

DOCUMENT Nº1: PROJECT REPORT

CHAPTER 1: Introduction	1
1.1. Objectives of this project.....	1
1.2. Motivation. A layman’s Introduction to Robotics.....	3
1.3. Background: Fundamentals of Path Finding and Obstacle Avoidance	4
CHAPTER 2: Fundamentals of Robotic Manipulation: Rigid Body Transformations	7
2.1. Rigid Body Transformations	7
2.2. Properties of Rotation Matrices	8
2.3. Rotation Representations.....	10
2.4. Rigid Body Motions.....	14
2.5. Rigid Body Velocities	15
CHAPTER 3: A* Algorithm in Artificial Intelligence	17
3.1. Algorithms and Artificial Intelligence	17
3.2. A-STAR: Combination of Uniform Cost Search and Heuristic Search.....	17
3.3. Tree Construction	18
CHAPTER 4: Backbone of the Proposed Methodology	21
4.1. The Proposed Methodology	21
CHAPTER 5: Implementation of the A* Algorithm for our target robot	25
5.1. The Proposed Technique	25
5.2. Implementation.....	26
5.3. Piecewise Cubic Hermite Interpolating Polynomial.....	31
CHAPTER 6: Forward and Inverse Kinematics, and Robot Configuration	33
6.1. Introduction to Robot Kinematics	33
6.2. Introduction to Forward Kinematics.....	33
6.3. Introduction to Inverse Kinematics.	34



6.4. Robot Configuration.....	34
6.5. Forward Kinematics with Differential Configuration Model.....	35
6.6. Inverse Kinematics with Differential Configuration Model.....	36
CHAPTER 7: Position Estimation and Control of Mobile Robots.....	37
7.1. Position Estimation.....	37
7.2. Control of Mobile Robots.....	38
CHAPTER 8: Simulink Visualization using S-Functions and Simscape Multibody Simulation	41
8.1. S-Functions.....	41
8.2. Simscape Multibody.....	45
CHAPTER 9: Implementation Platforms for EV3.....	55
9.1. LEGO MINDSTORMS EV3 Support for Simulink.....	56
9.2. RobotC Programming Language and Platform.....	58
CHAPTER 10: Computer Vision System for Obtention of the Environment Configuration ..	61
10.1. Modeling the Environment Configuration by means of a Monocular Camera	61
10.2. Perspective Modeling	62
10.3. Camera Calibration	64
CHAPTER 11: Application of the Developed Methodology and System to a Representative Case Study	69
11.1. Obstacle Detection by Camera Vision System and Transference to Matlab	70
11.2. Computing the Optimal Path Trajectory using A* Algorithm.....	71
11.3. Simulation of the Optimal Path	74
11.4. Simulink Implementation for EV3.....	77
11.5. RobotC Implementation for EV3.....	82
CHAPTER 12: Conclusion and Evaluation.....	83
12.1. Conclusion	83
12.2. Evaluation	84
CHAPTER 13: References	87



CHAPTER 1. INTRODUCTION

1.1. OBJECTIVES OF THIS PROJECT

The problem of path planning for a mobile robot is the ability to compute collision-free feasible routes from a specified start location to a desired goal destination through an unknown environment with unknown obstacles, while satisfying certain optimization criteria. This MSc Thesis develops a path planning methodology and system for a prototype industrial robot that deploys optimal non-colliding robot trajectories in a real environment. The process of obstacle avoidance is controlled by using a heuristic informed search algorithm, called A* (pronounced as "A star"), which is generally used to efficiently plot paths of minimal cost (e.g., shortest time or least distance travelled) from a starting point to a target point in a bounded area. The project also addresses the effective implementation of the path planner on a real system, the mobile terrestrial robot LEGO MINDSTORMS EV3. To reach the target without collisions, the developed robotic system has been endowed with perception, decision-making, path planning, and actions capabilities.

To validate the developed control strategy, two types of simulations have been developed: the first one consists of a trajectory visualization by using S-Functions and the second, more realistic and faithful one, by using the Simscape Multibody framework which considers the robot's physical properties and friction. Starting from a matrix of input values that correspond to an image of the environment obtained by using a monocular camera, the image is segmented and processed in Matlab and an optimal non-colliding trajectory is computed that reaches the goal. The path trajectory is evaluated by using co-simulation to visually analyse the estimated path trajectory followed by the robot. Then, the technique is implemented in a real robotic system, the LEGO MINDSTORMS EV3. Two implementations are provided. The first one relies on LEGO MINDSTORMS EV3 support library for Simulink, which allows the robot real-time behavior and control action on the wheels to be evaluated with regard to the Simulink reference model. The second one is developed in the robotics programming environment RobotC, which is used to provide an efficient, C-based implementation of the controller for EV3 to correctly trace the optimal collision-free output vector trajectory. Without loss of generality, the obstacles are statically fixed, which allows us to focus on the optimization problem and key important aspects of the process such as real-time obstacle detection, optimal control, and co-simulation.

There are many applications where mobile terrestrial robots that are controlled by the designed and implemented methodology and system can be used. This includes inspection and



maintenance applications, world modelling (environment mapping), intelligent surveillance, reconnaissance actions in military environments, material handling and transport applications, and exploration and mining robots, among others.

The thesis manuscript is organized into thirteen chapters:

CHAPTER 1. This chapter provides an introduction to the objectives, motivation and background of the master thesis.

CHAPTER 2. This chapter contains a brief introduction to *Rigid Body Transformations*, a basic mathematical theory for robotic manipulation that is needed to understand the project.

CHAPTER 3. This chapter summarizes the algorithm A* that is used to generate the fittest path from a start point to a target point while avoiding any obstacles in the environment.

CHAPTER 4. This chapter describes the methodology that is used to construct the robotic system developed in this project.

CHAPTER 5. This chapter explains how the typical pathfinding A* algorithm is adapted to compute the optimal collision-free path to allow mobile robots automatically avoid obstacles.

CHAPTER 6. This chapter summarizes the main concepts of *Forward and Inverse Kinematics* that allow the cartesian space and the joint space of the robotic problem to be related by suitable composing transformations.

CHAPTER 7. This chapter presents the fundamentals of position estimation and control in mobile robotics.

CHAPTER 8. This chapter provides a detailed explanation of the visualization of robot trajectories by means of S-functions. Then, realistic simulations are addressed by using the Simscape Multibody framework that allows friction to be considered by observing the main physical properties of the robot, such as body mass, inertia, and density.

CHAPTER 9. This chapter describes the implementation of the proposed navigation technique on a real system, the EV3. Navigation is decomposed into three tasks: mapping and modelling the environment; path planning; and path following with collision avoidance.

CHAPTER 10. This chapter focusses on the computer vision system used for perceiving the environment configuration, including obstacle detection.

CHAPTER 11. This chapter presents a complete review of the developed methodology and its application to a real case study.



CHAPTER 12. Evaluation and Conclusions.

CHAPTER 13. References.

1.2. MOTIVATION. A LAYMAN'S INTRODUCTION TO ROBOTICS

In a worldwide constantly evolving economy, robots have become crucial artifacts that make life easier. Robots are extensively used in many hazardous industrial fields where there may be dangers for people, such as the nuclear power industry, the mining industry, disaster sites exploration, and aerospace research.

In the past 30 years, the number of robots has grown exponentially in the world. Looking at the current cutting-edge developments in robotics, they range from industrial robots for assembly lines that are revolutionizing manufacturing, to surgical robots, legged robots and fully automated drones that are used for intelligent surveillance and reconnaissance. Since this rapidly growing field is just at the beginning of a vertiginous evolution process, there is widespread interest in robotics, with a huge potential of growth in the next years and big expectations for novel robot applications.

As mentioned in the *Harvard Business Review* (Lyll et al, 2018): “*Robots are improving productivity and margins in retail warehouses and fulfillment centers. Delivery drones and self-driving vehicles aren't far off. Rio Tinto, the global mining-and-metals company, is exploring how digital technologies can automate mine-to-port operations. Using driverless trains, robotic operators, cameras, lasers, and tracking sensors, the company will be able to manage the whole supply chain remotely — while improving safety and reducing the need for workers in remote locations. Many other leading companies are also exploring these possibilities, using robotics and artificial intelligence to digitize and automate labor-intensive, repetitive and transactional tasks and processes*”.

In this project, we focus on mobile robots although the proposed techniques could eventually be extended to other types of robotic structures with little effort (Valera Fernandez, Á., 2017a). Mobile robots appeared from the need to extend robotic applications. They are currently used in industry not only for material transport, but also for the recreation of human movements and for exploration and rescue missions. Their use has become popular in the army, where unmanned robots are not only remotely driven but can be fully autonomous

For robots to become entirely autonomous, collision avoidance is a key essential requirement that may allow it to find a safe path in a dangerous environment. This master thesis relies on an informed search approach that is known as A* for automatic collision avoidance in a robotic scenario, and validates the proposed robotic control by applying a cooperative simulation methodology. Cooperative simulation (a.k.a. co-simulation) is a methodology that allows individual components to be simulated with a variety of tools that run at the same time and exchange information in a collaborative environment. This



methodology has proven its effectiveness in different industrial fields such as chemistry, high-level computing, multibody dynamics, mechatronics, and structural mechanics. These developments have proven several advantages of co-simulation, such as the cooperative use of distinct specialized modeling tool environments, collaborative design, model obtainment for faster prototypes, immediate availability of the new model and development of processes that can be executed simultaneously.

1.3. BACKGROUND. FUNDAMENTALS OF PATH FINDING AND OBSTACLE AVOIDANCE

Mobile robot navigation refers to the robot's ability to move to a target position from an initial position given a prior knowledge of the environment. Path planning is a fundamental problem in robot navigation in order to obtain a non-collision route between two points in the presence of any kind of obstacles. There are several general methodologies to address the problem of path planning in computing, but they all have in common that a given map and goal location are needed to generate the geometric path towards the target position. In robotics, it must also be considered that, during its motion, the robot can face a number of obstacles (either static or moving), that may create an a-priori uncertainty about the optimal path. Hence, obstacle detection and avoidance must be considered for tracing the best path.

Throughout time, different algorithms for obstacle avoidance have been developed:

1. **Bug's algorithm:** In this method the robot moves towards the target position, unless an obstacle is found. In that case, the robot contours the obstacle until motion to target position is again achievable.
2. **Path planning using artificial potencial fields:** This method relies on the idea that the target position and obstacles generate a potencial field in the environment, in which the robot, which is considered as a particle, moves. The obstacles generate a repulsive potencial and the target position generates an attractive potencial. Also, a velocity vector of the robot must be defined to get to the target position without encountering any obstacle.
3. **The Vector Field Histogram:** Developed in 1991 by Johann Borenstein and Yoram Koren, this method is based on the generation of a polar histogram for the area close to the space occupied by the robot, being this histogram used to select the lowest polar obstacle density sector in order to direct the robot in that direction.
4. **Elastic Bands:** This methodology consists on a combination of real-time based robot control and global path planning, where the elastic band is just an obstacle free path, that is able to deform when an obstacle is detected to keep the trajectory collision-free.

5. **Dynamic approach:** This approach considers the robot's behavior as a non-linear dynamic system, where the obstacles are considered as an unstable equilibrium point, while the direction to the goal is considered as stable.
6. **Sampling based methods:** Two types must be distinguished:
 - a. **Probabilistic roadmap (PRM):** This method is based on the idea of building a graph that characterizes the free configuration space in a probabilistic manner and, afterwards, use a graph search algorithm to find the path. Even though this methodology can cope with high-dimensional systems and it is completely based on probability, it must be considered that, if only a limited amount of samples are used to train the system, a suboptimal solution is obtained, and that detection of possible collisions takes up the greatest part of the time.
 - b. **Rapidly exploring random tree (RRT):** This methodology builds up a tree while generating a next step configuration from the initial point configuration. Even though this methodology is simple, it is prone to be probabilistically incomplete. Due to this reason, a rewire function is introduced that is used to swap a new point in as a parent a node for the nearby vertices that can be reached along the shortest path through the new point. The extended methodology is called RRT*, and is asymptotically optimal (Karaman, S., and Frazzoli, E., 2011).
7. **Search-based methods:** For each search problem, there is a corresponding state space graph that is associated to a given search algorithm. However, in the search-based methods, a search tree of outcomes is considered for different plans, where each node in the search tree is an entire path in the program graph. By simply back-tracking a node in the search tree you return to the initial state. The goal of this type of methodology is to find the best path to the goal as soon as possible, without having to build up the whole tree:
 - a. **General graph search DFS, BFS:** Based on the use of a queue containing all the nodes for termination check, the queue is first initialized with the starting state and then the set of nodes is expanded within a generation loop until the path is found and the target state is achieved.
 - i. **Depth First Search (DFS):** The deepest node is removed (or expanded) by first considering a "last in first out" queue (LIFO).
 - ii. **Breadth First Search (BFS):** The shallowest node is expanded first, maintaining a "first in first out" (FIFO) queue. In practical search problems, there is a cost for moving from the current state to the next one, and, in case all weights are equal to one, this methodology finds the least-cost path with a minimal number of steps.
 - b. **Uniform Cost Search (UCS):** This methodology, also known as Dijkstra's algorithm, maintains a priority queue and expands the cheapest accumulated cost $g(n)$ node first, while exploring increasing



cost contours. However, it can only access the cost accumulated so far and no information regarding the target location.

- c. **Search Heuristics:** This methodology infers the least cost to goal and overcomes the limitations of uniform cost.
- d. **A* search:** This is the chosen technique that is deeply applied in this project. It is based on a combination of uniform cost search with a suitable heuristic. This methodology is fully explained in Chapter 3.

CHAPTER 2: FUNDAMENTALS OF ROBOTIC MANIPULATION: RIGID BODY TRANSFORMATIONS

2.1. RIGID BODY TRANSFORMATIONS

The movement of particles in an Euclidean space is defined by giving its location, at each instant of time, in regards to an inertial Cartesian coordinate frame. However, in the robotic field, the focus of interest lies in the collective movement of a set of particles rather than in individual particles. Therefore, we consider a rigid body as a group of particles with the distance between particles being constant, regardless of any forces applied to the body or body movements. Consequently, if p and q are two points on a rigid body, as the body moves, p and q must satisfy the following equational axiom (Murray, R., Li, Z. and Sastry, S., 1994):

$$\|p(t) - q(t)\| = \|p(0) - q(0)\| = C, \text{ for a given constant } C$$

The position and orientation of a rigid body is associated with a reference frame with three linearly independent basis vectors that are commonly denoted as a_1 , a_2 and a_3 for reference frame A , being a_1 , a_2 and a_3 orthogonal to each other. Each of these axes denotes the direction vectors, meaning that their norm is one. In robotics, it is commonly used the right-handed coordinate frames. Any vector can be written as a linear combination of basis vectors $v = v_1a_1 + v_2a_2 + v_3a_3$, where v_1 , v_2 , v_3 are the vector component of v on the coordinate frame a_1 , a_2 , a_3 respectively.

Let us introduce the notation that is used in this project, which follows the following norms:

- Vectors: \mathbf{x} , \mathbf{y} , \mathbf{a} , ...
- Reference frames: A, B, C or a, b, c
- Matrices: \mathbf{A} , \mathbf{B} , \mathbf{C}
- Transformations: ${}^A\mathbf{A}_B$, ${}^A\mathbf{R}_B$, \mathbf{A}_{ab} , $g_{ab}(\cdot)$

A rigid body displacement is any transformation of points between two positions and orientations. Given an object $O \in \mathbb{R}^n$, a rigid body displacement occurs in \mathbb{R}^n , and given a vector $v \in O$ from point p to q , its transformation is based on a displacement or a transformation of points, denoted as $g_*(v) = g(p) - g(q)$, where the distance of point p to q remains the same, which means that the length of the vector v in the original and transformed frames is the same. Given two vectors on the original frame, the transformed vectors of the transformed frame will have the same cross product as the original ones, as the cross products are preserved due to the fact that internal reflection is eliminated $(x, y, z) \rightarrow (x, y, -z)$ as shown in Figure 1 (Shen, 2017). In other words, rigid body transformations are mappings that satisfy these important properties:

- Lengths are preserved $\|g(p) - g(q)\| = \|q - p\|$
- Cross products are preserved $g_*(v) \times g_*(w) = g_*(v \times w)$
- Inner products are preserved $g_*(v) \cdot g_*(w) = v \cdot w$

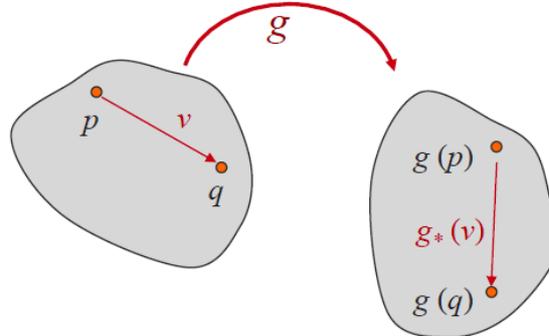


Figure 1. Rigid Body Displacement

2.2 PROPERTIES OF ROTATION MATRICES

Rotation matrices have two key properties:

1. Given rotation matrix $R \in \mathbb{R}^{3 \times 3}$ with $r_1, r_2, r_3 \in \mathbb{R}^3$ being its columns, since the columns of R are orthogonal, we have

$$r_i^T \cdot r_j = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

$$R \cdot R^T = I$$

2. Since the coordinate frame is right-handed: $r_2 \times r_3 = r_1$, which means that

$$\det R = r_1^T \cdot (r_2 \times r_3) = r_1^T \cdot r_1 = 1 \text{ (Special orthogonal group)}$$

The set of all 3x3 matrices which satisfy these two properties is denoted as $SO(3)$.

The set of all rotations form the *Special Orthogonal Algebraic* group, where :

$$SO(n) = \{R \in \mathbb{R}^{n \times n} \mid R \cdot R^T = I, \det R = 1\}$$

In this project, we consider $SO(3) = \{R \in \mathbb{R}^{3 \times 3} \mid R \cdot R^T = I, \det R = 1\}$, where $SO(3)$ is a group under the operation of matrix multiplication. This group satisfies the following axioms:

1. Closure: If $R_1, R_2 \in SO(3)$, then $R_1 \cdot R_2 \in SO(3)$
2. Identity: The identity element is the identity matrix I , that is $R_1 \cdot I = I \cdot R_1 = R_1$
3. Inverse: If $R \in SO(3)$, then $R^{-1} \in SO(3)$
4. Associativity : $R_1 \cdot (R_2 \cdot R_3) = (R_1 \cdot R_2) \cdot R_3$

After choosing a reference frame, a rigid body is described in the space by its orientation and position in regards to that frame, considering all coordinate frames are right-handed. Given a reference frame A , that is aligned to a reference frame B (same origin of axis), a

rotation matrix is required in order to transform any point $p \in \mathbb{R}^3$ from frame B to frame A. The principal axes of frame A follow the next nomenclature: $x = [1 \ 0 \ 0]^T$, $y = [0 \ 1 \ 0]^T$ and $z = [0 \ 0 \ 1]^T$ as shown in Figure 2; while the principal axes of frame B are $x_{ab}, y_{ab}, z_{ab} \in \mathbb{R}^3$ and the rotation matrix of the coordinates of principle axes of B related to A are denoted as R_{ab} .

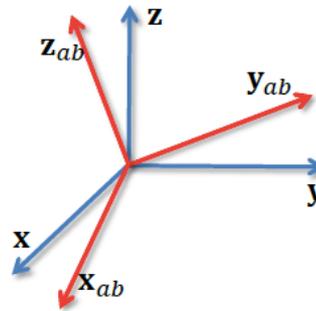


Figure 2. Rotation axes

Therefore, in order to transform point p from body frame B to inertial frame A, a rotation matrix R that shows the rotation of coordinate space B in frame A is used:

- $b_1 = R_{11}a_1 + R_{12}a_2 + R_{13}a_3$
- $b_2 = R_{21}a_1 + R_{22}a_2 + R_{23}a_3$
- $b_3 = R_{31}a_1 + R_{32}a_2 + R_{33}a_3$
- ${}^A R_B = [b_1 \ b_2 \ b_3] = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$
- $p_A = p_1 b_1 + p_2 b_2 + p_3 b_3 = [b_1 \ b_2 \ b_3] p_B$

Also, considering a third coordinate space C, the composition rule can be used to go from frame C to A as shown in Figure 3:

$$R_{ac} = R_{ab} \cdot R_{bc}$$

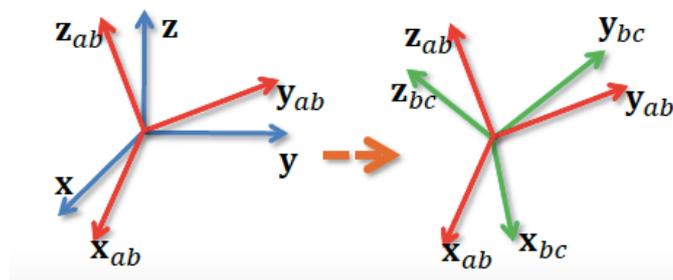


Figure 3. Composition Rule

where $R_{ab} = [x_{ab}, y_{ab}, z_{ab}]$ preserves the following axioms:

- Length: $\|R_{ab}(pb - qb)\| = \|pb - qb\|$
- Cross product: $R_{ab}(v \times w) = (R_{ab}v) \times (R_{ab}w)$, considering $R(v) \hat{=} RT = (Rv) \hat{}$, where

$$(a)^\wedge = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \text{ is the skew-symmetric matrix, and } a \times b = (a)^\wedge b$$

2.3 ROTATION REPRESENTATIONS

Considering $SO(3)$ a continuous group, where the multiplication and inverse operations are also continuous, $SO(3)$ is a smooth manifold. A manifold of dimension n is a set M , that presents near each point, a locally resemblance to Euclidean space \mathbb{R}^n , as shown in Figure 4 (Shen, 2017):

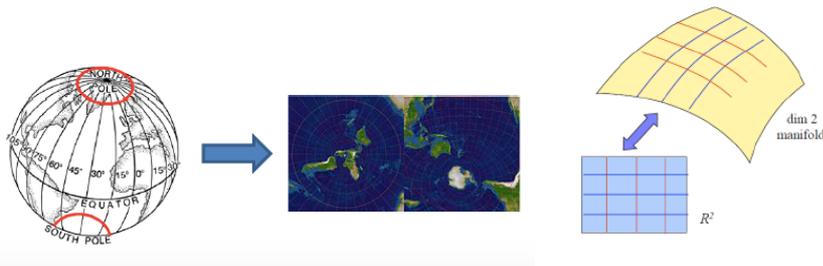


Figure 4. Examples of differential manifolds locally resembled to \mathbb{R}^2

There are several ways to represent rotations:

1. Rotation matrix: classically described as

Table 1. Rotation Matrix

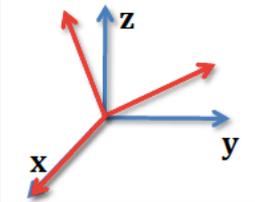
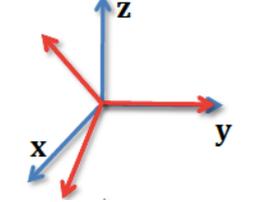
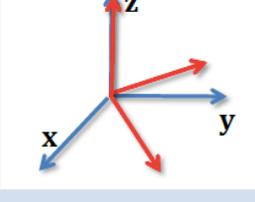
Rotation Matrix	Representation
$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$	
$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$	
$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	

2. Euler angles: based on elementary rotations, where any rotation can be described by three successive rotations about linear independent axes. Nevertheless, this is an almost one-to-one transform with singularities:

$$R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi) \Rightarrow R$$

$$R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi) \nLeftarrow R$$

Table 2. Euler Angles

Elementary Rotations	Representation
$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$	
$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$	
$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	

There are different Euler angle conversions, such as the ones shown in Table 3:

Table 3. Euler Angle Conversions

Proper Euler Angles	
$X_1 Z_2 X_3 =$	$\begin{bmatrix} c_2 & -c_3 s_2 & s_2 s_3 \\ c_1 s_2 & c_1 c_2 c_3 - s_1 s_3 & -c_3 s_1 - c_1 c_2 s_3 \\ s_1 s_2 & c_1 s_3 + c_2 c_3 s_1 & c_1 c_3 - c_2 s_1 s_3 \end{bmatrix}$
$X_1 Y_2 X_3 =$	$\begin{bmatrix} c_2 & s_2 s_3 & c_3 s_2 \\ s_1 s_2 & c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_2 c_3 s_1 \\ -c_1 s_2 & c_3 s_1 + c_1 c_2 s_3 & c_1 c_2 c_3 - s_1 s_3 \end{bmatrix}$
$Y_1 X_2 Y_3 =$	$\begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & s_1 s_2 & c_1 s_3 + c_2 c_3 s_1 \\ s_2 s_3 & c_2 & -c_3 s_2 \\ -c_3 s_1 - c_1 c_2 s_3 & c_1 s_2 & c_1 c_2 c_3 - s_1 s_3 \end{bmatrix}$
$Y_1 Z_2 Y_3 =$	$\begin{bmatrix} c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 & c_3 s_1 + c_1 c_2 s_3 \\ c_3 s_2 & c_2 & s_2 s_3 \\ -c_1 s_3 - c_2 c_3 s_1 & s_1 s_2 & c_1 c_3 - c_2 s_1 s_3 \end{bmatrix}$
$Z_1 Y_2 Z_3 =$	$\begin{bmatrix} c_1 c_2 c_3 - s_1 s_3 & -c_3 s_1 - c_1 c_2 s_3 & c_1 s_2 \\ c_1 s_3 + c_2 c_3 s_1 & c_1 c_3 - c_2 s_1 s_3 & s_1 s_2 \\ -c_3 s_2 & s_2 s_3 & c_2 \end{bmatrix}$
$Z_1 X_2 Z_3 =$	$\begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_2 c_3 s_1 & s_1 s_2 \\ c_3 s_1 + c_1 c_2 s_3 & c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 \\ s_2 s_3 & c_3 s_2 & c_2 \end{bmatrix}$

3. Exponential coordinates

Some important ingredients to be considered are:

- The scalar differential equation:
 - $\begin{cases} \dot{x}(t) = ax(t) \\ x(0) = x_0 \end{cases} \Rightarrow x(t) = e^{at} x_0$
- The matrix differential equation:
 - $\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases} \Rightarrow \mathbf{x}(t) = e^{\mathbf{A}t} \mathbf{x}_0$
 - $e^{\mathbf{A}} = \mathbf{I} + \mathbf{A} + \frac{1}{2}\mathbf{A}^2 + \frac{1}{3!}\mathbf{A}^3 + \dots + \frac{1}{n!}\mathbf{A}^n + \dots$
- The degree-of-freedom of SO(3)

R has only 3 independent parameters, considering 6 constraints from:

$$r_i^T \cdot r_j = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}, \text{ being } R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

- The motion of a point about a rotating link ω at constant unit velocity as shown in Figure 5

$$- \begin{cases} \dot{\mathbf{p}}(t) = \boldsymbol{\omega} \times \mathbf{p}(t) = \hat{\boldsymbol{\omega}} \cdot \mathbf{p}(t) \\ \mathbf{p}(0) = \mathbf{p}_0 \end{cases} \Rightarrow \mathbf{p}(t) = e^{\hat{\boldsymbol{\omega}}t} \mathbf{p}_0$$

