Politecnico di Torino

Master of Science Degree in Mechatronic Engineering

Master Thesis

Multi-body code and software control co-simulation for the dynamic behaviour analysis of rover exploration vehicle

Supervisors
Prof. Marcello Chiaberge
Ing. Gerlando Augello
Ing. Andrea Merlo

Student
Alessandra Barbero

A.A. 2017/2018

To my mum.

Contents

Abstract 1

1 INTRODUCTION **1**

1.1 Motivation 1

1.2 Objective of the Thesis 2

1.3 Organisation of the Thesis 3

2 THE EXPLORATION OF MARS **5**

2.1 General Overview 5

2.2 The ExoMars Mission 6

2.2.1 The ExoMars Rover 8

3 MOBILE ROBOTS **9**

3.1 Introduction 9

3.2 Locomotion 10

3.2.1 Wheel types 11

3.3 Mobile Robot Kinematics 12

3.3.1 Unicycle 14

3.3.2 Differential drive mobile robots 16

3.3.3 Car-like mobile robots 18

3.3.4 Inverse Kinematic problem 21

3.4 Navigation, Supervision and Control 23

4 ACTIVATE MODEL OVERVIEW **25**

4.1 Introduction 25

4.2 Co-simulation with Multi-Body systems 26

4.3 The Plant 28

4.4 The Controller 29

4.5 The Locomotion Control 30

4.6 Simulation Setting 30

4.6.1 Numerical ODE solver classifications and characteristics 30

4.6.2 Maximum, minimum, and initial step size 32

5 MOTIONSOLVE: EXOMARS ROVER MULTI-BODY MODEL 33
5.1 Introduction . . . 33
5.2 Multibody System Simulation . . . 33
5.3 MotionView - MotionSolve . . . 35
5.4 Rover vehicle system mechanical design . . . 36
5.4.1 Deployed configuration . . . 37
5.5 Building Up the Model with MotionView . . . 38
5.5.1 The *Locomotion* Subsystem . . . 39
5.5.2 Rover body subsystem . . . 53
5.6 Modifying the model for co-simulation . . . 56

6 THE ACTUATOR CONTROL 59
6.1 Introduction . . . 59
6.2 ExoMars Rover Vehicle Mobility System Overview . . . 59
6.3 The ExoMars Actuator Drive Electronics . . . 62
6.3.1 ExoMars Actuator Control Algorithm Overview . . . 63
6.4 The Controller . . . 66
6.4.1 PID Control Theory . . . 66
6.4.2 PID tuning operation . . . 72
6.5 Some clarifications . . . 72
6.5.1 Drive motors PID tuning . . . 73
6.5.2 Steering wheel motor PID tuning . . . 75
6.6 The Path planner . . . 77
6.6.1 Trapezoidal Velocity Profile . . . 77

7 THE INVERSE KINEMATICS 88
7.1 Introduction . . . 88
7.2 Locomotion capability of ExoMars rover . . . 89
7.3 Rover Inverse Kinematics . . . 90
7.3.1 Conventional Ackerman Kinematic Model . . . 91
7.3.2 Conventional Point Turn Kinematic Model . . . 93
7.3.3 Crab Kinematic Model . . . 95
7.4 Algorithm implementation in Activate . . . 96
7.4.1 Ackerman manoeuvre . . . 97
7.4.2 Point Turn and Crab manoeuvre . . . 98

8 MODEL VALIDATION 100
8.1 Real simulation of IVBB rover . . . 101
8.1.1 Locomotion Software GUI . . . 102
8.2 Model adaptation . . . 106

8.2.1 Modified Ackerman Kinematic Model 107
8.3 IVBB simulation results 108
8.3.1 Manoeuvre1 - Ackerman 108
8.3.2 Manoeuvre2 - Point turn 112
8.3.3 Manoeuvre3 - Crab 116
8.3.4 Manoeuvre4 - Straight forward motion 120

9 CONCLUSIONS **124**

Acknowledgements 128

List of Figures

1.1 The ExoMars Rover designed by the European Space Agency belonging to the ExoMars Mission 2020 2

3.1 Bipedal walking system 10
3.2 Four basic wheel types 11
3.3 Wheel configurations for rolling vehicles 12
3.4 Mobile robot reference frame representation 13
3.5 Differential drive robot model 15
3.6 Differential drive robot model 17
3.7 Bicycle model of a car 18
3.8 General control scheme of a mobile robot 23
3.9 Low-level and high-level controls of a mobile robot 24

4.1 Co-simulation between Activate and MotionSolve 26
4.2 General scheme of the complete model built for co-simulation 27
4.3 Complete multi-body model built with MotionView 28

5.1 Interaction between MBS and other technologies. 35
5.2 Rover module coordinate system 37
5.3 Deployed rover configuration 38
5.4 Real locomotion system of ExoMars rover that has to be modelled in MotionView *Locomotion* subsystem 41
5.5 ExoMars Rover BEMA wheels 42
5.6 MotionView *Wheels* subsystem 43
5.7 MotionView *Bogie* subsystem 44
5.8 MotionView panel of the joint between front left wheel and corresponding leg 45
5.9 MotionView *Locomotion* joints overview 45
5.10 IMPACT model 47
5.11 Contact panel of the wheel-ground contact with impact model parameters 48
5.12 Contact panel of the wheel-ground contact with friction coefficients . . . 49

5.13 Simulation result highliting the wrong tangential components F_t of contact forces 49
5.14 Comparison between different mesh refinement levels 50
5.15 Simulation results highlighting friction force and its components during three different intervals 51
5.16 ExoMars Rover BEMA actuator 52
5.17 MotionView force/torque panel 52
5.18 MotionView *Locomotion* force/torque entities overview 53
5.19 Complete simplified rover model build in MotionView 54
5.20 Differnt views of the complete rover model build in MotionView with real body geometry 55
5.21 Steps of the model construnction in MotionView 58

6.1 Overview of the mobility system functional architecture 60
6.2 Actuator's algorithm overview 63
6.3 Path planner behaviour 64
6.4 ExoMars Controller block diagram 64
6.5 General scheme of the complete model built for co-simulation 66
6.6 Block scheme of a control loop with proportional-integral-derivative action 67
6.7 Block scheme of a control loop with proportional action 68
6.8 Output DRV rotational speeds resulting from a P control 74
6.9 Output DRV rotational speeds resulting from a PI control 74
6.10 Output STR angular positions resulting from a P control 75
6.11 Output STR angular positions resulting from a PI control 76
6.12 Example of a continuous trajectory with velocity and acceleration constraints 77
6.13 Example of a continuous linear trajectory 79
6.14 Example of a continuous parabolic trajectory 80
6.15 Example of a continuous trapezoidal velocity trajectory 82
6.16 Complete Activate model 84
6.17 General Activate model of the *Profile Generator* 84
6.18 Activate implementation of the *Profile Generator* algorithm 86
6.19 *Profile Generator* output esample 87

7.1 General cycle of the ExoMars mobility control 89
7.2 Kinematic model for Ackerman manoeuvre 91
7.3 Kinematic model for Point Turn manoeuvre 94
7.4 Kinematic model for Crab manoeuvre 95
7.5 *Inverse Kinematics* block overview 96

7.6 Ackerman inverse kinematic algorithm implementation in Activate. . . . 98
7.7 Description of how R_i and θ_i are computed 98
7.8 Point Turn inverse kinematic algorithm implementation in Activate. . . 99
7.9 Crab inverse kinematic algorithm implementation in Activate. 99

8.1 IVBB Locomotion Software GUI 102
8.2 Overview of the control modes and their internal states 104
8.3 Modified kinematic model for Ackerman manoeuvre when using IVBB . 107
8.4 Activate block scheme of the modified *Inverse Kinematics* block 107
8.5 M1 Ackerman manoeuvre: wheel steering positions 109
8.6 M1 Ackerman manoeuvre: rover CM x, y position time evolution and xy trajectory 110
8.7 M1 Ackerman manoeuvre: wheel rotational velocities 111
8.8 M2 Point Turn: wheel steering positions 113
8.9 M2 Point Turn manoeuvre: rover CM x, y position time evolution and xy trajectory 114
8.10 M2 Point turn manoeuvre: wheel rotational velocities 115
8.11 M2 Point turn manoeuvre: rover heading 115
8.12 M3 Crab manoeuvre: wheel steering positions 117
8.13 M3 Crab manoeuvre: rover CM x, y position time evolution and xy trajectory 118
8.14 M3 Crab manoeuvre: wheel rotational velocities 119
8.15 M4 Straight Forward motion: wheel steering positions 121
8.16 M4 Straight Forward motion: rover CM x, y position time evolution and xy trajectory 122
8.17 M4 Straight Forward motion: wheel rotational velocities 123

Abstract

Since antiquity, humans have dreamed about spaceflight. When rockets became powerful enough to overcome the force of gravity and reach orbital velocities, space was opened to human exploration. Mars has always captured humankind imagination, being a source of inspiration for explorers and scientists. A considerable number of missions have attempted to reach the Red Planet, with varying degrees of success, focused primarily on understanding the planet geology and habitability potential. In the context of planetary surface exploration, mobile robots are the main characters, due to their capability of moving in unstructured environments. *ExoMars 2020* is the mission managed by the *European Space Agency* (ESA) intended to deliver a rover on Mars in search of a sign of life. The mission will send the European ExoMars rover and a Russian surface platform to the surface of Mars with the aim of performing scientific research looking for organic molecules and surface characterisation.

When dealing with mobile robot design, multi-body software are fundamental tools supporting engineering activities. *Multi-body System Simulation* (MBS) is a numerical simulation method for the study of mechanical systems motion caused by external forces acting on it.

One challenging activity of the Mechanical CAE (Computer Aided Engineering) group of *Thales Alenia Space Italia S.p.A.*, where this thesis has been developed, addresses exactly at building a dynamic model of the six wheeled planetary rover belonging to ExoMars Mission. Moreover, the CAE group requires to drive the MBS model in order to study the rover when performing different manoeuvres. This can be achieved by controlling torques applied to drive and steering wheel axes.

The model is built in *MotionView* environment, a general pre-processor for MBS simulation, and simulated through *MontionSolve* solver. These tools guarantee optimal performances when dealing with MBS simulation, but they are not enough to implement complex controller. In order to incorporate functions of sensing, actuation and control the *Activate* software tool is used. Thanks to its co-simulation interface it is possible to simulate this complex system including both the rover MBS model, simulated with MotionSolve, and the control systems, simulated with Activate.

The main objective of this thesis is to develop the Activate model for co-simulation, implementing the low level control loop of the ExoMars rover wheel actuators (DC motors) and the higher level locomotion function, which allow driving the rover given the manoeuvre commands set by the user, rather than acting on the actuator speeds and position.

Chapter 1

INTRODUCTION

1.1 Motivation

> "Back in the days of Apollo, sending humans to the moon was the only viable way to get the scientific data we wanted. But now, with our computer and robotics technology, there's very little an astronaut can do on Mars that a well-designed rover can't."
>
> Andy Weir

Over the course of centuries, human beings have constantly attempted to seek substitutes that would be able to mimic their behaviour. The term *robot* appeared for the first time in a novel written by Karel Capek in 1921. [13] The fantasy associated with robotics offered by science, movies and printed and animated cartoons is so far from reality. Actual industrial robots seem primitive compared with what many people expect a robot to be. No actual robot could compete with C3P0, R2-D2 and BB8 (from the movie *Star Wars*) or Sonny (from the movie *I Robot*). The only sure thing is that robotics will be an important technology in this century.

One of the most amazing areas of robotics is the use of robots in space. These machines give the possibility to explore space doing lots of things that astronauts can't or that are too risky for them; for instance, some can withstand harsh conditions, like extreme temperatures or high levels of radiation. [22]

All the mind-boggling qualities of planetary rovers are achieved starting from a design activity planned in the last details. Rover mobility is generally a primary objective during the design phase, since it could prevent the goal achievement; moreover, operating on soft-soil and in unstructured environments like the planet surface makes the rover goal extremely challenging.

From this perspective, software tools are used to support engineering activities. The general approach to rover design and study involves the modelling of the multi-body

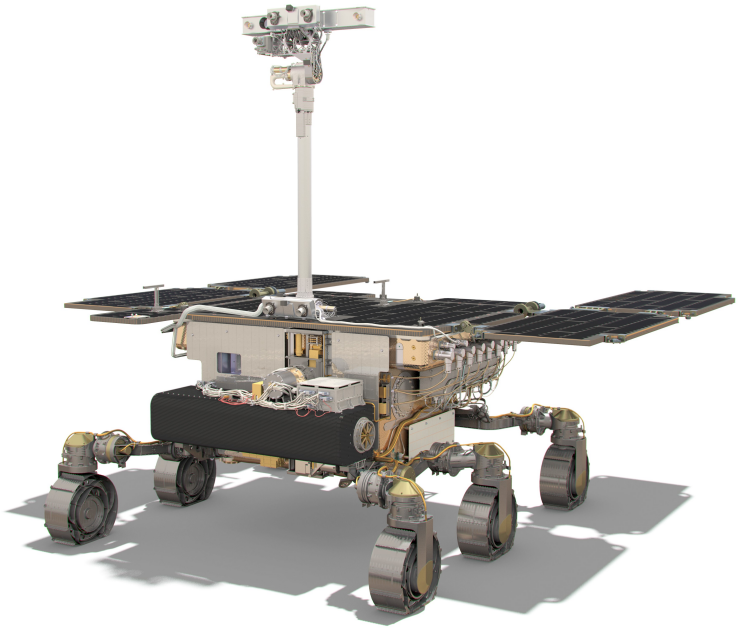


Figure 1.1: The ExoMars Rover designed by the European Space Agency belonging to the ExoMars Mission 2020

model with dedicated software like MCS Adams, SIMPACK, Mathworks SimMechanics and a probably lesser-known software, MotionView, being part of the Altair software-house. This last is starting to be used in some among the major space project, like the *ExoMars* mission managed by the *European Space Agency*. The possibility of performing co-simulation between multi-body software and software for the implementation of control systems, get researcher interested in studying the rover stresses when performing different manoeuvres.

1.2 Objective of the Thesis

The main objective of this thesis is to develop and implement a model of the ExoMars Rover control system. This will be done in Activate environment which provides the capability of performing co-simulation with MotionView/MotionSolve multi-body modelling software; both software tools are provided by Altair software-house. The controller is responsible of drive and steer the rover wheels in a way that it follows specific trajectory, achieved performing one of the following manoeuvres: Ackerman steering, point turn and crab.

The model provides the possibility to the user to choose the desired manoeuvre and set the corresponding rover level commands. Therefore, one has to take into account the transformation of the manoeuvre commands from rover level to wheel level, which

are the reference signal for the controllers. This operation requires the solution of the Inverse Kinematic problem.

In the final analysis, this work comprises the building of the multi-body model, the implementation of the controller model, the joining of these two and the customization of the model in a way that the user can easily set the input without knowing the model characteristics, i.e. without considering the kinematic model of the rover.

1.3 Organisation of the Thesis

The thesis is composed of eight chapters, each of them dealing with different aspect of ExoMars rover co-simulation process; it is organized as follows.

Chapter 1 is introductory and outlines the motivations that stimulate scientists and get them interested in studying planetary rovers. Afterwards, the main objectives of this work are illustrated, and finally a brief description of the thesis structure is given.

Chapter 2 provides an overview of the space exploration through the years, particularly focusing on the advantages of this activity. In the context of Mars exploration the *ExoMars* mission managed by the European Space Agency is then presented, and the planetary rover involved in it is described.

Chapter 3 offers an outline of Mobile Robotic Science. After a rough introduction, particular emphasis is placed on the rover kinematics topic that will be applied later to the ExoMars rover model.

Chapter 4 aims to introduce the main software employed during the thesis development. Moreover, it introduces the model obtained through this work, describing in broad outlines the constitutive blocks. These will be presented and described in details in subsequent chapters.

The following three chapters have similar structure; each of them assesses one particular block constituting the complete model, firstly presenting the main characteristics of the real system being part of the ExoMars rover, then moving to see how these were modelled in the different software.

The Chapter 5 consists of three parts. The first part focuses on the software used to build the multi-body model. The second part examines the real ExoMars rover, with specific regard to its mechanical structure. Finally, the third part addresses the issue of model realization, explaining the methodology used in this thesis.

The ExoMars rover actuator control is the subject of Chapter 6. Therefore, a brief introduction about control theory opens the chapter, followed by the presentation of the real system. The goal of this chapter is to propose a model for both low level control loop and profile generator.

Chapter 7 deals with the remaining component, that is the inverse kinematics block. This one is a major topic since makes the model easily handled, allowing the user

to drive the rover simulation in several ways ignoring the model characteristics, thus without solving any equation.

In Chapter 8 a real system is analysed. Using IVBB rover as the case study, its behaviour is compared with the one resulting from co-simulation. Firstly, a general overview of the rover mechanical structure and mobility capability is provided; then the procedure followed in performing the test on the real system is described and the results coming from it are reported. The final part of the chapter is intended to highlight the similarities between IVBB results and the one obtained through Activate-MotionSolve co-simulation.

Conclusions are drawn in Chapter 9. The main purpose of the thesis has been reached. The developed Activate model allows to simulate and analyse rover structures when performing different manoeuvres. Moreover, this chapter outlines the major advantages and possible application of the work. Finally, it proposes ideas for further studies and future works.

Chapter 2

THE EXPLORATION OF MARS

2.1 General Overview

> "It's human nature to stretch, to go, to see, to understand. Exploration is not a choice, really; it's an imperative."
>
> Michael Collins

Space exploration helps facing significant scientific and philosophic issue about our place in the universe and the presence of life besides Earth. Moreover, facing to the challenging questions encourages the development of new technologies and leads to international collaboration.

For ages, humans have dreamed to leave Earth and travelling in search of new worlds. The first step was accomplished by Soviet Union in 1957, when they launched the first satellite: *Sputnik*. It was the first human-made object into space. The space age began. Lots of plans were made to bring men in space, till Yuri Gagarin, in 1961 took the first orbital spaceflight around Earth. From this moment more than 500 cosmonauts had travelled the space. Later on July 20, 1969, American astronaut Neil Armstrong took "a giant leap for mankind"
being the first man stepping onto the moon; until now twelve astronauts have walked on the Moon surface. [6]

Even though Moon exploration stopped in 1972, human imaginary remained still alive. A considerable number of spacecraft have been launched towards all planets of the Solar System to gather useful informations for future missions realization. In particular, a lot of plans have been made for launching astronauts to explore Mars. The *Red Planet* has always captured the imagination of humankind, sparking interest in scientists and inspiring artists.

In the 19th century the Italian astronomer Giovanni Schiaparelli observed the planet surface and find out bright and dark straight-line features that he called *canals*. This contributed to new thinking of associating Mars with life. For a long time, it was popularly believed that these canals had been built by intelligent beings to form a huge irrigation network on the Red Planet. However, the myth of the Martians vanished during the early part of the 20th century, when better telescopes provided scientists a clearer view of the planet surface.

Later that century, the start of the space age brought about a change in how the search for extraterrestrial life was carried out. Scientists no longer look for intelligent beings, but for evidence for the presence of water, an essential element for the formation of life, either on the surface or hidden underground. [16]

In the mid 20th century, Joshua Lederberg, an American geneticist Nobel Prize winner in Medicine coined a new term: *exobiology*, also referred to as bio-astronomy or astrobiology. It describes the study of the existence of life outside the Earth and outlines the risk of bio-contamination related to space flights that might, in the future, contaminate and ruin bacterial ecosystems in outer space and on other planets.

The main purpose of Exobiology is to answer the questions regarding the origin, evolution and distribution of life in the Universe [29]. In the second half of the 20th century, when Space Agencies worked towards the development of new space technologies and encouraged the planning of new scientific programmes, exobiology officially became a science.

In this scenario, Mars remained a target planet where exobiologists hope to find signs of primitive life. This is due to the fact that, according to current scientific knowledge, it is thought Mars to have (now or even in the past) environmental conditions (moderate temperatures and liquid water) similar to those that probably have lead to the development of life on Earth. Whit this goal several missions have been organised, and in particular we will address to the ExoMars one.

2.2 The ExoMars Mission

With the purpose of exploring Mars and providing a continuous flow of scientific information, the European Space Agency ESA intended to deliver a European rover and a Russian surface platform to the surface of Mars. The missions, named *ExoMars*, will search for traces of past and present life, characterize the Mars geochemistry and water distribution as a function of subsurface depth, improve the knowledge of the Martian atmospheric environment and geophysics, and identify possible surface hazards to future human exploration missions. The ExoMars program includes two missions utilising the 2016 and 2020 launch opportunities to Mars, both under ESA leadership [19].

The first mission of the ExoMars programme consisted of launching a Proton M

rocket [1] carrying the Trace Gas Orbiter (TGO) and an Entry, Descent and Landing (EDL) demonstrator module, known as Schiaparelli. The main objectives of this mission were to search for evidence of some trace atmospheric gases (e.g. methane) that could denote the presence of active biological or geological processes. Moreover, ESA could test key technologies in preparation to subsequent missions to Mars.

The launch took place on 14 March 2016 and, thanks to the favourable positioning of Earth and Mars, the journey lasted about 7 months, with the arriving at Mars in October. Three days before reaching the atmosphere of Mars, Schiaparelli was ejected from the Orbiter towards the Red Planet. It coasted towards its destination, entered the Martian atmosphere and decelerated using aerobraking and a parachute. The established plain had foreseen to brake with the aid of a thruster system before landing on the Mars surface, but due to an anomaly in the measurement and navigation system the module crashed, while landing on the surface of the planet.

By contrast, the TGO was inserted into an elliptical orbit around Mars and later it performed a number of manoeuvres in order to shift its angle of travel. This action provided optimum coverage of the surface for the science instruments, while still offering good visibility for relaying data from current and future landers. Moreover, it will allow communication from the Rover Operations Control Centre (ROCC) to the future ExoMars rover.

Indeed, the second mission belonging to ExoMars program will take place in 2020. It consists in the launch of a Proton rocket carrying a rover, a surface platform and an Entry, Descend and Landing module. The system is expected to arrive to Mars after a nine-month journey. The rover will carry a comprehensive suite of analytical instruments dedicated to exobiology and geological research, the so-called Pasteur Payload. It is planned to let the rover 'live' and work on Mars for about 6 month, during which it will ensure a regional mobility of several kilometres, searching for traces of past and present life. It will do this by collecting samples with a drill from within surface rocks, and from underground, to a depth of 2 meters, and analysing them in the analytical laboratory [18].

The mission plain is the following: during launch and cruise phase, a single aeroshell will transport the surface platform and the rover; the EDL module will separate from the carrier shortly before reaching the Martian atmosphere. Finally, the landing on the surface of Mars will be controlled by parachutes, thrusters, and damping systems responsible of reducing the speed. Once the system has landed, the rover will egress from the platform to start its science mission, travelling several kilometres.

A dedicated control centre, Rover Operation Control Center ROCC, located in Turin, will monitor and control the ExoMars rover operations taking advantages of the

[1] The Proton-M is a heavy-lift launch vehicle provided by Roscosmos, the Russian State Space Corporation

previously launched TGO (2016 mission).

The strong relevance of ExoMars mission consists in the fact that it will be the first one combining the capability to move across the surface and to study Mars at depth, accessing locations where organic molecules might be well preserved. [29]

2.2.1 The ExoMars Rover

As expressed before, the ExoMars mission will include an exploration rover providing key mission capabilities: surface mobility, subsurface drilling and automatic sample collection, processing, and distribution to instruments.

The Rover uses solar panels to generate the required electrical power, and is designed to survive the cold Martian nights with the help of novel batteries and heater units. The ExoMars rover is highly autonomous since the comunications opportunities are very infrequent, only 1 or 2 short sessions per sol (Martian day).

Thus the cameras mounted on the rover mast will acquire stereo images on whose basis scientists on Earth will designate target destinations. As a consequence, the Rover mobility manager will calculate navigation solutions and safely travel (approximately 100 m per sol). To achieve this, it creates digital maps from navigation stereo cameras and computes a suitable trajectory. Moreover, it takes care of ensuring safety using close-up collision avoidance cameras.