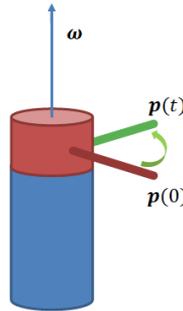


Figure 5. Exponential Coordinates

- The motion of a point about a rotating link ω at constant unit velocity:

$$- \begin{cases} \dot{\mathbf{p}}(t) = \boldsymbol{\omega} \times \mathbf{p}(t) = \hat{\boldsymbol{\omega}} \cdot \mathbf{p}(t) \\ \mathbf{p}(0) = \mathbf{p}_0 \end{cases} \quad \mathbf{p}(t) = e^{\hat{\boldsymbol{\omega}}t} \mathbf{p}_0$$

$$- \hat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

- Rotating about ω at unit velocity for θ units

$$- \mathbf{R}(\boldsymbol{\omega}, \theta) = e^{\hat{\boldsymbol{\omega}}\theta}$$

- The vector space of all 3x3 skew-symmetric matrices, denoted as $\text{so}(3) = \{S \in \mathbb{R}^{3 \times 3} : S^T = -S\}$

- The exponential map, which follows the following equation:

$$\mathbf{R}(\boldsymbol{\omega}, \theta) = e^{\hat{\boldsymbol{\omega}}\theta} = \mathbf{I} + \hat{\boldsymbol{\omega}} \sin \theta + \hat{\boldsymbol{\omega}}^2 (1 - \cos \theta)$$

- Hence, $e^{\hat{\boldsymbol{\omega}}\theta} \in SO(3)$, with $\det e^{\hat{\boldsymbol{\omega}}\theta} = 1$

4. Angle axis parameterization and Quaternions, not considered in this project.

2.4 RIGID BODY MOTIONS

General rigid body motions include not only rotation but also translations, forming the product space of \mathbb{R}^3 and $SO(3)$, which is denoted by $SE(3)$ and known as the *Special Euclidean group*:

$$- SE(3) = \{(\mathbf{p}, \mathbf{R}): \mathbf{p} \in \mathbb{R}^3, \mathbf{R} \in SO(3)\} = \mathbb{R}^3 \times SO(3)$$

Each point in $SE(3)$ can be transformed between different coordinate frames as shown in Figure 6:

$$- \mathbf{p}^a = \mathbf{R}_{ab}\mathbf{p}^b + \mathbf{p}_{ab} = \mathbf{g}_{ab}(\mathbf{p}^b)$$

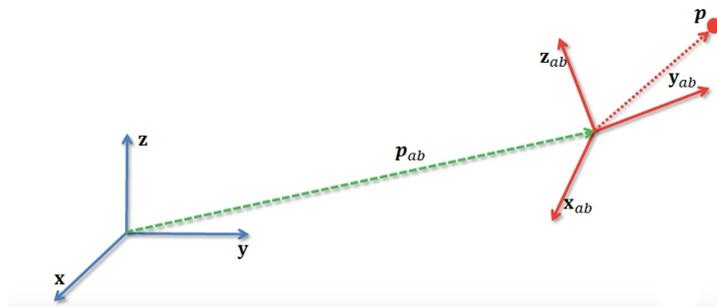


Figure 6. Transformation between different coordinate frames

Next, the composition rule for rigid body motion is presented and illustrated in Figure 7:

$$- \bar{\mathbf{g}}_{ac} = \bar{\mathbf{g}}_{ab} \cdot \bar{\mathbf{g}}_{bc} = \begin{bmatrix} \mathbf{R}_{ab}\mathbf{R}_{bc} & \mathbf{R}_{ab}\mathbf{p}_{bc} + \mathbf{p}_{ab} \\ 0 & 1 \end{bmatrix}$$

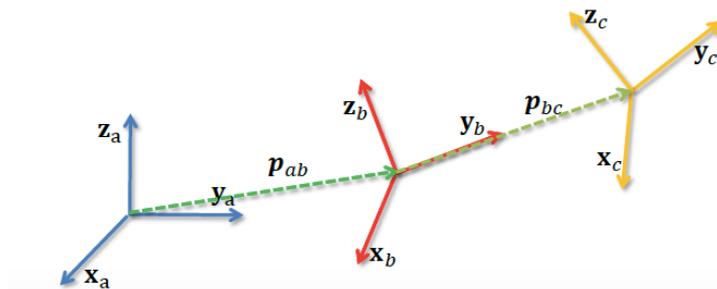


Figure 7. Composition of rigid body motion

Considering $SE(3) = \{(\mathbf{p}, \mathbf{R}): \mathbf{p} \in \mathbb{R}^{3 \times 3}, \mathbf{R} \in SO(3)\} = \mathbb{R}^3 \times SO(3)$, $SE(3)$ is a group under the operation of multiplication, satisfying the axioms of closure, identity, inverse and associativity explained before and $\mathbf{g} \in SE(3)$ is a rigid body transformation, where the cross products and lengths are preserved.

2.5 RIGID BODY VELOCITIES

Given a spatial frame A and a body frame B, considering a point attached to the rigid body following a rotational path in the spatial frame $p_a(t) = R_{ab} p_b$, the velocity of the point in the spatial frame can be denoted as:

$$- \quad v_p^a(t) = \frac{d}{dt} p^a(t) = \dot{R}_{ab}(t) p^b$$

$$\text{Considering } RR^T = I \Rightarrow \dot{R}R^T + RR^T \dot{R} = 0$$

$$- \quad v_p^a(t) = \dot{R}_{ab}(t) R_{ab}^{-1}(t) R_{ab}(t) p^b, \text{ where } \dot{R}_{ab}(t) R_{ab}^{-1}(t) \text{ is a skew-symmetric matrix.}$$

In regards to the angular velocity:

- The instantaneous spatial angular velocity is $\hat{\omega}_{ab}^a = \dot{R}_{ab} R_{ab}^{-1}$
- The instantaneous body angular velocity is $\hat{\omega}_{ab}^b = R_{ab}^{-1} \dot{R}_{ab}$
where $\hat{\omega}_{ab}^b = R_{ab}^{-1} \hat{\omega}_{ab}^a$ and $\omega_{ab}^b = R_{ab}^{-1} \omega_{ab}^a$
- The velocity induced by rotational motion is
 - o $v_p^a = \hat{\omega}_{ab}^a \cdot R_{ab} \cdot p^b = \omega_{ab}^a \times p^a$
 - o $v_p^b = R_{ab}^T \cdot v_p^a = \omega_{ab}^b \times p^b$

Therefore, by considering numerical integration, we have:

$$\dot{R} = R \hat{\omega}^b \text{ and } \dot{R} = \hat{\omega}^a R$$

The velocity of a point in the spatial frame or body frame, is respectively given by:

- $v_p^a = \omega_{ab}^a \times p^a - \omega_{ab}^a \times p_{ab} + \dot{p}_{ab}$
- $v_p^b = \omega_{ab}^b \times p^b + R_{ab}^T \cdot \dot{p}_{ab}$

These concepts are crucial for a real understanding of robot's kinematics, explained in Chapter 5. For further details, please refer to (Murray, R., Li, Z. and Sastry, S., 1994).



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots



CHAPTER 3: A* ALGORITHM IN ARTIFICIAL INTELLIGENCE

3.1. ALGORITHMS AND ARTIFICIAL INTELLIGENCE

An algorithm is just a structured method runned by computers. Artificial Intelligence (AI) is a research area of Computer Science that is based on the construction of specific algorithms which behave in a way that recreates human behavior and can be considered *intelligent*. There are lots of subareas that conform AI, such as: reasoning, perception, planning, learning, natural language processing, knowledge representation, social intelligence, general intelligence or motion and manipulation. When the artificial intelligent systems are implemented for working in the real world, then the robotic field gets involved. Due to an algorithm running in a computer, mechanical interactions are obtained in the real world, with algorithms telling individual motors when and how to move.

In AI, search techniques are typically used for problem solving. One of the best known AI search techniques is based on the A-star algorithm, which is widely used in path-finding and graph traversal. The A* algorithm searches all possible routes from an initial point until it finds the path with the cheapest cost to a goal (e.g., the shortest path).

3.2. A-STAR: COMBINATION OF UNIFORM COST SEARCH AND HEURISTIC SEARCH

The A* algorithm implemented in this MSc Thesis is the result of combining Dijkstra's algorithm (Uniform Cost Search) and search heuristics using search trees. On the one hand, the Uniform Cost Search expands the cheapest accumulated cost $g(n)$ node first, maintaining a priority queue, and explores increasing cost contours, being the node expanded guaranteed to have the smallest cost from the start state. Afterwards, the accumulated costs $g(m)$ of all the neighbours "m" of node "n" are updated. However, it can only see the cost $g(n)$ accumulated so far and has no information about the target location. On the other hand, the search heuristics are able to infer the least cost to goal, $h(n)$ and overcomes the limitations of uniform cost. Therefore, in the A* methodology, the estimated cost from the initial state to the target state passing through node n is $f(n) = g(n) + h(n)$, where $g(n)$ is the current best estimation of the accumulated cost from the initial state to node n and $g(n)$ is the estimated least cost from node n to the target state. Hence, the node with the cheapest cost estimated $f(n)$ is expanded in the A* Search (and then the accumulated costs $g(m)$ for all the unexpanded neighbours m are calculated). Then, the accumulated cost $g(n)$ and $f(n)$ are



obtained for the node's children that have not been expanded yet, where a node that has been expanded is guaranteed to have the smallest cost from the start state. This methodology does not end when the target position is enqueued, but when it is dequeued.

In order to obtaining an optimal A* path, the estimate d cost has to be less than the actual least cost to goal for all the nodes, which means that $h(n) < h^*(n)$, being $h^*(n)$ the real least cost to reach the goal from node n.

3.3. TREE CONSTRUCTION

As mentioned above, in order to apply the A* algorithm a search tree must be constructed. For the construction of a search tree, a closed container must be maintained to store all the nodes already visited: when a node is found, it is stored in the memory of the container. Also, a priority queue is kept in order to store all the nodes to be expanded. Each node can only be visited/expanded once. The priority queue is initialized with the initial state. The heuristic function $h(n)$ for each node is pre-defined, with the cheapest accumulated cost $g(n) = \infty$ for all nodes in the graph except for the initial state, where $g(Xs) = 0$. A loop is required to remove from the queue each visited node (the node with the lowest cost estimate $f(n)$), where the visited node is marked as expanded and added to the container. Then, all the unexpanded neighbours of the node, where each neighbor node cost $g(m) = \infty$, are pushed into the priority queue, but if $g(m) > g(n) + C_{nm}$, then, $g(m) = g(n) + C_{nm}$. The loops ends whenever the queue is empty, or the reached node "n" is the goal state.

For a better understanding, UAB's pseudocode of the A* Algorithm is provided in Figure 8, where the goal node is denoted by "node_goal" and the source node is denoted by "node_start" (UAB, 2018):

We maintain two lists: **OPEN** and **CLOSE**:

OPEN consists on nodes that have been visited but not expanded (meaning that successors have not been explored yet). This is the list of pending tasks.

CLOSE consists on nodes that have been visited *and* expanded (successors have been explored already and included in the open list, if this was the case).

```
1 Put node_start in the OPEN list with  $f(\text{node\_start}) = h(\text{node\_start})$  (initialization)
2 while the OPEN list is not empty {
3   Take from the open list the node node_current with the lowest
4      $f(\text{node\_current}) = g(\text{node\_current}) + h(\text{node\_current})$ 
5   if node_current is node_goal we have found the solution; break
6   Generate each state node_successor that come after node_current
7   for each node_successor of node_current {
8     Set successor_current_cost =  $g(\text{node\_current}) + w(\text{node\_current}, \text{node\_successor})$ 
9     if node_successor is in the OPEN list {
10      if  $g(\text{node\_successor}) \leq \text{successor\_current\_cost}$  continue (to line 20)
11    } else if node_successor is in the CLOSED list {
12      if  $g(\text{node\_successor}) \leq \text{successor\_current\_cost}$  continue (to line 20)
13      Move node_successor from the CLOSED list to the OPEN list
14    } else {
15      Add node_successor to the OPEN list
16      Set  $h(\text{node\_successor})$  to be the heuristic distance to node_goal
17    }
18    Set  $g(\text{node\_successor}) = \text{successor\_current\_cost}$ 
19    Set the parent of node_successor to node_current
20  }
21  Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

Figure 8. A* Algorithm Pseudocode

To illustrate the behavior a simple example is shown in Figures 9-11, for an environment where the initial position is (0.4, 0.4) and the target position is (1.8, 1.8). As shown in Figure 9, by intuition it is expected that the generated trajectory goes from the initial position to the target position following a straight line. If an obstacle is encountered in this trajectory in the position (0.6, 0.6), the result obtained with the A* algorithm will be the one shown in Figure 10, where the A* algorithm creates a path to avoid this obstacle considering the lowest distance from the start node (G cost) and distance from end node (Heuristic H cost), which gives the F cost. Figure 11 illustrates the result given by the algorithm in case a new obstacle is encountered in its way.

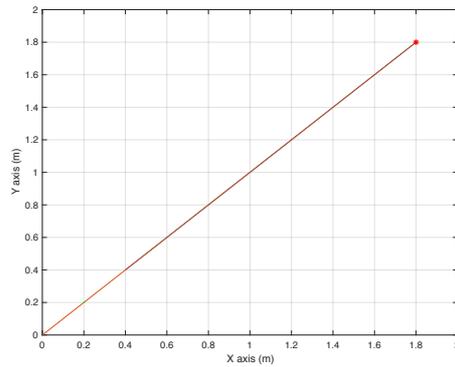


Figure 9. Intuitive Trajectory from Start Cell (0.4, 0.4) to Target Cell (1.8, 1.8)

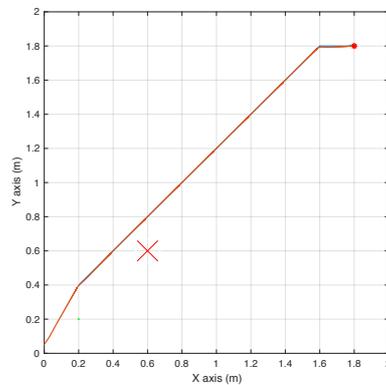


Figure 10. Path generated to avoid one obstacle

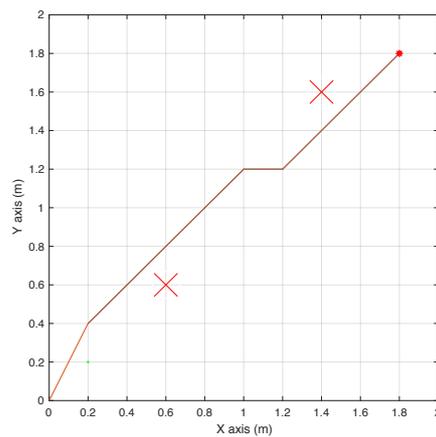


Figure 11. Path generated to avoid two obstacles

CHAPTER 4: BACKBONE OF THE PROPOSED METHODOLOGY

This chapter describes the methodology that is used to construct the robotic system developed in this project.

4.1. THE PROPOSED METHODOLOGY

The methodology developed in this project is summarized in Figure 12:

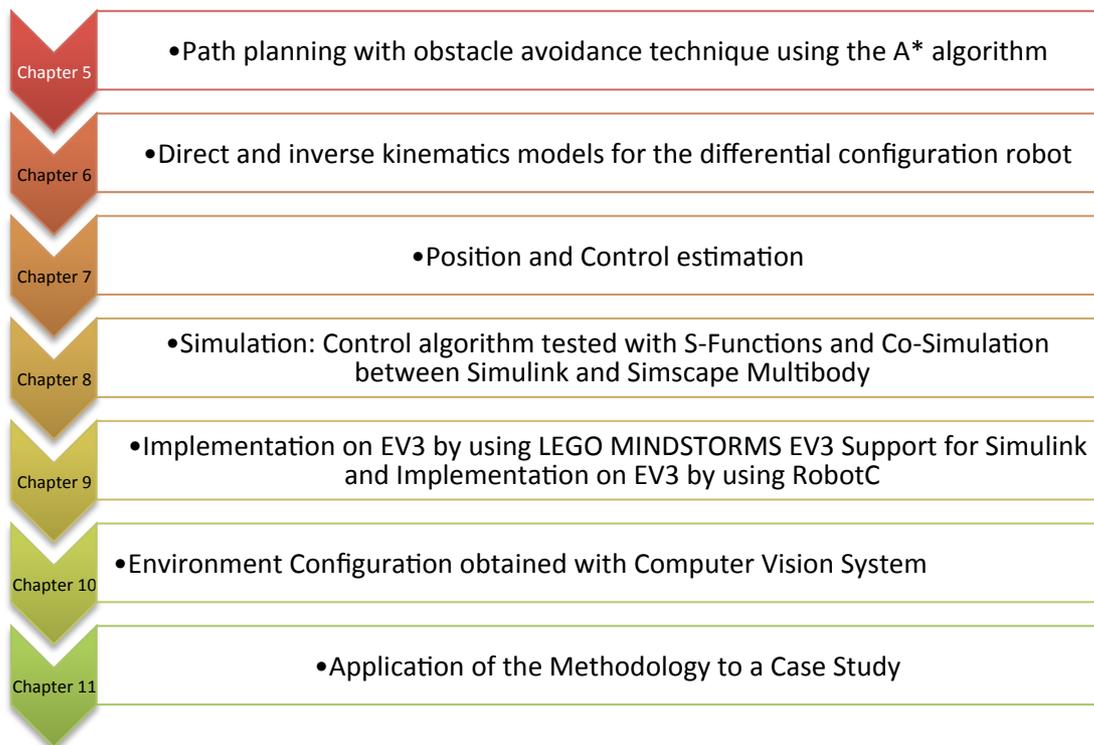


Figure 12. The proposed methodology

In more detail, the proposed methodology is structured as follows:



1. In an initial stage, the path planning technique with obstacle avoidance is designed and implemented by relying on the A* algorithm as described in Chapter 5. This chapter explains how the classical Artificial Intelligence algorithm A* is adapted to compute the optimal collision-free path to allow mobile robots automatically avoid obstacles by computing the minimal cost paths from a start point to a target point in a bounded area.
2. Next, a Simulink control structure is developed by considering: 1) the direct and inverse kinematics models for a differential configuration robot (Chapter 6), which allow the cartesian space and the joint space of the robotic problem to be related by suitable composing transformations; and 2) the robot position and control estimation model configurations (Chapter 7).
3. Third, the trajectory is thoroughly tested by visualizing the results from Simulink with S-Functions and by a co-simulation between Simulink and Simscape Multibody (Chapter 8). In the simulation, the system receives a vector input values that are of calculated by Matlab and interacts with a second software to automatically validate the robot's behavior in both a simple visualization of Simulink's results (with S-Functions) and a more realistic and faithful one that considers the effect that friction has on the robot (the co-simulation between Simulink and Simscape Multibody).
4. Once the models are validated, the technique is implemented on the EV3. Two independent implementations are provided: on the one hand, by using LEGO MINDSTORMS EV3 Support for Simulink, which supports academic applications and allows the robot real-time behavior to be evaluated; on the other hand, the by using the C-based robotics programming language RobotC, which supports industrial-strength applications.
5. Then, computer vision system is used to set the particular workspace configuration. This includes not only perceiving the starting point and target point in the environment configuration but also the process of obstacle detection (Chapter 10).
6. Lastly, the methodology is applied to a particular case study that is independent from any particular industrial bias (Chapter 11).

As shown in Chapter 11, the whole methodology is applicable in a simple and automated manner by following the following steps:

Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots

1. Get the environment configuration by running `vision_detect` (explained in Chapter 10) and compute the path trajectory for obstacle avoidance by implementing the path planning technique based on the A* Algorithm (explained in Chapter 5). Save the created map in workspace "Map" as a matrix, and the generated trajectory as "Traject_optimal" for further use in the code.

As requirements for this step, it must be considered that an image acquisition system is needed. In this project the type of camera used is monocular, which implies the depth of the environment was disregarded. In order to consider the depth, using a stereo camera or a RGB-D sensor should be needed. In the case when there is no camera available that is compatible with the given workspace, the starting point, target point and obstacle coordinates could be introduced manually as will be fully explained in Chapter 5.

Make sure that the dimensions of the industrial space where the methodology is going to be implanted correspond to maximum coordinates identified in the A* Algorithm as "maximum_x" and "maximum_y". For the working environment of this project, the dimensions were a 3x2.5 m2 grid.

2. Make sure the robot configuration of the industrial robot going to be used is differential, as the one used in this project. Otherwise, change the control system in Simulink explained in Chapter 6 and 7 before proceeding to step 3.

Visually check the behaviour automatically (using the control explained in Chapter 6 and 7) with S-Functions by running `sim('tfm_sfuns')` (developed in Chapter 8.1)

REMARK:

If SM Contact Forces Library is not installed, please before step 4, run:

```
run startup_Contact_Forces
```

(This step is fully constructed in Chapter 8.1)

3. In order to faithfully simulate a realistic behaviour which considers robot parameters such as weight, inertia moments and friction, then as explained in Chapter 8.2, run `sim('simulationwithsimscape')`
Be aware that initial and target positions and well as obstacles location must be set manually in the environment for each working configuration.
4. Make the EV3 follow the trajectory by running `run tfm_lego`
which allows the robot behaviour to be observed.
To change the robot's IP adress go to Model Configuration Parameters -> Hardware Implementation. This step is fully explained in Chapter 9.1.

It must be considered that the firmware version of the EV3 robot that enables Wifi



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots

connection is version 1.06H (latest version 1.09H is incompatible) and that the required connection protocol is WPA2 Personal (or none). It is also required to install the Netgear N150 wifi dongle.

5. As explained in Chapter 9.2, the methodology can be easily applied to any industrial robot. Fed the system with the path trajectory obtained in step 2 and run "ALGORITHM_A" (eventually, the controller can be easily rewritten using a similar C-based robot programming language if needed).

CHAPTER 5: IMPLEMENTATION OF THE A*

ALGORITHM FOR OUR TARGET ROBOT

5.1. THE PROPOSED TECHNIQUE

This chapter focuses on developing a suitable technique of path finding and obstacle avoidance for our target robot EV3. The underlying algorithm is A* algorithm, a combination of Dijkstra's algorithm and heuristic search described in Chapter 3. In order to obtain the least estimated cost $f(n)$ from the start state to the goal state passing through node "n", we consider the sum of the accumulated cost $g(n)$ of each node (which is the current best estimate of the accumulated cost from the start node to "n") and the heuristic $h(n)$, being $h(n)$ the estimated least cost from node n to the goal state. The developed strategy is based on expanding the node with the cheapest cost $f(n)$ and then update the accumulated cost $g(m)$ for all the unexpanded neighbors "m" of node "n", considering that a node that has been expanded is guaranteed to have the smallest cost from the start state.

In order to implement this algorithm, the first module to be defined is the declaration of the 2D grid map array representation that includes the obstacles, starting point and target point, as well as a given initialization of the location of the obstacles and target point.

Aftwards, a priority queue must be created in order to store all nodes that are pending to be expanded (variable "queue"), which is considered as an open container. Also, a close container is needed to keep the set of nodes that have been already evaluated (variable "set_closed"). The heuristic function $h(n)$ for all nodes is pre-defined using function "dis" to calculate the distance between nodes. The priority queue is initialized with the start state ("value_of_x", "value_of_y"), for which the function "add_to_queue" is used; similarly, the obstacles are added to the closed container "set_closed". The accumulated cost for the rest of the nodes in the graph is set to infinite and the accumulated cost for the starting node is set to zero. The main loop is based on the fact that nodes with the cheapest estimated cost $f(n)$ are expanded until the path is determined, using function "queue_of_expanded" to explore the search space and function "add_to_queue" to add a new node to the queue. As soon as the priority queue is empty, the loop is exited; meanwhile, the node "n" with the lowest cost $f(n)$ from the priority queue is removed and marked as expanded by introducing it into the closed container.

In order to identify the node with the lowest cost $f(n)$, the function "minimum_function" is introduced. The node in the open container with the lowest $f(n)$ is selected as the current node. Once processed, this node is removed from the open container

and transferred to the closed container "set_closed". Afterwards, its parent nodes are subsequently processed.

A whistle blows when the goal state is reached, and once the path is completed, a Boolean parameter "centinel" is used to get out of the loop. If a neighbour node of the current node is already set in the closed container, or not traversable, the routine skips to the next node. Also, for all the unexpanded neighbors "m" of node "n", if $g(m) = \infty$, the node "m" is pushed into the queue, and if $g(m) > g(n) + C_{nm} \rightarrow g(m) = g(n) + C_{nm}$ (considering it has not been added to the closed container yet), the least cost $f(n)$ of the neighbor node must be set. Also, considering that the neighbor node was selected as the current node in the open container, its parents are then considered as current nodes. When all nodes have been processed, the loop ends, and the path is backtraced by using the "find_predecessor_node" function that stepwisely gathers all predecessor nodes.

5.2. IMPLEMENTATION AND VALIDATION

To validate the implemented algorithm, different configuration maps were created. These maps represent the initial state, the obstacles and the target state. Due to technical limitations of the robot, it can only move forwards at a speed of 0.1 m/s , so the initial trajectory that considered a distance of 10 m had to be downscalated accordingly. A set of seven maps were initially considered to test the implemented algorithm. These maps are represented by using a matrix structure, where the first row of the matrix represents the coordinates of the initial point and the last row represents the target point. The intermediate rows represent the obstacles in the environment :

```
MAP1 = [0.2 0.2; ...% initial point
        0.4 0.2; ... % obstacles
        0.6 1.0; ...
        0.6 0.4; ...
        1.4 0.8; ...
        1.0 1.4; ...
        0.8 1.2; ...
        1.0 1.2; ...
        1.8 1.6]; % target point

MAP2 = [0.2 0.2; ... % initial point
        0.4 0.2; ... % obstacles
        0.6 1.0; ...
        0.6 0.4; ...
        1.4 0.8; ...
        1.0 1.4; ...
        0.8 1.2; ...
        1.0 1.2; ...
        0.6 0.6; ...
        0.4 1.2; ...
        1.8 1.6]; % target point
```

Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots

```
MAP3 = [0.2 0.2; ... % initial point
        0.4 0.2; ... % obstacles
        0.6 1.0; ...
        0.6 0.4; ...
        1.4 0.8; ...
        1.0 1.4; ...
        0.8 1.2; ...
        1.0 1.2; ...
        1.0 1.2; ...
        0.6 0.6; ...
        0.4 1.2; ...
        1.6 1.4; ...
        1.8 1.6]; % target point
```

```
MAP4 = [1.6 0.4; ...% initial point
        0.8 0.8; ... % obstacles
        1.0 0.8; ...
        1.2 0.8; ...
        1.4 0.8; ...
        1.6 0.8; ...
        0.8 1.0; ...
        1.0 1.0]; % target point
```

```
MAP5 = [1.6 0.4; ...% initial point
        0.8 0.8; ... % obstacles
        1.0 0.8; ...
        1.2 0.8; ...
        1.4 0.8; ...
        1.6 0.8; ...
        1.8 0.6; ...
        0.8 1.0; ...
        1.0 1.0]; % target point
```