"The locomotion is achieved through six wheels. Each wheel pair is suspended on an independently pivoted bogie (the articulated assembly holding the wheel drives), and each wheel can be independently steered and driven. All wheels can be individually pivoted to adjust the Rover height and angle with respect to the local surface, and to create a sort of walking ability, particularly useful in soft, non-cohesive soils like dunes. In addition, inclinometers and gyroscopes are used to enhance the motion control robustness. Finally, Sun sensors are utilised to determine the Rover's absolute attitude on the Martian surface and the direction to Earth." [17]

The goal of finding traces of past biological activity requires to investigate specific areas of Mrs surface, thus the suitable drilling location will be identified by scientists on-ground taking advantages of the camera system images and the data coming from the ground penetrating radar collected during the travel.

Then, the Rover subsurface sampling device will autonomously drill to the required depth (maximum 2 m) and will collect a small sample. This last will be delivered to the analytical laboratory in the heart of the vehicle where several analysis will be performed.

Chapter 3

MOBILE ROBOTS

3.1 Introduction

This chapter presents an overview of the main characteristic of mobile robots and provide a kinematic model of the most popular rover configurations. The main concept have been taken from [24], [13] and [21].

Robotics has achieved its first greatest success in the industrial manufacturing world. Starting from '70s robot arms, or *manipulators*, started to appear with the purpose of helping humans in simple and repetitive or dangerous tasks. They can move with great speed and accuracy, but suffer from a fundamental disadvantage: lack of mobility. Fixed manipulators have limited range of motion depending on their configuration. By contrast, a *mobile robot* is a structure capable to move and act, autonomously or remotely operated, in the surrounding environment, which could be structured, partially structured or unstructured. An environment is called *structured* when one knows its type and the geometric characteristics.

The fundamental problems in mobile robotics are:

Locomotion how the robot moves in the environment

Perception how the robot perceives the environment

Representation how the robot organize the knowledge about the environment

Mapping how the robot built the map of the environment

Localization where the robot is in the map

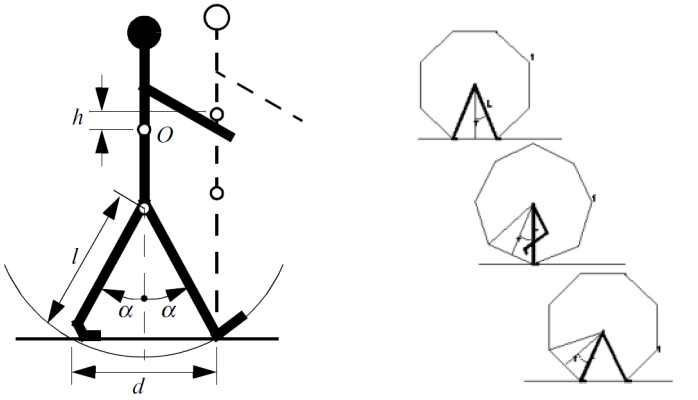
Path planning/action planning what the robot shall do to go from a point to an other; what are the actions to be performed to complete a specific task

Supervision and control how the commands to actuators are generated to perform simple or complex task

These topics together constitute the *mobility* problem.

3.2 Locomotion

A mobile robot needs locomotion mechanisms that enable it to move throughout the environment. Since there are a large variety of possible ways to move, the selection of the locomotion mechanism is an important aspect when designing mobile robots. Most of the these mechanisms have been inspired by the nature. For example, our bipedal walking system can be approximated by a rolling polygon, with sides equal in length d to the span. As the step size decreases, the polygon approaches a circle or wheel with the radius l, as shown in Figure 3.1; *Walking robots* are capable to adapt to variable environments, but they are low on flat terrain and extremely over-actuated.



(a) Bipedal walking system

(b) Rolling polygon approximation of the bipedal walking system

Figure 3.1: Bipedal walking system

The majority of mobile robots are *wheeled robots*, that exploits the fully rotating, actively powered joint technology. The wheel has been by far the most popular locomotion mechanism in mobile robotics since it allows achieves extremely high efficiency on flat ground with a relatively simple mechanical implementation. Moreover, is not usually needed to take care of balance problem because wheeled robots are almost always designed so that all wheels are in ground contact at all times. Thus, three wheels are sufficient to guarantee stable balance, although, two-wheeled robots can also be stable. When more than three wheels are used, a suspension system is required to allow all wheels to maintain contact with ground when the robot encounters uneven terrain. [24] Thus, traction and stability, manoeuvrability, and control are the main focus of wheeled robot research. One should study the robot wheels in order to find out if it provides sufficient traction and stability for the robot to cover all of the desired terrain,

and if the configuration enable sufficient control over the velocity of the robot.

3.2.1 Wheel types

There are four major wheel classes which differ widely in their kinematics. Therefore the overall kinematics of the mobile robot is highly affected by the choice of wheel type.

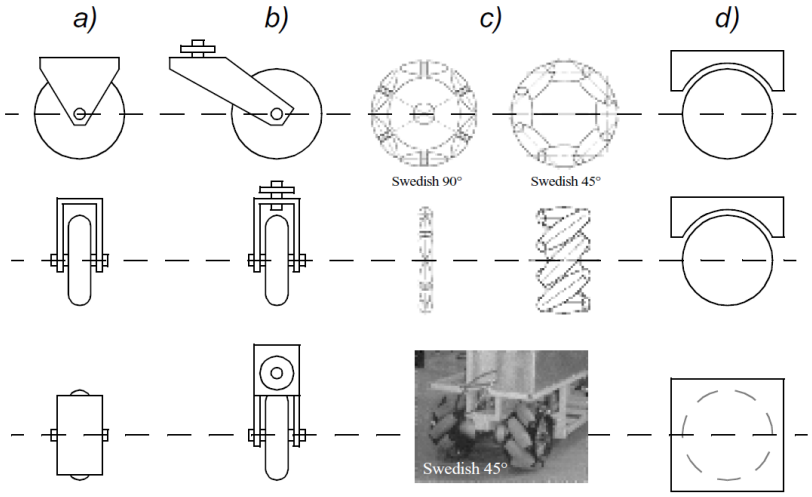


Figure 3.2: Four basic wheel types: (a) Standard wheel; (b) castor wheel; (c) Swedish wheel; (d) Ball or spherical wheel.

The *standard* wheel and the *castor* wheel are highly directional since they are characterised by a primary axis of rotation constraining the wheel to move in a well defined direction. If the wheel have to be moved in a different direction, it must be steered first along a vertical axis. These two wheels differ between each other in the way of accomplishing this steering motion. When steering standard wheels no side effects arise, because the center of rotation passes through the contact point with the ground; whereas, the castor wheel rotates around an offset axis, causing a force to be imparted to the robot chassis during steering.

The designs of the *Swedish* wheel and the *spherical* wheel make them less constrained by directionality than the conventional standard wheel. The Swedish wheel functions as a normal wheel, but provides low resistance in another direction as well. It has small rollers attached around its circumference which are passive; the only actively powered joint remains the wheel primary axis. The key advantage of this design is that the wheel can kinematically move with very little friction along many possible directions, not just forward and backward, even if the wheel rotation is powered only along the one principal axis.

The spherical wheel is a truly omnidirectional wheel, often designed so that it may be actively powered to spin along any direction. One mechanism for implementing this spherical design imitates the computer mouse, providing actively powered rollers that

rest against the top surface of the sphere and impart rotational force.

Regardless of the used wheel type, in robots designed for all-terrain environments and in robots with more than three wheels, the contact with ground is normally maintained by a suspension system, that in the simplest approaches is designed as a flexibility into the wheel itself [24].

3.3 Mobile Robot Kinematics

Kinematics is the most basic study of how mechanical systems behave, since it does not consider the mass of the system or the forces that cause the motion. In mobile robotics, we need to understand the mechanical behaviour of the robot in order to appropriate design them for accomplish the desired tasks and to understand how to control their operations.

The understanding of robot motions starts with the description of the contribution of each wheel provided for it. Each wheel has a role in enabling the whole robot to move, but it also imposes constraints. Therefore, robot kinematics depends on the wheel types and on the adopted wheel arrangement, or wheel geometry; for each arrangement a kinematic model exists. Some examples are reported in Figure 3.3

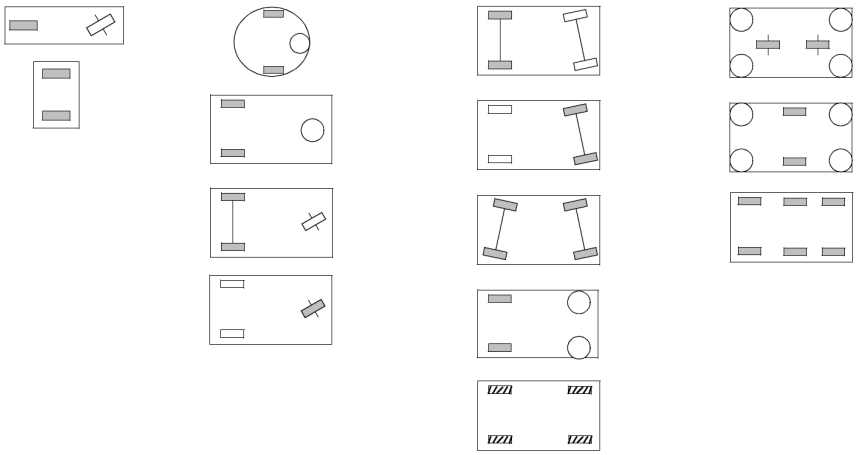


Figure 3.3: Wheel configurations for rolling vehicles

The assumption made in this thesis when modelling a robot is to consider it as a rigid body on standard wheels, operating on the horizontal *xy plane*.

The time evolution of vector $\boldsymbol{q}$, containing the *generalized coordinates* required to univocally identify the robot configuration, defines the motion of the robot and could be subject to holonomic and/or non-holonomic constraints. A *holonomic constraint* is an equation that can be written in terms of the position variables $\boldsymbol{q}$. Constraints in which time explicitly enters into the equation are called *rheonomic*, while constraints in which

time is not explicitly present are called *scleronomic*. A *non-holonomic constraint* can only be written in terms of the derivatives of the configuration variables $\dot{\boldsymbol{q}}$ and cannot be integrated to a constraint in terms of configuration variables. Mobile robots belong to the so-called *non-holonomic vehicles*, i.e. systems that cannot move directly from one configuration to another, but must perform a manoeuvre or sequence of motions. The non-holonomic constraint characterising the kinematic model of a wheeled robot is due to the *rolling without slipping* condition between the wheels and the terrain. The robot pose in the *xy plane* is described through a relationship between the global (*fixed*) reference frame $\mathcal{R}_0$ and the local (*mobile*) reference frame of the robot $\mathcal{R}_r$, as in Figure 3.4

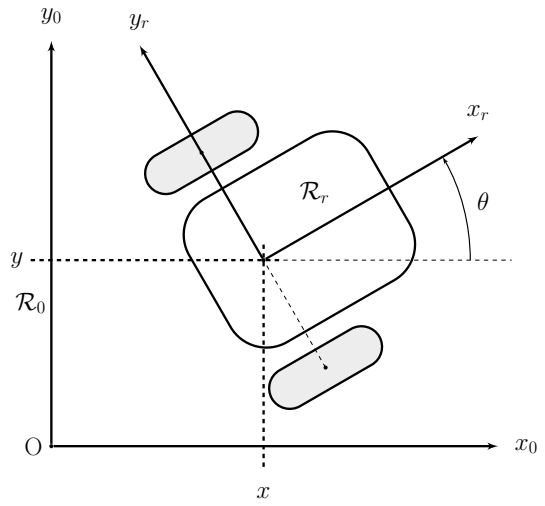


Figure 3.4: Mobile robot reference frame representation

The robot pose in the global reference frame is defined by coordinates x and y, and orientation angle θ, i.e. the rotation angle between the global and local reference frames. Thus, the pose of the robot in $\mathcal{R}_0$ is given by:

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{3.1}$$

To describe the robot motion we need rotation matrices. A *rotation matrix* R_b^a provides the description of the mobile reference frame $\mathcal{R}_b$ with respect to the fixed reference frame $\mathcal{R}_a$. A rotation matrix is also defined as the generic rotation of an angle θ around an axis represented by a unit vector $\boldsymbol{u}$.

$$R_b^a = Rot(\boldsymbol{u}, \theta)$$

In our case we need the rotation matrix R_r^0 that allows to map motion in the global reference frame to motion in terms of the local reference frame. It is defined as:

$$\boldsymbol{R}_r^0 = Rot(\boldsymbol{z}, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

Using rotation matrices it is possible to perform *transformation* of vectors from one reference frame to another. For example, given the velocities $\dot{\boldsymbol{q}}_0$ in the global RF it is possible to compute the components of motion $\dot{\boldsymbol{q}}_r$ in the rover RF:

$$\dot{\boldsymbol{q}}_r = Rot(\boldsymbol{z}, \theta)\dot{\boldsymbol{q}}_0 \tag{3.3}$$

In the simplest cases, the mapping described by this equation is sufficient to solve the *Forward Kinematics* problem of the mobile robot, i.e. how does the robot move given its geometry and the speeds of its wheels $\dot{\boldsymbol{q}}_0 = f(\dot{\boldsymbol{q}}_r)$.

For example, consider the robot shown in Figure 3.4: due to the particular robot configuration it is called *differential drive robot*. It has two wheels, each one with radius r_w, while the distance between the two wheel centers, called *wheel track* is d. Given r_w, d, θ and the rotational speed of the wheels ω_L and ω_R, a forward kinematic model allows to compute the robot overall speed in the global RF:

$$\dot{\boldsymbol{q}}_0 = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(r_w, d, \omega_L, \omega_R) \tag{3.4}$$

This means that we can compute the motion of the robot in the global reference frame given the motion in its mobile RF. Firstly, we have to compute the contribution of each wheel rotational speed ω_i to the robot CM motion in the local reference. Then, we apply the following relation:

$$\dot{\boldsymbol{q}}_0 = Rot(\boldsymbol{z}, \theta)^{-1}\dot{\boldsymbol{q}}_r \tag{3.5}$$

Thus, the Forward Kinematics can be considered a trivial problem, but this is not true for the reverse problem: *Inverse kinematics*, which is intended to find the wheel speeds that make the robot move as desired.

The following sections are devoted to introduce the kinematic model of the main mobile robot configurations.

3.3.1 Unicycle

We start considering the simplest case of a unicycle represented in Figure **??**. It consists of a single wheel that rolls on a plane (*xy plane*) while keeping its body vertical.

As stated before the robot configuration is described by vector $\boldsymbol{q}$ of the generalized coordinates, i.e. the position coordinates x and y of the point of contact with ground in

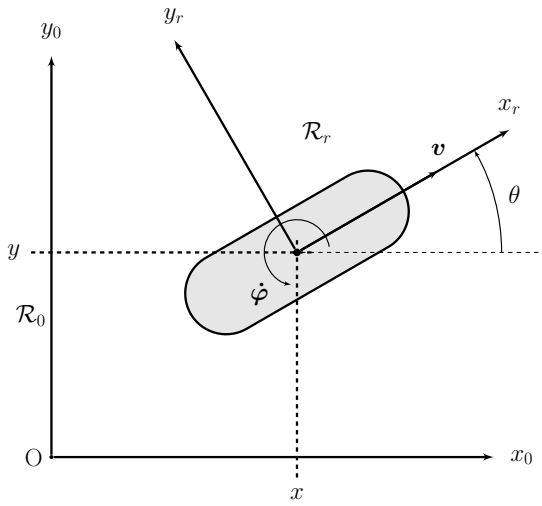


Figure 3.5: Differential drive robot model

the fixed reference frame, and the angle θ measuring the wheel orientation with respect to the global x *axis*.

A relationship exists between the wheel velocity components $\dot{x}$ and $\dot{y}$ [23]:

$$\begin{cases} \dot{x} = v_t \cos\theta + v_n \cos(\frac{\pi}{2} - \theta) \\ \dot{y} = v_t \sin\theta + v_n \sin(\frac{\pi}{2} - \theta) \end{cases} \tag{3.6}$$

However, to avoid slippage the tangential velocity v_t at the wheel contact point must be equal to the advance velocity of the wheel, and thus the lateral velocity v_n is null. The (3.6) becomes simply

$$\begin{cases} \dot{x} = v_t \cos\theta \\ \dot{y} = v_t \sin\theta \end{cases} \tag{3.7}$$

Therefore, the non-holonomic constraint is

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{3.8}$$

that is a typical example of a Pfaffian constraint. A *Pfaffian* constraint is a constraint written in the form

$$A(\boldsymbol{q})\dot{\boldsymbol{q}} = 0$$

. This type of constraint allows generating the allowed velocities from a matrix $G(\boldsymbol{q})$ defined as

$$Im(G(\boldsymbol{q})) = Ker(A(\boldsymbol{q}))$$

. The general formulation is

$$\dot{\boldsymbol{q}} = G(\boldsymbol{q})\boldsymbol{v} \tag{3.9}$$

which establish a relationship between the velocities $\boldsymbol{v}$ in the operational space (linear velocity and angular velocity around the vertical axis of the wheels) and velocities $\dot{\boldsymbol{q}}$ in the configuration space.

Therefore, (3.7) written in Pfaffian form is

$$\begin{bmatrix} \sin\theta & -\cos\theta & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{3.10}$$

and

$$Ker(A(\boldsymbol{q})) = span\left(\begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) = Im(G(\boldsymbol{q})) \tag{3.11}$$

Thus, all admissible generalized velocities can be define by a linear combination of two vectors

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v_1 + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} v_2 \tag{3.12}$$

where

- $v_1 = v$ is the linear velocity of the wheel, equal to the product between the wheel rotational speed $\dot{\omega}$ around its horizontal axis y and its radius r

$$v = r\dot{\omega}$$

- $v_2 = \dot{\varphi}$ is the angular velocity of the robot, i.e. the rotational velocity of the wheel around its vertical axis z

Acting on v and ω it is possible to change the robot configuration. The (3.12) is referred to as *Kinematic model* since it describes the velocities of the vehicle but not the forces or torques that cause the velocity.

3.3.2 Differential drive mobile robots

Another typical example of wheeled robot is the so-called differential drive robot, depicted in Figure 3.6.

We define [12]:

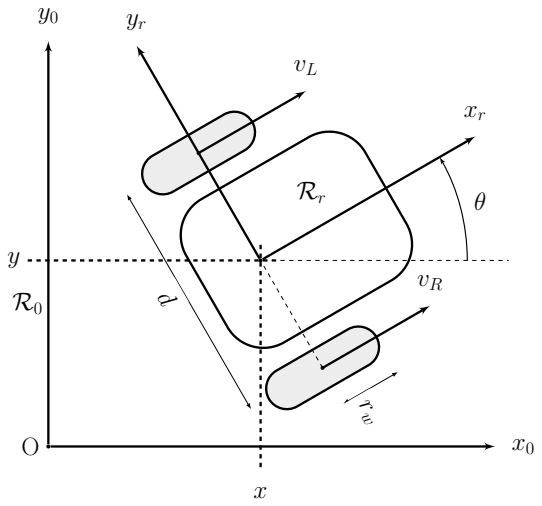


Figure 3.6: Differential drive robot model

ω_R = right wheel rotational speed around the wheel axis y

ω_L = left wheel rotational speed around the wheel axis y

d = distance between the wheels

r_w = wheel radius

R = curvature radius considered in the middle of the wheel axis

Considering ω_R and ω_L as inputs

$$v_L = \dot{\varphi}\left(R - \frac{d}{2}\right) \tag{3.13}$$

$$v_R = \dot{\varphi}\left(R + \frac{d}{2}\right) \tag{3.14}$$

$$v_R - v_L = \frac{2\dot{\varphi}d}{2} \longrightarrow \dot{\varphi} = \frac{v_R - v_L}{d} = \frac{r_w\omega_R + r_w\omega_L}{d} \tag{3.15}$$

$$v_R + v_L = 2\dot{\varphi}R \longrightarrow R = \frac{d(v_R + v_L)}{2(v_R - v_L)} \tag{3.16}$$

where $\dot{\varphi}$ is the angular velocity of the wheels around the center of rotation.

Moreover the linear velocity of the robot v is expressed by

$$v = \frac{v_R + v_L}{2} = \frac{r_w\omega_R + r_w\omega_L}{2} \tag{3.17}$$

The model in matrix form is thus

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r_w}{2} & \frac{r_w}{2} \\ \frac{r_w}{d} & \frac{r_w}{d} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} \tag{3.18}$$

and rewritten in the state space

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r_w}{2} & \frac{r_w}{2} \\ \frac{r_w}{d} & \frac{r_w}{d} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} = \begin{bmatrix} \frac{r_w \cos\theta}{2} & \frac{r_w \cos\theta}{2} \\ \frac{r_w \sin\theta}{d} & \frac{r_w \sin\theta}{d} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} \tag{3.19}$$

(3.19) represents the kinematic model of the differential drive robot.

3.3.3 Car-like mobile robots

We will consider now a robot having the same kinematics of an automobile. The theoretical notations refer to [13] and [12]. A commonly used model for a four-wheeled car-like vehicle is the bicycle model shown in Figure 3.7. The bicycle has a rear wheel fixed to the body and the plane of the front wheel rotates about the vertical axis to steer the vehicle.

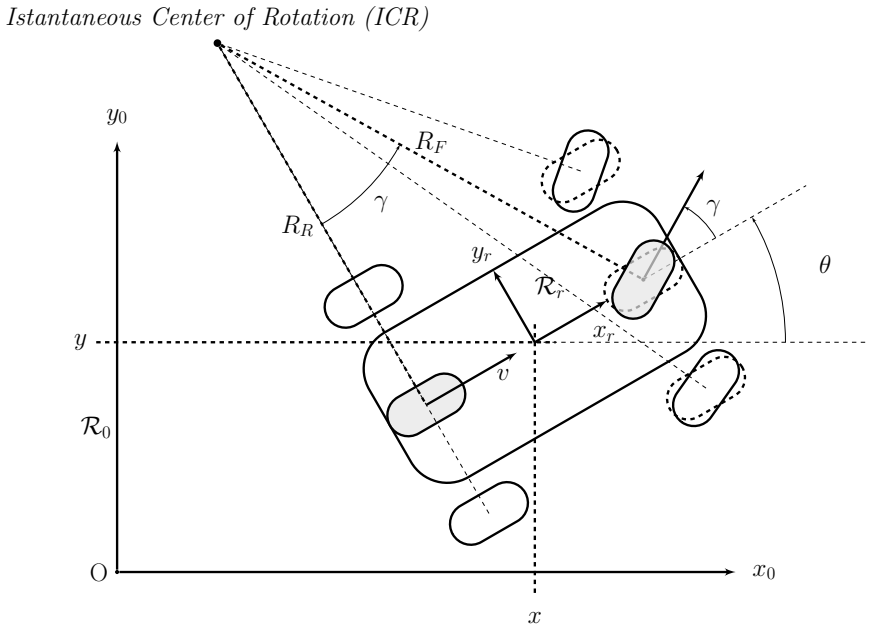


Figure 3.7: Bicycle model of a car. The bicycle approximation is shown in grey. The steering wheel angle is γ and the velocity of the back wheel, in the x_r direction, is v. The wheel axes are extended as dashed lines and intersect at the Instantaneous Centre of Rotation (ICR) and the distance from the ICR to the back and front wheels is R_R and R_F respectively

The pose of the vehicle is represented by the reference frame $\mathcal{R}_r$ with its *x axis* in the vehicle forward direction and its origin at point C, i.e. in the middle between

the rear wheels; the *y axis* is to the left side (lateral motion) which implies that the *z axis* is upward, completing the right-handed reference frame. The configuration of the vehicle is represented by the generalised coordinates $\boldsymbol{q} = [x\ y\ \theta\ \gamma]^T$, where γ is the *steering angle*. The vehicle velocity $\boldsymbol{v}_{Rover}$ is zero in the y direction, since the wheels cannot slip sideways; thus in $\mathcal{R}_r$ RF

$$\{\dot{x}\}_r = v_{Rover}; \qquad \{\dot{y}\}_r = 0$$

The dashed lines show the wheel axes; if they meet in a common point, as in this case, the wheel rotates without slippage; the intersection point is called *Instantaneous Curvature Center* (ICC). The origin of $\mathcal{R}_r$ thus follows a circular path and its angular velocity is

$$\dot{\theta} = \frac{v_{Rover}}{R} \tag{3.20}$$

R is the *turning radius* and it is defined as

$$R = \frac{L}{\tan\gamma} \tag{3.21}$$

where L is the distance between the wheel centres.

The velocity of the rover in the global reference frame $\mathcal{R}_0$ is

$$\{\dot{x}\}_0 = v_{Rover}\cos\theta; \qquad \{\dot{y}\}_0 = v_{Rover}\sin\theta$$

The bicycle is subject to two non-slipping constraints, one for each wheel (non-holonomic constraints)

$$\begin{cases} \dot{y}\cos\theta - \dot{x}\sin\theta = 0 \\ \dot{y}_f\cos(\theta+\gamma) - \dot{x}_f\sin(\theta+\gamma) = 0 \end{cases} \tag{3.22}$$

As defined before, these equations cannot be integrated to form a relationship between position variables x, y and θ.

Moreover, considering the rigid body constraints

$$\begin{cases} x_f = x + L\cos\theta \quad \rightarrow \quad \dot{x_f} = \dot{x} - L\dot{\theta}\sin\theta \\ y_f = y + L\sin\theta \quad \rightarrow \quad \dot{y_f} = \dot{y} + L\dot{\theta}\cos\theta \end{cases} \tag{3.23}$$

we can rewrite (3.22) as function of the generalized coordinates.

$$
\begin{aligned}
0 &= \dot{y}_f \cos(\theta+\gamma) - \dot{x}_f \sin(\theta+\gamma) \\
&= \dot{y}_f(\cos\theta\cos\gamma - \sin\theta\sin\gamma) - \dot{x}_f(\sin\theta\cos\gamma + \cos\theta\sin\gamma) \\
&= (\dot{y} + L\dot{\theta}\cos\theta)(\cos\theta\cos\gamma - \sin\theta\sin\gamma) - (\dot{x} - L\dot{\theta}\sin\theta)(\sin\theta\cos\gamma + \cos\theta\sin\gamma) \\
&= \dot{y}\cos\theta\cos\gamma - \dot{y}\sin\theta\sin\gamma + L\dot{\theta}\cos^2\theta\cos\gamma - L\dot{\theta}\cos\theta\sin\theta\sin\gamma + \\
&\quad - \dot{x}\sin\theta\cos\gamma - \dot{x}\cos\theta\sin\gamma + L\dot{\theta}\sin^2\theta\cos\gamma + L\dot{\theta}\sin\theta\cos\theta\sin\gamma \\
&= \dot{y}\cos(\theta+\gamma) - \dot{x}\sin(\theta+\gamma) + L\dot{\theta}\cos\gamma
\end{aligned}
\tag{3.24}
$$

Thus, the constraints are

$$
\begin{cases}
\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \\
\dot{x}\sin(\theta+\gamma) - \dot{y}\cos(\theta+\gamma) - L\dot{\theta}cos\gamma = 0
\end{cases}
\tag{3.25}
$$

The non-holonomic constraint for the rear wheel is satisfied by

$$
\dot{x} = v_{rover}\cos\theta \qquad \text{and} \qquad \dot{y} = v_{rover}\sin\theta
$$

Applying this to the constraint on the front wheel, we can obtain a solution for θ (to simplify $v_{rover} = v$)

$$
\begin{aligned}
0 &= \dot{x}\sin(\theta+\gamma) - \dot{y}\cos(\theta+\gamma) - L\dot{\theta}cos\gamma \\
&= (v\cos\theta)\sin(\theta+\gamma) - (v\sin\theta)\cos(\theta+\gamma) - L\dot{\theta}cos\gamma \\
&= (v\cos\theta)(\sin\theta\cos\gamma + \cos\theta\sin\gamma) - (v\sin\theta)(\cos\theta\cos\gamma - \sin\theta\sin\gamma) - L\dot{\theta}\cos\gamma \\
&= v\cos\theta\sin\theta\cos\gamma + v\cos^2\theta\sin\gamma - v\sin\theta\cos\theta\cos\gamma + v\sin^2\theta\sin\gamma - L\dot{\theta}\cos\gamma \\
&= v\sin\gamma - L\dot{\theta}\cos\gamma
\end{aligned}
$$

$$
\dot{\theta} = \frac{v\sin\gamma}{L\cos\gamma} = \frac{v\tan\gamma}{L}
\tag{3.26}
$$

The overall equations governing the kinematic model of the bicycle robot are

$$
\begin{cases}
\dot{x} = v\cos\theta \\
\dot{y} = v\cos\theta \\
\dot{\theta} = \frac{v\tan\gamma}{L} \\
\dot{\gamma} = \omega_{STR}
\end{cases}
\tag{3.27}
$$

And in matrix form

$$
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \frac{1}{L}\tan\gamma \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega_{STR}
\tag{3.28}
$$

where v is the linear driving velocity of the rover and ω_{STR} is the input steering velocity.

The rate of change of heading $\dot{\theta}$ is referred to as *turn rate*, *heading rate* or *yaw rate* and can be measured by a gyroscope. It can also be deduced from the angular velocity of the wheels on the left- and right-hand sides of the vehicle which follow arcs of different radius and therefore rotate at different speeds.

The (3.28) is referred to as a forward kinematic model since it describes the velocities of the vehicle but not the forces or torques that cause the velocity. It captures an important characteristic of a wheeled vehicle: when $v = 0$ then $\dot{\theta} = 0$, that is, it is not possible to change the vehicle orientation when it is not moving, *non-holonomic* system. [21]

3.3.4 Inverse Kinematic problem

When controlling a mobile robot, one wants to known which are the reference speeds of the wheels that make the rover follow a desired reference trajectory. This implies solving the Inverse Kinematic problem:

$$\text{configuration space velocities } (\dot{x}, \dot{y}) \quad \longrightarrow \quad \text{operational space velocities } (v, \omega_{STR})$$

Here, we consider the problem of generating reference speeds for the car-like robot. Assume that the model desired output trajectory is given in terms of cartesian components $x_d(t), y_d(t)$ of the car rear wheel. Thus

$$\dot{x}_d = v_d \cos\theta_d \tag{3.29}$$

$$\dot{y}_d = v_d \sin\theta_d \tag{3.30}$$

$$\dot{\theta}_d = \frac{v_d}{L} \tan\gamma_d \tag{3.31}$$

$$\dot{\gamma}_d = \omega_{STR} \tag{3.32}$$

From (3.29) (3.30)

$$1 = \cos^2\theta_d + \sin^2\theta_d = \frac{\dot{x_d}^2}{v_d^2} + \frac{\dot{y_d}^2}{v_d^2} \qquad \longrightarrow \qquad v_d(t) = \pm\sqrt{\dot{x}_d^2(t) + \dot{y}_d^2(t)} \tag{3.33}$$

where the sign refers to the desired vehicle motion , forward or backward respectively.

Dividing (3.30) by (3.29), we can compute the robot orientation

$$\frac{\dot{y}_d}{\dot{x}_d} = \frac{v_d \sin\theta_d}{v_d \cos\theta_d} = \tan\theta_d \qquad \longrightarrow \qquad \theta_d = \text{atan2}(\dot{y}_d(t), \dot{x}_d(t)) \tag{3.34}$$

Differentiating (3.30) and (3.29) we can compute the orientation of the vehicle as

$$\ddot{x}_d = -v_d\dot{\theta}_d \sin\theta + \dot{v}_d \cos\theta_d \qquad \longrightarrow \qquad \dot{v}_d = \frac{\ddot{x}_d + v_d\dot{\theta}_d \sin\theta_d}{\cos\theta_d} \tag{3.35}$$

$$\ddot{y}_d = v_d\dot{\theta}_d \cos\theta_d + \dot{v}_d \sin\theta_d \qquad \longrightarrow \qquad \dot{v}_d = \frac{\ddot{y}_d - v_d\dot{\theta}_d \cos\theta_d}{\sin\theta_d} \tag{3.36}$$

$$\begin{aligned}
\frac{\ddot{x}_d + v_d\dot{\theta}_d \sin\theta_d}{\cos\theta_d} &= \frac{\ddot{y}_d - v_d\dot{\theta}_d \cos\theta_d}{\sin\theta_d} \\
\sin\theta(\ddot{x}_d + v_d\dot{\theta}_d \sin\theta)_d &= \cos\theta_d(\ddot{y}_d - v_d\dot{\theta}_d \cos\theta)_d \\
\ddot{x}_d \sin\theta_d + v_d\dot{\theta}_d \sin^2\theta_d &= \ddot{y}_d \cos\theta_d - v_d\dot{\theta}_d \cos^2\theta_d \\
\ddot{x}_d \sin\theta_d + v_d\dot{\theta}_d &= \ddot{y}_d \cos\theta_d
\end{aligned}$$

$$\dot{\theta}_= \frac{\ddot{y}_d \cos\theta_d - \ddot{x}_d \sin\theta_d}{v_d} = \frac{\ddot{y}_d(\frac{\dot{x}_d}{v_d}) - \ddot{x}_d(\frac{\dot{y}_d}{v_d})}{v_d} = \frac{\ddot{y}_d\dot{x}_d - \ddot{x}_d\dot{y}_d}{v_d^2} \tag{3.37}$$

Then, plugging this last relation into (3.31) we obtain the desired steering angle

$$\dot{\theta}_= \frac{v_d}{L}\tan\gamma_d = \frac{\ddot{y}_d\dot{x}_d - \ddot{x}_d\dot{y}_d}{v_d^2}$$

$$\gamma_d = \arctan\frac{L(\ddot{y}_d\dot{x}_d - \ddot{x}_d\dot{y}_d)}{v_d^3} \tag{3.38}$$

Finally, differentiating (3.38)

$$\omega_{STR} = \dot{\gamma} = Lv_d\frac{(\dddot{y}_d\dot{x}_d - \dddot{x}_d\dot{y}_d)v_d^2 - 3(\ddot{y}_d\dot{x}_d - \ddot{x}_d\dot{y}_d)(\dot{x}_d\ddot{x}_d + \dot{y}_d\ddot{y}_d)}{v_d^6 + L^2(\ddot{y}_d\dot{x}_d - \ddot{x}_d\dot{y}_d)^2} \tag{3.39}$$

All these equations provide the input needed to reproduce the desired output trajectory. They depend only on the values of the operational space coordinates (x_d and y_d) and its derivatives up to the third order.

3.4 Navigation, Supervision and Control

As stated before, a mobile robot is a structure capable to move autonomously in the surrounding environment exploiting its perceptions and performing required actions.

A crucial prerequisite for a mobile robot to perform its tasks is the capability to autonomously navigate. *Navigation* requires the following capabilities:

- Localization: to determine its pose with respect to a given reference frame
- Mapping: to build a consistent and meaningful representation of the environment
- Path planning: to plan the motion strategy to reach a given target position

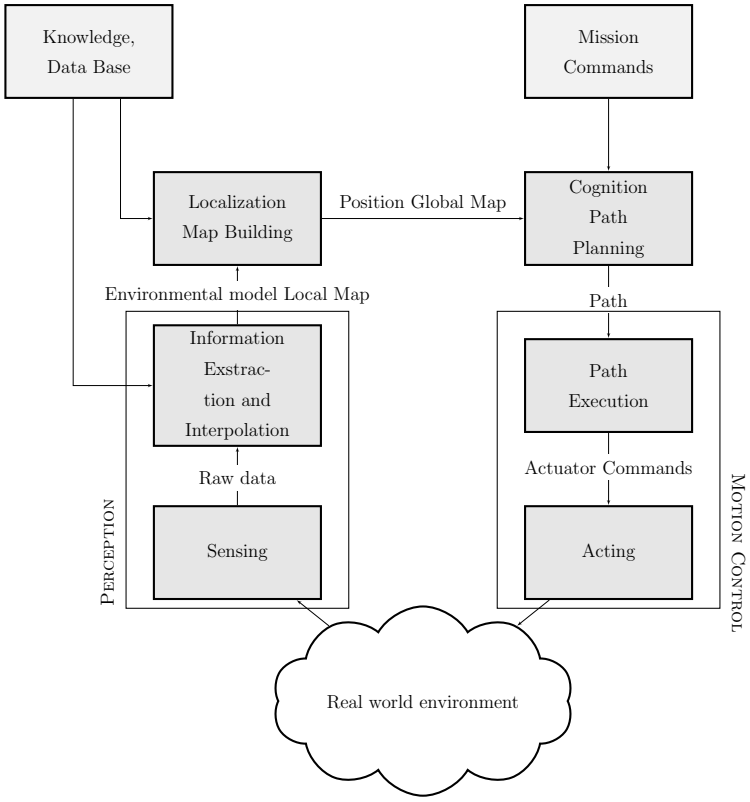


Figure 3.8: General control scheme of a mobile robot

A general control scheme is reported in Figure 7.1. It comprises

High-level control : it is responsible of planning the robot motion. Given the data provided by sensors, it elaborates the reference signal to be provided as input to the low-level control.

Low-level control : PID controllers are in charge of drive the robot actuators (usually DC motors) in a way that it moves according to the desired commands. The low level control loop concerns only the actuator control, according to the instructions

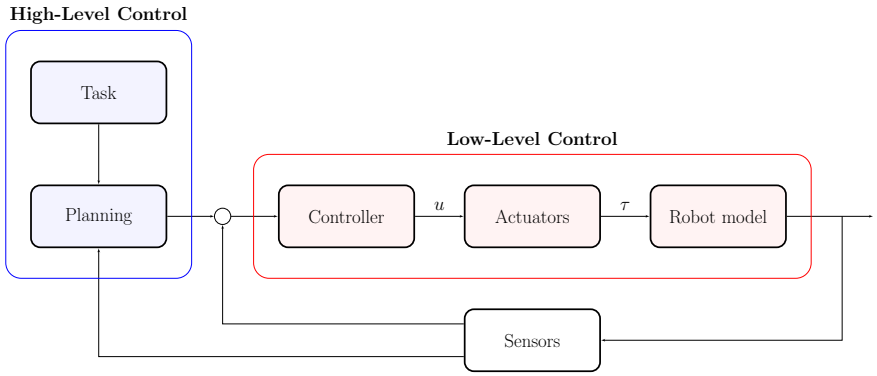


Figure 3.9: Low-level and high-level controls of a mobile robot

coming from the high-level control. Usually the PIDs parameters are high enough to make the robot a purely kinematic system.

Chapter 4

ACTIVATE MODEL OVERVIEW

4.1 Introduction

The purpose of this thesis is to model the ExoMars actuator controller driving the multi-body vehicle model, in order to analyse the mechanical structure of the rover when performing specific manoeuvres. In this chapter the main characteristics of the software used to build and simulate the model are presented. The technical description of the software is get from [26].

The difficulty of modelling and simulating complex multi-disciplinary system (mechanics and control) in one simulation tool alone, led to the choice of couple different simulation tools with each other. *Co-simulation* is a general approach for the joint simulation of models developed with different tools where each tool treats one part of a modular coupled problem. In this specific case, we can simulate a complex system that includes a multi-body system and control subsystems. When co-simulating, the data exchange (input and output variables, status information) between subsystems is restricted to discrete communication points; in the time between two communication points, the subsystems are solved independently from each other by their individual solvers. Considering the *master-slave* approach, a common method in co-simulation, the slave simulates the sub-model whereas the master is responsible for both coordinating the overall simulation as well as transferring data. The slaves are the simulation tools, which are prepared to simulate their subtask. They are able to communicate data, execute control commands and return status information.

For this purpose *solidThinking Activate* was chosen. It is a software solution for multi-disciplinary, dynamic system modelling and simulation, and it is especially useful for signal-processing and controller design. One of the key functions of this tool is precisely the possibility of performing co-simulation with multi-body dynamics.

4.2 Co-simulation with Multi-Body systems

Co-simulation enables multi-body dynamic models and Activate models to communicate with each other during simulation. The multi-body system (MBS) is simulated with **MotionSolve**, a multi-body modelling, analysis, visualization and optimization solution for performing multi-disciplinary simulations that include kinematics and dynamics, statics and quasi-statics, linear and vibration studies, stress and durability, loads extraction, co-simulation, effort estimation and packaging synthesis. In the Activate-MotionSolve interface, Activate is the master and MotionSolve is the slave.

In co-simulation, inputs to the MotionSolve model is provided and the output of the MotionSolve block is requested. The way a co-simulation is performed is shown in Figure 4.1. The Activate simulator performs the following steps for the co-simulation and synchronisation between slaves:

1. The Activate simulator advances the time from t_n to t_{n+1}, and updates the output of all blocks (except slaves)
2. Slaves are called one by one and asked to advance their time from t_n to t_{n+1} and their output is requested at t_{n+1}
3. The Activate simulator takes a new step from t_{n+1} to t_{n+2} using the new output value of slaves computed at t_{n+1}
4. Go on

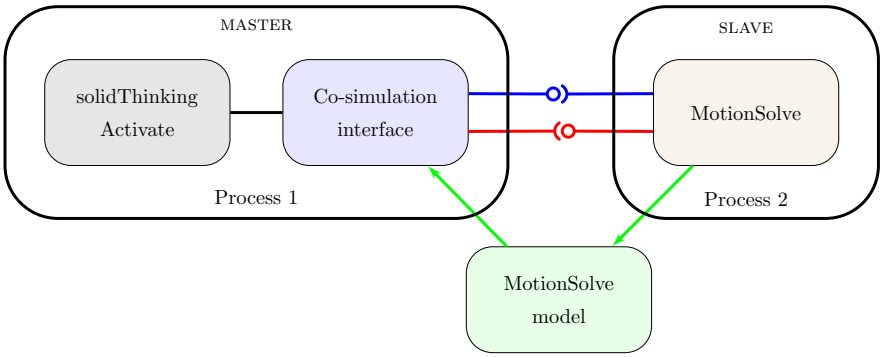


Figure 4.1: Co-simulation between Activate and MotionSolve

The procedure required to perform co-simulation is here described:

1. Build the multi-body model of the system in MotionView, a general pre-processor for multi-body dynamics.
2. Modify the multi-body model for co-simulation: the MotionView model that acts as a plant, must be prepared by adding *solver variables*, such as control plant

input and control plant output. These variables are either used as inputs to the multi-body model, such as forces or torques generated by the controller modelled in Activate, or measured outputs, such as displacements or velocities.

3. Build the control system model with Activate.

4. Integrate the multi-body model in the Activate model.

An general scheme of the model that will be implemented in Activate is presented in Figure 4.2.

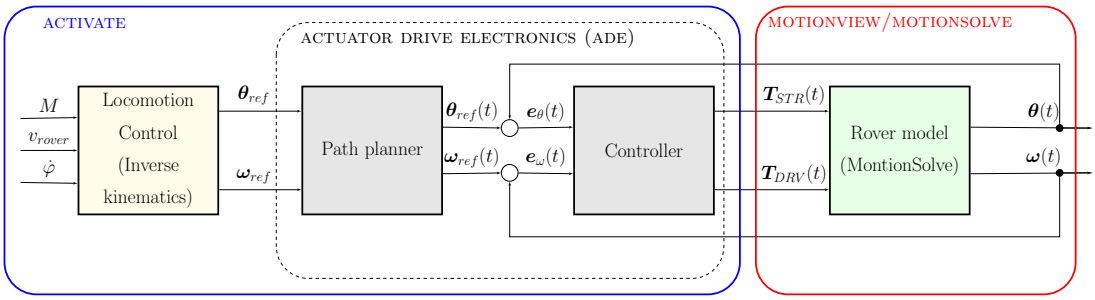


Figure 4.2: General scheme of the complete model built for co-simulation

where

M = the manoeuvre type the rover is asked to perform

v_{rover} = the rover velocity in $[m/s]$

φ = rover heading in $[rad]$

$\boldsymbol{\theta}_{ref}$ = the angle reference values of the six wheels in $[rad]$

$\boldsymbol{\omega}_{ref}$ = the angular speed reference values of the six wheels in $[rad/s]$

$\boldsymbol{\theta}_{ref}(t)$ = the time laws of $\boldsymbol{\theta}_{ref}$

$\boldsymbol{\omega}_{ref}(t)$ = the time laws of $\boldsymbol{\omega}_{ref}$

$\boldsymbol{e}_{\theta}(t)$ = the differences between the six reference angles and the six current ones

$\boldsymbol{e}_{\omega}(t)$ = the differences between the six reference angular speeds and the six current ones

$\boldsymbol{T}_{STR}(t)$ = the six command torques appplied to the steering axes

$\boldsymbol{T}_{DRV}(t)$ = the six command torques appplied to the drive axes

$\boldsymbol{\theta}(t)$ = the six current angles

$\boldsymbol{\omega}(t)$ = the six current angular speeds

The constitutive blocks are:

1. The *Trajectory Planner*
2. The *Controller*
3. The multi-body model of the rover

They will be presented in general terms in the following section, while a more detailed description will be the subject of the following chapters.

4.3 The Plant

As stated before, the multibody model of the ExoMars rover was built in MotionView environment on the basis of the actual rover mechanical design. The MotionView graphical user interface allows to add ENTITIES and set their values directly from specific panels. The final model is reported in Figure 4.3; it consists of POINTS, BODIES connected through JOINTS, which constraint them to move in specific direction, GRAPHICS offering visualization for entities during pre and post processing and CONTACTS between bodies. Contacts between wheels and terrain are the most tricky entities, since their properties have to be as accurate and realistic as possible, in order to have reliable results and reasonably short simulation time. A whole science, called *Terramechanics* exists, addressing the study of soil properties and the interaction of wheeled or tracked vehicles on various surfaces; this makes it evident how much intricate this topic is. The procedure used to build this model is presented in details in Chapter 5.

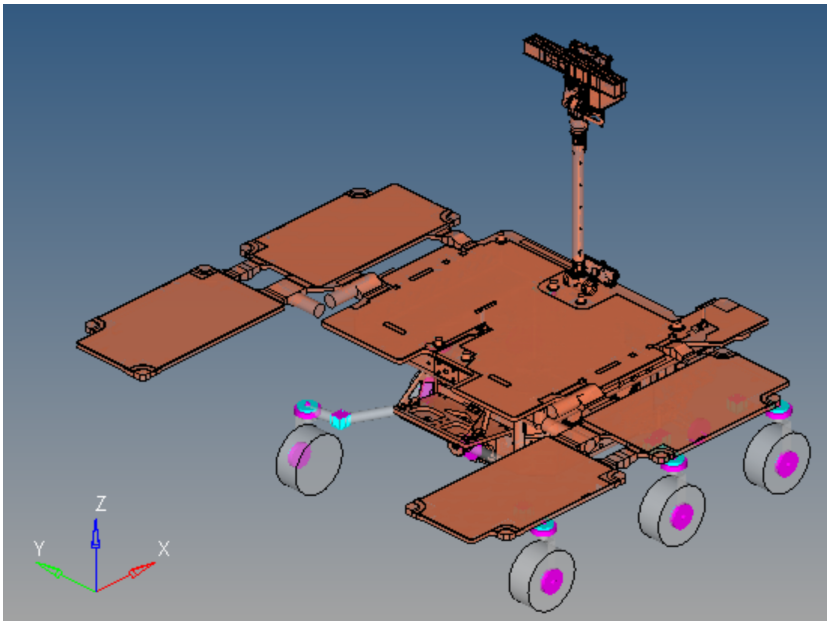


Figure 4.3: Complete multi-body model built with MotionView

As shown in Figure 4.2 the MotionSolve *Rover model* block has inputs and outputs; these have to be added to the model by creating SOLVER VARIABLES and collecting them into SOLVER ARRAYS. The solver variables contain the individual plant input and output values, while the solver arrays define the plant input and output to communicate with Activate. In particular, two input vectors have been defined, called $\boldsymbol{T}_{STR}(t)$ and $\boldsymbol{T}_{DRV}(t)$, containing respectively the six steering torques and the six drive torques, which are computed by the *Controller* block. By contrast, the output vectors, called $\boldsymbol{\theta}(t)$ and $\boldsymbol{\omega}(t)$, are the steering position and driving speed of the six wheels, computed by MotionSolve.