```
MAP6 = [1.6 0.4; ...% initial point
        0.8 0.8; ... % obstacles
        1.0 0.8; ...
        1.2 0.8; ...
        1.4 0.8; ...
        1.6 0.8; ...
        1.8 0.6; ...
        0.8 1.0; ...
        1.4 1.0; ...
        1.0 1.0]; % target point
```

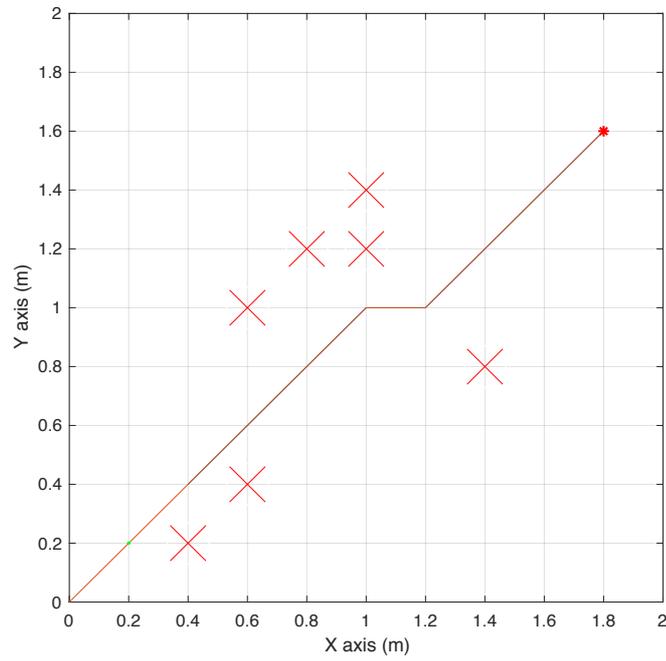


Figure 13. Example 1, A* Trajectory for Map 1

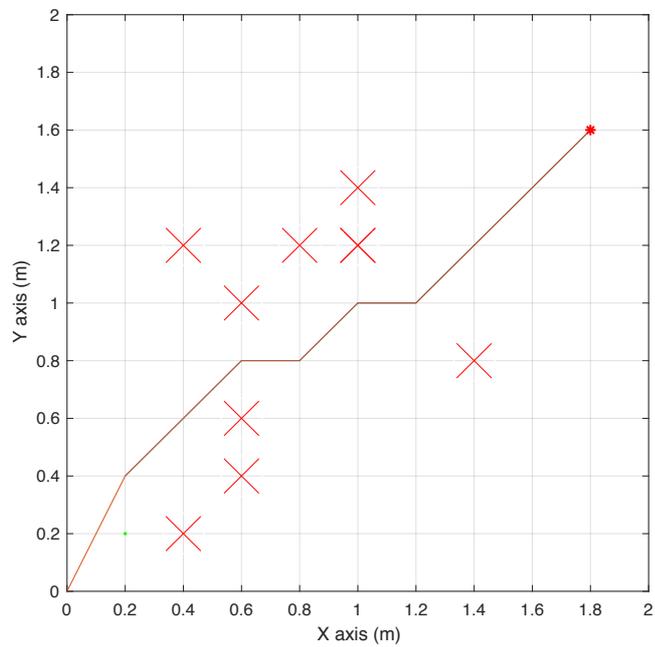


Figure 14. Example 2, A* Trajectory for Map 2

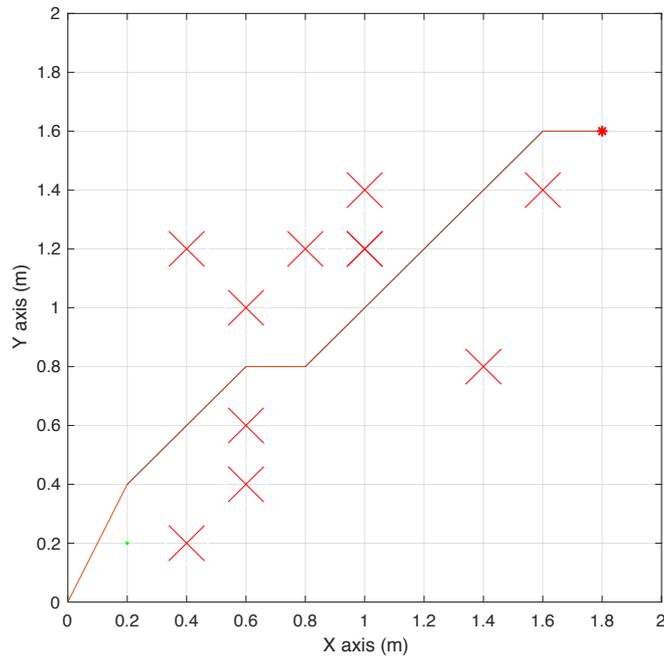


Figure 15. Example 3, A* Trajectory for Map 3

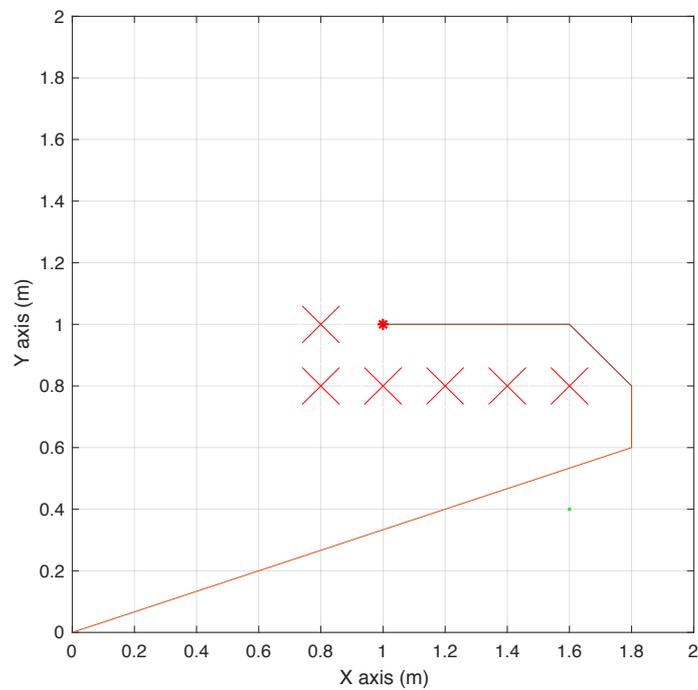


Figure 16. Example 4, A* Trajectory for Map 4

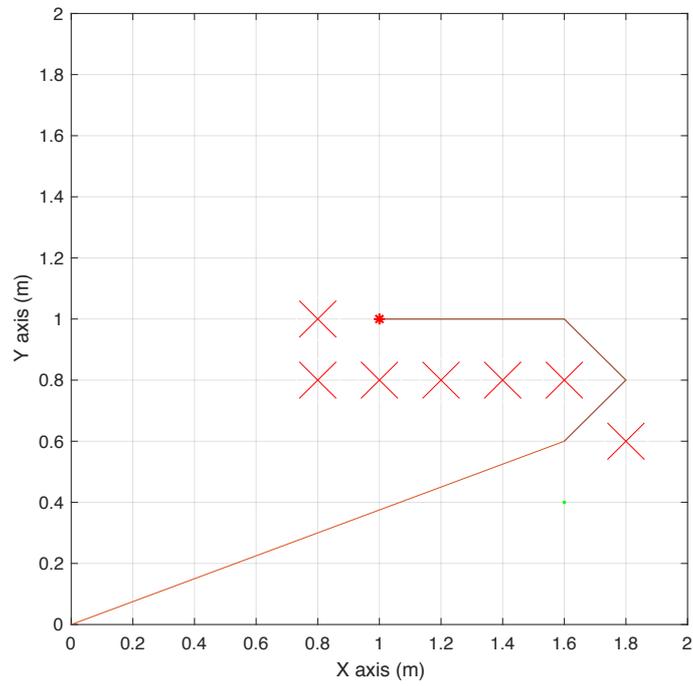


Figure 17. Example 5, A* Trajectory for Map 5

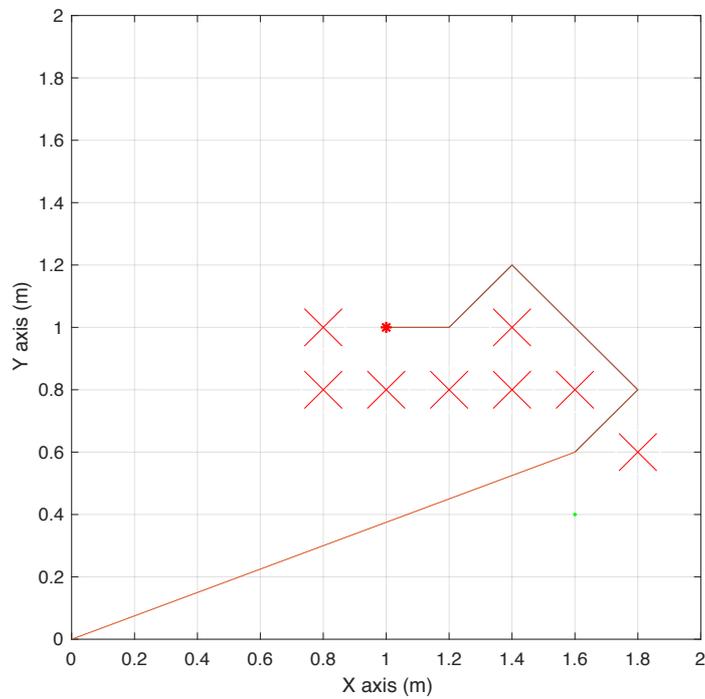


Figure 18. Example 6, A* Trajectory for Map 6

5.3. PIECEWISE CUBIC HERMITE INTERPOLATING POLYNOMIAL METHOD

The trajectories previously computed by the implemented A* algorithm consist of a limited number of points. In order to increase the accuracy of the generated trajectory, the piecewise cubic hermite interpolating polynomial method has been used to compute intermediate points between each two points that increase the number of points in the trajectory path.

This method returns a piecewise polynomial structure by finding out the values of an underlying interpolating function $P(x)$ at intermediate points, where on each subinterval $x_k \leq x \leq x_{k+1}$, $P(x)$ represents the cubic Hermite interpolant to the slopes at the two endpoints and the given values (Izmiran, 2005). $P(x)$ interpolates y , where $P(x_j) = y_j$, and the first derivative $P'(x)$ is continuous. It must be considered that $P''(x)$ is probably not continuous. The slopes at the x_j are chosen in such a way that $P(x)$ preserves the shape of the data, while respecting its monotonicity.

This type of polynomial was chosen due to the fact that it presents no overshoots and less oscillation if the data are not smooth with respect to the spline function polynomial, and is less expensive to set up. Initially, the spline polynomial was also considered, as it presented some advantages (Izmiran, 2005): by obtaining a continuous second derivative it may produce smoother results whenever the considered data contains values of a smooth function, as shown in Figure 19. Since this is not fulfilled in our case, the piecewise cubic hermite interpolating polynomial was chosen instead. Also, the spline method presents a more curvy trajectory, but for obstacle avoidance is really important that the points are directly joined without an overshoot; otherwise, the robot could encounter an obstacle, as can be seen in Figure 20.

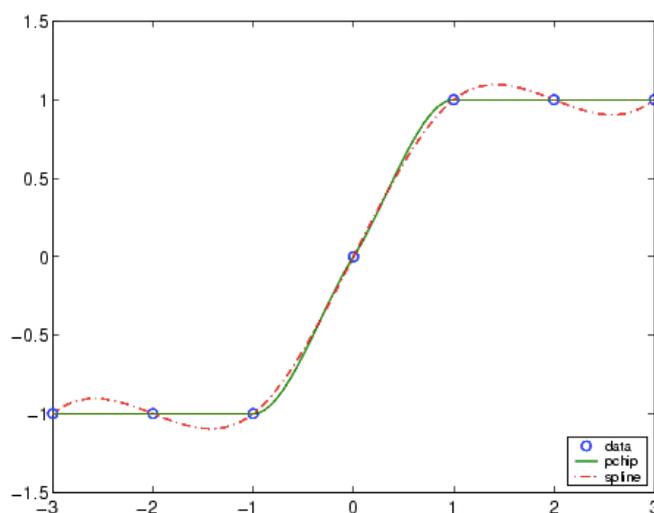


Figure 19. Pchip vs Spline

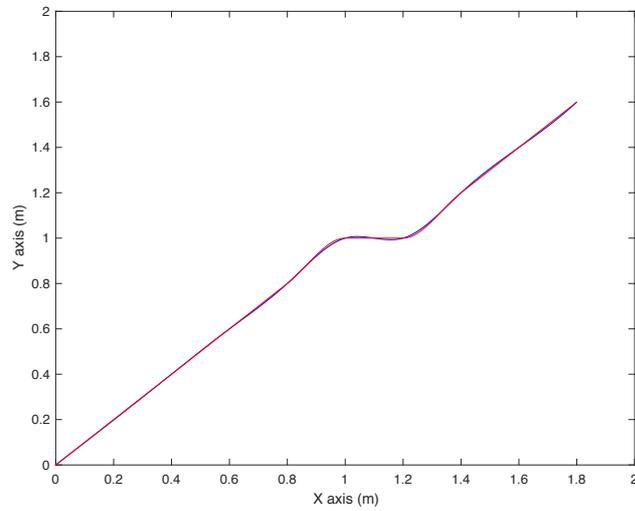


Figure 20. Pchip (red) vs Spline (blue) for our target robot

By considering the time required to move from the initial point to the target point and the required intervals, each set of points is broken down to make the distance between each pair of consecutive points smaller.

CHAPTER 6: FORWARD AND INVERSE KINEMATICS, AND ROBOT CONFIGURATION.

6.1. INTRODUCTION TO ROBOT CINEMATICS

As it is known, kinematics is the study of the movement of mechanical systems, without considering the forces that caused that movement. The kinematics theory is used to obtain the expressions that relate the position and orientation of a given element of the robot with the adopted values of the node's variables. There exist two types of kinematic problems to solve: forward and inverse kinematics. In order to solve these two types of problems, it must be considered that the robot is composed by a set of rigid bodies, where each element has an associated Cartesian reference frame. Therefore, the orientation and position of each element of the robot can be obtained from the reference frames, and from these, the orientation and position of the end part of the robot can also be obtained.

6.2. INTRODUCTION TO FORWARD KINEMATICS

Forward kinematics allows the position and orientation of the outermost part of the robot to be obtained by considering the variables in each joint for robotic manipulators, and to also obtain the position and orientation of the robot by considering the position and velocities of the wheels. It is based on a vector function $f_R(l_i, q)$ that relates the geometric properties of the mechanical system l_i and the joint coordinates $q \in \mathbb{R}^3$ with the Cartesian coordinates $[x, y, z]^T \in \mathbb{R}^3$ and the robot's orientation $[\theta, \phi, \psi]^T \in \mathbb{R}^3$ as shown in Figure 21, borrowed from (Valera Fernández, Á., 2017b):

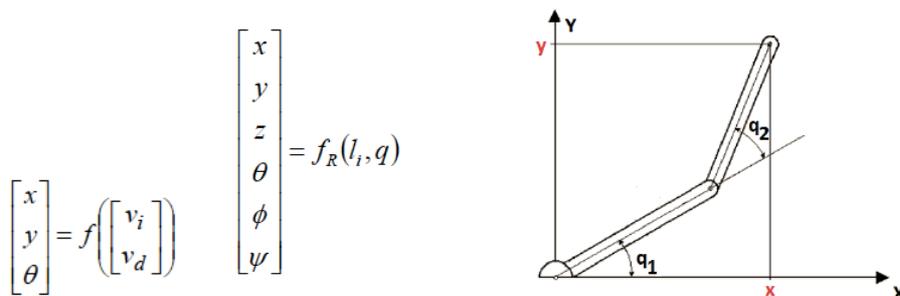


Figure 21. Forward Kinematics

Therefore, the forward kinematics problem provides only one solution for each set of variables of the considered joint. There are several ways to solve this problem, such as using geometric methods or homogenous matrices.

6.3. INTRODUCTION TO INVERSE KINEMATICS

Inverse kinematics allows the values of the variables in each joint to be obtained by considering the position and orientation of the outermost part of the robot. This type of kinematics is based on a non-linear problem that, in the case of robotic manipulators, creates a relationship between the orientation and joint coordinates of the robot's tools at its end, as shown in Figure 22, taken from (Valera Fernández, Á., 2017b). In mobile robotics, it allows the position and velocities of the wheels to be obtained, which allows the robot to be given the desired position and orientation:

$$q = f_R^{-1}(x, y, z, \theta, \phi, \psi)$$

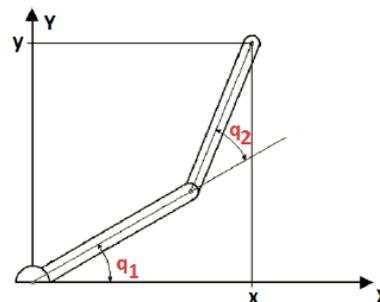


Figure 22. Inverse Kinematics

Therefore, by considering a known cartesian position, the inverse kinematics problem consists in finding the articulation coordinates $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\} = f(x, y, z, \alpha, \beta, \gamma)$. Considering it is generally a redundant problem as it provides more than one solution. There are several methods to solve this problem, such as using geometric methods, taking the homogenous matrix as a start point, applying kinematic decoupling (being this one the most commonly used) or by iteration. Taking into consideration that the explicit solution of the system is required, Pieper's theorem must be taken into account: any 6 dof (degrees of freedom) robot with 3 consecutive axes meeting at a point has solvable inverse kinematics.

6.4 ROBOT CONFIGURATION

There are several kinematic configurations for mobile robots, such as differential configuration, caterpillar configuration, tricycle and ackerman configuration. The configuration of the project robot EV3 is the differential configuration, which presents two traction wheels and a front support wheel.

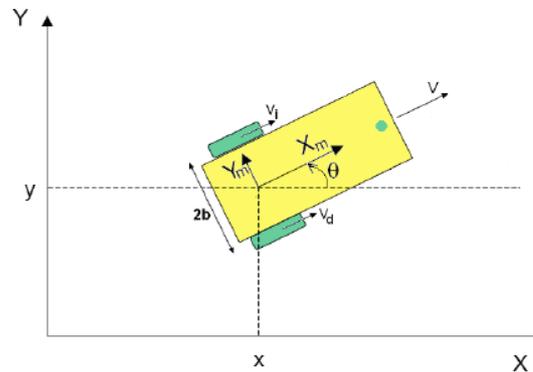


Figure 23. Differential Configuration

6.5 FORWARD KINEMATICS WITH DIFFERENTIAL CONFIGURATION MODEL

For implementing the forward kinematics problem with a differential configuration model, and considering the wheels' velocity v_d and v_i , the linear velocity of the robot v and the angular velocity of the robot ω , the following equations are generated:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix} = \frac{1}{2b} \begin{bmatrix} b \cos \theta & b \cos \theta \\ b \sin \theta & b \sin \theta \\ -1 & 1 \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

$$= \begin{bmatrix} \cos \theta \frac{v_i + v_d}{2} \\ \sin \theta \frac{v_i + v_d}{2} \\ \frac{v_d - v_i}{2b} \end{bmatrix}$$

From these equations, the Simulink structure can be built:

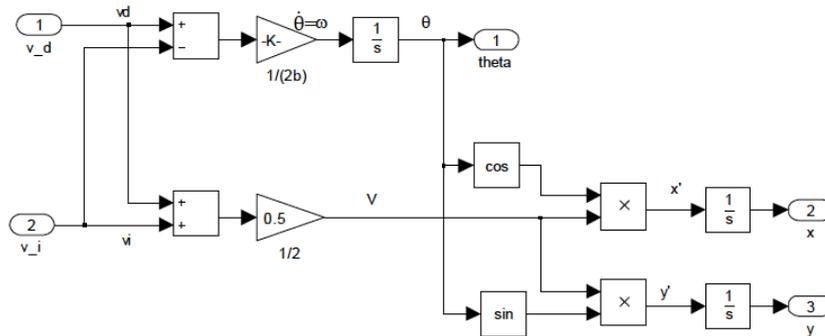


Figure 24. Simulink Forward Kinematics

6.6 INVERSE KINEMATICS WITH DIFFERENTIAL CONFIGURATION MODEL

For implementing the inverse kinematics, the following equations must be considered, leading to the Simulink structure shown in Figure 25:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$\downarrow A^* = (A^T A)^{-1} A^T$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \begin{bmatrix} 1 & -b \\ 1 & b \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \begin{bmatrix} 1 & -b \\ 1 & b \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & -b \\ \cos \theta & \sin \theta & b \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \begin{bmatrix} \dot{x} \cos \theta + \dot{y} \sin \theta - b \dot{\theta} \\ \dot{x} \cos \theta + \dot{y} \sin \theta + b \dot{\theta} \end{bmatrix}$$

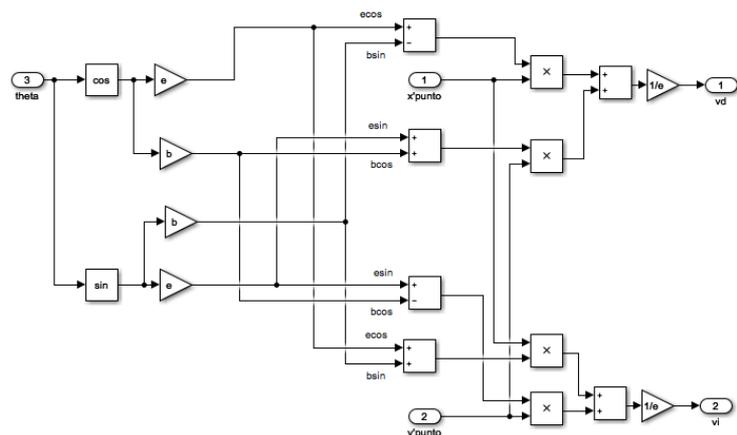


Figure 25. Inverse Kinematics Simulink Structure

CHAPTER 7: POSITION ESTIMATION AND CONTROL OF MOBILE ROBOTS

7.1. POSITION ESTIMATION

Taking into consideration the rigid body motion described in Chapter 2, and considering the robot's velocity and the integral of the robot's differential equation in recursive form, and doing an integral approximation of the differential equation, considering Δt or θ small enough, it can be considered:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta(t)) & 0 \\ \sin(\theta(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \quad \begin{bmatrix} x(t+\Delta t) \\ y(t+\Delta t) \\ \theta(t+\Delta t) \end{bmatrix} = \begin{bmatrix} x(t) + \frac{2v(t)}{\omega(t)} \sin\left(\frac{\omega(t)\Delta t}{2}\right) \cos\left(\theta(t) + \omega(t)\frac{\Delta t}{2}\right) \\ y(t) + \frac{2v(t)}{\omega(t)} \sin\left(\frac{\omega(t)\Delta t}{2}\right) \sin\left(\theta(t) + \omega(t)\frac{\Delta t}{2}\right) \\ \theta(t) + \omega(t)\Delta t \end{bmatrix}$$

$$\left. \begin{array}{l} \sin\left(\omega(t)\frac{\Delta t}{2}\right) \approx \omega(t)\frac{\Delta t}{2} \\ \cos\left(\theta(t) + \omega(t)\frac{\Delta t}{2}\right) \approx \cos(\theta(t)) \\ \sin\left(\theta(t) + \omega(t)\frac{\Delta t}{2}\right) \approx \sin(\theta(t)) \end{array} \right\} \begin{bmatrix} x(t+\Delta t) \\ y(t+\Delta t) \\ \theta(t+\Delta t) \end{bmatrix} \approx \begin{bmatrix} x(t) + v(t)\Delta t \cos(\theta(t)) \\ y(t) + v(t)\Delta t \sin(\theta(t)) \\ \theta(t) + \omega(t)\Delta t \end{bmatrix}$$

In discrete position estimation, the robot's position is given by the following equations:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} \approx \begin{bmatrix} x_k + v_k T_s \cos(\theta_k) \\ y_k + v_k T_s \sin(\theta_k) \\ \theta_k + \omega_k T_s \end{bmatrix}$$

where the angular and linear velocities of the wheels satisfy the following equations:

$$\left\{ \begin{array}{l} w_{d_k} = pul2rad \frac{enc_{d_k} - enc_{d_{k-1}}}{T_s} \\ w_{i_k} = pul2rad \frac{enc_{i_k} - enc_{i_{k-1}}}{T_s} \end{array} \right\} \left\{ \begin{array}{l} v_{d_k} = r \cdot w_{d_k} \\ v_{i_k} = r \cdot w_{i_k} \end{array} \right.$$

$$v_k = \frac{v_{d_k} + v_{i_k}}{2}$$

$$\omega_k = \frac{v_{d_k} - v_{i_k}}{2b}$$