4.4 The Controller

The *Controller* block is in charge of making the rover follow a specific path at a specific speed. In order to do this, the wheels have to be properly driven and steered. The control system implemented in this thesis is based on simple PID controllers that drive the wheels, making them rotate at the required angular speed, and steering them as needed to guarantee that the rover performs the requested manoeuvre. Thus two controller categories have been implemented:

STR controllers receive as input the error

$$\boldsymbol{e}_{\theta}(t) = \boldsymbol{\theta}_{ref}(t) - \boldsymbol{\theta}(t)$$

where $\boldsymbol{\theta}_{ref}(t)$ are the reference steering angles coming from the *Profile Generator* block, which generates the reference values profiles on the basis of the target values computed by the *Locomotion Control*; and $\boldsymbol{\theta}(t)$ are the current angles computed by MotionSolve, which receives as input the control variables (control torques), runs the multi-body model and computes the wheel instantaneous angular positions. Based on its control law, each one of the six STR controllers generates the required control torque $\boldsymbol{T}_{STR}(t)$.

DRV controllers receive as input the error

$$\boldsymbol{e}_{\omega}(t) = \boldsymbol{\omega}_{ref}(t) - \boldsymbol{\omega}(t)$$

where $\boldsymbol{\omega}_{r}(t)$ are the reference wheel velocities and $\boldsymbol{\omega}(t)$ are the current ones, exactly as in the case of STR controllers, but considering velocity variables. The difference with respect the previously presented controllers lies in the controller parameters which are tuned in order to obtain the desired behaviour.

The control law and the detailed design of these controllers are described in details in Chapter 6.

4.5 The Locomotion Control

The *Locomotion Control* block interfaces between the user, which sets the input parameters (i.e. the manoeuvre type, vehicle speed and heading) and the low-level control algorithm which drives the rover actuators. Since the actual mobility actuators are the wheels with their driving and steering motors, it is necessary to convert the *vehicle level* manoeuvre commands into *wheel level* commands, i.e. driving velocities and steering positions of the wheels. This is the Locomotion Control task. Chapter 7 will describe how this problem, the so-called *Inverse Kinematics* has been solved. It consists in the definition of the rover wheel drive speeds and steering angles as functions of the rover speed and heading.

4.6 Simulation Setting

Before running the complete model, some simulation parameter must be set. This section is in charge of providing the basic concepts about these with reference to the user manual [26]. The simulation parameters panel allows to customize the simulation parameters and choosing a *solver*. In Activate several numerical ODE solvers are available. Each one implements a specific numerical algorithm for solving the model. They have many control parameters that should be set to optimize the simulation in different situations. Thus, the choice of the solver to use and how to set its parameters is an crucial operation.

4.6.1 Numerical ODE solver classifications and characteristics

Stiff vs non-stiff solvers

First of all it is important to know if a particular ODE solver is appropriate for solving a stiff ODE [1]. Based on this information the appropriate solver can be chosen for a particular problem. In order to simulate an ODE, the numerical solver divides the simulation time into small intervals of length *size* and finds the solution at each interval. Smaller step size generally results in higher accuracy of the solution, but it also means higher computation time. Even if the model to be simulated is stable, the stability of the solution provided by the numerical method depends directly on the step size. In many cases smaller step sizes should be used to ensure the stability of the numerical method. Whatever the numerical method, for large step sizes the solution tends to be unstable and stability may be achieved by using sufficiently small step sizes. [26]

[1] A stiff equation is a differential equation for which certain numerical methods for solving it are numerically unstable, unless the step size is taken to be extremely small.

Explicit vs implicit solvers

Numerical solvers for differential equations are divided into two categories: explicit and implicit methods. The former calculates the new state of a system as a function of current state and current state derivatives. The latter computes the state by solving a nonlinear equation involving both the new state and the new derivatives as a function of the current state. [26]

Consider the time step size h, the current system state $x(t)$ and the unknown new state $x(t+h)$. Then, for an explicit method we have

$$x(t+h) = F(x(t))$$

while for as implicit method we have a nonlinear equation like the following

$$G(x(t), x(t+h)) = 0$$

Implicit solvers should solve a nonlinear equation at each time step which makes these solvers slower compared to explicit solvers. However, they are used because many problems in real life are stiff, and, in these cases, explicit methods need to take small step sizes for a slowly varying solution. By contrast, implicit methods can integrate much faster by taking much larger step sizes. For stiff problem an explicit method requires very small time steps to keep the error small.

Fixed-step vs. variable-step solvers

Another aspect of numerical solvers is whether they are fixed-step or variable-step. The former uses the same step size all over the simulation: it is defined once and kept fixed. This type of solver solves the model at regular intervals of time that cannot be changed. Decreasing the integration step increases the accuracy of the results, while increases the time required to simulate.

The latter adjusts the step size during the simulation, as a function of the estimated error in the solution. The advantage of this solution is that the integration step becomes either smaller or larger depending on the dynamics of the system. This is an important feature that increases the efficiency of the solvers. Variable step solvers are more reliable to handle fast changes in the solution by reducing the time step, where constant time step methods may give wrong results.

On the other hand, in some situations where the computational effort should be limited in each step, such as realtime applications, fixed step-size solvers are privileged. The user will often want the solution at specified points in time, for example at equally spaced intervals, to produce tables or plots. For these cases, we do not need using fixed-step solvers. [26]

4.6.2 Maximum, minimum, and initial step size

Variable step solvers adjust the step size during simulation, and the step size value can be very small or very large, based on the dynamics of the system. Activate provides the possibility of setting high and low limits on the step size chosen by the solver. The maximum step size can be set to *auto*, and in this case it is computed by Activate according to the following formula [26] :

$$h_{max} = \frac{T_{final} - T_{initial}}{100}$$

where $T_{initial}$ and T_{final} are the initial and finale simulation time.

However, setting manually a high limit on the step size may be very useful. For example, consider a simple non-stiff model containing a zero-crossing. The solver tries to advance the time by taking large step sizes. If it happens that the zero-crossing function changes the sign several times during a large step, the solver will miss the sign change in the zero-crossing function and it may introduce error in the result.

By contrast, the size of the very first step after a cold-restart in variable step solvers does not usually have much importance, since the solver can adjust it and find an appropriate value. In some situations, the solver is unable to find a good value for the step-size and fails. In order to help the solver, the initial value of the step-size can be set manually.

The simulation setting chosen in this thesis work are listed here:

- Solver: **Lsoda**. It is a variable step and variable order solver. This solver is the best choice when the user does not know if the model is stiff or not. The solver switches automatically between stiff and non-stiff methods during the integration.
- Absolute and relative tolerance: 0.000001. The absolute and relative error tolerances specified for the integration of an ODE affects the speed and accuracy of the computation for variable step solvers. Small error tolerances may result in higher accuracy, but the simulation time will become large. On the other hand looser error tolerance values, speeds up the simulation but the error introduced in the solution will be higher. As a consequence, it is important to find a compromise between speed and accuracy.
- Maximum step size: 0.001, it is a good compromise between accuracy and time required to simulate.

Chapter 5

MOTIONSOLVE: EXOMARS ROVER MULTI-BODY MODEL

5.1 Introduction

This chapter focuses on the implementation of the ExoMars rover multi-body model. Before describing the procedure followed in the model construction, a brief introduction is given in order to clarify some key aspects about Multi-body System (MBS) Simulation and to present the software used during the thesis development. The informations are getting from to the user manual [28].

5.2 Multibody System Simulation

> "There are multiple ways to look at a problem, formulate the underlying equations and solve them. We have the freedom to pick the methods that seem most natural for a specific problem. In addition to Newton's Laws of Motion, Virtual Work based methods like D'Alembert's Principle, Energy based methods like Euler-Lagrange Equations and Variational methods such as Hamilton's Principle can be used to formulate problems. These are only a few of the known methods! Similarly there are multiple approaches to solve the equations of motion. Each method has its advantages and disadvantages and there is no "one best method for all problems". The richness of the theory and breadth of available mathematics to solve multi-body problems has always fascinated me."
>
> Rajiv Rampalli (*Vice President – MBS software development, Altair)

Multi-Body System Simulation is the study of motion of mechanical systems caused by external forces and motion excitations acting on it. The term *multi-body* refers to complex systems that can undergo large overall motion, i.e. the extent of the relative

motion between the components can be larger than or comparable to the overall dimensions of the system. The mechanical system may consist of rigid and flexible bodies connected by various kinds of kinematic constraints and flexible connectors.

Software for MBS simulation is of growing fundamental importance to modern Mechanical Computer Aided Engineering (MCAE), since physics based modelling tools represent a very good trade-off between mathematical complexity and accuracy. These tools allow engineers to build, test, evaluate, and improve product designs without building any hardware prototypes. With MBS software, it is possible to reduce product development costs, evaluate more design alternatives, and decrease the time it takes to bring a new product to market.

A MBS simulator is a system level tool, which can effectively solve complex, multi-physics problems that are characteristic of real-world situations. It achieves this primarily by collaborating with other synergetic technologies, as shown in Figure 5.1:

1. Geometry information from Computer-Aided Design (CAD) software is used to define the basic design of a mechanical model for MBS.

2. The stresses, strains, deformations, and material models in a Finite Element Analysis (FEA) are used to build high fidelity component models for MBS.

3. Software that simulates multi-domain actuators is used to create complex MBS subsystems that contain electrical, hydraulic, pneumatic, and mechanical subsystems.

4. Control system design packages can apply the techniques of classical and modem control theory to design controllers that manage overall system behaviour in MBS models. Simplified MBS models are commonly used to design control systems.

5. Optimization and Design of Experiments (DOE) software are used to determine design parameters that optimize system level behaviour and improve the performance of a mechanical system.

A Multi-Body system study generally involves the following steps:

- Constructing Models
- Executing Solvers
- Post-Processing

Particularly, in this thesis work, MotionView (model building), MotionSolve (analysis), HyperGraph and HyperView (post-processing) will be employed.

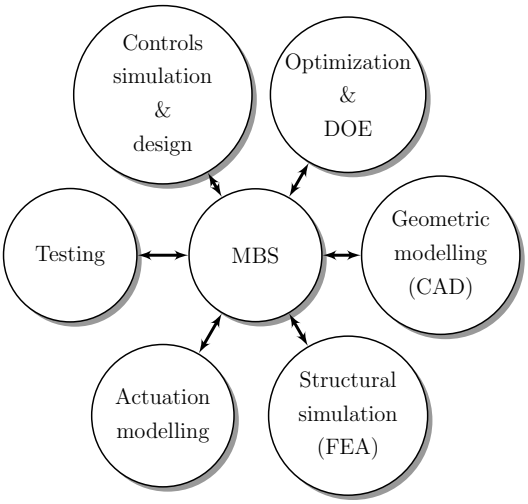


Figure 5.1: Interaction between MBS and other technologies.

5.3 MotionView - MotionSolve

MotionView is a general pre-processor for multi-body system simulation, while *MotionSolve* is its solver. The multi-body models shall be created through the user interface, where entities can be added and deleted and their values set.

The model construction starts with the definition of *points*. These are the fundamental construction elements for MotionView models, since almost all the entities that can be created in MotionView need to use points either for defining their location or orientation. Points exist only in MotionView, not in MotionSolve files. They are only used to specify locations for other entities that define MotionSolve models (e.g. markers).

The following step consists in the *body* creation. A body is the same as a link characterised by mass properties, that are essential for their definition. Nevertheless, *graphics* can be associated with a body if desired, but it is not essential.

Then, the bodies are constrained through *joints* and *motions*. This means that there are algebraic equations restricting the relative motion between them, since constraints only allow the connected bodies to have relative motion in certain specific directions; motions in all the other directions are forbidden. Finally, one can set the external *forces* acting on the system.

Once the model is built up, the solver *MotionSolve* is invoked. It is a set of computation algorithms that solve equations of motion of the system provided to it by MotionView; it is the heart of the simulation software program. MotionSolve can solve several different classes of problems; the most general one consists of problems in *dynamic* analysis, where the system can have more than one uncontrolled degree of freedom (dof). By contrast, *static* analysis is most often used to compute the equilibrium configuration of a mechanism. Moreover, *kinematic* analysis is also provided by

MotionSolve; it is used for systems that have no uncontrolled dofs, and it is typically used early in the design cycle, at the concept stage. Finally, *quasi-static* analysis is applicable when the forces change with time, but do so slowly. This means inertial forces can be ignored, and the static equilibrium equations can be solved at each instant of time. Stability analysis is a good example of its usage.

Once the solution is completed by the solver, this generates different types of output files among which there are animations of movement and plots of forces (.h3d), where accelerations, velocities, displacements against time are the main data generated. [28]

In the following section the ExoMars rover vehicle design will be described, in order to understand the main characteristics of its structure. Then, it will be outlined how the system has been modelled in MotionView environment.

5.4 Rover vehicle system mechanical design

The ExoMars Rover, developed by the European Space Agency ESA, provides key mission capabilities: surface mobility, subsurface drilling and automatic sample collection, processing, and distribution to instruments.

The locomotion is achieved through six wheels. Each wheel pair is suspended on an independently pivoted bogie (the articulated assembly holding the wheel drives), and each wheel can be independently steered and driven. In addition, all wheels can be individually pivoted to adjust the rover height and angle with respect to the local surface, and to create a sort of walking ability, particularly useful in soft, non-cohesive soils like dunes.

The ExoMars rover module is comprised of the following subsystems [4]:

- Rover vehicle
  - Structure and mechanical system
  - Mobility system
  - Thermal system
  - Electrical system
  - Communication system
  - Rover software
- Sample acquisition system (SAS) "Drill" and Sample Preparation and Distribution System (SPDS)
- Analytical laboratory

- Mission management software (MMS)
- Scientific payloads consisting of set of instruments

When describing the mechanical structure of a system, the first thing to do is to define its reference frame; in particular, the ExoMars rover coordinate system is showed in Figure 5.2: it is fixed with the rover, thus moves as the Rover moves, and it is defined as follows:

- The x *axis*, X_{RB}, lies towards the front of the rover in the nominal direction of travel.
- The z *axis*, Z_{RB}, lies vertically upwards, antiparallel to the gravity vector when the rover is on flat, horizontal terrain.
- The y *axis*, Y_{RB}, completes the orthogonal right-handed reference frame, and will lie to the left of the Rover.

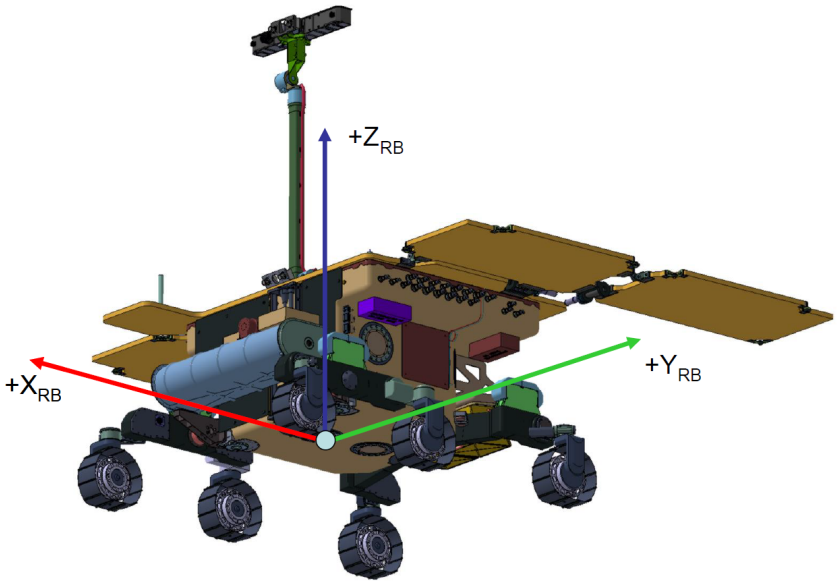


Figure 5.2: Rover module coordinate system

5.4.1 Deployed configuration

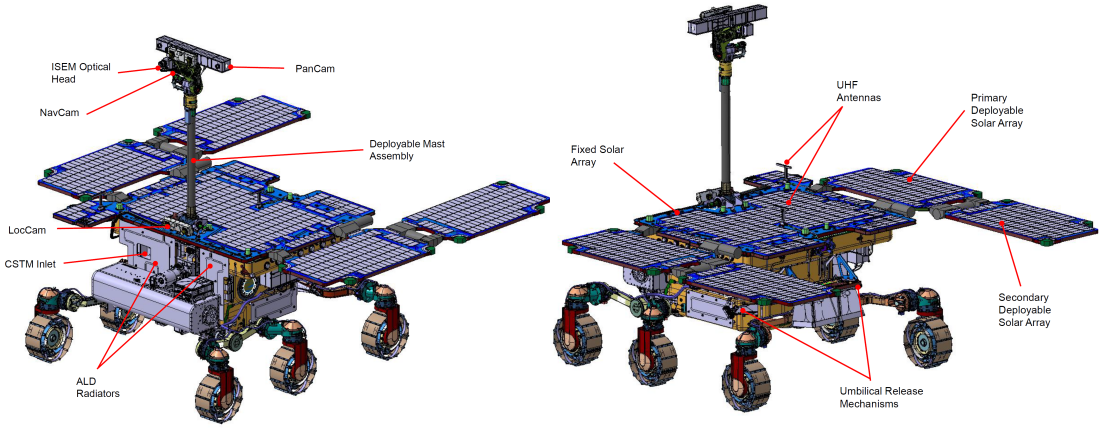
The rover in its deployed on-surface configuration is depicted in Figure 5.3. The rover body is supported by the *3-bogie locomotion suspension system*, with side bogie pivots at the lower front corners and the rear bogie pivot at the centre of the lower rear edge.

The upper part of the body incorporates overhanging balconies at the sides and rear to support a fixed solar array panel. Primary and secondary deployable solar panels

are hinged from the side edges of the fixed panel and the rear edges of the primary panels.

The *Actuator Drive Electronics* (ADE) are suspended below the left and right overhanging balconies on the rover body, and a navigation camera is mounted to the pan and tilt mechanism at the top of the deployable mast assembly (DMA). The mast is deployed by a single hinge mechanism attached at the centre of the front edge of the solar array to permit the cameras to view the area immediately in front of the rover for normal forwards locomotion and sample acquisition and transfer operations. However, viewing of the ground immediately below the rover to the sides and rear, including the middle and rear wheels, is restricted by the presence of the deployed solar array.

During mobility, the drill is stowed horizontally across the front of the rover body. Its positioning mechanism is mounted to the rover body front face to permit the drill to be deployed vertically for drilling and translated and swung upwards to deposit samples into the analytical laboratory drawer inlet. A localisation camera is mounted at the front edge of the fixed solar panel, ahead of the mast hinge, with a forwards and downwards view over the top of the stowed drill [4].



(a) (b)

Figure 5.3: Deployed rover configuration

5.5 Building Up the Model with MotionView

Once the main characteristics and dimensions of the rover mechanical structure are known, one shall start building up its MBS model. The multi-body model built in MotionView was structured in subsystems, in order to differentiate the elements composing the rover body and those ones related to the locomotion system.

The MotionView model structure is the following:

- Model
  - Locomotion
    - Lateral bogies
    - Rear bogie
    - Steering axes
    - Wheels
  - Rover body

The model construction followed the steps explained in Section 5.3. The creation of each subsystem will be described in details in the following subsections.

5.5.1 The *Locomotion* Subsystem

The Bogie Electro-Mechanical Assembly BEMA is the actuation system the rover uses to move across the Martian surface. It consists of the following key parts: wheels, actuators (drive, steering and deployment motor/gearboxes), bogies, angular speed and position sensors, and Hold-Down and Release Mechanisms (HDRMs) [1]. The BEMA provides the mechanical means for the rover to achieve the mobility requirements.

The key mobility capabilities are:

Self-deployment including raising the rover body from the stowed to its nominal operating height above the lander platform. The locomotion subsystem is deployed and the rover body raised from the Lander Platform by synchronised operation of the BEMA 6 deployment and 6 wheel drive actuators. The baseline deployment scenario is the following: the middle wheels remain locked to the Lander at the wheel HDRMs and the front and rear wheel legs are pre-deployed before and after respectively to lengthen the overall front-rear distance. This confers stability for deployment when the Lander Platform is at steep landing inclinations and significantly reduces the torques required from the deployment actuators. Following pre-deployment, the front wheel legs and wheels are driven back towards the middle of the Rover and the rear and middle wheel legs are driven forwards simultaneously, causing the body to lift. The middle wheels remain fixed at the

[1] The set of three Body Hold Down and Release Mechanisms provides the primary mechanical connection between the rover and lander during launch, cruise and Entry, Descend and Landing. They are configured as an iso-static mounting system in order to minimise the effects of differential strain between the two connected bodies. After landing, the Body HDRMs provide a release function to permit the rover to deploy and egress from the Lander.

HDRMs to prevent any forwards or aft motion due to imperfect synchronisation or slippage. Operation of the middle wheel drive actuators therefore contributes to the deployment motorisation of the wheel leg.

Egress from the lander platform for a range of scenarios which include descents down inclined ramps and steps down to the ground. Guide rails on the lander platform ramps prevent the rover driving off the side of the ramp during egress. Once the Rover is deployed and ready to egress, the middle wheel HDRMs are released one at a time. The BEMA actuators are controlled by the two ADE units mounted to the outside of the rover body. The drive and sensor circuits are allocated so that the left ADE operates the left side bogie (front and middle wheels) and the left rear wheel of the rear bogie and the right ADE operates the right side bogie (front and middle wheels) and the right rear wheel of the rear bogie to minimise harness lengths.

On surface mobility including:

- Point turning capability to drive across difficult terrain on the surface by changing heading in a way decoupled from the Rover position.
- Double Ackerman steering for smooth turns during forward and backward motion without having to stop to perform point turns.
- Crab manoeuvres for fine adjustment of the Rover position without heading implications, and to enhance path following accuracy particularly by allowing compensation of lateral slippage when traversing across slopes.
- Climb over rock obstacles and crevasses.
- Climb, cross-hill traverse and descend various loose soil slopes.

The nominal deployed configuration of the ExoMars locomotion subsystem is shown in Figure 5.4. The design is symmetrical between the left and right side of the Rover.

As already stated, the Rover has a six wheel locomotion suspension in a 3-bogie arrangement which comprises two longitudinal side bogies, and a transverse bogie mounted across the rear. All the bogies are passively pivoted at central bearings and there are no springs. The 3-bogie arrangement is kinematically similar to the rocker-bogie system employed by all US Mars rovers in the fore-after longitudinal plane, but no transverse differential linkage is necessary so no part of the locomotion subsystem penetrates or crosses the rover body to link the left and right sides. The front of the rover is kept clear for accommodation and operation of the drill. Dog legs are incorporated in the rear transverse bogie beam to permit the WISDOM GPR horn antenna apertures to have a clear view of the ground, whilst ensuring the wheels are behind the horns to give good ground clearance when driving off step obstacles.

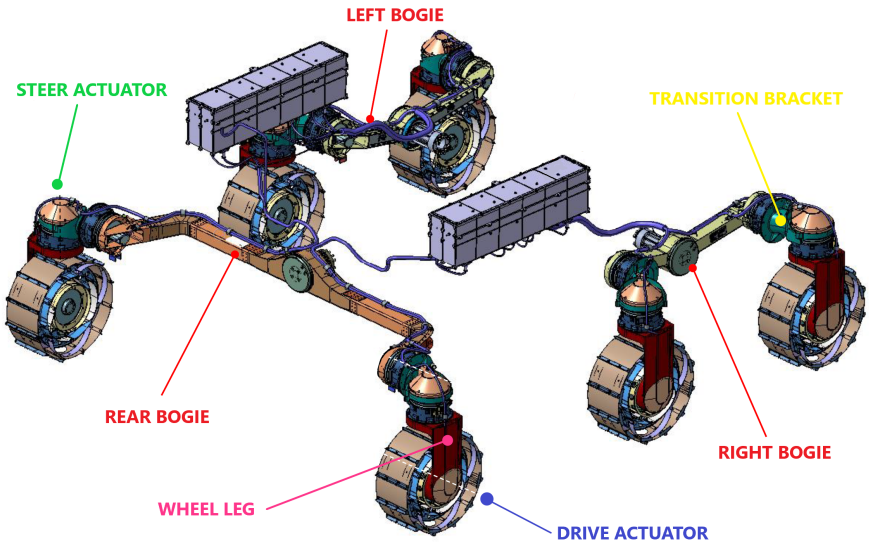


Figure 5.4: Real locomotion system of ExoMars rover that has to be modelled in MotionView *Locomotion* subsystem

The front to rear wheel separation is maximised within the constraints of the stowage envelope to confer good uphill and downhill longitudinal static and dynamic stability when descending obstacles forwards or in reverse. The longitudinal position of the middle wheels is aligned slightly forward of the expected longitudinal position of the rover centre of mass to equalise the wheel loading on level ground.