This yields the following equations for the robot's velocities:

7.2. CONTROL OF MOBILE ROBOTS

Two types of controls are required for mobile robots: kinematic and dynamic. It corresponds to a cascade control, where the dynamic control represents the outer loop and the kinematic control the inner one, as shown in Figure 26 (Valera Fernández, Á., 2017b):

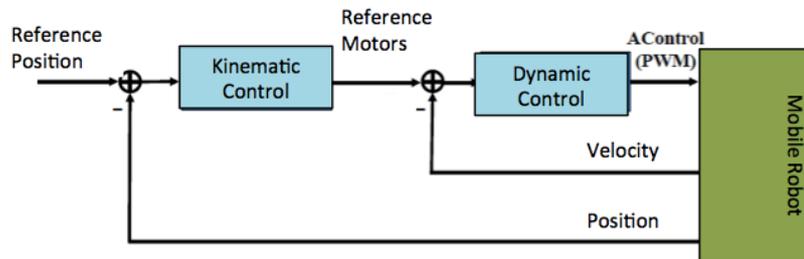


Figure 26. Mobile Robot Control

1. DYNAMIC CONTROL: This type of control is generally obtained with a basic type of control (P, PD, PI or PID of velocity and/or angular positions of the motos of the robot). For the differential configuration, the dynamic control establishes the wheel's velocity control, as shown in Figure 27:

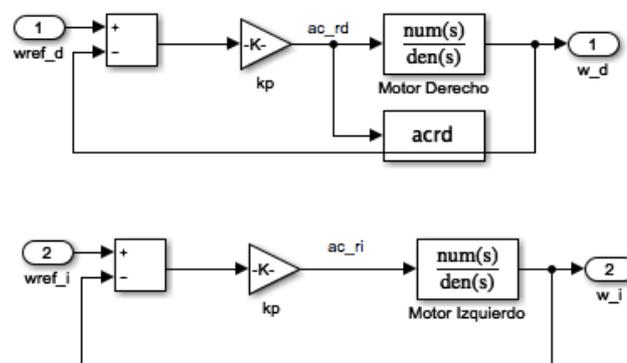


Figure 27. Dynamic Control Simulink Structure

2. KINEMATIC CONTROL: This type of control is generally more complex than the dynamic one. It is based in the determination of the required robot's actions to take it from the actual position to a target final position, taking into consideration its orientation and velocities as shown in Figure 28 that will be fully explained below:

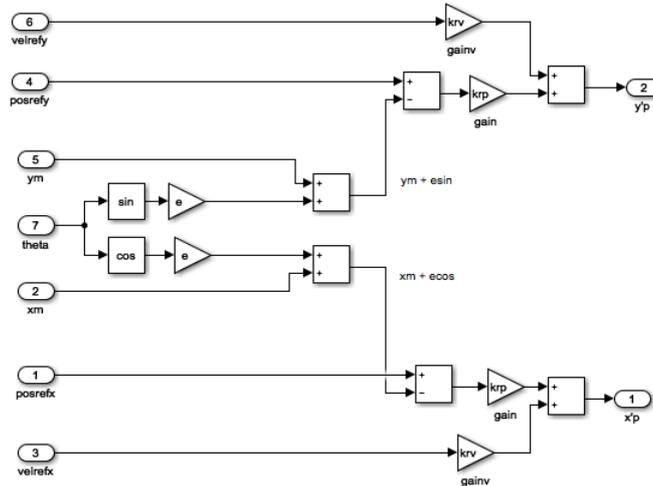


Figure 28. Dynamic Control Simulink Structure

3. TRAJECTORY CONTROL: There are different ways to control trajectories, such as proportional control with pre-feeding, proportional-integral with pre-feeding or position control by decentralized point, being this last option the one considered in this project. For this type of control, the control is established from the position and velocity of a point at distance e from the traction axis of the robot. The derivative of the position of the decentralized point follows the following equations:

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & -g \sin(\theta) \\ 0 & 1 & g \cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -g \sin(\theta) \\ 0 & 1 & g \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

being the kinematic of the differential robot as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ -1/2b & 1/2b \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

where, by substitution of the kinematic equation of the differential robot, the following equations are obtained:

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \frac{1}{2b} \begin{bmatrix} b \cos(\theta) + g \sin(\theta) & b \cos(\theta) - g \sin(\theta) \\ b \sin(\theta) - g \cos(\theta) & b \sin(\theta) + g \cos(\theta) \end{bmatrix} \begin{bmatrix} v_i \\ v_d \end{bmatrix}$$

Therefore, the wheels have the following reference velocities:

$$\begin{bmatrix} v_i \\ v_d \end{bmatrix} = \frac{1}{g} \begin{bmatrix} g \cos(\theta) + b \sin(\theta) & g \sin(\theta) - b \cos(\theta) \\ g \cos(\theta) - b \sin(\theta) & g \sin(\theta) + b \cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix}$$

being the velocity of the decentralized point (kinematic control), as follows :

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \begin{bmatrix} k_{r_v} \dot{x}_{ref} \\ k_{r_v} \dot{y}_{ref} \end{bmatrix} + \begin{bmatrix} k_{r_p} & 0 \\ 0 & k_{r_p} \end{bmatrix} \begin{bmatrix} x_{ref} - (x + g \cos \theta) \\ y_{ref} - (y + g \sin \theta) \end{bmatrix}$$

and being the dynamic control as follows:

$$\begin{bmatrix} acontrol_i \\ acontrol_d \end{bmatrix} = \begin{bmatrix} k_{m_p} \frac{(v_{i_{ref}} - v_{i_{robot}})}{radio} \\ k_{m_p} \frac{(v_{d_{ref}} - v_{i_{robot}})}{radio} \end{bmatrix} = \begin{bmatrix} k_{m_p} \cdot (w_{i_{ref}} - w_{i_{robot}}) \\ k_{m_p} \cdot (w_{d_{ref}} - w_{d_{robot}}) \end{bmatrix} = \begin{bmatrix} k_{m_p} \cdot error_wi \\ k_{m_p} \cdot error_wd \end{bmatrix}$$

Hence, the position control by decentralized point presents the following structure:

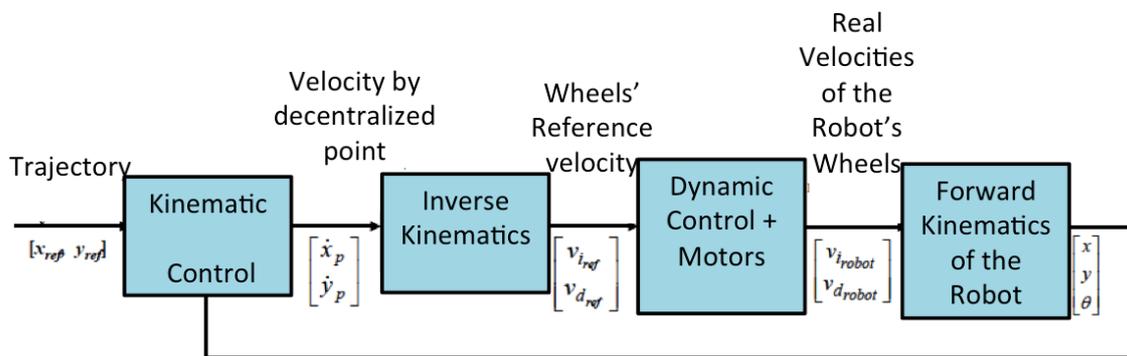


Figure 29. Position Control by Decentralized Point

4. PATH CONTROL: The path control for the proposed obstacle avoidance methodology is postponed to Chapter 8.

CHAPTER 8: SIMULINK VISUALIZATION USING S-FUNCTIONS AND SIMSCAPE MULTIBODY SIMULATION

Before implanting the developed control system for obstacle avoidance based on the programmed algorithm, two types of system visualization were considered to test the robot's behavior: a simple one by using S-Functions that can easily be applied to any system in an automated way, and a more complex one based in a co-simulation, specially tailored for a particular system, where the robot's physical parameters and the environment structure are considered.

8.1. S-FUNCTIONS

System functions (S-functions) consist of an extension of the capabilities of Simulink's environment for discrete, continuous and hybrid systems. As previously explained in Chapter 5, in order to validate the optimal paths, different maps have been created. Remember that the optimal path is obtained by considering the sum of the current best estimate $g(n)$ of the accumulated cost from the start node to "n" and the heuristically estimated least cost $h(n)$ from node n to the given goal state.

S-Functions consist on two elements: a Matlab function and a Simulink structure. The code used in this project is structured in 3 parts: case 0, where the variables are initialized, case 3, where the outputs are calculated and case 9, which represents the end of the visualization.

Simulink's structure for the S-Functions of Figure 30 represents a combination of the kinetic control, inverse kinematics control and the Real Lego Robot structures previously explained:

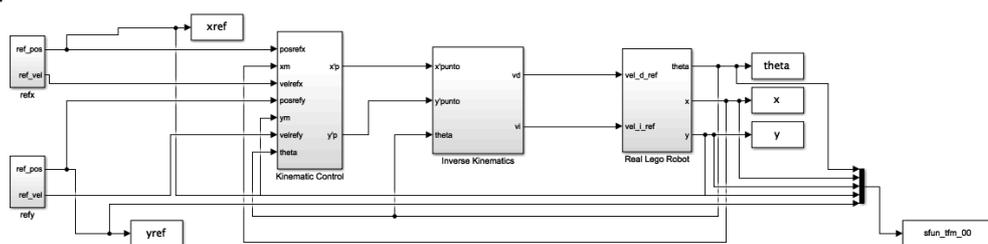


Figure 30. Simulink Global Structure for S-Functions

With this structure, different visualizations for each map can be obtained as shown in Figures 31-37, where the robot moves by following the optimal trajectory in both a XY and XYZ plane view, and where the trajectory can be left hidden if wanted by the user. At the middle top of each figure, the user can also see the general frame coordinates followed by the robot as it moves. This methodology can be extrapolated to any type of real world environment to test, in an easy and safe manner, if the robot will eventually collide with an obstacle while following its planned trajectory from the starting point to the target point (in meters).

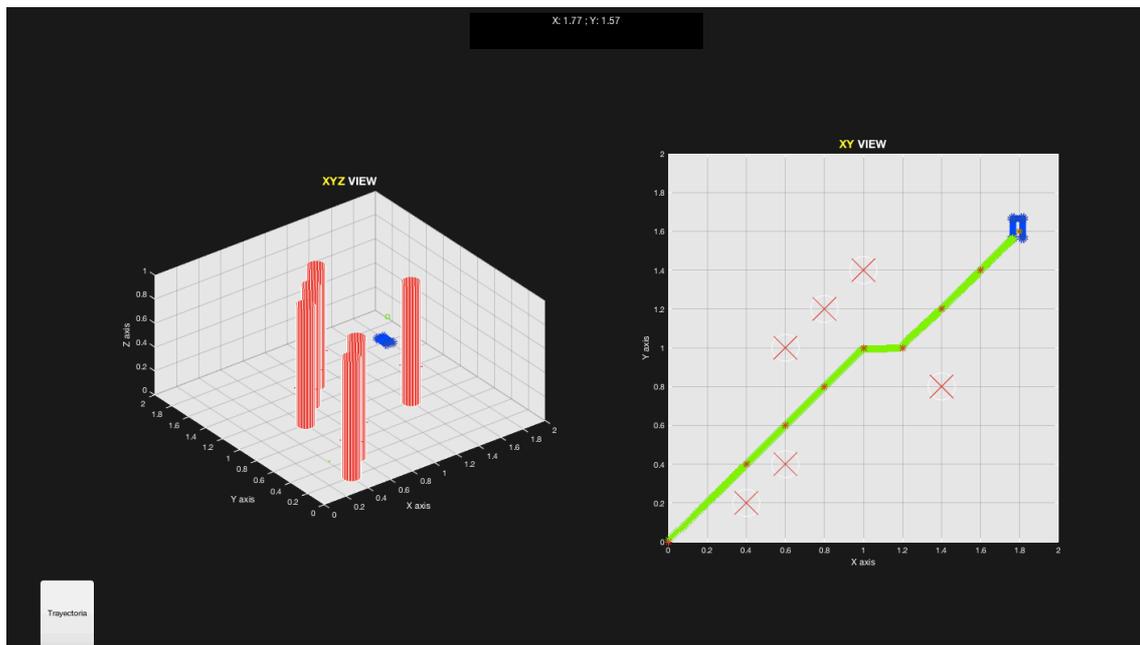


Figure 31. Example 1, S-Functions Map 1

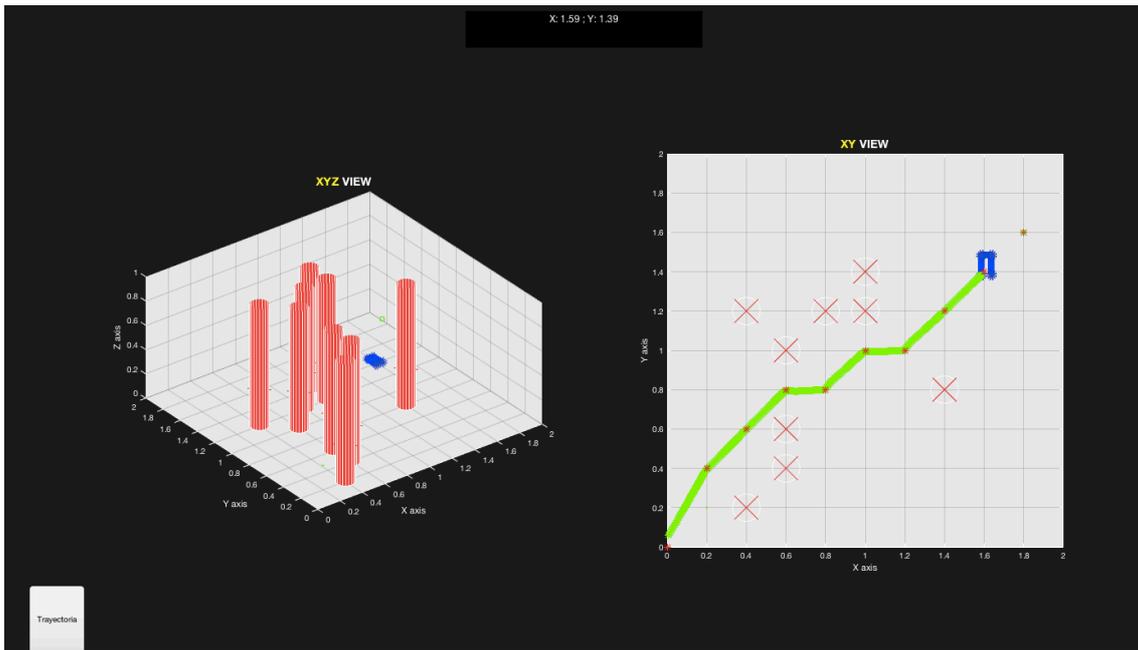


Figure 32. Example 2, S-Functions Map 2

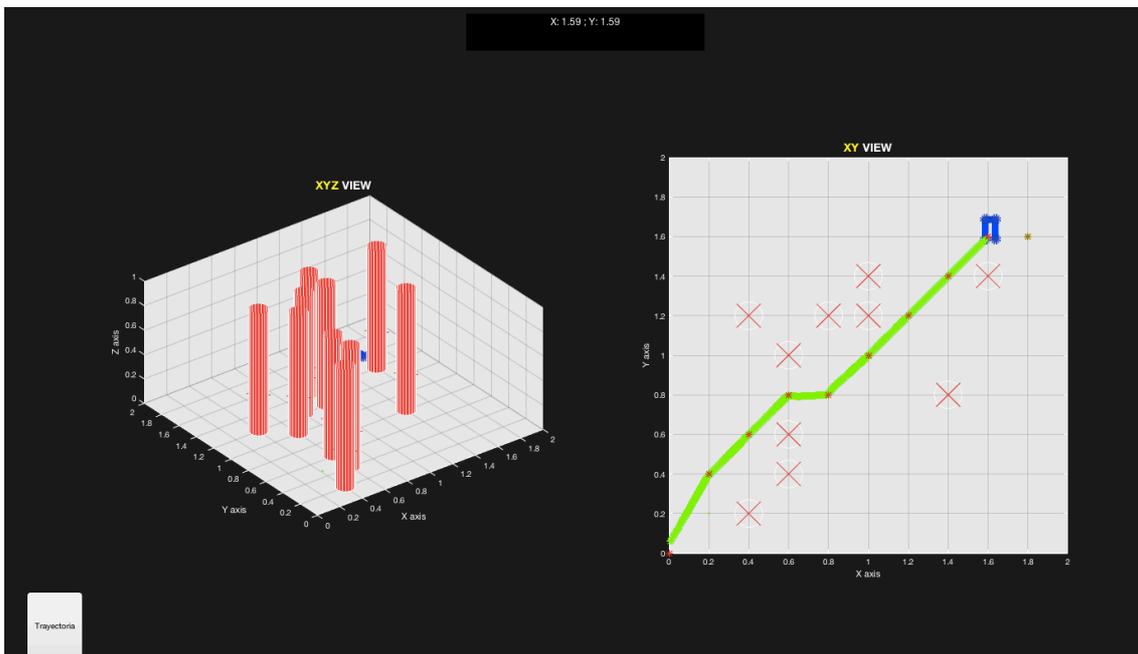


Figure 33. Example 3, S-Functions Map 3

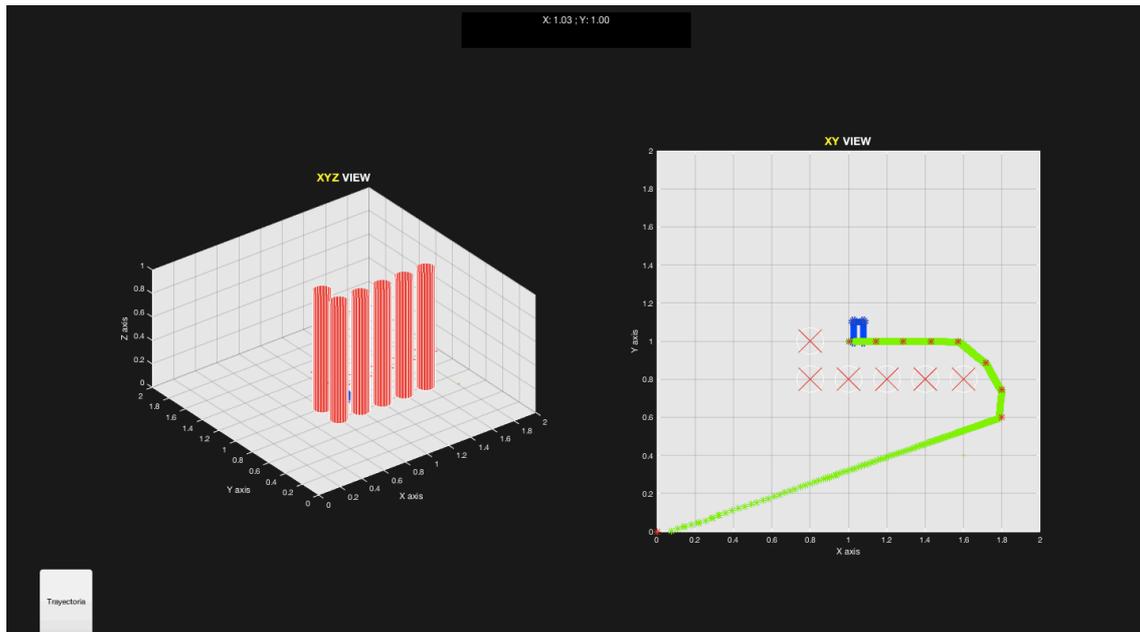


Figure 34. Example 4, S-Functions Map 4

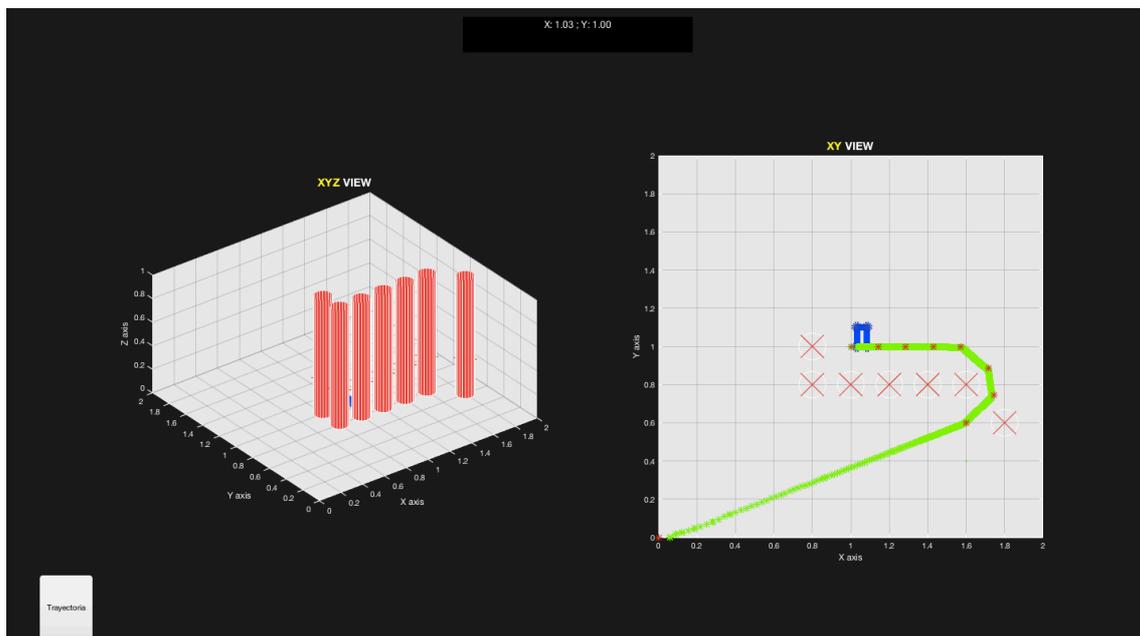


Figure 35. Example 5, S-Functions Map 5

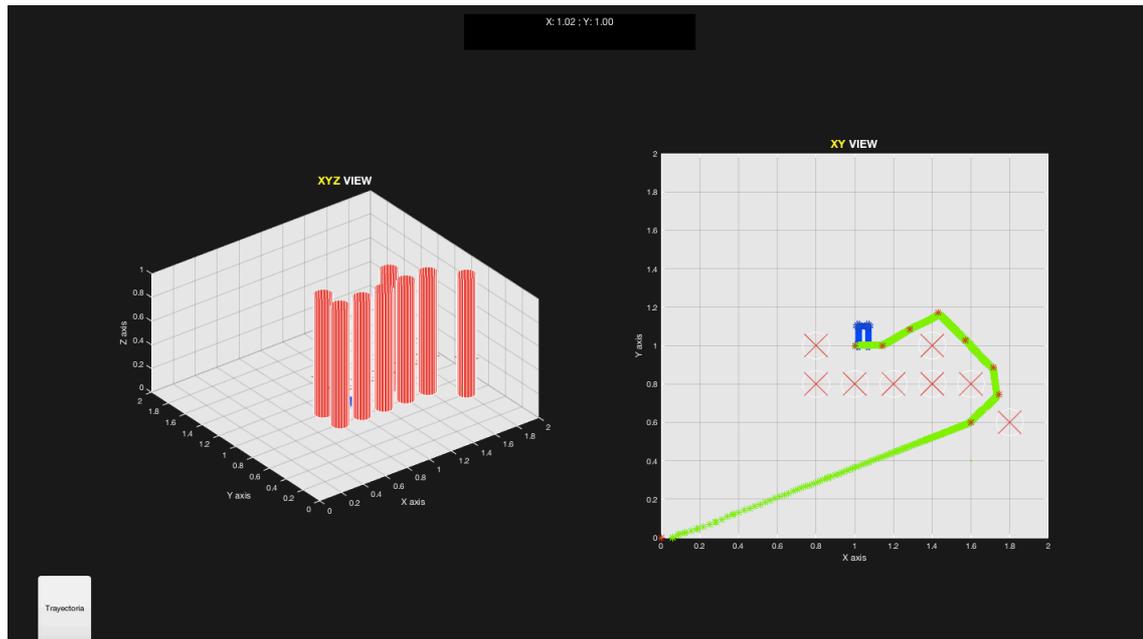


Figure 36. Example 6, S-Functions Map 6

8.2. SIMSCAPE MULTIBODY

Simscape Multibody provides a simulation environment for the Lego Robot EV3 by resolving the differential equations of the mechanical system. This type of simulation yields a more realistic approach to modeling the behavior of the real robot that considers its structure, dimensions, weight, inertia and other general properties. Therefore, the first task required for this type of simulation was to obtain the Solidworks model of the robot, whose general structure was obtained from Grabcad (GrabCad site, 2018). However, the real Lego Robot (shown in Figure 37) has a slightly different structure, so the structure had to be modified in order to turn it into the configuration used in this project, being the result shown in Figure 38.



Figure 37. EV3 Real Model

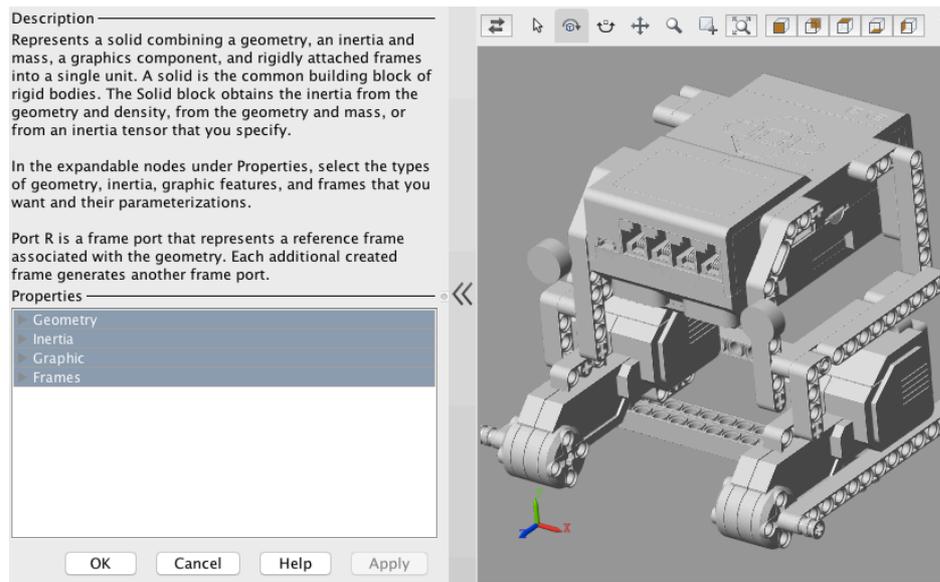


Figure 38. Body Structure of the Robot in .stl

Once the model was modified, the structure was split into several parts before being introduced to Simulink. In order to obtain a basic view, a .stl type of model was initially introduced, which considered each part of the robot's structure as a point mass in the center of masses (reason why four wheels were considered, whereas the real robot has a differential configuration), instead of considering its real properties (mass, inertia...). Following the previous S-Functions visualization, the position vector was introduced to the body structure joint. The images shown in this section correspond to Map 1 (Figure 13), but it can be extrapolated to any environment and set of obstacles. The result was quite accurate as shown in Figure 39.

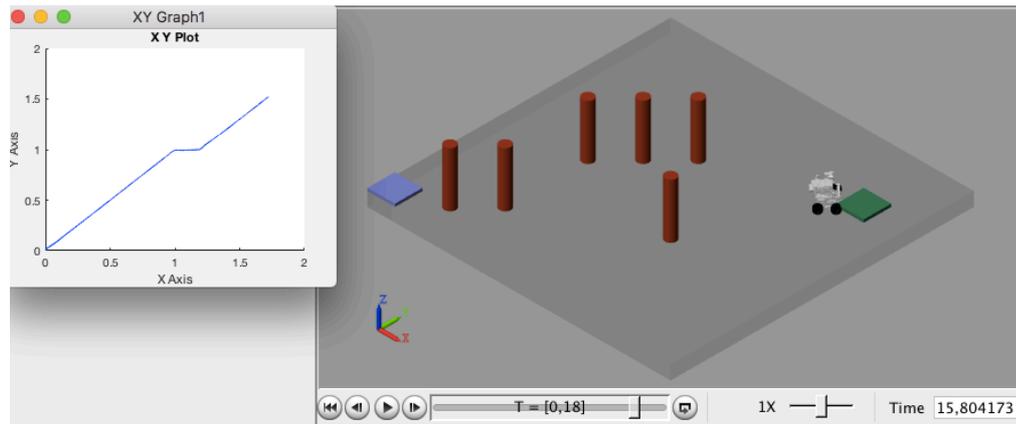


Figure 39. The Robot following the Path Simulation

Nevertheless, in order to obtain a more realistic approximation of the robot's real world behavior, the model had to include the weight, inertia, density, center of mass and moment of inertia of each structure, so it had to be considered the material's density, which considering it is PP plastic, is 0.91g/cm^3 as shown in Figure 40. As the STEP file is introduced directly, the dimensions of the rigid body are directly evaluated and inertia moment and weight are immediately calculated by considering the given density.

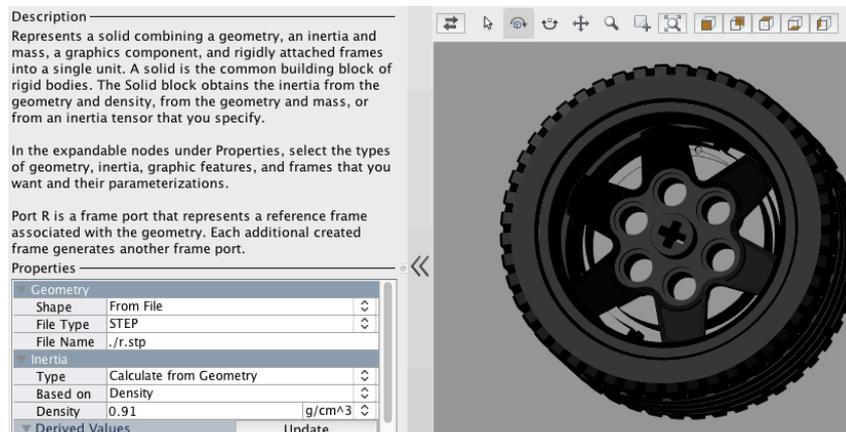


Figure 40. The wheel's properties

To ensure compability with Simscape Multibody (in particular, strict limitations on the size of files), the previous body structure had to be slightly modified to work properly. Also, SM-Contact Forces were considered in order to simulate the wheel's friction with the floor as shown in Figure 47. Moreover, following a professional methodology, a new Simulink structure was created in order to introduce the wheel's reference angular velocity to the wheels and simulate its behavior (instead of introducing the position coordinates, as explained previously). The general structure is shown in Figures 41-49, where each figure

follows a hierarchical structure, from a general structure higher structure to a more specific structure.

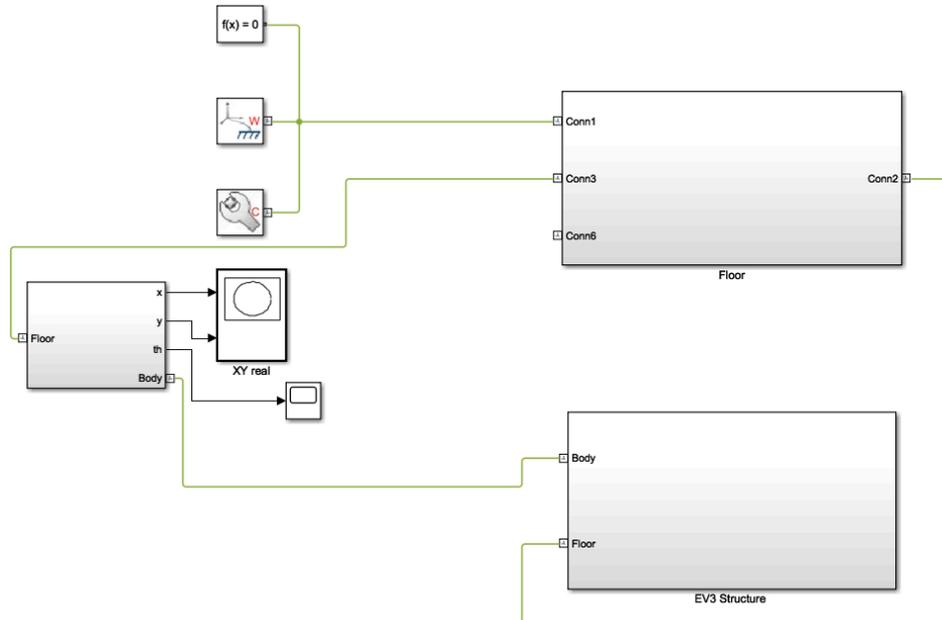


Figure 41. General Simulink Structure

As can be observed, Figure 41 presents the higher hierarchical level of the Simulink structure, with three main subsystems: the floor subsystem, containing the floor platform, the obstacles, target position and initial position which can be observed in Figure 43, the EV3 Structure, which includes the controller structure (which sends the control action into the wheels structure), the body structure (connected to the wheels and its corresponding SM-Contact forces blocks which consider the friction of the wheels with the floor), as shown in Figure 46 and the subsystem that connects the body structure to the floor, shown in Figure 42. It is important to note that the World frame is connected to the floor to fix the position and considers an inertial coordinate frame. A mechanism configuration block is considered to set the gravity parameter.

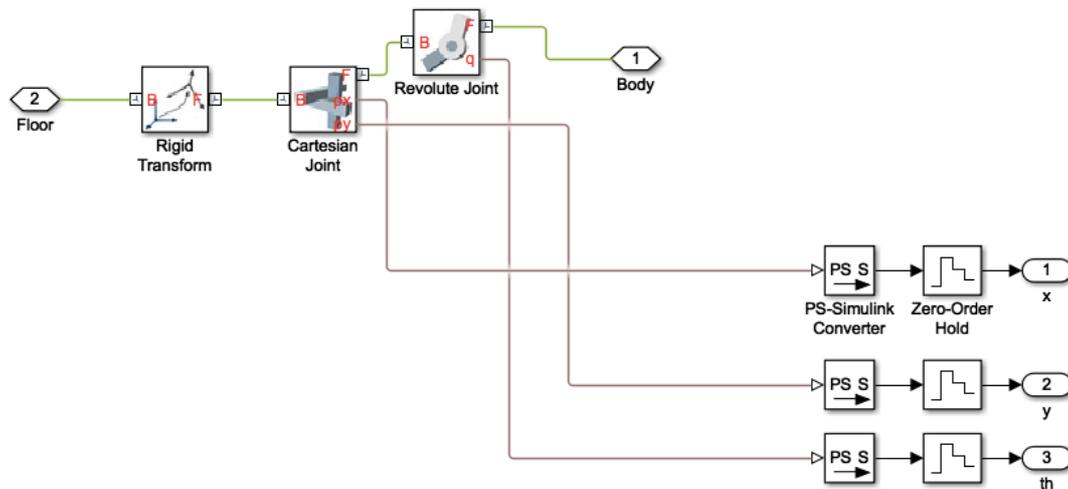


Figure 42. Joints used in the body, considering gravity

As shown in Figure 42, there are different structures to be considered in Simscape: a rigid body (introduced as STEP file), a rigid transform between points (where the rotation and translation between two points are considered), different types of joints, depending on the degrees of freedom of the body and the conversion blocks from Simscape back to Simulink.

In case the environment structure would change, the only required changes in the configuration of this Simscape file structure should be done in the general floor structure: in particular, the floor dimensions might need to be changed or the start point, target point or obstacles should be modified. These parameters are simply changed by introducing in each rigid transform structure its coordinates in regards to the World Frame.

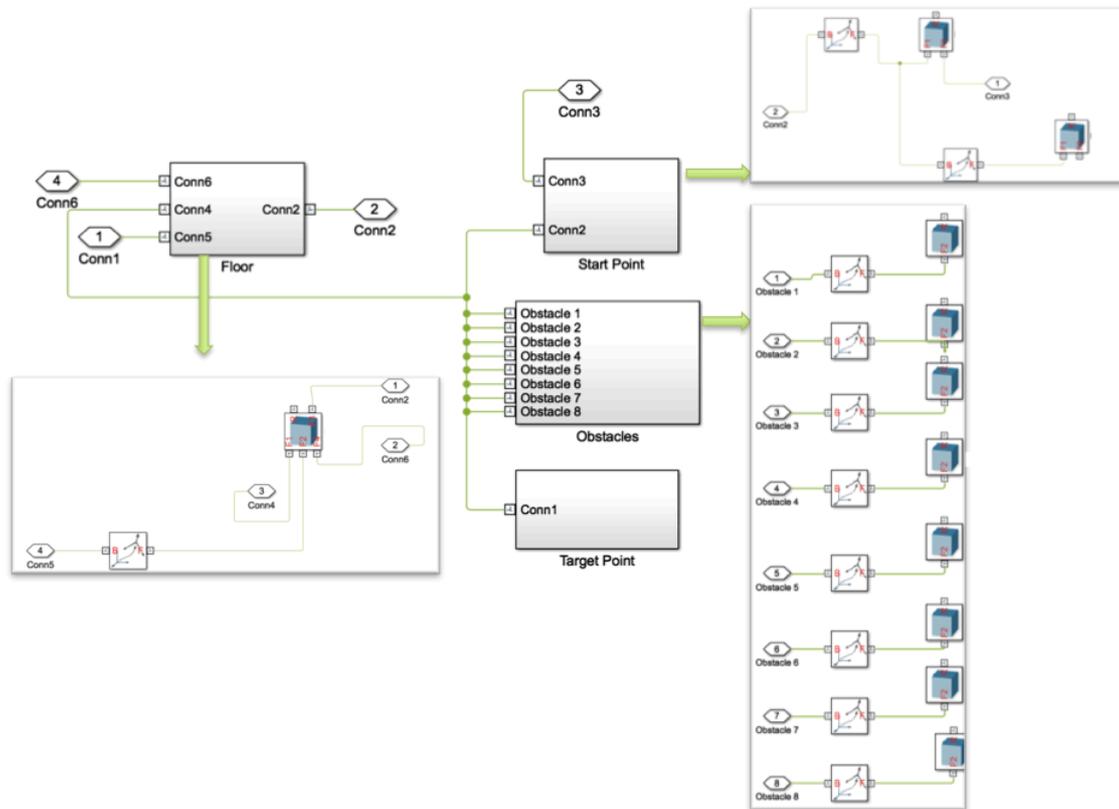


Figure 43. General Floor structure with third hierarchical level subsystem

Figure 44 presents the General EV3 structure, where the velocity for each wheel (left and right) is obtained from the controller structure (equivalent to the one in Figure 30), converted from Simulink to Simscape type of block and introduced in the revolute joint of each wheel as shown in Figure 45, with a resulting velocity shown in Figure 46. At the same time, each wheel is connected to the body structure of Figure 38 and connected to the SM-Contact Forces blocks shown in Figure 50.

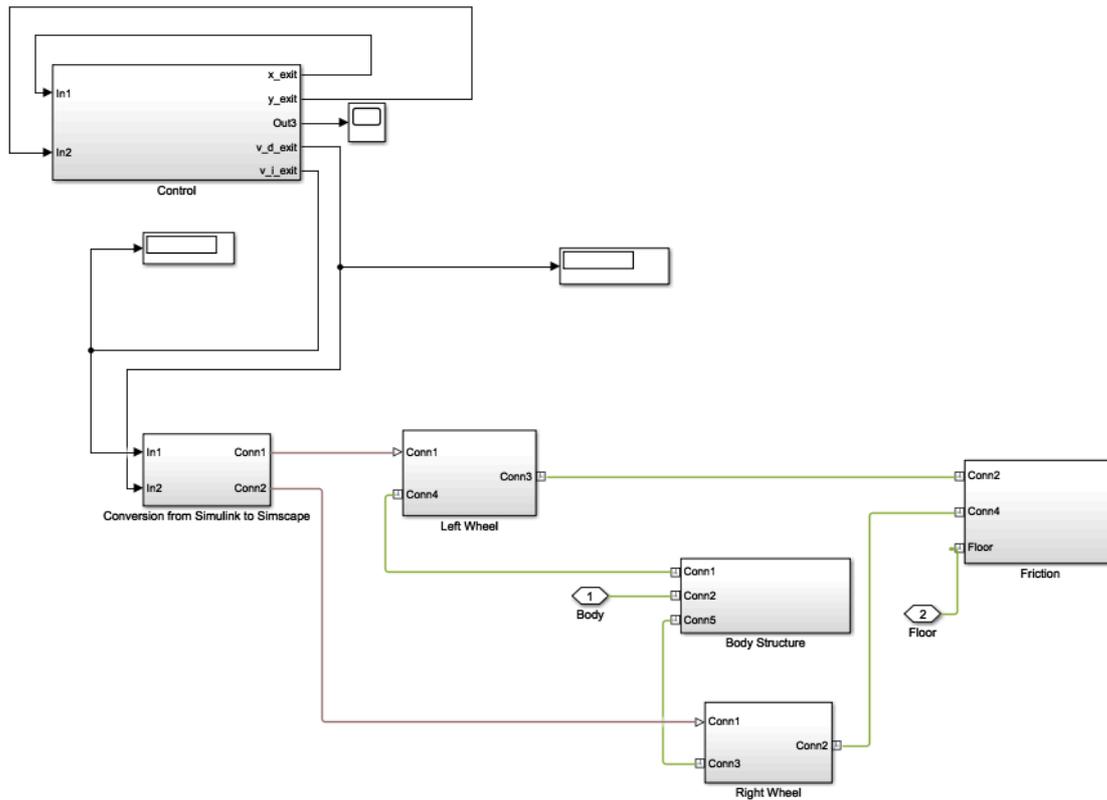


Figure 44. General EV3 structure

The EV3 structure subsystem is crucial for this project. It presents an innovative co-simulation structure between the controller and the mechanic structure, allowing the behavior of the robot to be checked in respect of its physical parameters (center of mass, moments of inertia and products of inertia calculated from the introduced density of the material) and friction before being implemented on the real system. For this project, since the parameter introduced into the wheel from the controller is the velocity of the robot, the type of motion actuation considered is provided by the input, and the torque is automatically computed as shown in Figure 47.

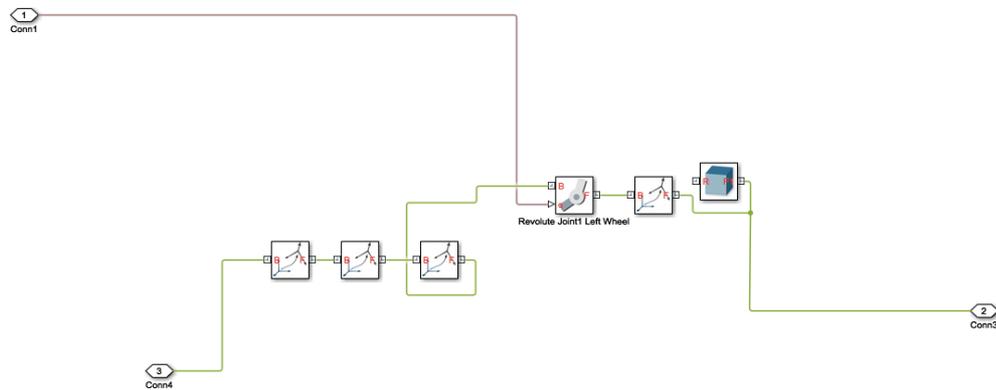


Figure 45. Each of the wheel's structure model

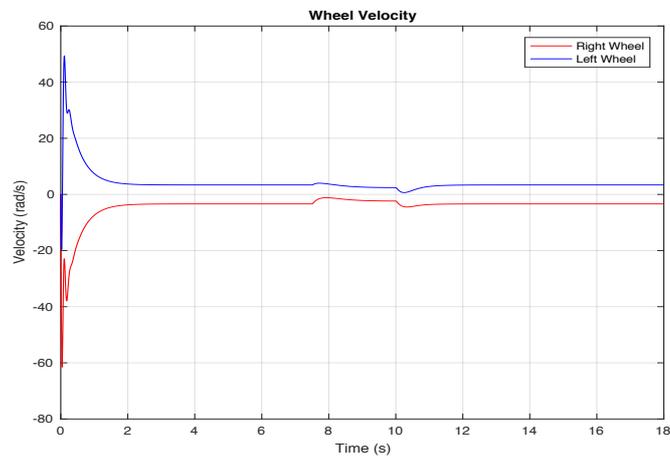


Figure 46. The Wheel's Behavior

A contact force is then considered between the wheel and the floor as shown in Figure 47. Considering the wheel radius, the sphere around it is considered to have a radius of 0.03 m.

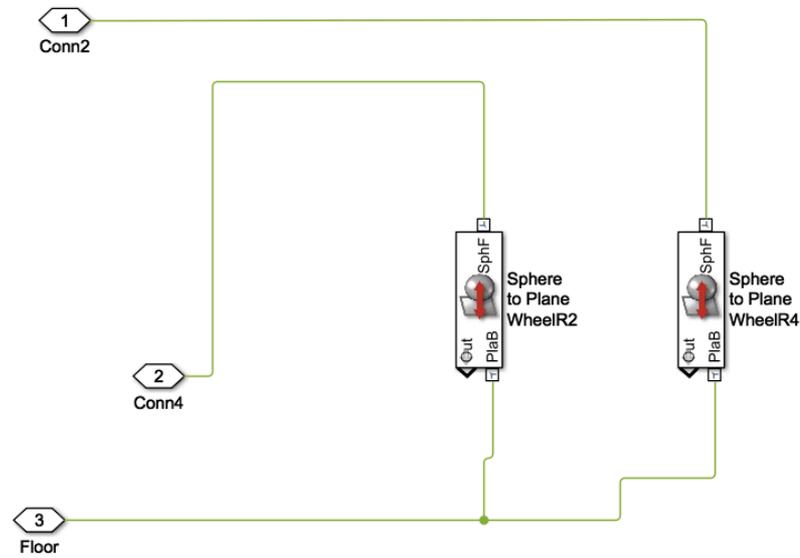


Figure 47. SM-Contact Forces for each wheel

Figures 48 and 49 shows the result of this co-simulation between Simulink and Simscape Multibody, which has allowed the behavior of the robot to be precisely proved before implementation, as shown in Figure 50.

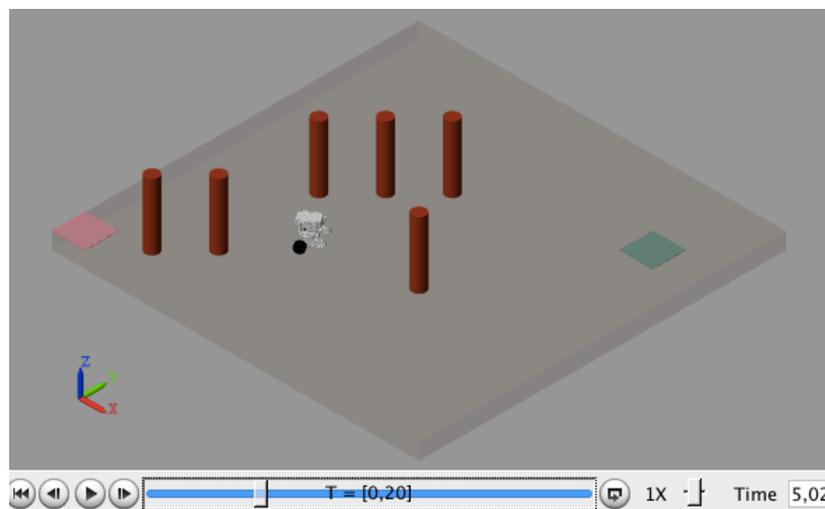


Figure 48. The Resulting Simulation Side View

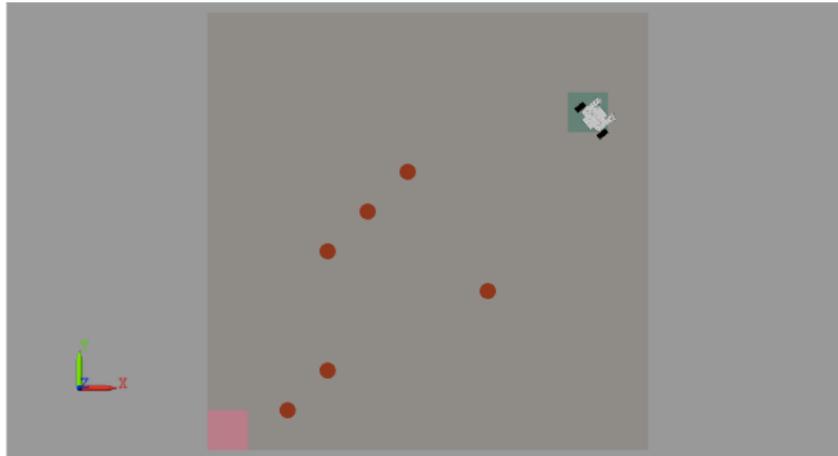


Figure 49. The Resulting Upper Simulation

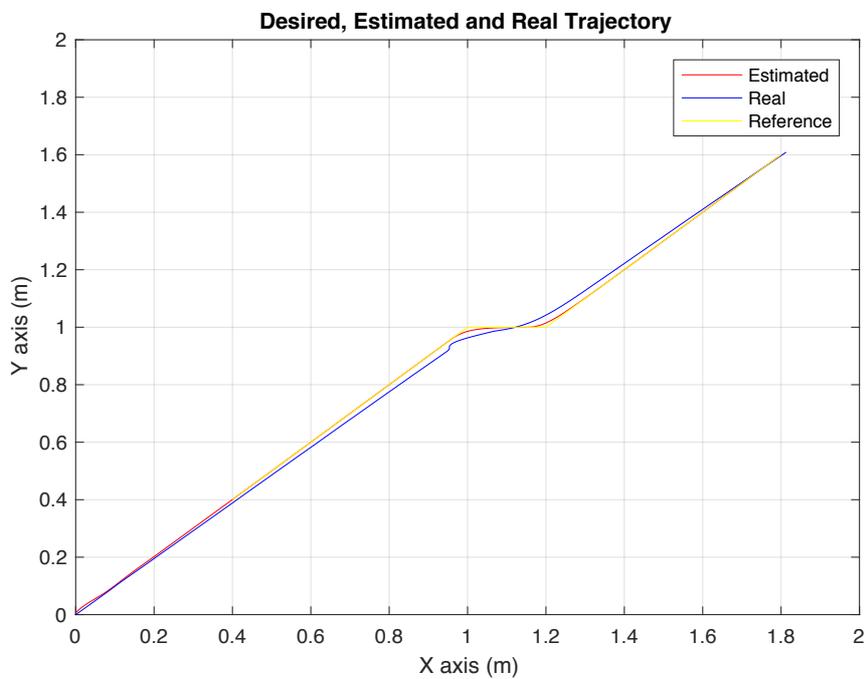


Figure 50. Simulation Behaviour: Desired, Estimated and Real Trajectory

As illustrated in Figure 50, the obtained result is quite accurate and models the robot's real behavior better than the previous visualization.

CHAPTER 9: IMPLEMENTATION PLATFORMS FOR EV3 ROBOT

After having tested the robot's behavior with several simulations, and in particular, with the more precise one that considers the robot's friction with the floor, its density, inertia moments and mass, the proposed techniques are implemented in the EV3 real robot. Two independent implementations have been developed: the first one uses LEGO MINDSTORMS EV3 Support for Simulink, whereas the second one uses the cross-robotics-platform programming environment RobotC. In both implementations the following phases for parameter obtention are observed:

1. •Coordinates x and y for each point in the optimal path obtained from the A* algorithm, using Matlab. The sequence of points in the trajectory is completed until reading 1001 points using the pchip command of matlab, which stands for connecting a piecewise polynominal structure. These points are transfered to the corresponding platform in matrix form.
2. •For each dimension, x and y, the reference velocity is obtained by subtracting the reference position from the actual reference position, considering a time step of 0.02 s for a simulation time of 20s.
3. •Develop the two classical types of controls required for mobile robots: kinematic and dynamic. The global control corresponds to a cascade control, where the dynamic control represents the outer loop and the kinematic control, the inner one. By applying the kinematic control theory, the decentralized position points are obtained.
4. •By applying the inverse kinematics control theory, the velocity of each wheel is obtained (considering one right wheel and one left wheel with different velocities).
5. •By multiplying by the radius, the reference angular velocity is obtained for each wheel.
6. •By using a dynamic control structure consisting of a proportional $K_p=9$, the wheel's velocity control is obtained, where the velocity parameter is passed to the robot's motors in the wheels.
7. •The encoders get the wheel's pulse, that has to be subtracted by the wheel's previous time step state and divided by the step time in order to obtain the robot's real angular velocity (and transformed to radians).
8. •This angular velocity is transformed into linear by multiplying by the radius of the wheel and then, using the forward kinematics control for the differential model, the robot's real position x, y and its theta angle is obtained.
9. •This real parameters x, y and theta are used to feedback the initial kinematics control.

9.1 LEGO MINDSTORMS EV3 SUPPORT FOR SIMULINK

LEGO MINDSTORMS EV3 support for Simulink provides a hardware library where all the structure parts that compose the robot are considered, as shown in Figure 51. Therefore, in order to implement the system the first task was to download the library as an add-on to Simulink.

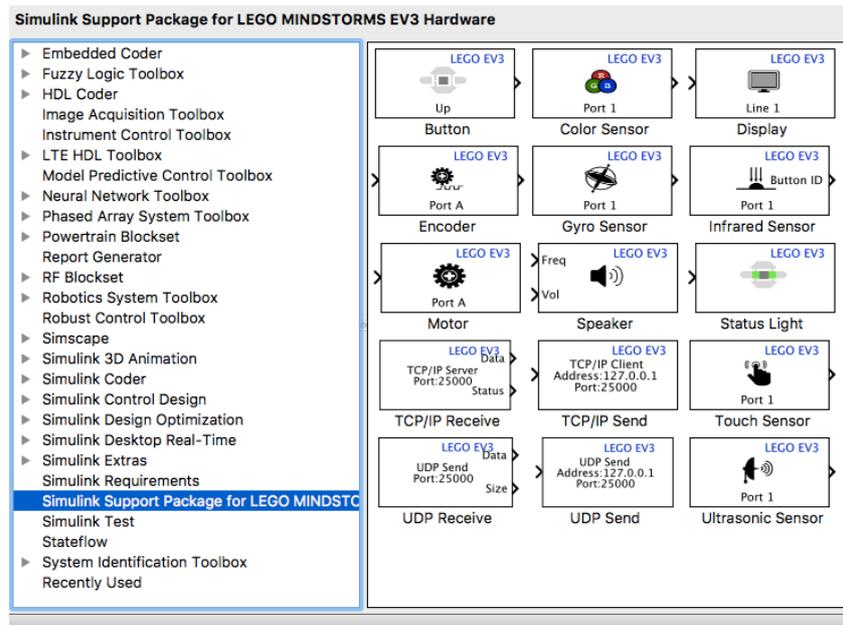


Figure 51. Simulink Support Package for LEGO MINDSTORMS EV3 Hardware

In order to connect to the robot via WIFI, the Netgear N 150 adapter was required. It also had to be installed the firmware version of the robot that allowed the connection, being the correct firmware version 1.06H. The required connection protocol in order to connect is WPA2 Personal (or none).

The general structure of the implementation is shown in Figure 52, showing a high level hierarchical structure, where the kinematics control as well as the inverse kinematics are inside the control subsystem (Figure 54) and the reference angular velocity is introduced to the wheel's motor in the subsystem for each wheel. Figure 53 shows the wheels' structure. After obtaining the robot real angular velocity and multiplying it by the radius in order to get the linear velocity, it is introduced into the Forward Kinematics control subsystem.

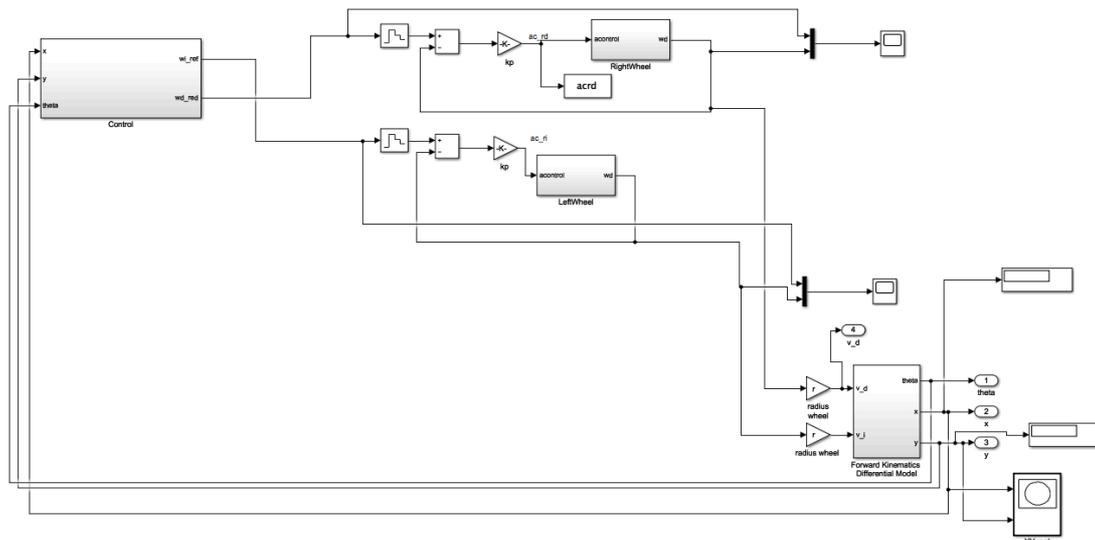


Figure 52. Global Structure

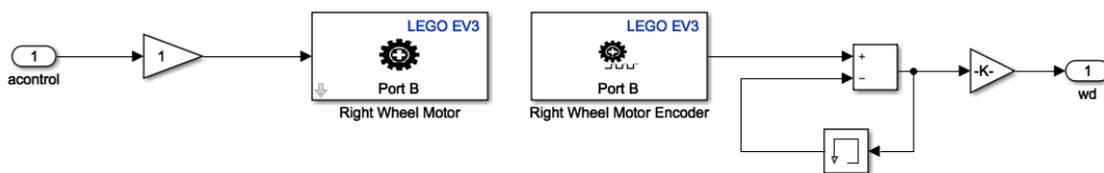


Figure 53. Wheel's Structure

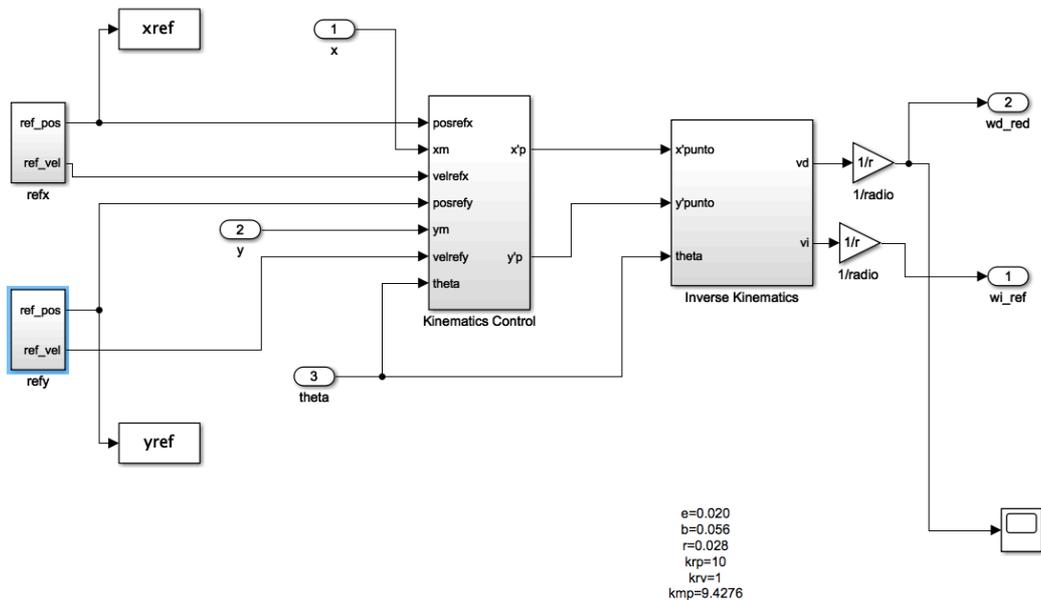


Figure 54. System Structure, considering Kinematics and Inverse Kinematics control

Figure 55 shows the robot's behavior in both the reference simulation and the real behavior of the robot.

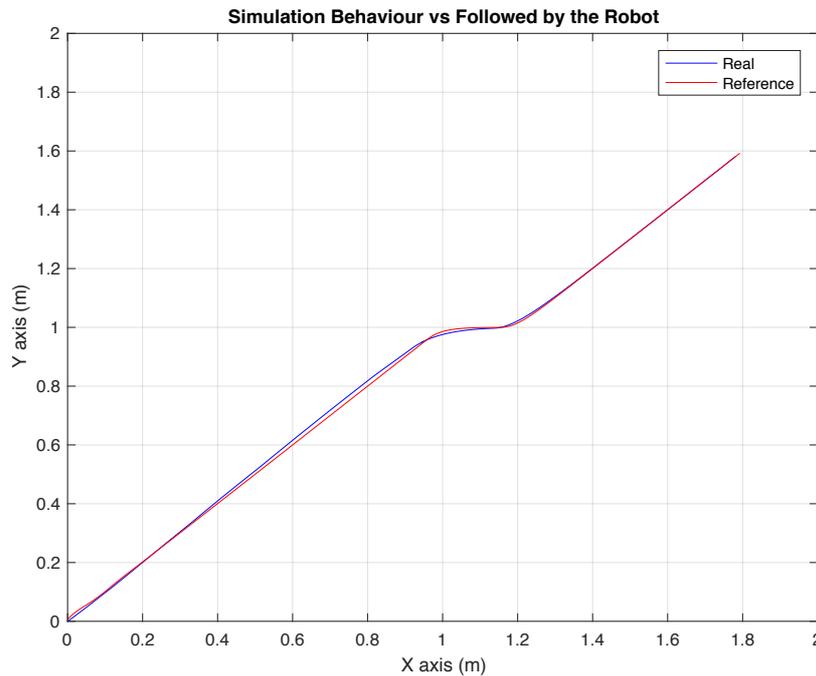


Figure 55. Simulation Behaviour vs Real Behaviour

9.2 ROBOTC PROGRAMMING PLATFORM AND LANGUAGE

C is one of the most widely used programming languages of all times, having influenced many programming languages later. RobotC is a C-based programming language, especially designed for robotics applications. To take advantage of the widespread of the RobotC language, the proposed methodology has been adapted to RobotC and implanted in the Robotics Lab of the ai2 Institute of the UPV.

For implementing the control structure, the following procedure is considered:

- a. Compute angular velocities of the wheels

$$wd = pul2rad * (wheeldPrevious - wheeldNow)/0.02$$

$$wi = pul2rad * (wheeliPrevious - wheeliNow)/0.02$$

- b. Compute linear velocities of the wheels

$$v = w * radius$$

$$vd = wd * radiuswheel$$

$$vi = wi * radiuswheel$$

- c. Compute linear velocity of the robot

$$velLin_robot = (vd + vi)/2$$

- d. Compute angular velocity of the robot

$$velAng_robot = (vd - vi)/(2 * b)$$

- e. Compute X-Y position and orientation of the robot

$$x = x + velLin_robot * \cos(theta)$$

$$y = y + velLin_robot * \sin(theta)$$

$$theta = theta + velAng_robot$$

- f. Implement the kinematic control

$$xp_point = kr_v * vel_xref + kr_p * (xref - (x + g * \cos(theta)))$$

$$yp_point = kr_v * vel_yref + kr_p * (yref - (y + g * \sin(theta)))$$

- g. Implement the inverse kinematics control: compute the velocity of each wheel (considering one right wheel and one left wheel with different velocities).

$$gcos_theta = g * \cos(theta)$$

$$gsin_theta = g * \sin(theta)$$

$$bsin_theta = b * \sin(theta)$$

$$bcos_theta = b * \cos(theta)$$

$$vd_ref = \frac{(gcos_theta - bsin_theta) * xp_punto + (gsin_theta + bcos_theta) * yp_punto}{g}$$

$$vi_ref = \frac{(gcos_theta + bsin_theta) * xp_punto + (gsin_theta - bcos_theta) * yp_punto}{g}$$

- h. Compute the linear velocity is obtained for each wheel

$$wd_ref = vd_ref / radiuswheel$$

$$wi_ref = vi_ref / radiuswheel$$

- i. Implement the dynamic control structure: compute the wheel's velocity controller and transfer the wheel velocity parameter to the robot's motors

$$acontrolA = error_wd * kmp$$

$$acontrolB = error_wi * kmp$$

- j. Compute the robot's real angular velocity (and transformed to radians):

Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots

- i. The value of the encoders give the value of the wheel's pulse.

$$wheelNow = getMotorEncoder(leftMotor)$$
$$wheeliNow = getMotorEncoder(rightMotor)$$

- i. Calculate the angular velocity of the wheels by subtracting from the actual value of the pulse given by the encoders, the wheel's previous pulse, and divide it by the step time:

$$wd = pul2rad * (wheelNow - wheelPrevious)/0.02$$
$$wi = pul2rad * (wheeliNow - wheeliPrevious)/0.02$$

- k. Estimate the wheel's angular velocity errors and apply the control actions to each wheel.

$$error_wd = wd_ref - wd;$$
$$error_wi = wi_ref - wi;$$
$$acontrolA = error_wd * kmp;$$
$$acontrolB = error_wi * kmp;$$

- l. Run the code transferred to the robot and retrieve the datalog text file to be subsequently transferred to matlab to graphically analyze the results.

CHAPTER 10: COMPUTER VISION SYSTEM FOR OBTENTION OF THE ENVIRONMENT CONFIGURATION