The pivots of the longitudinal side bogies are positioned centrally between the wheels. These are forward the front of the rover body, so the pivot shafts are supported by bogie attachment brackets cantilevered from the HDRM corner hard-points. The transverse bogie pivot is mounted to the same hard-point fitting which houses the rear body HDRM. All left-right wheel pairs have the same wheel track (distance between left and right wheel centers). The wheel track dimension is selected to provide the required lateral cross-hill stability. [4]

The wheels are flexible and their dimensions and radial stiffness are chosen to maximise traction and minimise energy consumption on soft soils, within the size constraints dictated by the rover stowage envelope. Flexible wheels provide the equivalent traction performance of a larger rigid wheel.

The MotionView *Locomotion* subsystem models the main elements involved in the ExoMars locomotion system, i.e. the bogie electromechanical actuators (BEMA) shown in Figure 5.4. It consists of wheels, lateral and rear bogie assemblies, drive and steer actuators.

Wheels

According to [15], the BEMA wheel is a flexible two-stage design. The first stage (*operational springs*) absorbs the operational loads to meet radial stiffness and contact pressure requirements. The operational springs are attached to the outer hub that has bump stop rings to limit the radial travel of the operational stage to loads of approximately $360N$. The rings also prevent the springs from deflecting out of plane. The second stage (*impact springs*) provides higher energy absorption capabilities to enable the system to limit loads during mobility impact events; thus, reducing impact to the actuator bearings. The impact springs connect the outer hub to the inner hub, which interfaces to the DRV actuator. Moreover, they allow the outer hub to move with respect to the inner hub. Each wheel also includes grousers to provide traction during operations.

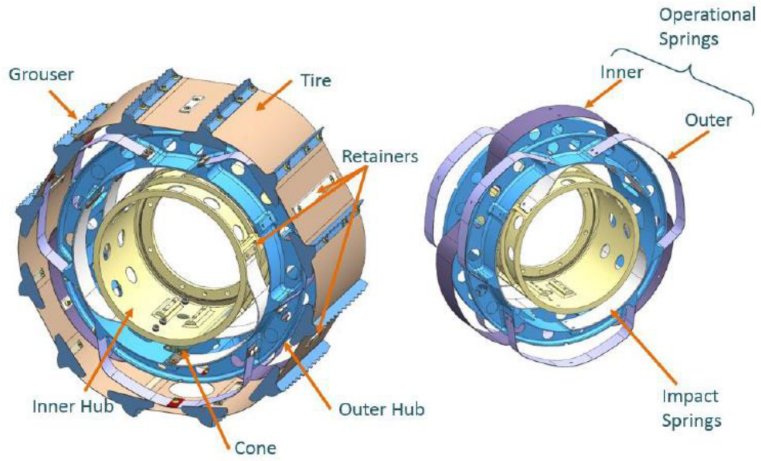


Figure 5.5: ExoMars Rover BEMA wheels

The overall wheel components are outlined in the Figure 5.5. The design is common to the left and right sides of the rover and the wheel assemblies are interchangeable with any other location on the rover. The BEMA wheel, excluding grousers, is 285 mm in diameter and 120 mm wide. There are a total of 9 operational springs per wheel; 6 outer springs and 3 inner springs. The outer springs are half the width of the inner ones but they are all the same thickness. To reduce stiffness variation as much as possible, the outer springs are mounted 60° out of phase to the inner springs.

The tire is made from a single sheet wrapped around the operational stage and attached each contact point with the operational springs. Each wheel includes 12 grousers

spaced at 30° around the circumference of the tire. Moreover, each grouser also includes a side grouser face to prevent lateral slip while the rover is cross hill driving.

The MotionView subsystem *Wheels* is composed of six point, six body and six graphic entities (actually the number of point and body entities is three since they were created as point and body pairs). Each body represents a wheel with center of mass defined by the corresponding point. In this thesis, the wheels are considered as rigid bodies, thus the graphic associated to each one of them is simply a cylinder of radius $r = 142,5\ mm$ and width $l = 120\ mm$ (these values are representative of the real ExoMars rover dimensions previously defined). The overall MotionView model of the wheels is reported in Figure 5.6.

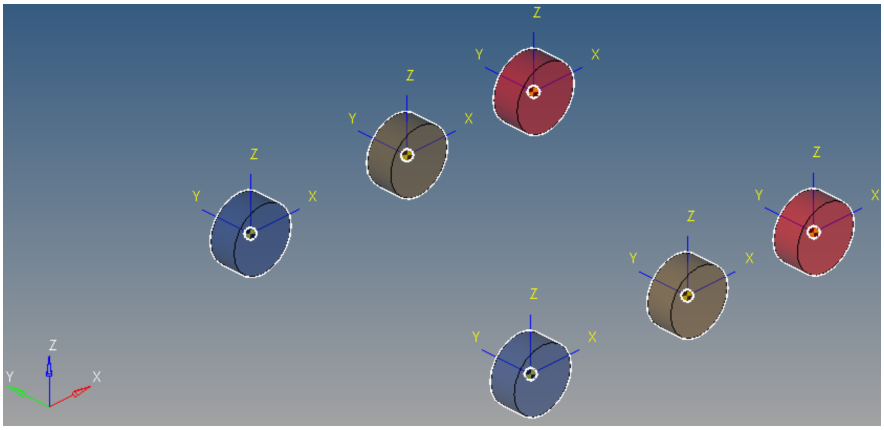


Figure 5.6: MotionView *Wheels* subsystem

Bogie

The *Bogie* subsystem models all the elements comprising the bogie assemblies, i.e. the two lateral beams and the rear beam, the wheel legs and the transition brackets. As for the wheels modelling, each component requires a point to define the center of mass of the body entity modelling it; then, graphics is added in order to visualize better the structure. They consist of simple cylinders of variable length and radius. The overall MotionView model of the *Bogie* subsystem is reported in Figure 5.7.

Joints

After having created and characterised all the bodies constituting the locomotion system, these have been linked together through joints. As defined in [28] "joints are used to specify an idealized connector between two bodies. Physically, the joint consists of two mating surfaces that allow relative translational and/or rotational movement in certain specific directions only. The surfaces are abstracted away, and the relationships

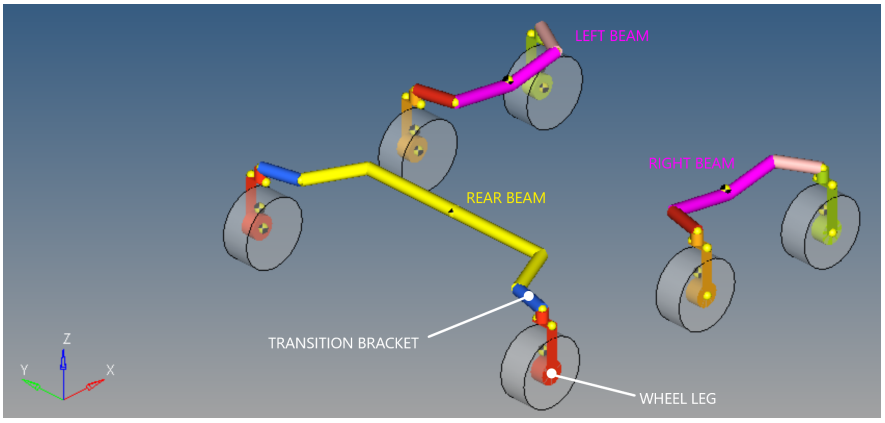


Figure 5.7: MotionView *Bogie* subsystem

are always expressed as a set of equations between points and directions on two bodies."Generally, constraints can involve displacements, velocities, and time; by contrast constraint-joints in MotionSolve involve displacements, but velocities or time are not explicitly involved. This means that they do not add or remove energy from the system.

A wide variety of joints with varying degrees of freedom is available to model system behaviour in MotionView environment:

InLineJoint constrains a point to be on a line

InPlaneJoint constrains a point to be on a plane

OrientationJoint constrains all rotational degrees of freedom

Parallel Axis Joint constraints two axis to be parallel

Perpendicular Joint constraints two axis to be perpendicular

For the purpose of this thesis, the necessary joints are :

- 6 revolute joints between wheel and relative leg, located at the wheel center where the DRIVE motors are. The rotational axis of these joints coincides with the wheel axis in order to allow the rotation around the *y axis* of the wheel reference frame.

  In order to manage this situation, one has to explicitly define the wheel reference frame in the BODY COORDSYS tab of the wheel body panel. Later, one has to set the ALIGNMENT AXIS of joint coincident to *y axis* of the just defined wheel RF.

  It is essential to refer to the wheel reference frame instead of the global one, since during motion the rover orientation could change, i.e. the rover longitudinal axis does not remain in the same direction of the global *x axis* and of course the wheel axis does no more coincide with the global *y axis*.

Note that also the cylinder graphics associated to wheel bodies must be defined setting the alignment axis coincident with the *y axis* of the wheel RF, such that when the rover move, or the wheels are steered, the graphics moves accordingly. This is a fundamental points, since contact (defined later in this section) are defined between graphics, thus they must be correctly set.

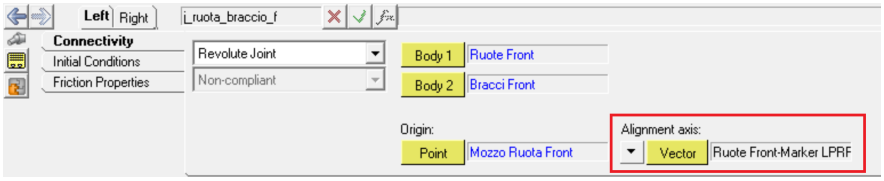


Figure 5.8: MotionView panel of the joint between front left wheel and corresponding leg

- 6 revolute joints between the leg and the relative transition bracket located where the STEER motors are. The joint axis have to be aligned to the *z axis* of the wheel reference frame. The same procedure explained for DRV joints have to be followed to force these two axes to be coincident.
- 6 fixed joints connecting the transition bracket to the beam
- 3 revolute joints, representing the pivots, connecting the lateral and rear beam to the rover body. In this case, the joint alignment axes are defined as the rover *y axis* and *x axis* for later joints and rear joint respectively.

The overview of the MotionView joints comprised in the *Locomotion* system is reported in Figure 5.9.

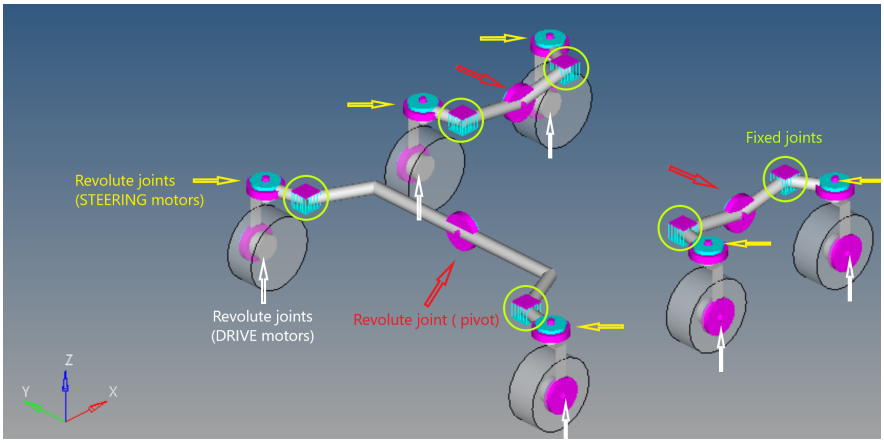


Figure 5.9: MotionView *Locomotion* joints overview

Contacts

Contacts are the most critical entities in the multi-body modelling procedure. MotionSolve provides a very sophisticated contact modelling capability that can handle complex scenarios between rigid bodies. [20]

In order to simulate rigid body contact, one needs firstly to identify the geometries of the two bodies that can come into contact with each other. There are several options for doing this. In the most simple way, we can use MotionSolve GRAPHICS entities. Moreover, complex geometries may be defined using a CAD tool and imported into MotionView. In the first case, graphics are tessellated automatically by MotionSolve, using 2-D triangular shell elements with no thickness; by contrast, if one needs specific geometry, it is necessary to mesh it before the import, taking care of create a close surface mesh. The quality of the mesh for the graphics strongly affects the quality and accuracy of the contact force calculated by MotionSolve.

Contacts are handled by MotionSolve as follows: when a collision event is detected, the dedicated algorithm returns the set of interfering mesh elements, from which MotionSolve computes the point of contact and surface normal vector; after, given the magnitude and direction of the normal and friction forces it computes the penetration depth.

The choice of the solver step size becomes important while trying to accurately capture the onset of first contact. If the step size is not small enough to detect the contact event, large penetrations may occur that result in large contact forces. However, small step sizes lead to long simulation time and they may be not necessary for the entire simulation. Thus, in order to have more realistic contact forces without reducing the global integration time, MotionSolve monitors the occurrence of contact and automatically changes the solver step size to accurately determine the first contact event.

Finally, the normal and friction force magnitudes are computed using one of the following models: [20]

Poisson model

$$F_n = kz^{\frac{3}{2}}(1 - \eta s) \tag{5.1}$$

where

F_n = normal force

k = penalty factor

z = penetration

$\eta = \dfrac{1 - COR^2}{1 + COR^2}$ with COR restitution coefficient

$s = step(\dot{z}, -\nu_t, -1.0, \nu_t, 1.0)$ with $\dot{z}$ penetration depth

Impact model The impact function models impact forces acting on bodies during collision. The elastic properties of the boundary surface between the two bodies can be tuned as desired. The function is

$$Impact(x, \dot{x}, x_1, k, e, c_{max}, d) \tag{5.2}$$

where

x = the independent variable

$\dot{x}$ = the time derivative of the independent variable

x_1 = the lower bound of the independent variable

k = the stifness of the boundary surface interaction

e = the exponent of the force-deformation characteristic

c_{max} = the maximum damping coefficient

d = he penetration at which the full damping coefficient is applied

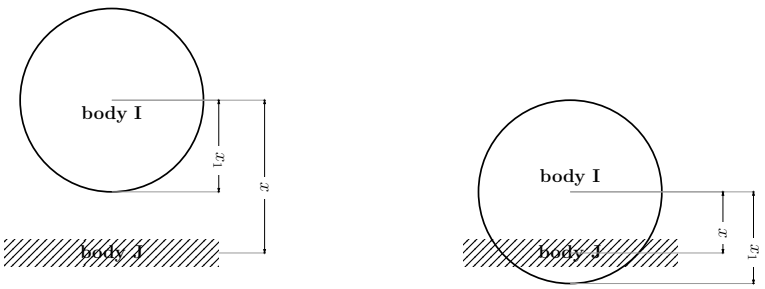


Figure 5.10: IMPACT model

Volume model this model assumes that both colliding bodies are surrounded by a layer of spring whose stiffness is determined by material elastic modulus properties and the depth of this layer. the normal force is modelled as the sum of two components: an elastic force (the same as in Impact model) and a damping force

$$F_{spring} = A_{con} K z^{exp} \tag{5.3}$$

$$F_{damping} = -c\frac{dz}{dt} \tag{5.4}$$

where

$$\begin{aligned} K &= \text{contact stiffness} \\ z &= \text{contact penetration depth} \\ exp &= \text{the exponent for the force-deformation characteristic} \\ c &= \text{the damping coefficient} \\ A_{con} &= \text{the area of contact} \end{aligned}$$

In this thesis the contacts between the rover wheels and the terrain is first modelled in the simplest way, using the Impact model without considering friction effects. The parameters are set in the MotionView contact panel as shown in Figure 5.11.

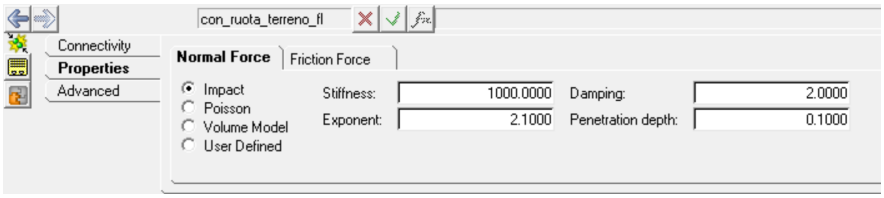


Figure 5.11: Contact panel of the wheel-ground contact with impact model parameters

The model accuracy shall be improved adding friction properties; however, it is a crucial parameter when modelling multi-body systems, since its wrong setting can results in unrealistic scenarios.

In particular, when introducing friction on the wheel-ground contact, we are adding a tangential component. When no friction is considered the contact force acts on the wheel only along the normal direction; by contrast, friction introduces a tangential component resulting in a deviation of the contact force. Problems come out when this tangential component becomes high enough to make the wheel rotate. This means that the rover moves without any force acting on it, that is completely unreasonable. In order to choose accurately the friction coefficients, several simulations have to be done

aiming to verify the tangential components do not introduce any absurd effect.

What have been done was to start from default values, run the simulation and look at the contact forces with post-processing software, *HyperView* and *HyperGraph*.

The first simulation was run adding friction to the previously described model whose parameters are shown in Figure 5.12. The model does not have any input, i.e. no forces act on the wheels or any other entity. The purpose of this test is to verify the correct application of contact forces.

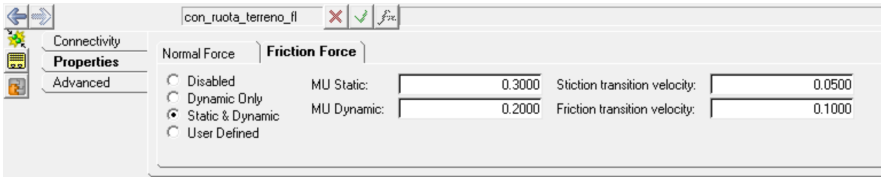


Figure 5.12: Contact panel of the wheel-ground contact with friction coefficients

Looking at the .h3d output file (animation) it was evident the results are completely unacceptable, since wheels start rotating, having no forces acting on them. The reason of this weird behaviour is explained in Figure 5.13.

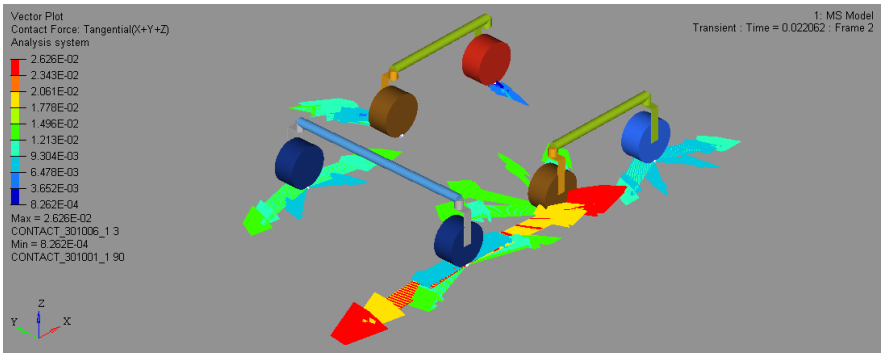


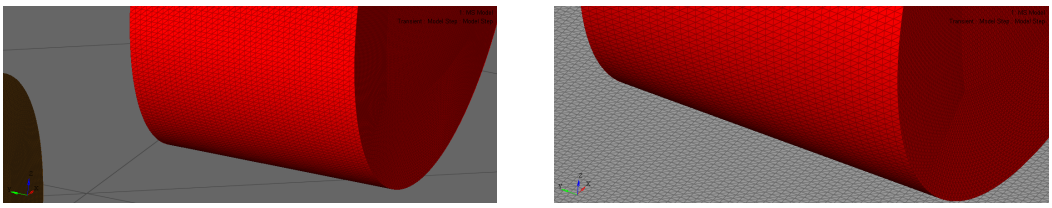
Figure 5.13: Simulation result highliting the wrong tangential components F_t of contact forces

During the fist time instant of the simulation the contact forces direction are completely wrong. It is reasonable to have tangential components, since we are in transient, so the contact still has to stabilize, but the direction of these ones have to remain along the longitudinal axis of the wheels.

The reason of this lack of correlation between simulation and reality is justified by the wrong mesh definition for the two elements in contact. This is clearly visible in Figure 5.14: the wheel elements have size of 2 mm while the elements of the ground are of size 30 cm. This is due to the fact that the contact takes place between two bodies whose graphics were defined in MotionView. Thus, as explained in before, the mesh is applied automatically by MotionSolve. This causes problems because the two

components have completely different sizes.

This issue was solved by increasing the mesh refinement level of the ground graphic. The same simulation is run and the new results are significantly improved: the contact force of each wheel has a tangential component only during the first time instants; once the contact is stabilized, the tangential component is null and the only acting forces are in the normal direction.



(a) Wrong mesh with element size $30cm$ (b) Correct mesh with element size $2mm$

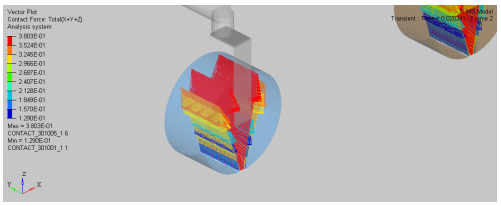
Figure 5.14: Comparison between different mesh refinement levels

Once the mesh was corrected and the resulting contact forces were become realistic, the following performed step was to make wheels rotate. In this new simulation condition one should verify if the tangential contact forces remain in the driving direction. The test was performed making the rover move starting from $t = 10\ s$; therefore, we can dived the simulation in three periods:

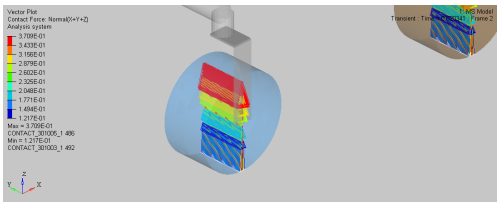
1. Transient: the contact is detected and the forces stabilize. It is expected to have both normal and tangential components of contact forces.
2. Rest period ($t < 10\ s$): before the application of the forces that make the rover move, it is expected to remain still. In this condition, contact forces acting on the wheels should be only in normal direction.
3. Driving ($t \geq 10\ s$): starting from $T = 10\ s$ the rover starts moving, therefore it is expected to have tangential components of contact forces, but only in the direction of the longitudinal axis of the wheel, i.e. only in the direction of motion.

The results reported in Figures 5.15 show the expected behaviour.

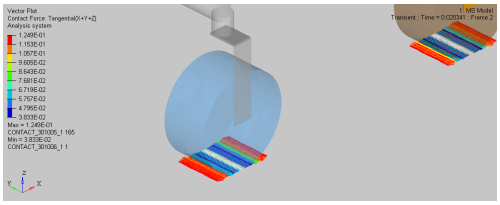
One could now consider the contact model to be correct. However, it does not represent any real and specific condition, since the contact parameters were set by default. In order to correctly correlate the model to reality it would be necessary a long study and characterization of the wheel-ground contact, but it is not the purpose of this thesis. Further studies on this challenging topic are therefore suggested in order to better correlate the model to real situations.