In previous chapters, the location of the starting point of the robot trajectory, as well as the target point and the obstacles in the environment were introduced manually. In this chapter, the locations of the obstacles into the environment are automatically recognized by processing the images that are captured by a camera by using a very simple computer vision system.

10.1. MODELING THE ENVIRONMENT CONFIGURATION BY MEANS OF A MONOCULAR CAMERA

For recreating an unknown industrial environment, a randomly located set of obstacles are automatically detected by using a monocular camera installed on the ceiling. Then, they are conveniently encoded and introduced in Matlab as the set of obstacles to be considered by the implemented A* algorithm. Considering a thin lens, where the thickness is negligible compared to the radius of lens surfaces curvature, rays through the center of the lens do not change direction. After surpassing the lens, rays that are parallel to the optical axis meet focus as shown in Figure 56. Considering perspective projection, the image will become bigger as the object gets closer (Shen, 2017).

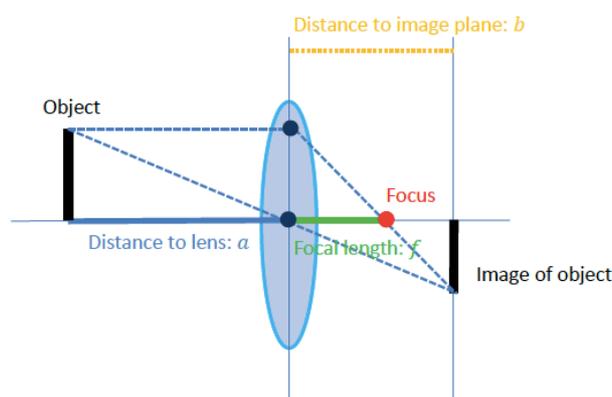


Figure 56. Thin lens perspective projection

Considering the pin-hole camera model shown in Figure 57 (Shen, 2017), the object image is upside down, so it is important to assume that the image plane is in front of the lens.

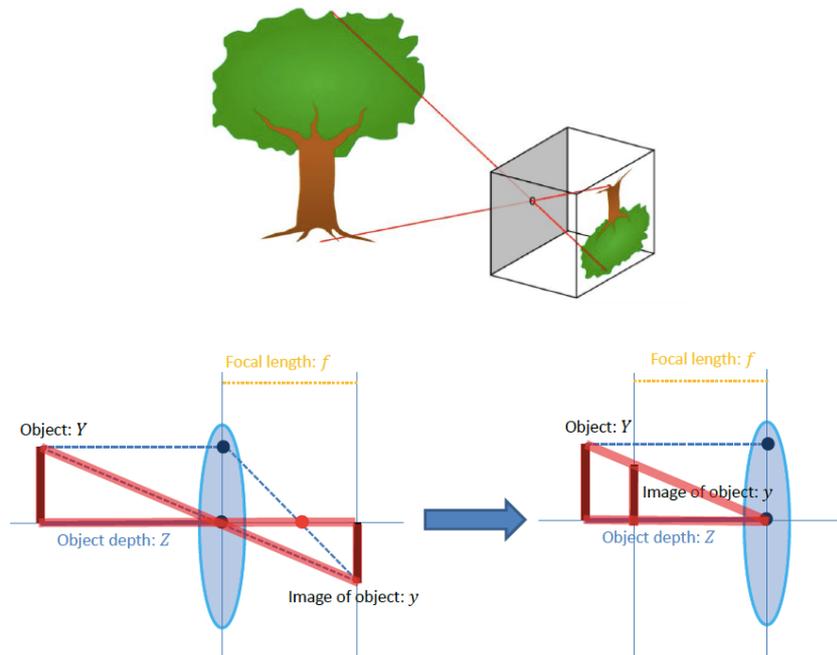


Figure 57. Pin -hole Camera Model

It is crucial to expect an offset in the x and y coordinates caused by the error and to calibrate the camera accordingly.

10.2 PERSPECTIVE PROJECTION

Considering a perspective projection as the one shown in Figure 58, the optical axis, being the z-axis, is perpendicular to the image plane(u,v), where the intersection of these two planes is the image center (u_0, v_0) and f is the distance in pixels of the image plane to the origin. The formulas to consider for computing the image plane coordinates are the following:

$$u = \frac{f \cdot X_c}{Z_c} + u_0$$

$$v = \frac{f \cdot Y_c}{Z_c} + v_0$$

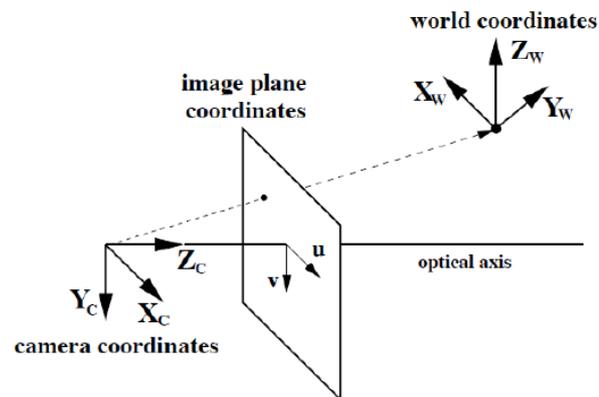


Figure 58. Perspective Projection

As a result, the following matrix form equation must be considered:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & u_o \\ 0 & f & v_o \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix}$$

Together with the equation that translate the camera parameters to world frame:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

The complete pin-hole camera model is presented in Figure 59, where the unknown depth can be obtained considering the pixel values, intrinsic camera calibration camera pose and world point (Shen, 2017):

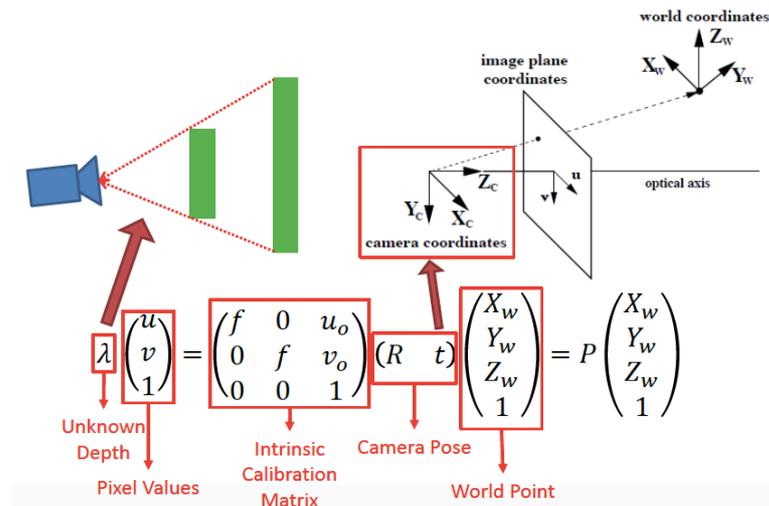


Figure 59. Pin-hole Camera Model

It must be considered that wide angle lenses have radial distortions, where the straight lines become curves, but this was not our case as the proposed camera had a thin lens.

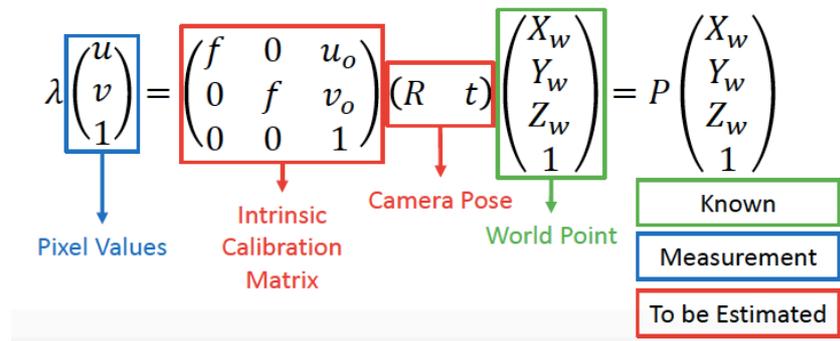
10.3 CAMERA CALIBRATION

For calibrating the image, a calibration object as required is shown in Figure 60:



Figure 60. Calibration object

Using this object, intrinsic parameters and poses of the cameras with respect to the calibration object are obtained. Using the following equation, where the pixel values are obtained by careful measurement, the world point is known and the intrinsic calibration matrix and camera posed is estimated.



The calibration matrix is:

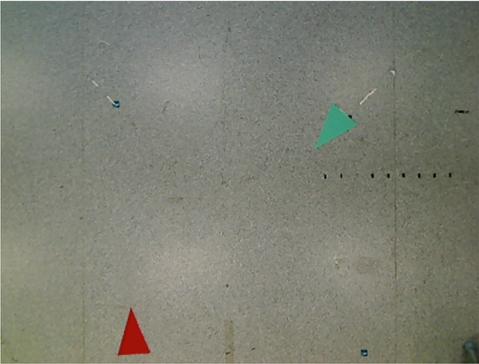
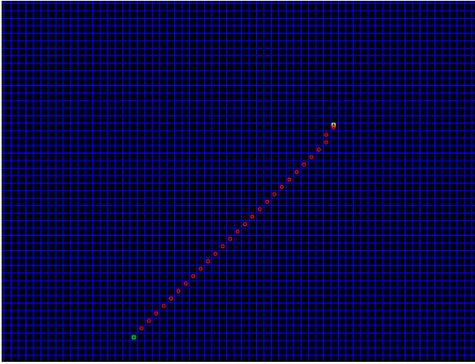
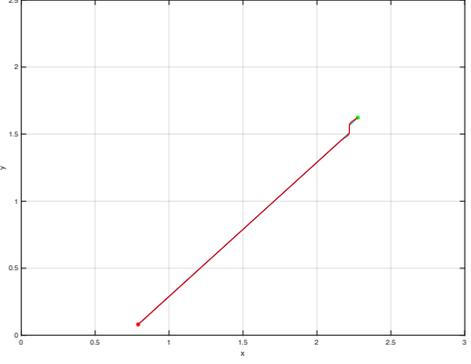
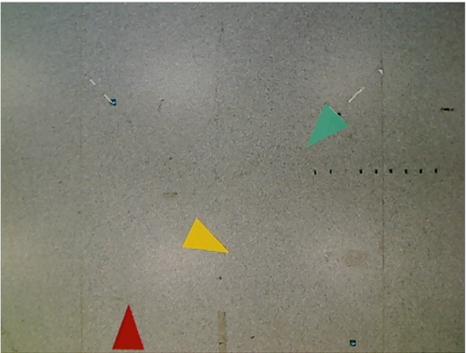
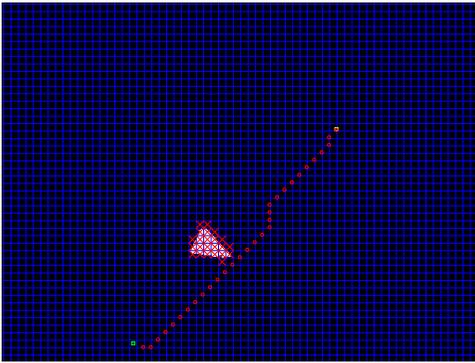
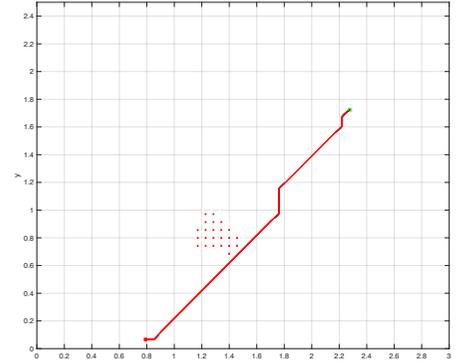
$$theta = \begin{bmatrix} -5.557190660439 & 0.142628475911 & 0 \\ 0.115215296172 & 5.533663392034 & 0 \\ 1521.5431137392655 & -190.250769088174 & 1 \end{bmatrix};$$

Once the camera is calibrated, the initial and target positions can be obtained, as well as the obstacle positions.

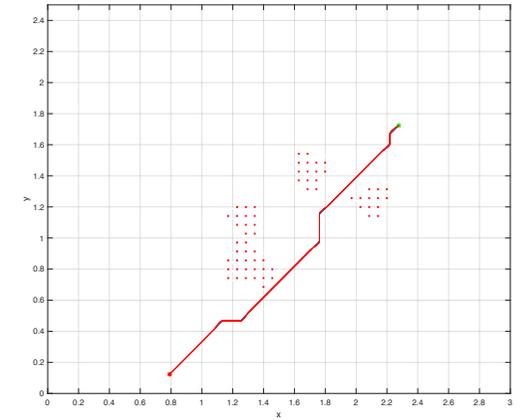
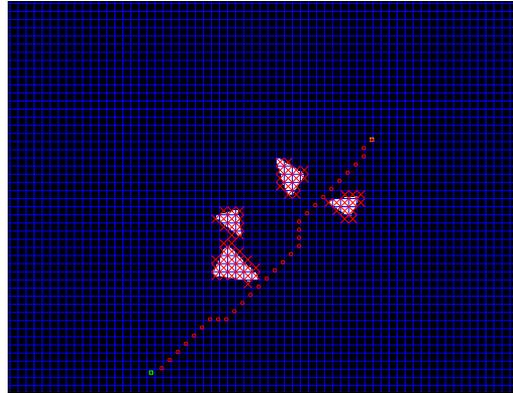
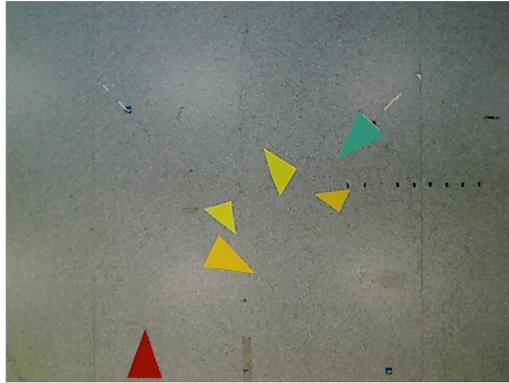
In this project, the initial position is considered a red triangle, the target position a green one, and the obstacles are signaled by yellow triangles. With the use of a webcam, a photo of the general environment is taken. Afterwards, it is required to segment the image so that it only considers the triangular red card that is the initial position of the robot, the green card as the final position, and the yellow cards as the obstacles while disregarding any other information. With the segmented and eroded image of the initial and final position of the robot, the perimeter of each card is obtained, as well as the distance between border points and the center of gravity of the object, and its initial orientation. In order to apply the algorithm, the center of the pixels is considered. For the obstacle cards, the image is also segmented and eroded, but afterwards, a matrix with the positions occupied by the obstacles (value equal to 1) is created, where the positions free of obstacle have a value equal to 0. A map matrix considering the initial and target positions and obstacles is created.

Several sequences of examples illustrating the extraction of the environment using the monocular camera are shown in Table 4, where the first image of the sequence is the image taken of the environment, the second image corresponds to the trajectory given in pixels and the third image of each sequence is the real trajectory given in meters.

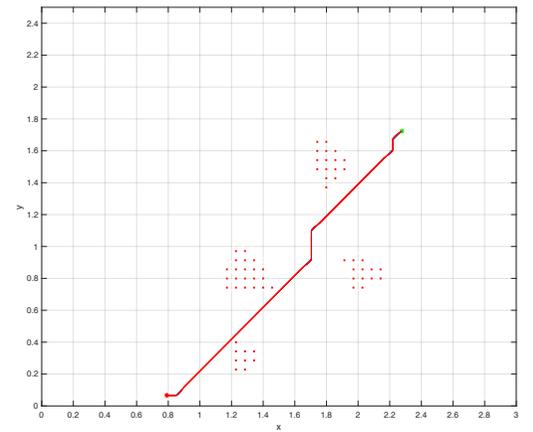
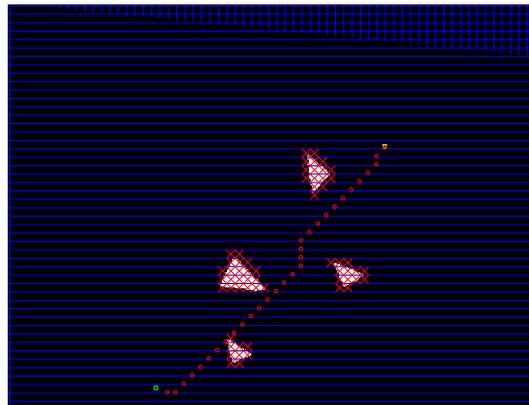
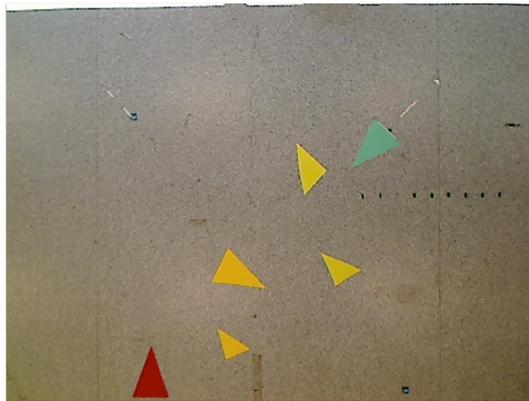
Table 4. Extraction of Environment Configuration using Monocular Camera

Photo of the Environment	Trajectory in Calibrated Image (in Pixels)	Computed Robot Trajectory (in meters)
Example 1		
		
Example 2		
		

Example 5



Example 6



CHAPTER 11: APPLICATION OF THE DEVELOPED METHODOLOGY TO A REPRESENTATIVE CASE STUDY

This chapter presents a complete overview of the application of the designed and implemented methodology and system to a representative case study that is developed in the Robotics Lab of the ai2 Institute of the UPV, shown in Figure 61.

For doing this, the following procedure is fully explained:

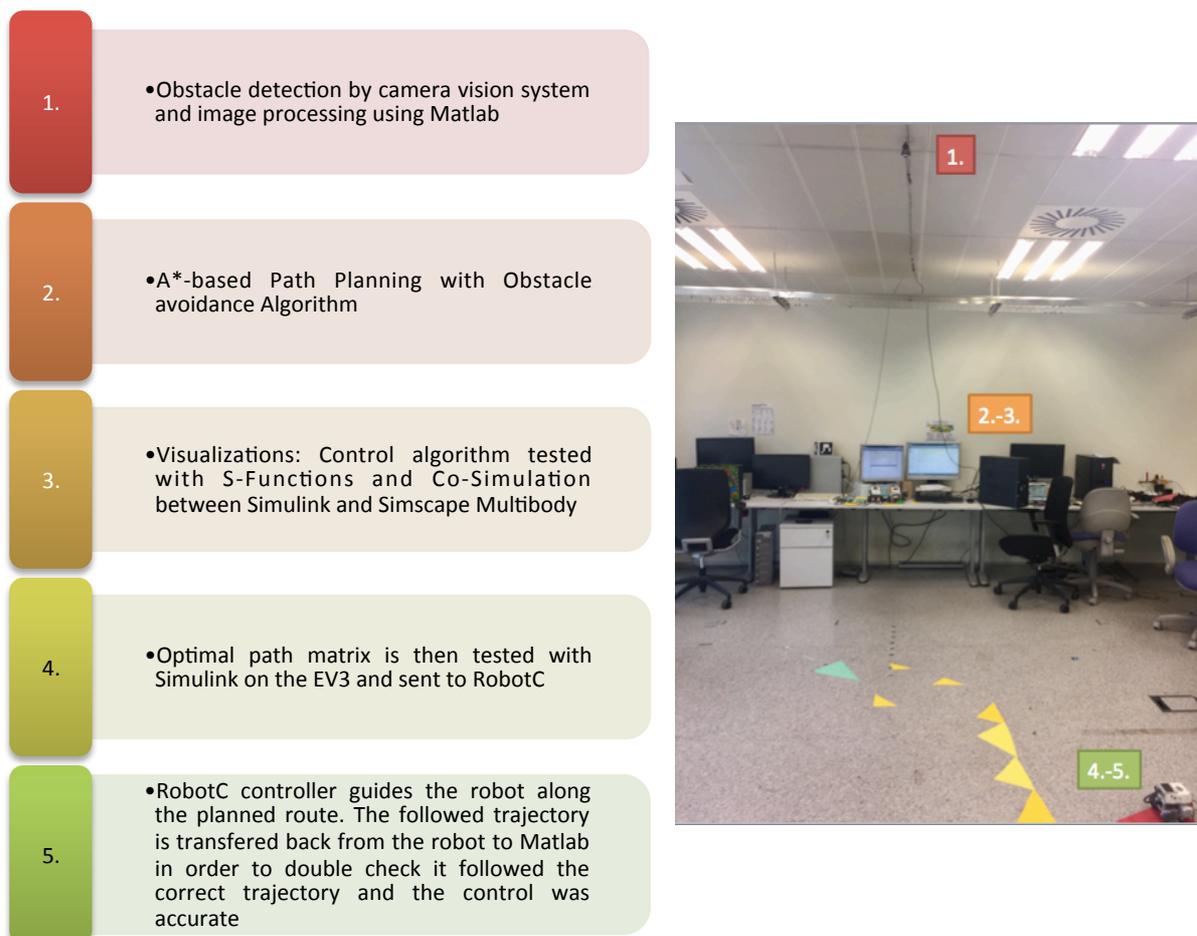


Figure 61. The applied Methodology

As already discussed, the co-simulation allows individual components to be simulated with diverse complementary tools running at the same time and exchanging information in a collaborative environment. Following this methodology, the leading component in the co-simulation receives a vector of input values that are calculated by using Matlab, which interacts with a second software, to automatically visualize the robot's behavior considering the implemented control and physical properties of the robot.

After having accurately simulated the robot's behavior with the co-simulation, the technique is implemented in a real robotic system, the LEGO MINDSTORMS EV3, by following a methodology that also combines two different software systems: a more academic one, by using LEGO MINDSTORMS EV3 support library for Simulink, which allows the robot real-time behavior to be evaluated, and a more professional one, by using the C-based robotics programming language RobotC, which supports industrial-strength applications and witnesses its applicability to any industrial system.

11.1 OBSTACLE DETECTION BY CAMERA VISION SYSTEM AND TRANSFERENCE TO MATLAB

The methodology is fully applied to two different environment configurations that are shown in Figures 62 and 63. As already explained, the initial position is marked by means of a red triangle, the target position by a green one, and the obstacles as yellow triangles as shown in Figures 62 and 63. The image is segmented by colors to obtain the initial and final position, as well as the obstacles as shown in Figure 62 (right) for Configuration 1 and Figure 64 for Configuration 2.

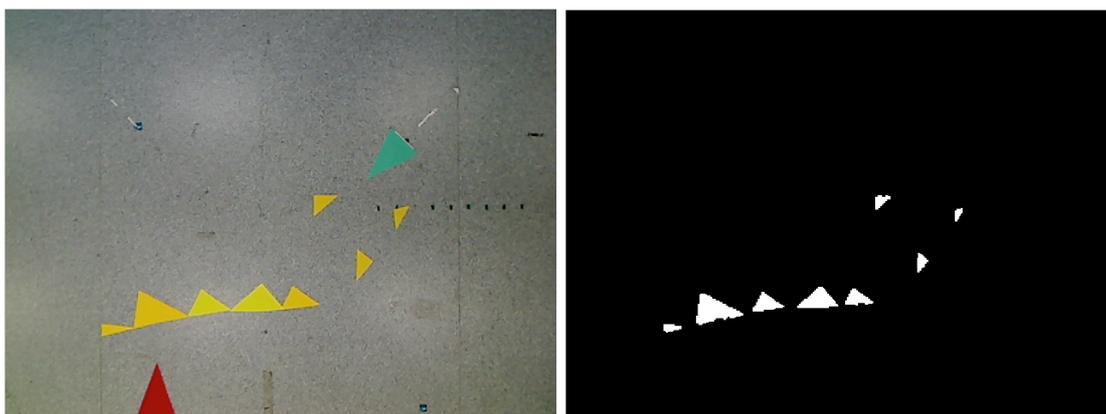


Figure 62. Case Study: Environment Configuration 1

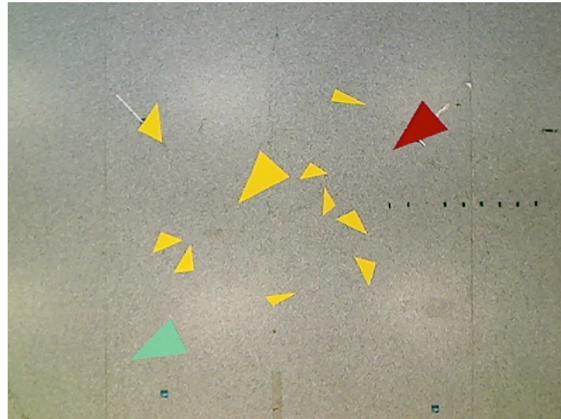


Figure 63. Case Study: Environment Configuration 2

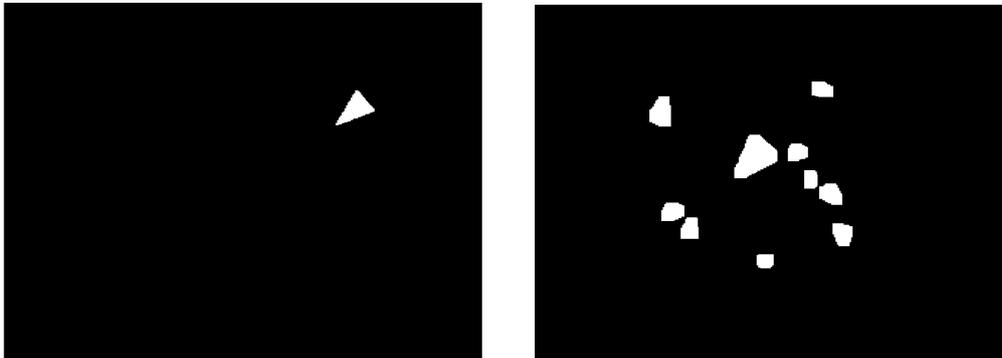


Figure 64. The Initial Position (left) and the Eroded and Segmented obstacles (right) of Configuration 2

11.2 COMPUTING THE OPTIMAL PATH TRAJECTORY USING A* ALGORITHM

Considering the fixed initial and final position of the robot, as well as a robot diameter of the robot = (10 cm), the minimum cost path from the initial to the target position is calculated using the previously explained A* algorithm. Afterwards, the coordinates are obtained from the calculated nodes and joined using a spline and using the calibration matrix. The pixel coordinates of the first configuration are shown in Figure 65 and in Figure 66 for the second one. The robot configuration trajectory (calibrated in meters) is shown in Figures 67 and 68.

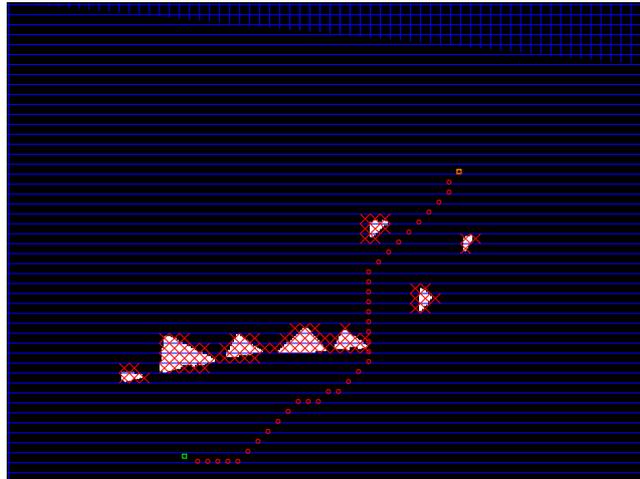


Figure 65. The found path in pixels for Environment Configuration 1

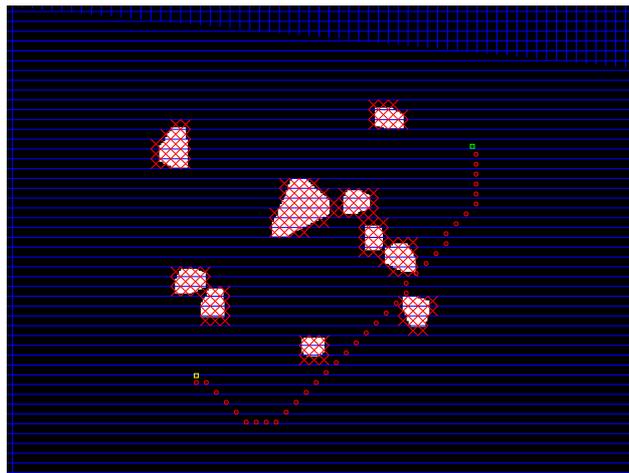


Figure 66. The found path in pixels for Environment Configuration 2

11.3 SIMULATION OF THE OPTIMAL PATH

The trajectory obtained with the A* algorithm in Matlab and controlled with Simulink, is tested by using S-Functions (as shown in Figure 69 for Configuration 1 and Figure 70 for Configuration 2). Then, in order to consider physical properties and friction, the optimal trajectory obtained for the case study configurations is tested using the Simscape Multibody framework using the model shown in Figure 41, considering the created map with the initial and target positions and obstacles, which are introduced into each system. This validates that the robot reaches its goal with no collision.

Similarly, for any new environment configuration acquired with the vision system, the variable matrices “trajectory” and “map” generated with Matlab in step 2 must be conveniently saved to be input to the S-Function visualization. Then, by simply running the Simulink model, a simulation is automatically obtained for the new environment configuration. This is a simple way to visually detect that the computed angle and position of the robot at each point of the path trajectory are adequate.

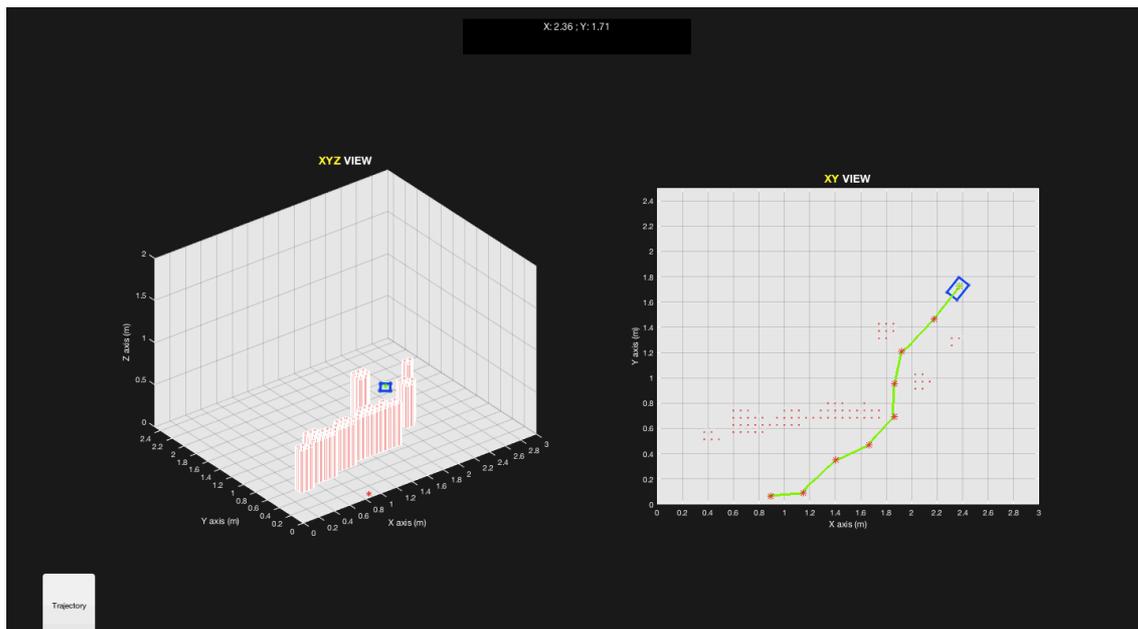


Figure 69. Visualization with S-Functions for Configuration 1

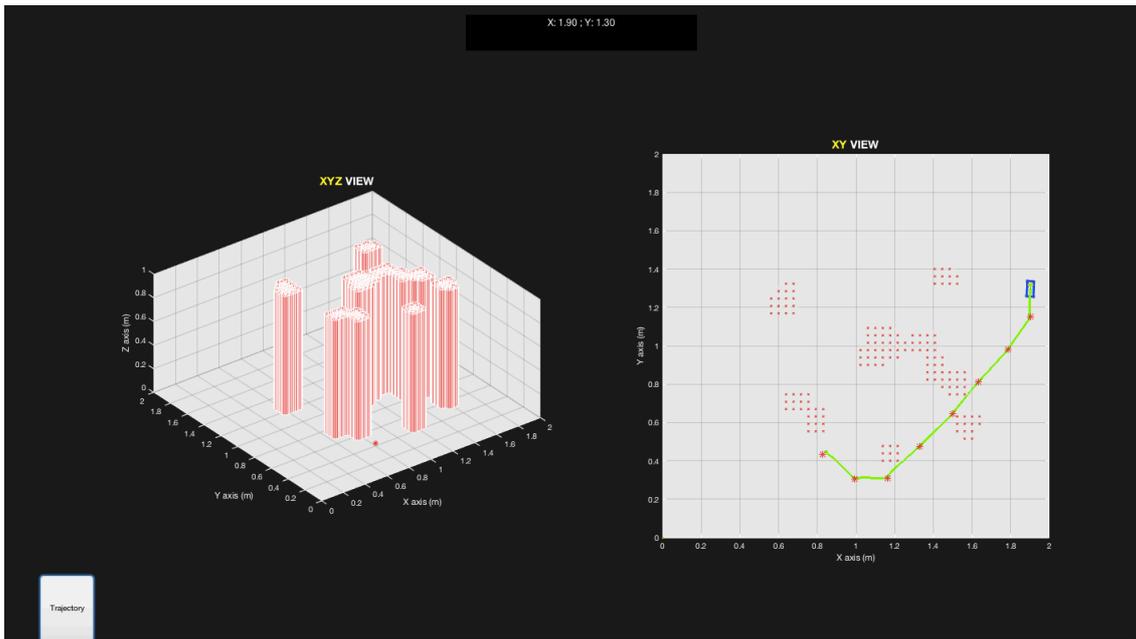


Figure 70. Visualization with S-Functions for Configuration 2

In regards to the more realistic visualization regarding Simscape Multibody simulation, it imports a step-type model of the robot and considers physical characteristics of the robot such as inertia moments, weight, precise dimensions and configuration structure. The visualization for a new environment is almost as simple and automated as the one for S-Functions. For the same type of differential configuration robot, only the positions of the target and initial position, as well as the obstacle location have to be conveniently replaced as shown in Figure 71.

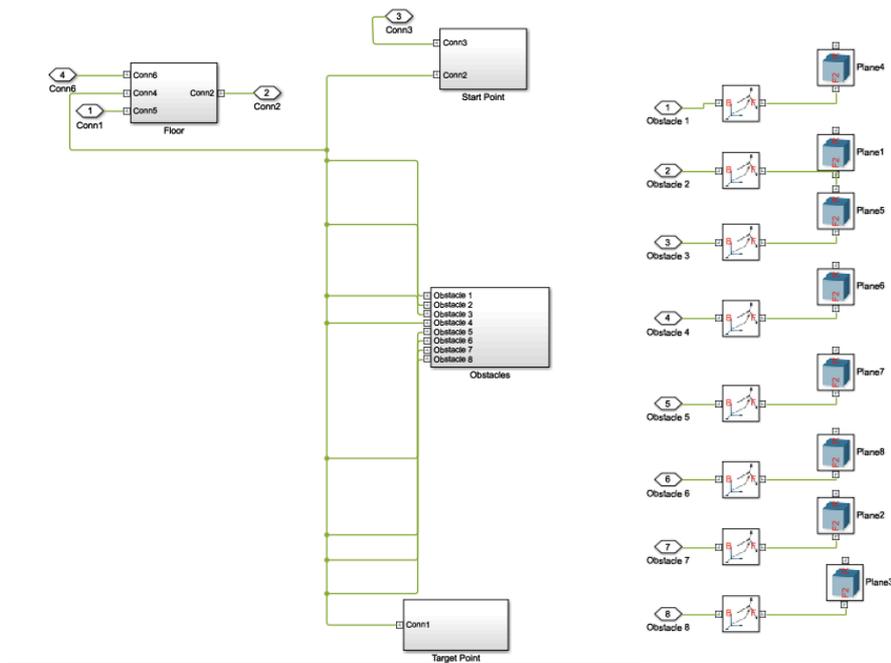


Figure 71. Simulink Structure requiring changes for a new environment configuration

Figures 72 and 73 show the simulated behavior of the robot for the case study Configuration 1 using Simscape simulation. The whole simulated environment is illustrated in Figures 74 and 75.

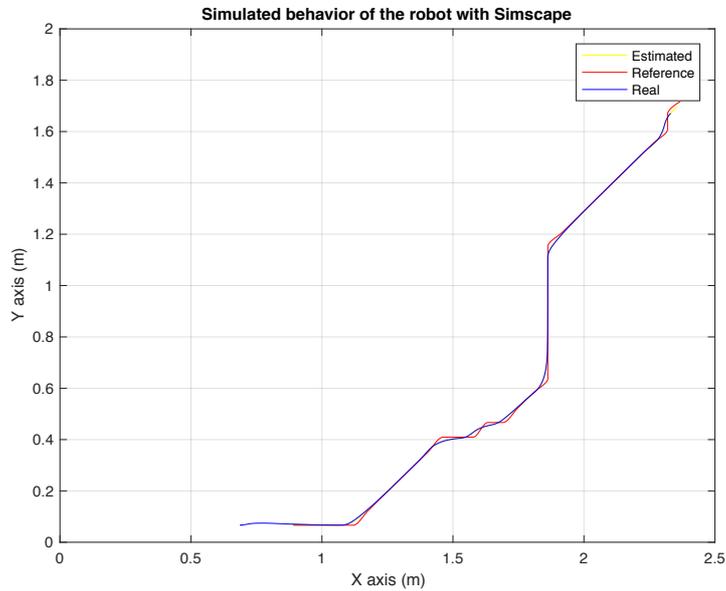


Figure 72. Simulated behavior of the robot with Simscape

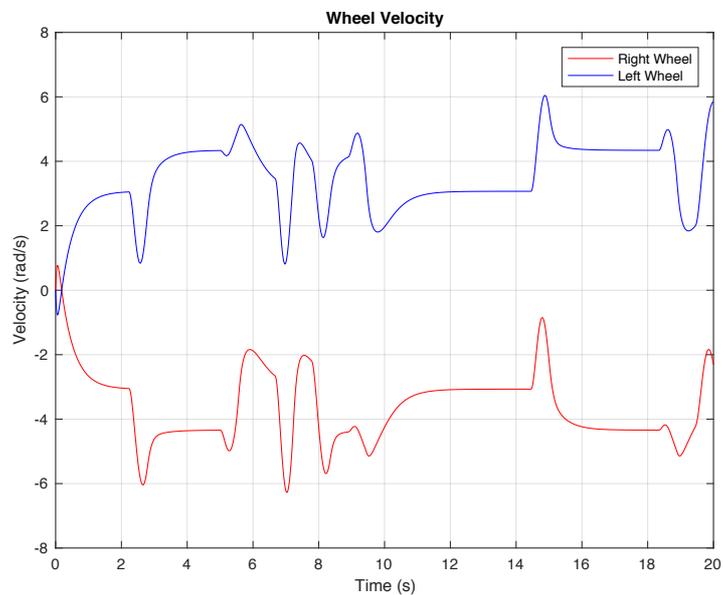


Figure 73. Wheel Velocity

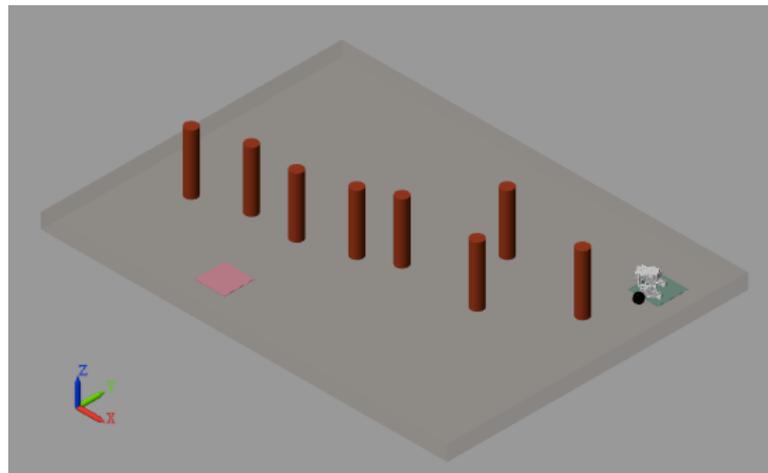


Figure 74. Simulation with Simscape Multibody Side View

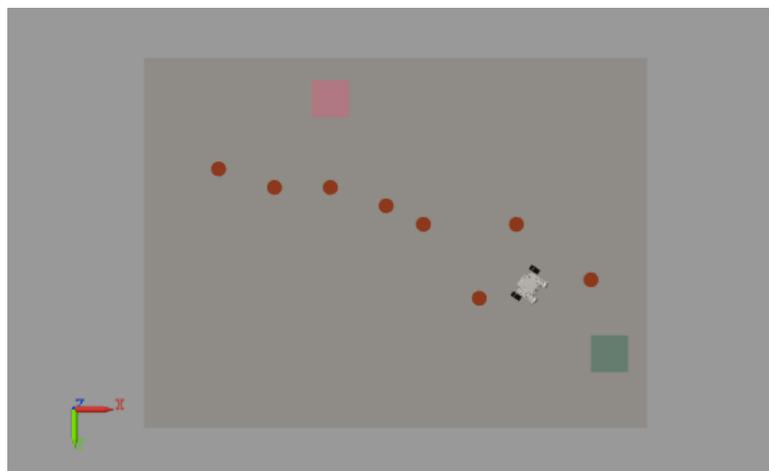


Figure 75. Simulation with Simscape Multibody Upper View

11.4 SIMULINK IMPLEMENTATION FOR EV3

By considering the initial controller and using the LEGO MINDSTORMS library for Simulink, the implementation structure corresponding to Figure 52 has been split into two parts, using the UPD structure: the Simulink model for Host computer (Figure 76), and the model to be executed on the LEGO MINDSTORMS EV3 (Figure 77), which allows to run the S-Function visualization in real-time. This way, via UPD, considering the IP address of the EV3 hardware, a communication system is established via Wifi using the Netgear adapter, where the robot control is applied to the robot by clicking on “run on target hardware” and the Simulink model runs in the host computer in real time, allowing the visualization via S-Functions to be computed.

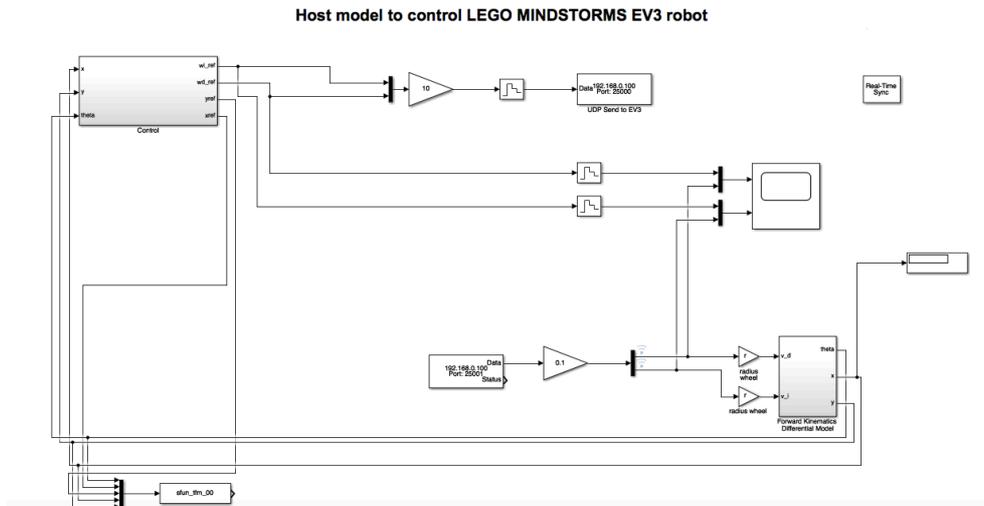


Figure 76. Simulink Model for Host computer

LEGO MINDSTORMS EV3 Robot Control

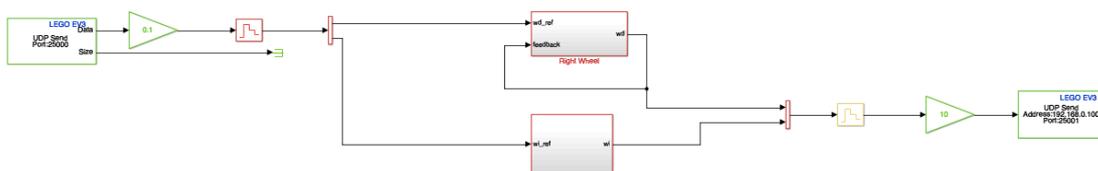


Figure 77. Model for LEGO MINDSTORMS EV3

Figures 78 and 79 illustrate the real trajectory followed by the EV3 robot in the environment. The graph shown in Figure 79 is generated in real-time as the robot moves, as well as the graphs shown in Figures 82 and 83, visually describing the wheel's control behavior.

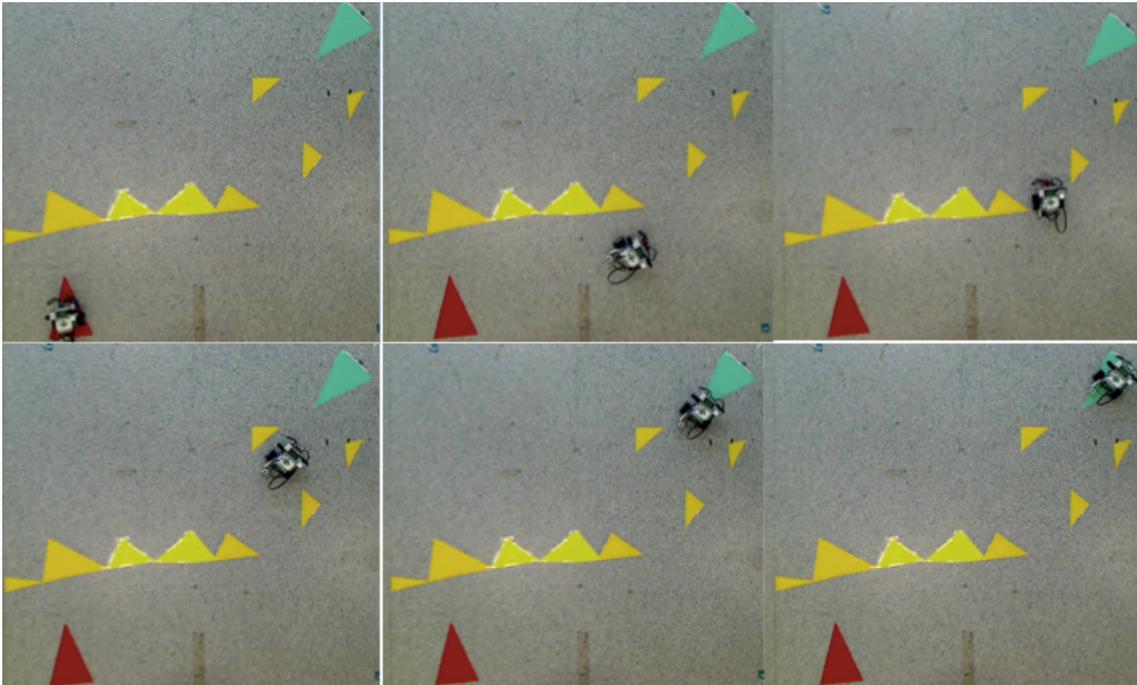


Figure 78. EV3 Robot following the path trajectory computed by the Simulink implementation

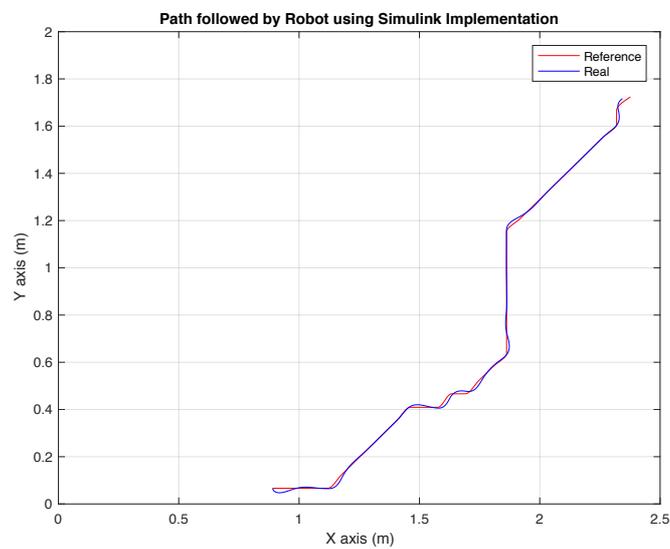


Figure 79. Path followed by the robot under the Simulink control implementation

Figure 80 and 81 show the variation with time of the position in x and y axis for the reference and real trajectory coordinates.

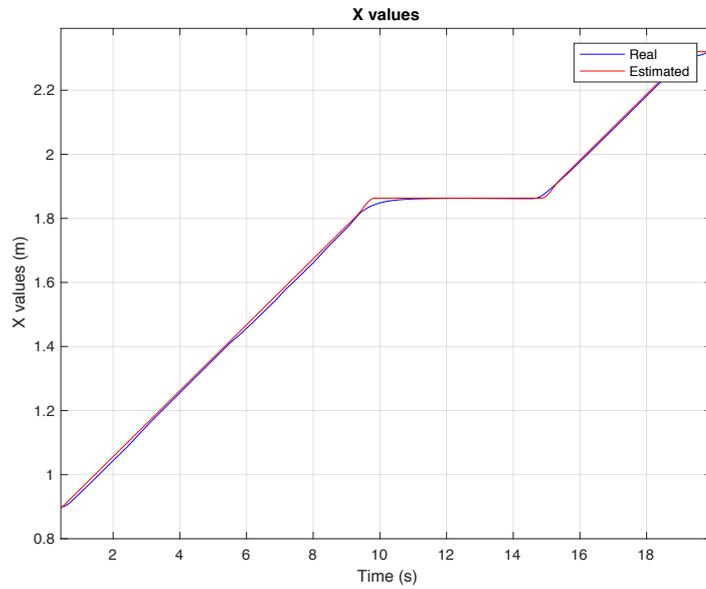


Figure 80. X Values with Simulink Implementation

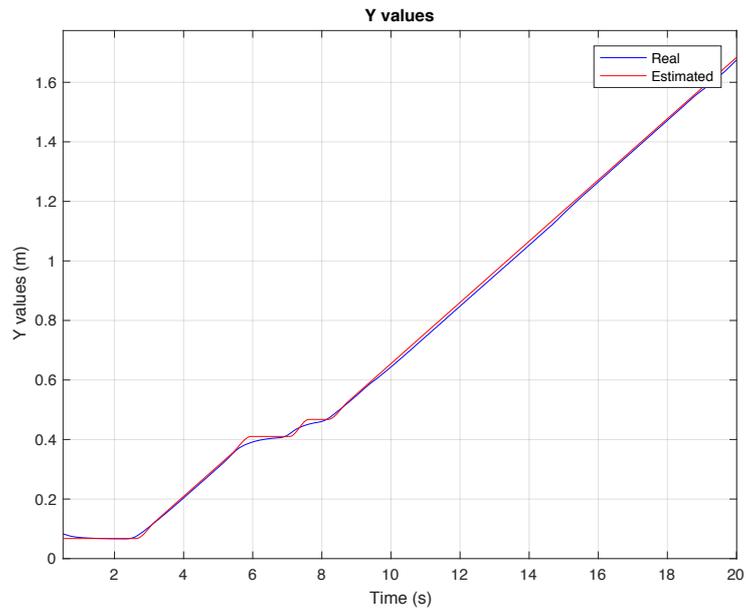


Figure 81. Y Values with Simulink Implementation

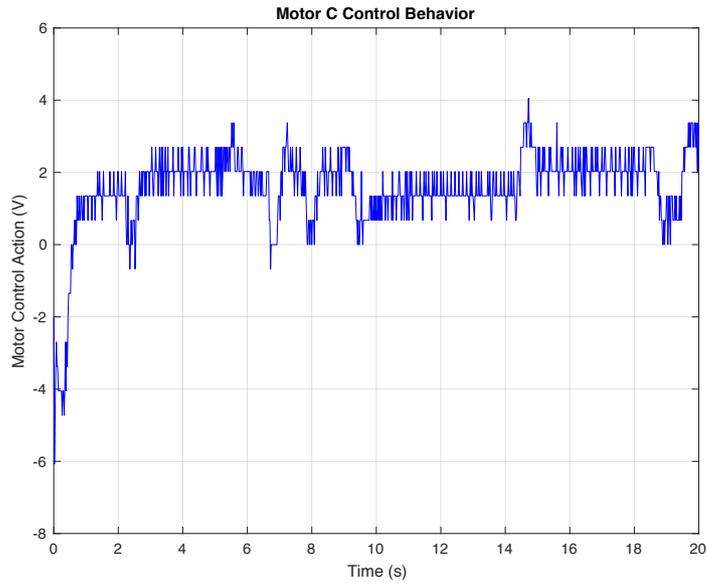


Figure 82. Motor C Control Behaviour

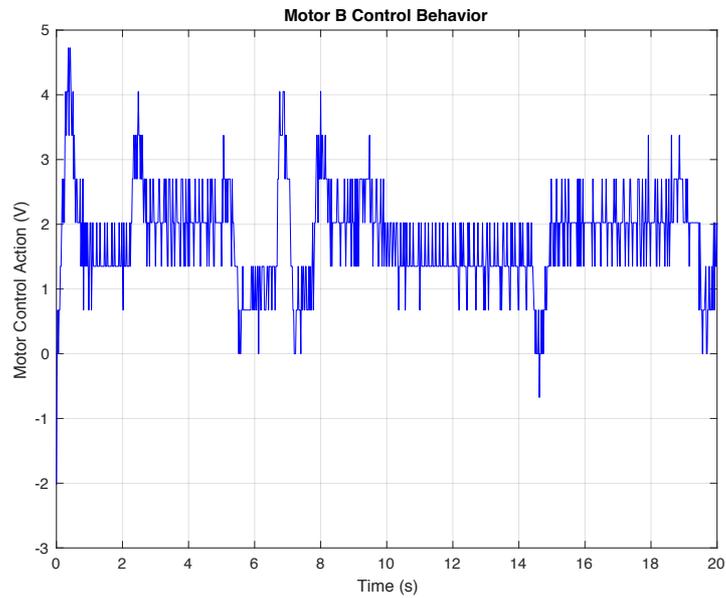


Figure 83. Motor B Control Behaviour

11.5 ROBOTC IMPLEMENTATION FOR EV3

This section reports on the robotC implementation of the controller, accordingly to the algorithm previously explained. Given the case study Configuration 1, the robot follows the computed trajectory that is shown in Figure 84.

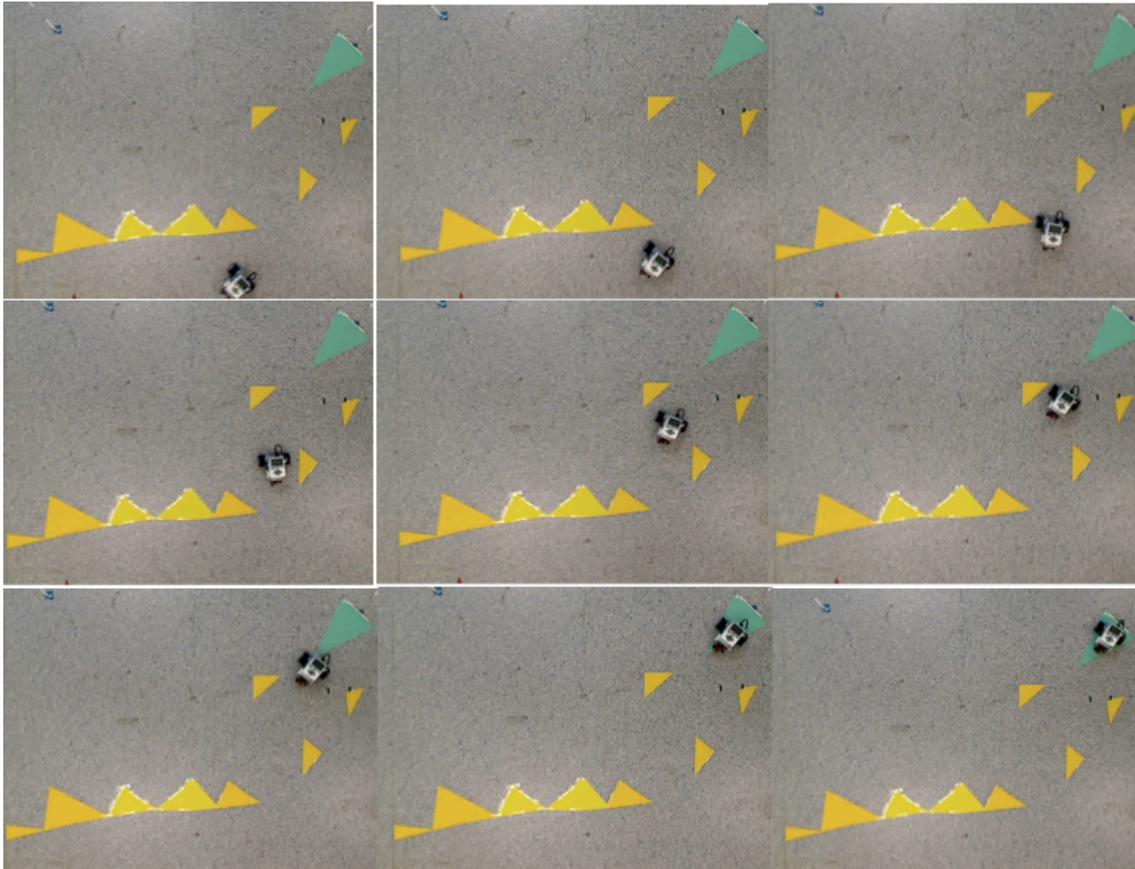


Figure 84. EV3 Robot following the path trajectory computed by the RobotC implementation

In order to double check the followed path, a datalog including the reference positions and the positions actually visited by the robot is created, and transferred back to Matlab for evaluation.

The complete code implementation consists of 1297 lines of Matlab code, 601 lines of RobotC code and 167 KB of Simulink code (structures).



CHAPTER 12: CONCLUSION AND EVALUATION

12.1 CONCLUSION

In the last years, mobile robots have become a subject of significant interest because they are open to a broad range of possible applications in many industrial sectors. In this project, a methodology and system have been developed that provide mobile terrestrial robots with a path finding navigation strategy in a grid-map forming static environment with unknown obstacles. In any navigation scheme, the desire is to reach a destination without crashing or getting lost. Moreover, when the mission is executed, it is necessary that a feasible and optimal path is planned that not only avoids obstructions in its way but also minimizes costs such as time, distance, or energy. The proposed path finding provides the mobile robot more autonomy and intelligence.

The robotic system developed in this project efficiently performs autonomous tasks including basic recognition, positioning, decision-making, trajectory planning, and action in an industrial enclave. The proposed methodology relies on a computer vision system that is able to acquire and interpret the working configuration. Such a configuration is then processed by a developed Matlab program that implements the classical, Artificial Intelligence A* Algorithm in order to synthesize an optimal, least-cost path trajectory from the initial position to the specified target position without collision based on the current map and reliable localization. Then, starting out from the start position in the grid, the mobile robot autonomously heads for its destination position in the grid to reach the target.

Several set up environments are tested in the simulations within a rectangular workspace map. The environment mapping depends on different colours that are identified in the environment set up. The robotic system can identify three colors inside the environment: red, yellow and green. The yellow color is interpreted as an obstacle area; the red colour designs the starting position; and the green colour stands for the target position.

In order to validate the generated paths from the starting point to the goal, they are tested with S-Functions and in co-simulation between Simulink and the Simscape Multibody framework. Considering a differential configuration robot EV3, a model configuration of its direct and inverse kinematics models are developed in Simulink, which allows its position to be visually controlled to validate the robot's behavior. Roughly speaking, this is done by feeding Simulink with the trajectory matrix generated by Matlab so that a co-simulation is automatically generated: first a simple visualization (S-Functions) and then a more dependable



one that considers the effect that ground friction causes on the robot based in a co-simulation between Simulink and Simscape Multibody.

The simulation part is an estimation of the real expected result. The algorithms are implemented in Matlab whereby the environment is studied in a two dimensional coordinate system. The algorithm permits the robot to move from the initial position to the desired position following an estimated trajectory.

Once the models are validated by testing the considered trajectories, the technique is implemented in a real robotic system, the LEGO MINDSTORMS EV3, by following a methodology that combines two different approaches and software systems. On the one hand, by using LEGO MINDSTORMS EV3 support for Simulink, which supports academic applications and allows to evaluate the robot's real-time behavior. On the other hand, to assess the validity of the methodology for real industrial robots, the algorithm is implemented and tested using RobotC, which provides for industrial-strength applications. Finally, the trajectory followed by the robot in our case study is sent back to Matlab for evaluation.

12.2 EVALUATION

In this present project, the problem of path planning in a 2-dimensional workspace with obstacles avoidance has been addressed. Given an environment configuration and goal position, the techniques proposed in this thesis support planning and deploying of complex robot maneuvers for obstacle avoidance. This is done by developing algorithms, interactive simulations and efficient implementations that have allowed the creation of a fully automated methodology that can be applied to any industrial environment.

It has been proven that this methodology can be applied to different industrial robots (with a differential configuration) in an automated manner. The vision system allows a precise description of the environment configuration to be acquired. In regards to the least-cost path finding algorithm A*, which is based on a combination of uniform cost search with a suitable heuristic, it has been proven well-suited for obstacle avoidance applications, as its application delivers highly precise, accurate path trajectories for obstacle avoidance. The obtained path is the shortest path from all possible free trajectories..

By using two types of simulations, the computed trajectory is thoroughly evaluated before implementing and deploying the real system. Considering the control system implemented in Simulink, a completely automated and simple (ideal) recreation of the robot's behavior is obtained by using S-Functions. In order to evaluate the behavior under lifelike conditions, critical parameters such as friction, mass, and inertia moments are considered with the Simscape Multibody Simulation. It is worth noting that the resulting trajectory for both types of simulations are very much in agreement, proving the accuracy of the developed algorithms and techniques. We have run our simulation in several set up environments where,



in each situation, the robot succeeds to avoid the obstacles and reach its target. Our navigation approach has an advantage of adaptivity as it works perfectly even in an unknown environment.

After simulation, the path finding methodology is implemented in a real robot, the LEGO MINDSTORMS EV3, by following an approach that combines two different robot programming platforms: Simulink and RobotC. The implementation in Simulink allows the robot's real time behavior to be evaluated as data are transferred from Simulink to the robot via Wifi, which has been proven to be really useful and it increased the safety of the system. The methodology has also been implemented in a C-based language within the cross-robotics-platform programming environment RobotC. The C programming language runs in almost all platforms, which makes our system virtually platform-independent and opens up a wider range of applications of the developed methodology and system for industrial robots. Also, it has been found that both implementations are accurate and work equally precisely, proving the benefits of the co-simulation technique.