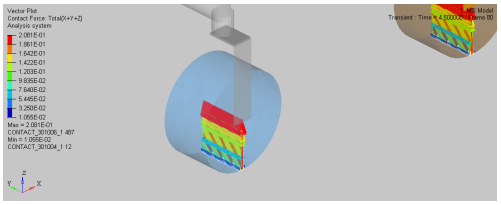
(a) Initial total contact force



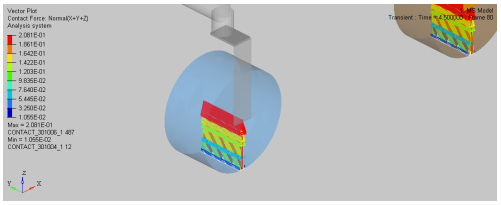
(b) Initial normal contact force



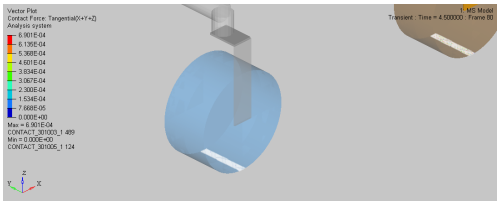
(c) Initial tangential contact force



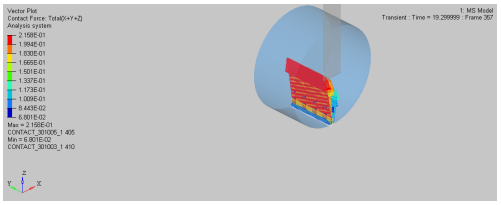
(d) Rest period total contact force



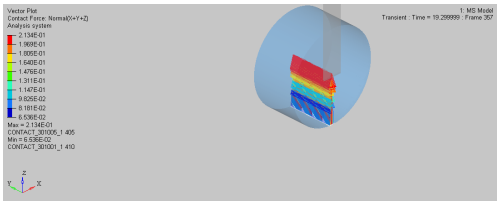
(e) Rest period normal contact force



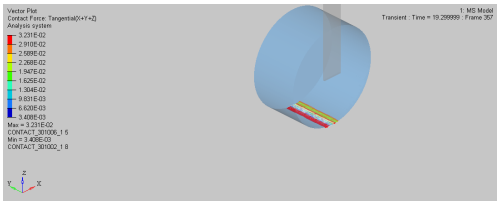
(f) Rest period tangential contact force



(g) Driving total contact force



(h) Driving normal contact force



(i) Driving tangential contact force

Figure 5.15: Simulation results highlighting friction force and its components during three different intervals

Actuators

The rover requires actuators for locomotion and steering, and obviously for deployment from the stowed configuration. The specific set of constraints for each of those operations necessitates three distinct actuator assemblies: dirve DRV, steer STR and deploy DEP. However, the overall BEMA actuator architecture is based on a brushed DC motor and planetary gearbox feeding into a Harmonic Drive (HD) at the low-speed/high-torque output; all actuators have the same overall geometry [15].

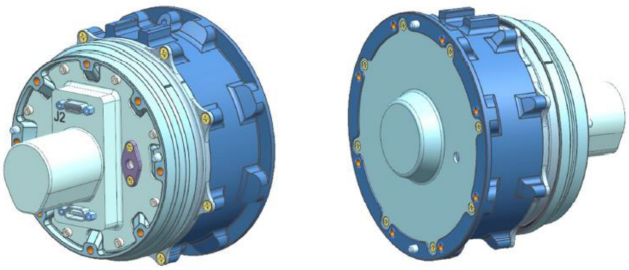


Figure 5.16: ExoMars Rover BEMA actuator

The output torque required from the actuator is different depending on its operation. The deployment operation is a one-time event which requires higher output torque than steering or driving. As a result the harmonic drive used in the DEP actuator has implemented a higher gear ratio. Deployment and steering require a sensor capable of indicating an absolute position, whilst drive operations only need a relative positioning sensor. This means that the DEP and STR actuators must include a potentiometer whilst the DRV actuator only includes an encoder (as part of the motor module).

In this thesis work, the deployment process was not of interest, therefore only drive and steering actuators have been modelled. The effect of these actuators have been considered adding MotionView forces entities acting on the wheels.

- DRIVE torques: 6 torques acting at each center of the wheels that make them rotate around their axis (*y axis* of the wheel reference frame).

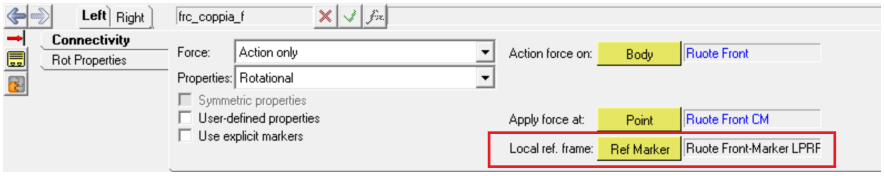


Figure 5.17: MotionView force/torque panel

As in the joint case, the usage of the wheel reference frame is a key aspect, since when modelling the drive torques, their axes must remain aligned with the

wheel rotational one, as the real actuators do, even if the rover is turning and its longitudinal axis is no more in the same direction of the global x *axis*; the force/torque panel is shown in Figure 5.17.

- STEERING torques: 6 torques acting on the wheel legs that allows the rotation of the wheels around their z *axes* in order to reach the required angular position that allow the rover follow a specific trajectory. .

The overview of the MotionView joints comprised in the *Locomotion* system is reported in Figure 5.18.

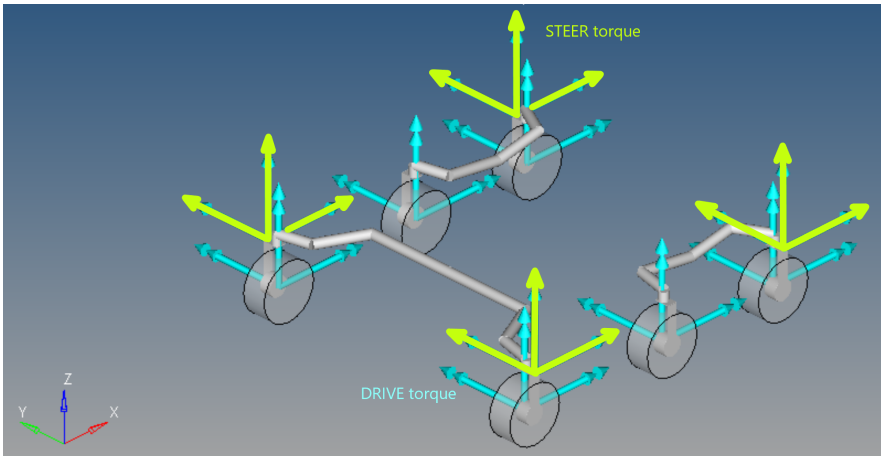


Figure 5.18: MotionView *Locomotion* force/torque entities overview

Sensors

In the development of this thesis, the sensor are not modelled. Actually, the variables that have to be measured, i.e. the wheel velocities and position, are computed by MotionSolve and sent to Activate adding SOLVER VARIABLES entities. The discussion about these variables will be presented in Section 5.6.

5.5.2 Rover body subsystem

The second subsystem constituting the ExoMars rover multi-body model is called *Rover Body*. It was decided to create a separate subsystem in order to collect the elementary entities composing the initial simplified model of the ExoMars body. Indeed, it was first modelled using basic elements available in MotionView library, in order to simplify the initial modelling and simulation. Thus, the first implementation is made up of six bodies:

- *rover*
- *head*

- *mast*
- *rear connection*
- *front-right connection*
- *front-left connection*

Boxes and cylinders graphics were associated to all these bodies. The finale model is reported in Figure 5.19.

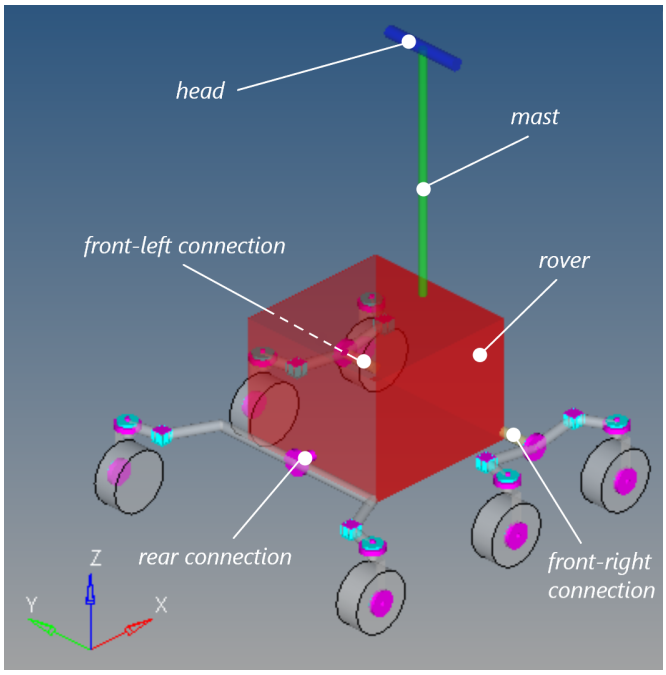


Figure 5.19: Complete simplified rover model build in MotionView

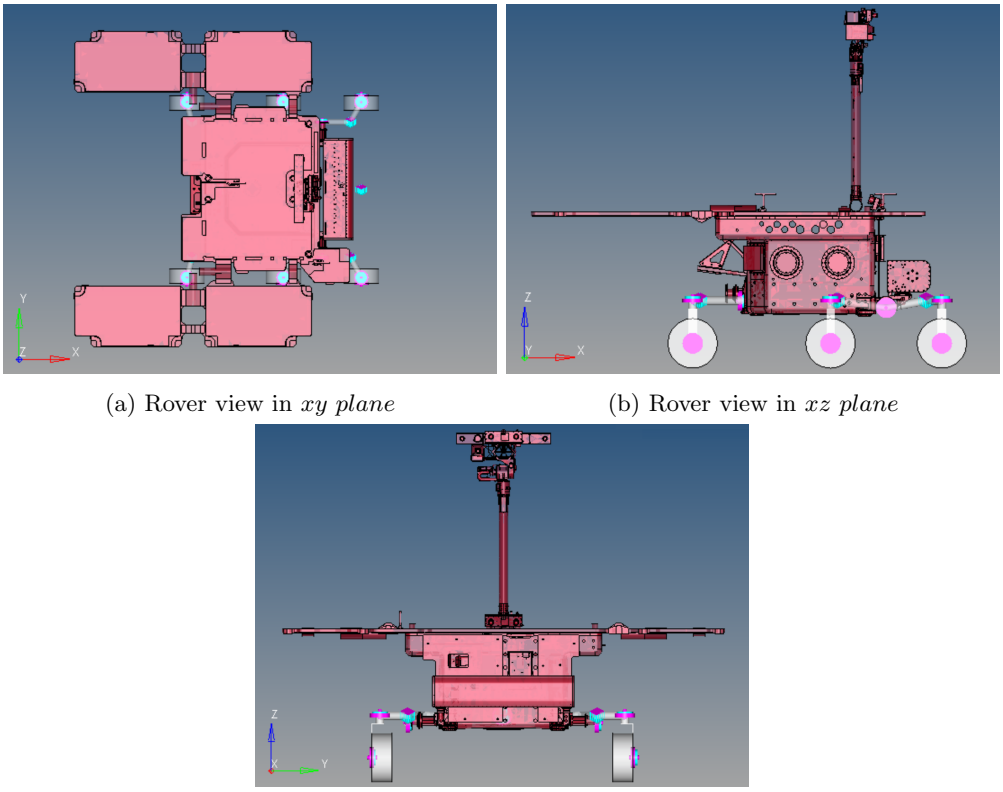
The real ExoMars rover body configuration is a single-piece moulded "bathtub" structure with Carbon Fiber Reinforced Polymer CFRP-faced honeycomb sandwich walls [2]. It provides the "warm" enclosure and structural support for several module, e.g. the Analytical Laboratory Drawer. The body size and shape are dictated by the accommodation of internal equipment and the constraints of the available stowage volume within the Descent Module; the Descent Module is a blunt capsule capable of landing on the Mars surface with a ballistic and controlled entry trajectory. The descent module's entry and deceleration through the Martian atmosphere will be assisted by

[2]Honeycomb structures are natural or man-made structures that have the geometry of a honeycomb to allow the minimization of the amount of used material to reach minimal weight and minimal material cost. The geometry of honeycomb structures can vary widely but the common feature of all such structures is an array of hollow cells formed between thin vertical walls. The cells are often columnar and hexagonal in shape. A honeycomb shaped structure provides a material with minimal density and relative high out-of-plane compression properties and out-of-plane shear properties.

heat shields and parachutes prior to a fully controlled descent and touchdown. The body height and width is driven by the accommodation of the ALD. The body length is constrained by the allowable envelope and the accommodation of the drill box at the front and WISDOM antennas at the rear.

At the rear of the body, a bracket provides the mounting for the WISDOM GPR antenna horns which overhang the rear bogie of the locomotion subsystem. The bracket is configured to provide sufficient space for rotational movement of the transverse rear bogie [4].

Having available the CAD model of the body structure, one shall modify the multi-body model and make it more accurate including those geometric informations. This is allowed in MotionView by adding a CADGRAPHICS entity, associated to the *Rover* BODY entity. Some modifications have to be done in order to match the geometry center (defined when modelling in CAD software) and the point defined in MotionView as the *Rover CM*. This was made performing some rotation and translation operations. The final results is shown in Figure 5.20



(a) Rover view in *xy plane*

(b) Rover view in *xz plane*

(c) Rover view in *yz plane*

Figure 5.20: Differnt views of the complete rover model build in MotionView with real body geometry

5.6 Modifying the model for co-simulation

The goal of this chapter is to describe the construction of a multi-body model of the ExoMars rover, such that it can be combined with a controls system model implemented in Activate. MotionSolve and Activate models can communicate with each other during co-simulation, but some arrangements have to be done in order to let the two software correctly communicate.

The basic MBS model described before has to be modified for co-simulation by adding MotionView entities called SOLVER ARRAYS and SOLVER VARIABLES. As defined in [20], the solver variables contain the individual plant input and output values, while the solver arrays define the plant input and output to communicate with Activate. In our case, the control acts on the drive and steering motors so that the wheels reach the desired rotational speed and they are steered as defined by the trajectory control.

Plant inputs Twelve solver variables have to be added to the model: six drive torques and six steering torques. These are collected in two solver arrays defining the control inputs, called *DRV Torques* and *STR Torques*. Then, we have to connect the solver variable to the TORQUES previously defined. This is done by setting the torque value in the property panel to

$$VARVAL(< solver.variable.id >)$$

This expression specifies the torque to be the solver variable specified in brackets <>. This variable sets the control signal sent from Activate to be the control force acting on the wheels.

Plant outputs Furthermore, other solver variables are required to specify model output. They have to be defined using the following MotionSolve functions:

(a) Wheel rotational and steering velocity:

$$\text{WY(I, J, K); WZ(I, J, K)}$$

The WY (WZ) function computes the y (z) component of the relative rotational velocity of marker I with respect to marker J, as resolved in the coordinate system of marker K. The first argument, marker I, must be specified. The second and third arguments, markers J and K, are optional. [20]

(b) Wheel steer angular position:

$$\text{AZ(I, J)}$$

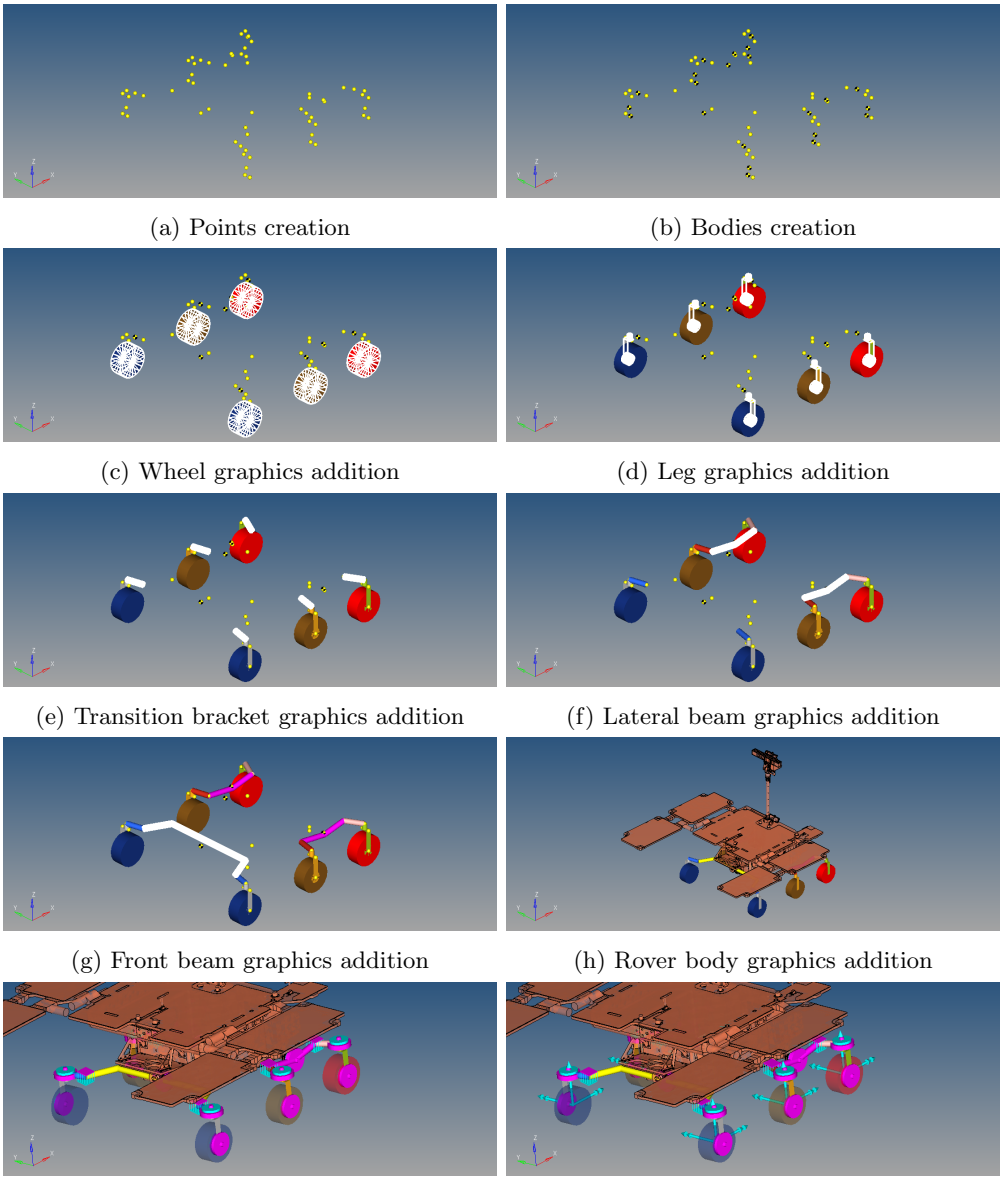
The AZ function computes the relative rotational displacement of marker I with respect to marker J about the z *axis* of marker J. The computed angle takes into

account the total number of revolutions between the two markers. While computing this angle, it is assumed that rotations about the other two axes of marker J (X and Z) are zero. Then, the AZ angle is the angle between the two X-axes measured counter-clockwise from the x *axis* of the J marker. The first argument, marker I, must be specified. The second argument, marker J, is optional. [20]

(c) Rover CM x and y coordinates and rover heading φ:

DX(I, J, K); DY(I, J, K); AZ(I, J)

The DX (DY) function computes the x (y) component of the relative translational displacement of marker I with respect to marker J, as resolved in the coordinate system of marker K. The first argument, marker I, must be specified. The second and third arguments, markers J and K, are optional. [20]



(a) Points creation (b) Bodies creation

(c) Wheel graphics addition (d) Leg graphics addition

(e) Transition bracket graphics addition (f) Lateral beam graphics addition

(g) Front beam graphics addition (h) Rover body graphics addition

(i) Joints creation (j) Forces/Torques addition

Figure 5.21: Steps of the model construction in MotionView

Chapter 6

THE ACTUATOR CONTROL

6.1 Introduction

In this chapter we will focus on the Actuator Drive Electronics (ADE) being part of the ExoMars rover Mobility System. Its main function consists in the actuator (motor) control loop process.

The first part of the chapter aims at describing the low level control loop, firstly presenting the general aspects of the PID control theory, and then illustrating how the actuator controllers have been modelled in Activate. The theoretic part refers to [9] and [10].

Usually, in control design courses, the reference signal generation is not considered, since typical signals, as step functions or sinusoidal are assumed, but here this topic has to be taken into account. Indeed, the actuator control system includes also a *Path Planner* (or *Profile Generator*) responsible of generating a profile for the reference variables sent to the controller. This component has to takes into account the constraints driven by the actuator limits. Therefore, the second part of the chapter concerns the generation of the profile, starting from the definition of the basic ones, and ending with a specific case, the trapezoidal velocity profile, which is the one used in this thesis. Finally a presentation of the Activate implementation is provided, with detailed explanation of the methodology followed to build the relative block diagram. For the theoretic concept one have made reference to [11].

6.2 ExoMars Rover Vehicle Mobility System Overview

The ExoMars mobility system is in charge of rover motion. It relies on a comprehensive and complex set of functions. The overview of the mobility system functional architecture is presented in Figure 6.1. It relies on the following elements [5]:

- A Bogie Electro-Mechanical Assembly (BEMA) which comprises the bogie struc-

tures including the wheels, wheel axes position and speed sensors. Actuation of the BEMA is necessary for the rover to move and follow the commanded path to the target.

- A set of sensors, which supply information about the rover state vector: absolute attitude, relative attitude, body angular velocity and angular acceleration, and position.
- An Actuator Drive Electronics (ADE) unit which is in charge of controlling the BEMA according to individual axis commands. It is the subject of this chapter.
- A data handling system (DHS) with its embedded Application Software which performs the management of numerical data and all computations required from the sensor processing to the elaboration and the transmission of the commands to the ADE through a CAN bus.
- A communications system to allow a command and telemetry interface to the Rover Operations Control Centre (ROCC).

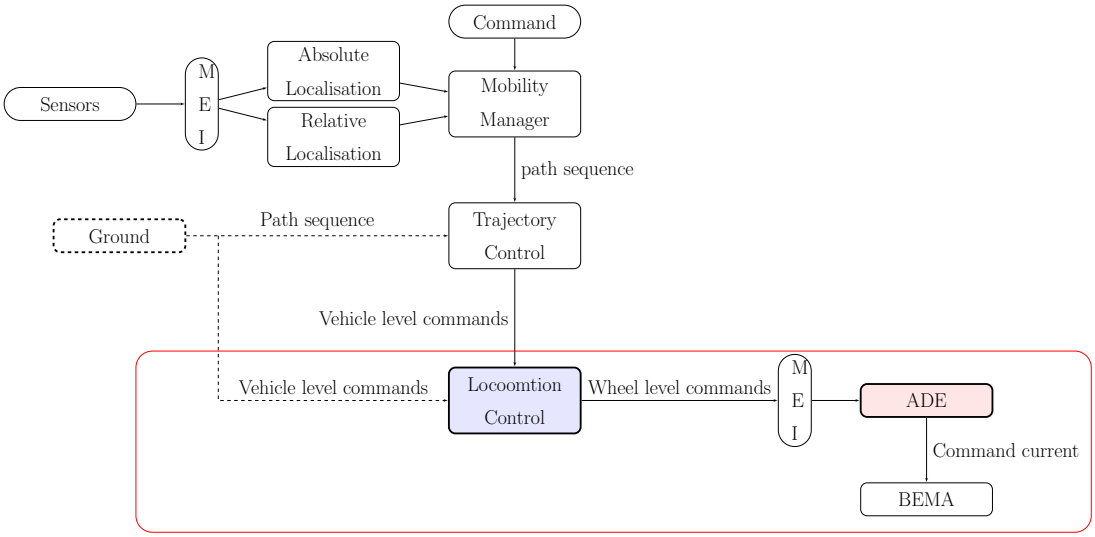


Figure 6.1: Overview of the mobility system functional architecture

Figure 6.1 comprises also the *Ground* block; indeed, Ground is allowed to send rover body level manoeuvre commands (i.e. Generic Ackerman, Generic Point Turn, Stop and respectively parameters) directly to the rover without the intervention of the Trajectory control. Moreover, Ground shall load a predefined path to the rover Trajectory Control function and request for its execution.

The red rectangle encloses the blocks modelled in this thesis. In particular, the red block ADE is the subject of this chapter.

The key functions of ExoMars rover mobility system are summarised as follows:

- Mobility equipment
  - Sensors
    - IMU (Inertial Measurement Unit) containing
      - Accelerometer: provides the linear acceleration vector
      - Gyroscope: provides the angular rates vector
    - Cameras (Navigation and Localisation): provide a stereo images of the surrounding environment
    - Pan and Tilt Mechanism sensors: provide the orientation of the mast head through ADE
    - Locomotion sensors: provide the locomotion axes angular position and/or angular rates through ADE
  - Actuators
    - Bogie Electro-Mechanical Assembly: 3 bogies connected to the rover body through a passive pivot each. Each bogie contains 2 legs with a wheel, with 3 active axes: wheel driving, wheel steering and leg deployment. The BEMAs are commanded by the ADE, performing the actuator closed-loop control, which will be discussed in this chapter.
    - Pan and Tilt Mechanism: though not part of the Mobility Subsystem it is used to point the NavCam at an aimed direction. It is located on top of the mast structure and the interfacing on the mast head.
- Mobility Equipment Interface (MEI): it interfaces between the Mobility equipment and the higher level mobility software functions.
- Localisation
  - Absolute localisation: uses accelerometer measurements to estimate the rover body pitch and roll on the Mars Local Geodetic frame by comparing the gravity measurement in this reference frame and the one in the rover body frame. The rover heading is uploaded by Ground [7].
  - Relative localisation: since absolute localisation does not provide rover position and cannot be executed whilst the rover is moving, relative localization is needed to propagate the attitude and the initial position (either zero or set by ground). This function includes a visual and a non-visual part [1].
    - Visual Localisation: is the most accurate estimate of the rover position and attitude when the rover moves. It is based on a pair of stereo images from the localisation cameras that are processed to find and track corners and/or edges on the images. The evolution of these features is

then processed by an estimator that determines the rover position and attitude. The processing is slower than the closed-loop trajectory control and therefore another estimator is needed between VisLoc estimates.