To conclude, the methodology developed in this MSc thesis can be considered highly precise and is potentially applicable to any industrial context and terrestrial mobile robot. The proposed path finding strategy has the advantage of being generic and can be parameterized at the user's demand. Since the algorithm is general for any obstacle detection the obstacles can take any shape. This approach works perfectly even in an unknown environment, where the robot is able to understand the structure of the environment and find a non-colliding way towards its goal.



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots



CHAPTER 13: REFERENCES

Books

(Murray, R., Li, Z. and Sastry, S., 1994): *A Mathematical Introduction to Robotic Manipulation*. CRC Press, California.

Journal articles

(Karaman, S., and Frazzoli, E., 2011): *Incremental Sampling-based Algorithms for Optimal Motion Planning*. International Journal of Robotics Research, 76.

Electronic material

(GrabCad site, 2018) Hudák, J., 2018. *LEGO MINDSTORMS EV3* [consulted 15th April 2018]. Available at: <<https://grabcad.com/>>

(Izmiran, 2005) *Pchip*. The MathWorks site [consulted 2nd and 3rd February 2018]. Available at: <<http://matlab.izmiran.ru/>>

(Lyall, A., Mercier, P., and Gstettner, S., 2018) *The Death of Supply Chain Management*. Harvard Business Review site [consulted 16th May, 2018]. Available at: <<https://hbr.org/>>

(UAB, 2018) Universitat Autònoma de Barcelona contributors. *A* Algorithm pseudocode* [consulted 2nd February 2018]. Available at: <<http://www.uab.cat/matematiques/>>

Miscellaneous

(Shen, S., 2017): *Introduction to Aerial Vehicles*. HKUST MSc on Electronic and Computer Engineering, Hong Kong University of Science and Technology.

(Valera Fernández, Á., 2017a): *Aplicaciones: Mini/Micro-Robótica*. ETSINF, Universitat Politècnica de València.

(Valera Fernández, Á., 2017b): *Robótica Industrial*. ETSII, Universitat Politècnica de València.



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for the automatic collision avoidance.
Application to terrestrial mobile robots

**MASTER DEGREE IN
INDUSTRIAL ENGINEERING
FINAL PROJECT**

DOCUMENT N°2:

PROJECT BUDGET



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for automatic collision avoidance.
Application to terrestrial mobile robots



DOCUMENT Nº2: PROJECT BUDGET

CHAPTER 1. Unit Prices	89
1.1. Analysis of the Path Planning and Obstacle Avoidance Techniques.....	89
1.2. Design of a Navigation Methodology	89
1.3. Simulation of the Methodology in two different environments	89
1.3. Application of the obstacle avoidance in the EV3	89
1.4. Food expenses and Meetings	90
1.5. Additional tasks and Activities.....	90
1.6. Other Concepts.....	90
CHAPTER 2: Measurements	91
2.1. Analysis of the Path Planning and Obstacle Avoidance Techniques.....	91
2.2. Design of a Navigation Methodology	91
2.3. Simulation of the Methodology in two different environments	91
2.4. Application of the obstacle avoidance in the EV3	92
2.5. Food expenses and Meetings	92
2.6. Additional tasks and Activities.....	92
2.7. Other Concepts.....	92
CHAPTER 3: Detailed Budget.....	93
3.1. Analysis of the Path Planning and Obstacle Avoidance Techniques.....	93
3.2. Design of a Navigation Methodology	93
3.3. Simulation of the Methodology in two different environments	93
3.4. Application of the obstacle avoidance in the EV3	94
3.5. Food expenses and Meetings	94
3.6. Additional tasks and Activities.....	94
3.7. Other Concepts.....	94
CHAPTER 4: Total Budget of the Project	95



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



**industrials
valència**
ESCOLA TÈCNICA SUPERIOR
ENGINYERS INDUSTRIALS
UNIVERSITAT POLITÈCNICA DE VALÈNCIA



Development and programming of algorithms for automatic collision avoidance.
Application to terrestrial mobile robots

This document contains the budget of this project for implementing the methodology and system for robot path planning with obstacle avoidance in industrial companies.

CHAPTER 1: UNIT PRICES

1.1. ANALYSIS OF THE PATH PLANNING AND OBSTACLE AVOIDANCE TECHNIQUES

Description	Unit Prices
Robot Manipulation and Obstacle Avoidance bibliography review	40€/hour
Planification of the activities to carry out in order to solve the problem	40€/hour
Typing and text editing	10€/day
Required Electricity	0.3€/day
Computing system wear and tear	1€/day

1.2. DESIGN OF A NAVIGATION METHODOLOGY

Description	Unit Prices
Planification of the activities to carry out in order to solve the problem	40€/hour
Development of path planning and obstacle avoidance trajectory based on the A* Algorithm	40€/hour
Typing and text editing	10€/day
Required Electricity	0.3€/day
Computing system wear and tear	1€/day

1.3. SIMULATION OF THE METHODOLOGY IN TWO DIFFERENT ENVIRONMENTS

Description	Unit Prices
Consulting experts of the industrial setting	40€/hour
Constructing the required CAD files	40€/hour
Bibliography study on S-Functions and Simscape Multibody	40€/hour
Planification of the activities to carry out in order to solve the problem	40€/hour
Designing the simulation	40€/hour
Implementation and experimental testing	40€/hour
Calculating time and presenting results	40€/hour
Typing and text editing	10€/day
Required Electricity	0.3€/day
Computing system wear and tear	1€/day

Development and programming of algorithms for automatic collision avoidance.
Application to terrestrial mobile robots

1.4. IMPLEMENTATION IN THE EV3

Description	Unit Prices
Learning/training on RobotC and the EV3 Library	40€/hour
EV3 Robot	349.18€/10 years
Netgear Wna1100 Adaptador	38.33€/year
Implementation	40€/hour
Typing and text editing	10€/day
Required Electricity	0.3€/day
Computing system wear and tear	1€/day

1.5 ADDITIONAL TASKS AND ACTIVITIES

Description	Unit Prices
Developing technical documentation and reporting	40€/hour
Typing and text editing	10€/day
Electricity required	0.3€/day
Computing system wear and tear	1€/day

1.6 OTHER CONCEPTS

Description	Unit Prices
Microsoft Office 2016 and Matlab	18€/month
General office items	45 €
Book printing and binding	50 €

CHAPTER 2: MEASUREMENTS

2.1 ANALYSIS OF THE OBSTACLE AVOIDANCE TECHNIQUES

Description	Measurement
Robot Manipulation and obstacle avoidance Bibliography review	10 hours
Planification of the activities to carry out in order to solve the problem	8 hours
Typing and text editing	2 days
Required Electricity	2 days
Computing system wear and tear	2 days

2.2 DESIGN OF A NAVIGATION METHODOLOGY

Description	Measurement
Planification of the activities to carry out in order to solve the problem	6 hours
Development of path planning and obstacle avoidance trajectory based on the A* Algorithm	30 hours
Typing and text editing	4 days
Required Electricity	4 days
Computing system wear and tear	4 days

2.3 SIMULATION OF THE METHODOLOGY IN TWO DIFFERENT ENVIRONMENTS

Description	Measurement
Consulting experts of the industrial setting	20 hours
Constructing the required CAD files	4 hours
Bibliography study on S-Functions and Simscape Multibody	15 hours
Planification of the activities to carry out in order to solve the problem	8 hours
Designing the simulation	8 hours
Implementation and experimental testing	70 hours
Calculating time and presenting results	20 hours
Typing and text editing	5 days
Required Electricity	5 days
Computing system wear and tear	5 days



2.4 IMPLEMENTATION IN THE EV3

Description	Measurement
Learning/training on RobotC and the EV3 Library	12 hours
EV3 Robot	1 unit/10 years
Netgear Wna1100 Adaptador	1 unit
Implementation	80 hours
Typing and text editing	30 days
Required Electricity	30 days
Computing system wear and tear	30 days

2.5 ADDITIONAL TASKS AND ACTIVITIES

Description	Measurement
Developing technical documentation and reporting	30 hours
Typing and text editing	7 days
Electricity required	7 days
Computing system wear and tear	7 days

2.6 OTHER CONCEPTS

Description	Measurement
Microsoft Office 2016 and Matlab	6 months
General office items	1 unit
Book printing and binding	1 unit

CHAPTER 3: DETAILED BUDGET

3.1 ANALYSIS OF THE OBSTACLE AVOIDANCE TECHNIQUES

Description	Unit Prices	Measurement	Cost
Robot Manipulation and obstacle avoidance Bibliography review	40€/hour	10 hours	400€
Planification of the activities to carry out in order to solve the problem	40€/hour	8 hours	320€
Typing and text editing	10€/day	2 days	20€
Required Electricity	0.3€/day	2 days	0.6€
Computing system wear and tear	1€/day	2 days	2€

3.2 DESIGN OF A NAVIGATION METHODOLOGY

Description	Unit Prices	Measurement	Cost
Planification of the activities to carry out in order to solve the problem	40€/hour	6 hours	240€
Development of path planning and obstacle avoidance trajectory based on the A* Algorithm	40€/hour	30 hours	1200€
Typing and text editing	10€/day	4 days	40€
Required Electricity	0.3€/day	4 days	1,2€
Computing system wear and tear	1€/day	4 days	4€

3.3 SIMULATION OF THE METHODOLOGY IN TWO DIFFERENT ENVIRONMENTS

Description	Unit Prices	Measurement	Cost
Consulting experts of the industrial setting	40€/hour	20 hours	800€
Constructing the required CAD files	40€/hour	4 hours	160€
Bibliography study on S-Functions and Simscape Multibody	40€/hour	15 hours	600€
Planification of the activities to carry out in order to solve the problem	40€/hour	8 hours	320€
Designing the simulation	40€/hour	8 hours	320€
Implementation and experimental testing	40€/hour	70 hours	2800€
Calculating time and presenting results	40€/hour	20 hours	800€
Typing and text editing	10€/day	5 days	50€
Required Electricity	0.3€/day	5 days	1.5€
Computing system wear and tear	1€/day	5 days	5€

3.4 IMPLEMENTATION IN THE EV3

Description	Unit Prices	Measurement	Cost
Learning/training on RobotC and the EV3 Library	40€/hour	12 hours	480€
EV3 Robot	349.18€/10 years	1 unit/10 years	34.918€
Netgear Wna1100 Adaptador	38.33€/year	1 unit/1 year	38.33€
Implementation	40€/hour	80 hours	3200€
Typing and text editing	10€/hour	30 days	300€
Required Electricity	1€/km	30 days	300€
Computing system wear and tear	1€/day	30 days	9€

3.5 ADDITIONAL TASKS AND ACTIVITIES

Description	Unit Prices	Measurement	Cost
Developing technical documentation and reporting	40€/hour	30 hours	1200€
Typing and text editing	10€/day	7 days	70€
Electricity required	0.3€/day	7 days	2.1€
Wear and tear of computing system	1€/day	7 days	7€

3.6 .OTHER CONCEPTS

Description	Unit Prices	Measurement	Cost
Microsoft Office 2016 and Matlab	18€/month	6 months	108€
General office items	45 €	1 unit	45€
Book printing and binding	50 €	1 unit	50€



CHAPTER 4: TOTAL BUDJET OF THE PROJECT

Description	Cost
Obstacle Avoidance Technique Analysis	742.6 €
Design of the A*Algorithm	1485.2 €
Simulation Designs	5856.5 €
Implementation on the EV3	4362.25 €
Tasks and additional activities	1279.1 €
Other Concepts	203 €
Project total budget without VAT	13 928.65€
Tax Value added (21%)	2925.02 €
Project total budget with VAT	16 853.66 €

The total budget of the project is 16 853.66 €.