  - Non-Visual Localisation: is the fastest estimate of the rover position and attitude but also the least accurate due to slippage. It uses the measurements from the gyroscopes and the BEMA sensors to estimate the relative motion. Though the attitude estimate remains accurate, the position estimate may be very inaccurate when the rover slips since the sensors used in this module cannot detect it.

- Mobility manager: the mobility functionality is based around a set of mobility modes (Idle, Absolute Localisation, Direct Driving, Locomotion and Localisation Only, Path Traverse). The Mobility Manager is in charge of deciding the way modes are grouped, the sequence they are called with associated transitions, and the checking of transition conditions.

- Trajectory Control: its objective is to drive the rover along a specific path sequence, consisting of a series of path and point turn segments. The trajectory control execution calculates the vehicle level commands, i.e. rover velocity and heading, required to follow a path sequence closely [3].

- Locomotion Manoeuvre Control: since commands issued by trajectory control are vehicle level commands and the actual mobility actuators are the wheels with their driving and steering actuators, they must be translated into wheel level command. The locomotion manoeuvre is in charge of taking vehicle level commands and adequately transforming them into driving and steering commands. This transformation also aims at ensuring that at any point in time all actuators are in a compatible configuration, i.e., not fighting each other [2].

- Locomotion Deployment Control: it is used for BEMA deployment operations. BEMA deployment is a one-off activity performed to stand the rover up on the lander platform. This control generates commands for the DEP and DRV actuators in order to deploy all six legs simultaneously minimising rover body tilt in the lander platform frame.

6.3 The ExoMars Actuator Drive Electronics

The Actuator Drive Electronics unit is in charge of controlling the BEMA according to individual axis commands. The ExoMars rover is equipped with two identical actuator drive electronics. One ADE is installed in the right side of the rover (ADE-R) and the other in its left side (ADE-L). The ADEs perform the control of most of the rover

actuators , by implementing the control operations required to make the motors move at the required velocity or toward the required position. To achieve this, the ADE will receive commands issued by the on-board computer to move the actuators in response to the higher level mobility algorithms [27].

6.3.1 ExoMars Actuator Control Algorithm Overview

The diagram shown in Figure 6.2 provides an overview of the algorithm involved in the actuator control loop.

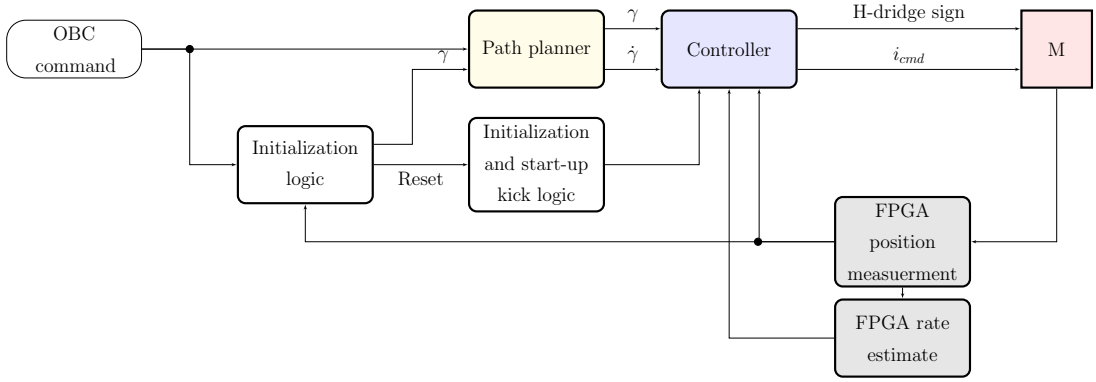


Figure 6.2: Actuator's algorithm overview

The **Initialization Logic** block governs when the path planner and control law are reset. [14] Essentially when a new command is received from the on-board computer (OBC) and the system is not moving (or direction of motion is reversed) the path planner and controller are reset to take advantage of the kick-start (Hard Reset). Otherwise only the path planner is reset (Soft Reset). Thus, the initialization logic is responsible for two things:

1. Handling incoming on-board computer commands and determining if the path planner must be entirely reset (kick-start the MOT control law) or only partially reset

2. What temperature dependent values should be

The **Path Planner** is in charge of generating a profile, according to the on-board computer target, and respecting the acceleration/speed constraints. [14] Data in the path planner flows as follows:

1. The parametrized allowed second rate of change $\ddot{x}$ is used to increase the magnitude of the live command first rate of change $\dot{x}$

2. Once the first rate of change has reached it's maximum magnitude it is commanded to hold this value

3. When the value of the present state minus the state of interest x is less than the value of the area of the triangle created by the slope of $\ddot{x}$ and $\dot{x}$, than the $\dot{x}$ is reduced by the $\ddot{x}$ to a 0 value.

This behaviour is better explained in Figure 6.3

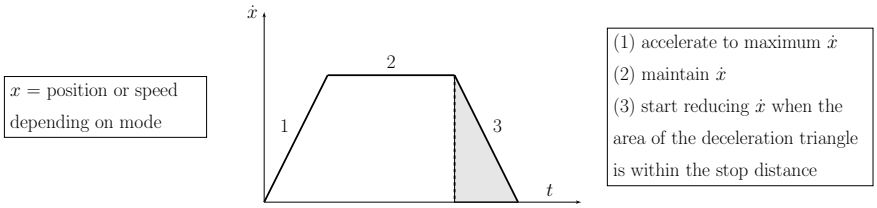


Figure 6.3: Path planner behaviour

The **Controller** receives position/speed target demands generated by the path planner, and according to a specific control law computes the command current sent to the DC motors driving the rover wheels [14].

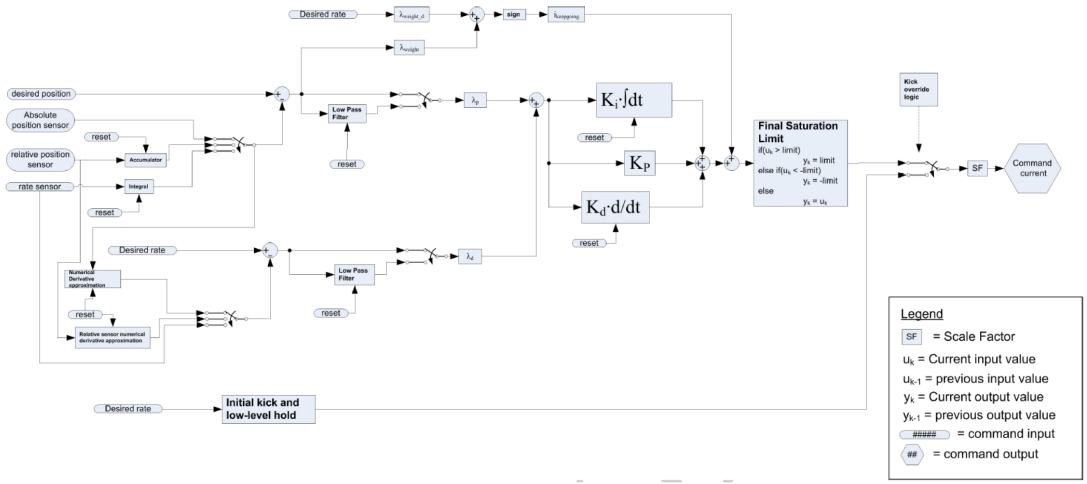


Figure 6.4: ExoMars Controller block diagram

The nominal operation is the following:

1. Matched position and rate commands arrive at the controller and are compared against the signals measurements

2. The difference between the desired and measured signals is then taken and signals are optionally low-pass

3. Both the rate and position errors are then multiplied by a weighting factor prior to being added together.

4. The summed signal is then put through a standard PID.

5. In parallel to calculation of the PID a weighted rate is calculated based on the sum of the desired rate and position error multiplied by weight terms. The sign of this weighted rate is then multiplied by a constant to calculate the friction boost and added to the output of the PID. This feed forward action is added in order to compensate friction disturbance, according to the following control law:

$$i_{cmd} = sign(\lambda_{weight_d}\dot{\gamma_d} + \lambda_{weight_p}(\gamma_d - \gamma))i_{keepgoing} \tag{6.1}$$

   where $i_{keepgoing}$ is a parametrised current value that approximately keeps the motor moving in the presence of friction.

6. The combined output is then limited.

7. Final output either uses this value directly or overrides with the kick-start logic

8. As a final step the signal can be negated and h-bridge sign set by the H-bridge logic.

The design of the controller block diagram includes several sources of feedback and possibility to inject or not low-pass filters. They are all included in order to allow flexibility of re-configuration in mission. Moreover, it has been included a weighted error signal through a common controller, in order to allow weight sensors differently, depending on their relative accuracy (e.g. on the STR axis the motor side rate sensor is more accurate than the joint side position sensor).

Finally, the **Initialization and start-up kick logic** block is in charge of providing a kick-start to the actuator, in order to cope with detent-brake characteristics of some of the Actuators.

As already stated, this thesis aims at develop a model implementing the main function of the ExoMars control, focusing on the ADE functionalities. Referring to Figure 6.1 we want to construct the model corresponding to the systems comprised in red window. The inputs are considered as vehicle level commands directly set by ground, while the outputs are the torques applied to the wheels of the MotionSolve rover model; therefore, the *BEMA* block corresponds to the MotionSolve one.

The ADE controller has been modelled in Activate as two separate blocks: a *Profile Generator* and a *Controller*. The complete Activate model is reported in Figure 6.5, where the grey blocks will be subject of the following sections, while the *Inverse kinematics* block will be described in detail in Chapter 7.

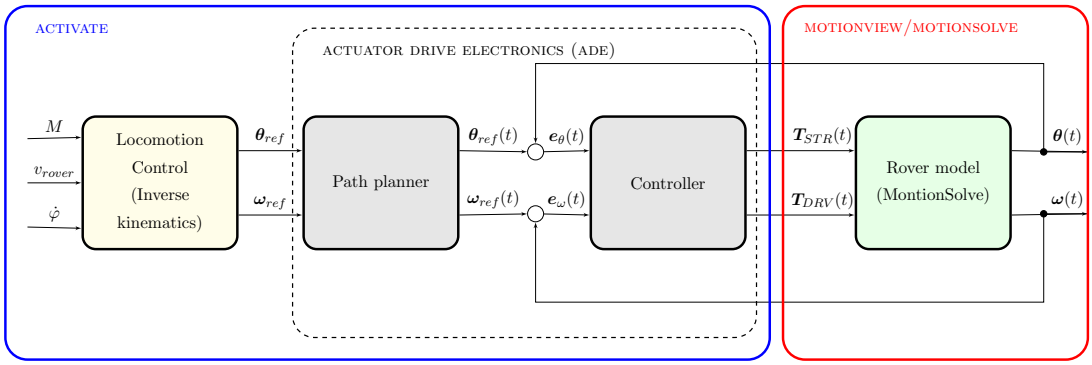


Figure 6.5: General scheme of the complete model built for co-simulation

6.4 The Controller

Considering the overall Activate model presented in Figure 6.5, the *Controller* receives the reference values for the steering angles and driving velocities from the *Profile Generator*, which generates a trapezoidal profile for these signals according to higher level commands. Based on their control laws, the controllers generate the command torques to be applied to the steering axes or drive axes of the wheels.

The first implementation of the control algorithm was based on the usage of PID controllers associated to the steering motors, which takes as input the error

$$\boldsymbol{e}_{\theta}(t) = \boldsymbol{\theta}_{ref}(t) - \boldsymbol{\theta}(t)$$

where $\boldsymbol{\theta}_{ref}(t)$ are the reference steering angles computed by the *Profile Generator* and $\boldsymbol{\theta}(t)$ are the current ones computed by MotionSolve; and PID controllers associated to the drive motors, which receive as input the error

$$\boldsymbol{e}_{\omega}(t) = \boldsymbol{\omega}_{ref}(t) - \boldsymbol{\omega}(t)$$

where $\boldsymbol{\omega}_r(t)$ are the reference wheel velocities and $\boldsymbol{\omega}(t)$ are the current ones.

In the next section the PID control theory is presented with reference to [9].

6.4.1 PID Control Theory

Having argued in the previous chapter that the actuator controllers have been implemented as PIDs, here a brief overview of the PID control theory is reported, making reference to [9].

The PID controller is the most common control loop feedback mechanism and it is widely used in industrial control systems. A PID controller continuously calculates an *error* $e(t)$ as the difference between a desired reference variable, often called *setpoint* $r(t)$ and a *measured process variable* $y(t)$, and applies a *control signal* $u(t)$, that is the sum of three terms:

- P-term proportional to the error
- I-term proportional to the integral of the error
- D-term proportional to the derivative of the error

The PID algorithm is described by equation (6.2) and a block diagram of a PID controller in a feedback loop is represented in Figure 6.6.

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)\, d\tau + K_d \frac{de(t)}{dt} = K_p \left(e(t) + \frac{1}{T_i} \int_{t_0}^{t} e(\tau)\, d\tau + T_d \frac{de(t)}{dt} \right) \quad (6.2)$$

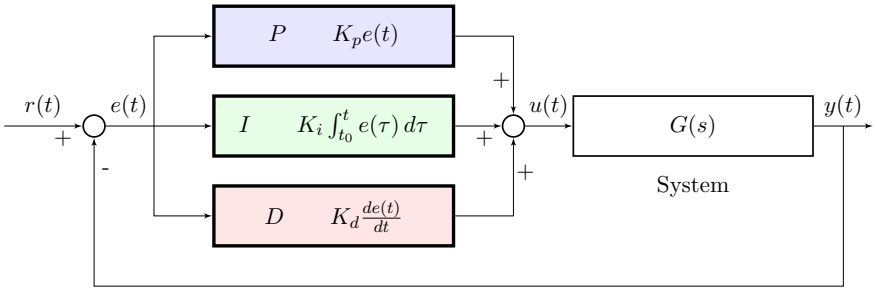


Figure 6.6: Block scheme of a control loop with proportional-integral-derivative action

The parameters characterizing the control are the following: proportional gain K_p, integral time T_i (or integral gain K_i), derivative time T_d (or derivative gain K_d).

Applying the Laplace Transformation to the previous equation (6.2), we obtain the following Transfer function:

$$\frac{U(s)}{E(s)} = L(S) = K_p + \frac{K_i}{s} + K_d s \quad (6.3)$$

Effect of proportional action

$$u(t) = K_p e(t)$$

The proportional control is obtained when $K_i = K_d = 0$, i.e. $T_i = \infty$ and $T_d = 0$. The control output will be proportional to the error, the larger the error the larger the control action, taking into account the gain factor K_p. Thus the P control leads always to a steady-state error between the setpoint and the actual signal value, because the controller requires an error to generate the proportional response. If there is no error, there is no correction.

The characteristic parameter K_p has a theoretical meaning, but this control type can be also defined from a practical point of view with a parameter called *proportional band* B_p. It is defined as the minimum variation of the input $e(t)$ in percentage, that makes the output $u(t)$ change from the minimum value to the full scale value.

When K_p is defined as the ratio between the signal normalized with respect to its full scale value, as in (6.4), it is possible to use relation (6.5) between B_p and K_p, from which it will be clear that increasing the proportional band will lead to an increase of the precision, i.e. it will be sufficient a smaller variation in the error to have a significant change in the output control signal.

$$K_p = \frac{u(t)/u(t)_{fullscale}}{e(t)/e(t)_{fullscale}} \tag{6.4}$$

$$K_p = \frac{100}{B_p} \tag{6.5}$$

Therefore, reducing the proportional band, the proportional gain will increase, improving the precision of the controller (the control response will be faster); moreover, the proportional gain increase corresponds to an increase of the output value reached by the P block. This value will gradually distance less and less from the reference value, but never reaching it due to the steady state error; it can be decreased increasing the gain, but this could lead to instability, forcing the output to oscillate. P controllers are acceptable only for those applications in which variances between the effective value of the controlled variable and the desired one are allowed.

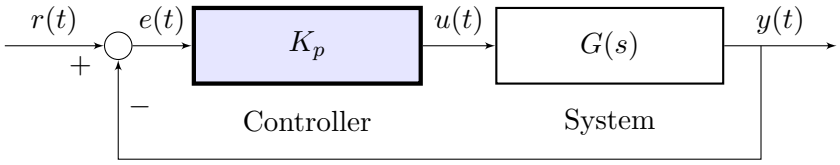


Figure 6.7: Block scheme of a control loop with proportional action

What was described before is now demonstrated from a mathematical point of view. Considering the general block scheme with a plant and a controller as in Figure 6.7. For example, we can associate to it functions with the following expressions:

$$R(s) = \frac{1}{s} \; ; \; G(s) = \frac{A_0}{1 + s\tau} \; ; \; C(s) = K_p$$

The transfer function of the closed loop system is thus:

$$W(s) = \frac{Y(s)}{R(s)} = \frac{C(s)G(s)}{1 + C(s)G(s) * 1} = \frac{\frac{K_p A_0}{1+s\tau}}{1 + \frac{K_p A_0}{1+s\tau}} = \frac{\frac{K_p A_0}{\tau}}{s + \frac{1+K_p A_0}{\tau}} \tag{6.6}$$

Applying to (6.6) the unitary step we obtain the following output signal:

$$Y(s) = R(s)W(s) = \frac{1}{s} \frac{\frac{K_p A_0}{\tau}}{s + \frac{1+K-pA_0}{\tau}} \tag{6.7}$$

The error function is:

$$E(s) = R(s) - Y(s) = \frac{1}{s} \frac{1 + s\tau}{1 + s\tau + K_p A_0} \tag{6.8}$$

Applying the residual theorem and the inverse Laplace Transform we obtain the system time response:

$$y(t) = \left(\frac{K_p A_0}{1 + K_p A_0} \right) \left[1 - e^{\frac{1+K_p A_0}{\tau} t} \right] \delta_{-1}(t) \tag{6.9}$$

From (6.9) it is evident that the increase of the proportional gain K_p will result in an increase of the gain $\left(\frac{K_p A_0}{1+K_p A_0} \right)$ tending toward 1, and at the same time the time constant of the exponential term will be reduced. In this way, the output value will be more and more closed to the desired one and the time required to reached this level will be smaller and smaller. This last is the *Rise Time* and it is the parameter characterising the readiness of the controlled system. However, the only way to reach the exact reference value is to have $K_p \to \infty$.

The steady state error characterises the system *accuracy*, i.e. the ability of following the reference signal at steady state. In order to analyse the accuracy we have to take into account some system characteristics:

- Steady state gain of the loop transfer function
- Error transfer function
- *System type number*: this parameter is essentially the number of poles at the origin of the *s plane*, and it can indicate whether the steady state error of the system will be zero, or a constant value, or infinity according to the input.

The system is of *type h* if the transfer function has a pole of multiplicity h at $s = 0$
It is possible to evaluate the steady state error with the following procedure:

$$e_\infty = \lim_{t \to \infty} e(t) = \lim_{s \to 0} sE(s) \tag{6.10}$$

with $E(s) = W_e(s)R(s)$

In the case of our example, $W(s)$ has no poles in the origin, thus the system is of type 0, and the steady state error is

$$e_\infty = \lim_{s \to 0} sE(s) = \frac{1}{1 + K_p A_0} \tag{6.11}$$

One more time, it is evident that it is required a infinitely large K_p to have $e_\infty = 0$.

Usually, the control specifications about the steady state accuracy impose constraints on the type of the open loop system, i.e. on the number of poles the open loop

transfer function, and on the minimum steady state gain of the closed loop transfer function.

Since the open loop transfer function is $C(s)G(s)$, known the properties of $G(s)$, the specifications constrain the number of poles the controller must have and its minimum steady state gain.

Let's consider another example, in which the system to be controlled is characterised by a second order transfer function:

$$R(s) = \frac{1}{s} \; ; \; G(s) = \frac{A_0}{1 + 2\zeta\left(\frac{s}{\omega_n}\right) + \left(\frac{s}{\omega_n}\right)^2} \; ; \; C(s) = K_p$$

Analogously to the previous example, the closed loop transfer function is:

$$W(s) = \frac{Y(s)}{R(s)} = \frac{C(s)G(s)}{1 + C(s)G(s) * 1} = \frac{K_p A_0}{1 + K_p A_0 + 2\zeta\left(\frac{s}{\omega_n}\right) + \left(\frac{s}{\omega_n}\right)^2} \tag{6.12}$$

And the error transfer function is:

$$E(s) = \frac{1}{s} \frac{1 + 2\zeta\left(\frac{s}{\omega_n}\right) + \left(\frac{s}{\omega_n}\right)^2}{1 + K_p A_0 + 2\zeta\left(\frac{s}{\omega_n}\right) + \left(\frac{s}{\omega_n}\right)^2} \tag{6.13}$$

From (6.13) we obtain the steady state error:

$$e_\infty = \lim_{s \to 0} sE(s) = \frac{1}{1 + K_p A_0} \tag{6.14}$$

Also in this case, there is a steady state error that decreases when K_p increases.

Effect of integral action

$$u(t) = K_i \int_{t_0}^{t} e(\tau)\, d\tau$$

The integral control is obtained when $K_p = K_d = 0$. The integral action accounts for past values of the error and integrates them over time to produce the integral term. Usually, the integral action is associated to the proportional one in order to obtain the so called PI controller.

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)\, d\tau = K_p \left(e(t) + \frac{1}{T_i} \int_{t_0}^{t} e(\tau)\, d\tau \right) \; ; \; U(s) = K_p \left(1 + \frac{1}{T_i s} \right) E(s)$$

The main function of the integral action is to make sure that the process output agrees with the setpoint in steady state. If there is a residual error after the application of proportional control, the integral term starts growing, adding a control effect that

tries to eliminate it. Since the error decreases, the proportional effect decreases too, but this effect is compensated by the growing of the integral action. So, the steady-state error disappears when using PI control, but increasing the integral action could lead to instability. Indeed, the proportional term introduces a pole and this leads to a worsening in terms of stability.

When using this type of controller, we need to tune the T_i variable, i.e. the integral time constant also called *reset time*. It is responsible of the integral action: the response grows faster towards the setpoint for small values of T_i.

Consider for example a system made up of a plant to be controlled and a PI controller, that receives as input a step signal r(t):

$$R(s) = \frac{1}{s}\ ;\ G(s) = \frac{A_0}{1+s\tau};\ C(s) = K_p\left(1 + \frac{1}{T_i s}\right)$$

The closed loop transfer function is therefore:

$$W(s) = \frac{Y(s)}{R(s)} = \frac{C(s)G(s)}{1 + C(s)G(s) * 1} = \frac{1}{1 + \frac{sT_i(1+s\tau)}{A_0K_p(1+sT_i)}} \tag{6.15}$$

And the error is:

$$E(s) = \frac{1}{s}\frac{sT_i(1+s\tau)}{sT_i(1+s\tau+A_0K_p) + A_0K_p} \tag{6.16}$$

As before we can compute the steady state error:

$$e_\infty = \lim_{s\to 0} sE(s) = 0 \tag{6.17}$$

Effect of derivative action

$$u(t) = K_d\frac{de(t)}{dt}$$

The integral control is obtained when $K_p = K_i = 0$. The derivative action is a best estimate of the future trend of the error, based on its current rate of change. The purpose of the derivative action is to improve the closed loop stability.

The PI parameters are chosen in such a way that the output of the system oscillates; successively, in order to damp the oscillations, the derivative term is introduced. Increasing the derivative time T_d will increase the damping, but if T_d becomes too large the derivative action ceases to be effective.

The derivative action introduces a zero in the origin of the *s plane*; this means a phase lead.

6.4.2 PID tuning operation

An important advantage of PID controllers is the possibility of use them without knowing in details the model of the system to be controlled. Users of control systems are frequently faced with the task of adjusting the controller parameters to obtain a desired behaviour. There are many different ways to do this. Since the PID controller has so few parameters, a number of special empirical methods have also been developed for direct adjustment of the controller parameters.

One among the developed techniques is the rule developed by Ziegler and Nichols. Their idea was to perform a simple experiment, extract some features of process dynamics from the experiment and determine the controller parameters from these features.

Anyway, the simplest way to perform PID tuning is to manually change the parameters. Manual control is the method used in the development of this thesis to set the parameters of the PID controllers responsible of the ExoMars rover actuator control.

Manual loop tuning

This is a simple method of tuning a PID controller. First of all, the system is configured in closed loop, the PID sends the control output $u(t)$ to the system, but the integral and derivative gain are set to zero. The proportional gain K_p is increased until the response oscillates and is fast enough.

After that, the integral term is set, so that the oscillation are gradually reduced. The I-parameter is also chosen in such a way that the steady state error is eliminated. This step will probably increase the overshoot, that usually is undesired, or limited.

Overall, the readiness and the overshoot of the system response are regulated increasing the derivative term.

6.5 Some clarifications

In the following sections it will be presented the tuning operation of the STR and DRV PID controllers and relative results. Some clarification should be given. As highlighted in Chapter 5, contact entities play an important role in the multi-body modelling. In order to build a first general model, the contacts have been modelled without considering specific real cases, keeping the default parameter values and verifying that the contact force are reasonable; thus, the PIDs are tuned accordingly.

Obviously, this does not properly represent real situations; however, it allows to construct a simple overall model that can be eventually improved and customized to more detailed scenarios. This is the great advantage of the developed model: it is simple to adapt it to specific analysis. For instance, if one desires to simulate the movement of the ExoMars rover on the Martian soft-soil terrain, it is sufficient to take the correct

MotionSolve model, with the accurate contact modelling, load it into the developed model and consequently tuning the PID controllers, without any other modification.

Certainly, the great challenge remains the development of the correct wheel-ground contact model.

6.5.1 Drive motors PID tuning

The PID controllers associated to the drive motor of the ExoMars rover have been tuned manually. In the Activate model they receive as input the error between the reference wheel velocity, computed by the prifile generator, and the current wheel velocity computed by MotionSolve. The PID gains have been tuned trying different solutions, and the several simulations results are reported in the following sections.

All tests were taken considering a straight line as trajectory, therefore the reference speed is the same for all the wheels and they behave in similar way. In particular, the input to the Inverse Kinematic block is the desired rover velocity

$$v_{rover} = 0.03\ m/s$$

By consequence, the reference wheel angular velocities computed by the Inverse Kinematics block are

$$\omega_i = 0.25\ rad/s$$

.

This is the value of the reference signal for all the drive PID controllers. Only the results relative to the front right wheel are reported, but all other wheels behave in the same way. The simulation parameters are set as illustrated in Chapter 4. As already stated, the tuning procedure is not exactly the standard one because of the simplification made as regard as contact model.

P term tuning

First of all the PID gains have been chosen to obtain a simple proportional action. The simulation results reported below have been obtained setting different values of K_p: 100, 500, 1000, 5000, 10000. The effect of the proportional gain K_p increase, is highly evident in Figure 6.8, where only the front right wheel velocity is plotted. The gain increase results in a increase of the readiness of the response, but a steady state error is always present. For this reason, an integrative term is inserted. Its tuning is described in the following paragraph.

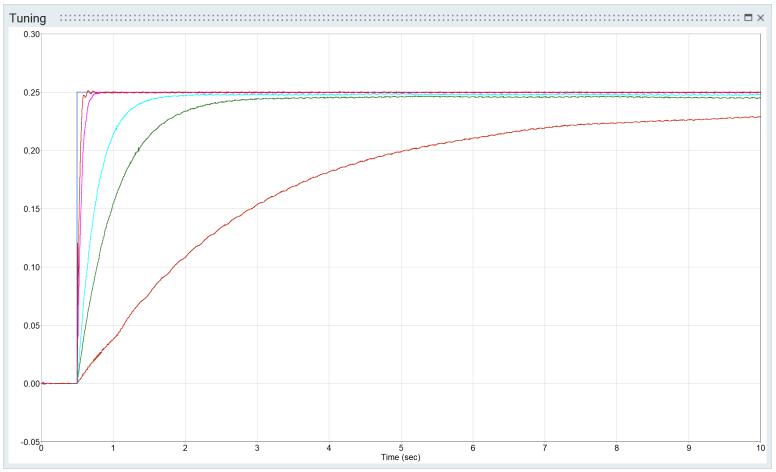


Figure 6.8: Output DRV rotational speeds resulting from a P control

I term tuning

For the I-term tuning, the simulations were done setting the proportional gain $K_p = 2000$, which value gives a reasonably acceptable result, as described in the previous section; whereas, the integral gain K_i is increased starting from 0 till 100. The output velocity of the front right wheel is plotted in Figure 6.9 which compares the results between simple proportional action and proportional-integrative action. Obviously, the steady state error can be eliminated by introducing the integral term.

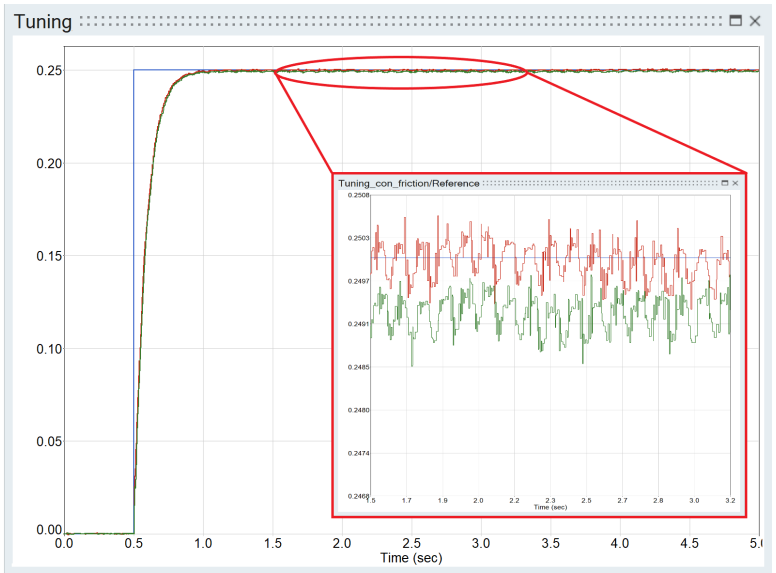


Figure 6.9: Output DRV rotational speeds resulting from a PI control

D term tuning

In this particular case the addition of derivative action is not needed, therefore the K_d term is null.

6.5.2 Steering wheel motor PID tuning

As already stated, the tuning procedure is not exactly the theoretical one, since the low accuracy of the contact model leads to strange behaviours. As regards the steering controllers, the PIDs are tuned being already connected to the profile generators; thus, the references signals for the PIDs are not step functions, but trapezoidal ones.

All tests were taken keeping still the rover and setting the input reference angular position to $\frac{\pi}{6}$. This signal is taken by the Profile Generator which gives as output a trapezoidal reference signal.

Therefore, the reference is the same for all the wheel axes and they behave in similar way. Only the results relative to the front right wheel are reported, but same applies to all the other ones. As already stated, the tuning procedure is not exactly the standard one because of the simplification made as regard as contact model.

P term tuning

First of all, the P term has been tuned. Starting from $K_p = 1000$, the values is increased till $K_p = 50000$. The resulting output angular positions are reported in Figure 6.10.

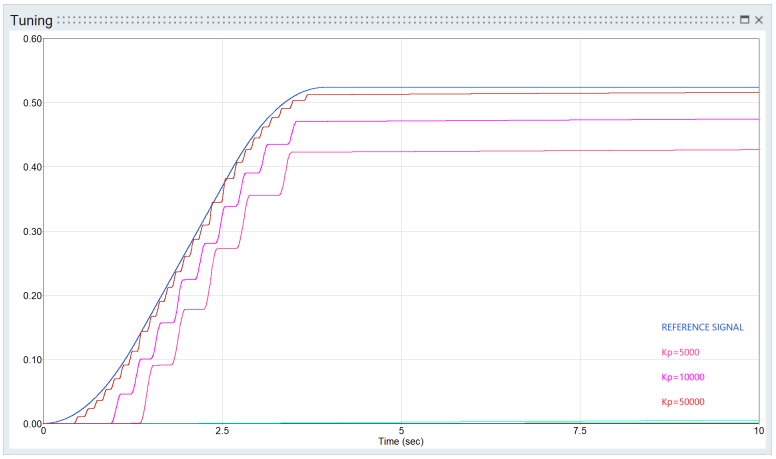


Figure 6.10: Output STR angular positions resulting from a P control

I tern tuning

Considering the results obtained in the previous section with only proportional term, it is evident that if one wants to reach the reference value it is needed to add an integral term. Figure 6.11 shows the resulting outputs with

$$K_p = 50000 \text{ and } K_i = 1000; 5000; 10000$$

.

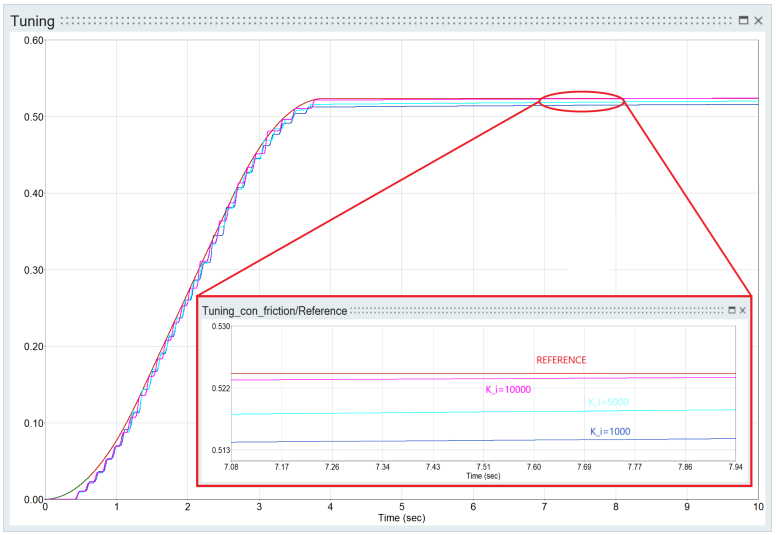


Figure 6.11: Output STR angular positions resulting from a PI control

D term tuning

Also in this case the derivative action is not needed, thus $K_d = 0$.

6.6 The Path planner

This section is dedicated to the the *Path Planner*, the second constitutive block of the actuator control. The Path Planner is a software node that, given the target values of the control variables, the kinematic constraints (maximum speed) and the dynamic constraints (maximum accelerations, maximum torques), designs the time law of the input variables. In the developed software, the *Path Planner* is a block located between the *Inverse Kinematics* block, from which receives the target values, and the *Controller* block, to which sends at any given time the current value.

The time law chosen for the ExoMars control was the so called *Trapezoidal velocity*; an example of continuous profile of a generic variables $s(t)$ is illustrated in Figure 6.12.

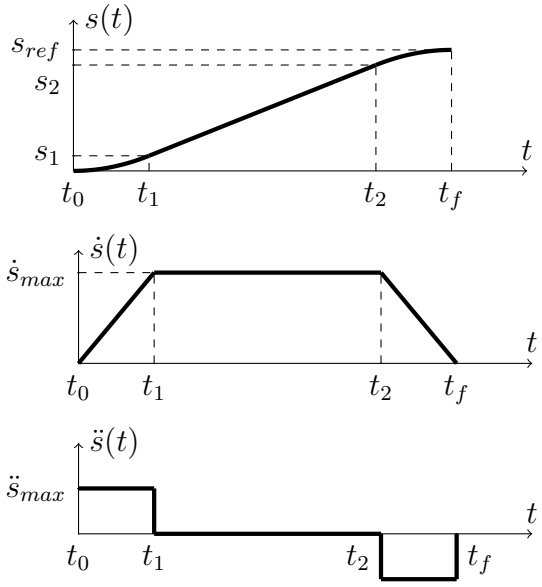


Figure 6.12: Example of a continuous trajectory with velocity and acceleration constraints

With reference to [11], we start defining how the trapezoidal velocity profile shall be defined and generated. This specific case can be consider as the combination of two more simple profiles, the so called *Linear Profile* and *Parabolic Profile*; therefore, one should start analysing them, and later applying the same concepts to the trapezoidal profile.

6.6.1 Trapezoidal Velocity Profile

A general motion can defined by assigning the initial and final time instant t_0 and t_f, and conditions on position, velocity and acceleration at those instants. From a mathematical point of view, the problem is then to find a function

$$s = s(t),\ t \in [t_0, t_f]$$

such that the given constraints are satisfied. This problem can be solved by considering a polynomial function

$$s(t) = a_0 + a_1(t) + a_2(t)^2 + ...a_n(t)^n$$

where the degree n of the polynomial, and thus the $n+1$ coefficients a_i, is depending on the number of constraints to be satisfied and the desired "smoothness" of the resulting motion.

Linear trajectory - Constant velocity

The most simple profile is the linear trajectory (constant velocity), defined as

$$s(t) = a_0 + a_1(t - t_0)$$

where t_0, t_f, $s(t_0)$ and $s_{ref} = s(t_f)$ are known. Thus the parameters a_0 and a_1 are computed by solving the following equations:

$$\begin{cases} s(t_0) = s_0 = a_0 \\ s(t_f) = s_f = a_0 + a_1(t_f - t_0) = a_0 + a_1 T \end{cases}$$

where $T = t_f - t_0$ is the time duration.

therefore

$$\begin{cases} a_0 = s_0 \\ a_1 = \frac{s_f - s_0}{t_f - t_0} = \frac{h}{T} \end{cases}$$

where $h = s_f - s_0$ is the displacement.

An example of linear continuous trajectory is presented in Figure 6.13.

Parabolic Trajectory - Constant acceleration

Another usual profile is the parabolic trajectory, characterized by an acceleration with a constant absolute value and opposite sign during the acceleration/deceleration periods. Mathematically, it is defined by two second degree polynomials, one from t_0 to t_1 (acceleration phase) and the second from t_1 to t_f (deceleration phase), where t_1 is the *flex point*.

Considering the case of a symmetric trajectory with respect to the middle point, we have:

$$t_1 = \frac{t_0 + t_f}{2}, \qquad s(t_1) = \frac{s_0 + s_f}{2}$$

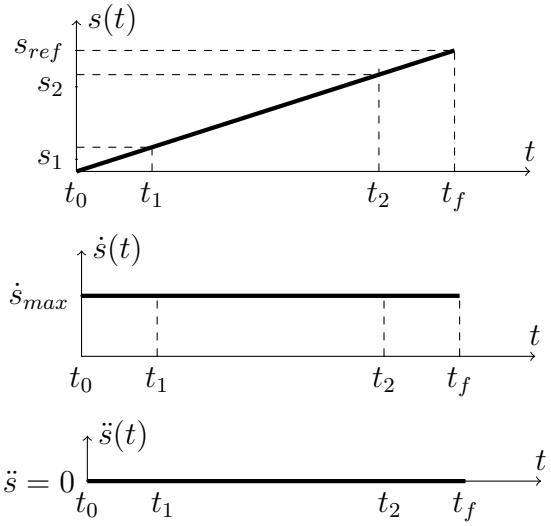


Figure 6.13: Example of a continuous linear trajectory

1. The first phase is described by

$$s(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2,\ t \in [t_0, t_1]$$

The parameters a_0, a_1 and a_2 are computed by imposing the constraints of the trajectory, i.e. the points s_0 and s_1, and the condition on the initial velocity $v(t_0) = v_0$

$$\begin{cases} s(t_0) = s_0 = a_0 \\ s(t_1) = s_1 = a_0 + a_1(t_1 - t_0) + a_2(t_- t_0)^2 \\ \dot{s}(t_0) = v_0 = a_1 \end{cases}$$

obtaining

$$a_0 = s_0, \qquad a_1 = v_0, \qquad a_2 = \frac{2}{T^2}(h - v_0 T)$$

with

$$T = 2T_1 = 2(t_1 - t_0),\ h = 2h_1 = 2(s_1 - s_0)$$

Therefore the trajectory in phase1 is defined as

$$\begin{cases} s(t) = s_0 + v_0(t - t_0) + \frac{2}{T^2}(h - v_0 T)(t - t_0)^2 \\ \dot{s}(t) = v_0 + \frac{4}{T^2}(h - v_0 T)(t - t_0) \\ \ddot{s}(t) = \frac{4}{T^2}(h - v_0 T) \end{cases}$$

2. The second phase is described by

$$s(t) = a_3 + a_4(t - t_0) + a_5(t - t_0)^2,\ t \in [t_1, t_f]$$

The parameters a_3, a_4 and a_5 are computer by assigning the final velocity v_f at $t = t_f$ and solving the following system

$$\begin{cases} s(t_1) = s_1 = a_3 \\ s(t_f) = s_f = a_3 + a_4(t_f - t_1) + a_5(t_f - t_1)^2 \\ \dot{s}(t_f) = v_f = a_4 + 2a_5(t_f - t_1) \end{cases}$$

obtaining

$$a_3 = s_1 = \frac{s_0 + s_f}{2}, \qquad a_4 = 2\frac{h}{T} - v_f, \qquad a_5 = \frac{2}{T^2}(v_f T - h)$$

Therefore the trajectory in phase2 is defined as

$$\begin{cases} s(t) = s_1 + (2\frac{h}{T} - v_f)(t - t_1) + \frac{2}{T^2}(v_f T - h)(t - t_1)^2 \\ \dot{s}(t) = 2\frac{h}{T} - v_f + \frac{4}{T^2}(v_f T - h)(t - t_1) \\ \ddot{s}(t) = \frac{4}{T^2}(v_f T - h) \end{cases}$$

An example of parabolic continuous trajectory is presented in Figure 6.14.

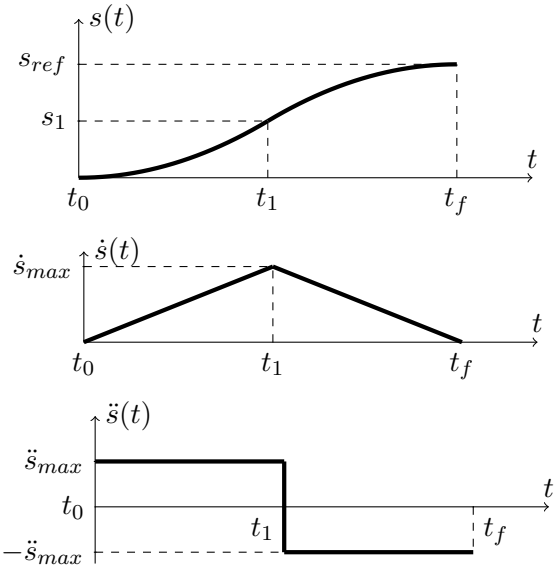


Figure 6.14: Example of a continuous parabolic trajectory

Once the characteristics of these basic profiles are understood, it is easy to note that the trapezoidal velocity trajectory mentioned before, is nothing but the combination of three polynomial profiles, of second order, first order and second order again (2-1-2):

Phase 1 Constant acceleration in the interval $[t_0, t_1]$

Phase 2 Constant velocity in the interval $[t_1, t_2]$

Phase 3 Constant deceleration in the interval $[t_2, t_f]$

We divide the trajectory in three time intervals defined as:

$$\mathcal{I}_1 := t : t_0 \leq t < t_1 = [t_0, t_1)$$

$$\mathcal{I}_2 := t : t_1 \leq t < t_2 = [t_1, t_2)$$

$$\mathcal{I}_3 := t : t_2 \leq t \leq t_f = [t_2, t_f]$$

Thus the time laws defining $s(t)$ in the corresponding intervals are:

- Acceleration equation:

$$\begin{cases} \ddot{s}_{max} & t \in \mathcal{I}_1 \\ 0 & t \in \mathcal{I}_2 \\ -\ddot{s}_{max} & t \in \mathcal{I}_3 \end{cases}$$

- Velocity equation:

$$\begin{cases} \ddot{s}_{max}(t - t_0) + \dot{s}_0 & t \in \mathcal{I}_1 \\ \dot{s}_{max} & t \in \mathcal{I}_2 \\ \dot{s}_{max} - \ddot{s}_{max}(t - t_2) & t \in \mathcal{I}_3 \end{cases}$$

- Position equation:

$$\begin{cases} \frac{1}{2}\ddot{s}_{max}(t - t_0)^2 + \dot{s}_0(t - t_0) + s_0 & t \in \mathcal{I}_1 \\ \dot{s}_{max}(t - t_1) + s_1 & t \in \mathcal{I}_2 \\ -\frac{1}{2}\ddot{s}_{max}(t - t_2)^2 + \dot{s}_{max}(t - t_2) + s_2 & t \in \mathcal{I}_3 \end{cases}$$

In the case of our problem, we desire to make the rover start a certain manoeuvre at a certain time t_0. We considered the case of STR control, therefore we have to define the time law of the steering angles $\boldsymbol{q}(t)$. The target final value $\boldsymbol{q}(t_f)$ is generated by the inverse kinematics block, and it is the reference value is $\boldsymbol{q}_{ref}$, i.e. the angular position with respect to the *z axis* the steering axes must assume in order to perform

the desired manoeuvre. Then, considering that each manoeuvre starts end finishes with zero velocity, we have the following boundary conditions:

$$q(t_0) = q_0, \qquad \dot{q}_0 = 0, \qquad q_f = q_{ref}, \qquad \dot{q}_f = 0$$

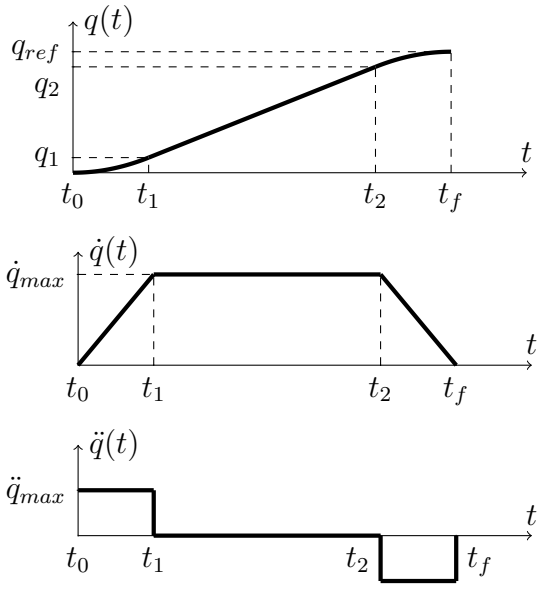


Figure 6.15: Example of a continuous trapezoidal velocity trajectory

Considering the continuity constraints at instants t_0,t_1,t_2,t_f, we can compute the following quantities:

$$q_1 = q_0 + \frac{1}{2}\ddot{q}_{max}(t_1 - t_0)^2$$

$$q_2 = \dot{q}_{max}(t_2 - t_1) + q_1$$

$$q_f = q_0 + \frac{1}{2}\ddot{q}_{max}(t - t_0)^2 + \dot{q}_{max}(t_2 - t_1) - \frac{1}{2}\ddot{q}_{max}(t_f - t_2)^2 = q_{ref}$$

$$\dot{q}_1 = \dot{q}_2 = \dot{q}_{max} = \ddot{q}_{max}(t_1 - t_0)$$

$$\dot{q}_f = \dot{q}_{max} - \ddot{q}_{max}(t_f - t_2) = 0$$

We can define a variable λ as a fraction of the total period T in which the acceleration and deceleration phases happen.

$$\lambda = \frac{T_1 - T_3}{T} \tag{6.18}$$

where T_1 and T_3 are the durations of the acceleration and deceleration phases respectively.

Then we define the acceleration and velocity coefficients C_a and C_v respectively, that allows to compute the time intervals the trajectory is composed of.

$$C_v = \frac{1}{1-\lambda}, \qquad C_a = \frac{1}{\lambda(1-\lambda)} \tag{6.19}$$

The parameter λ is defined at the beginning according to the desired trajectory; therefore C_a and C_v are known.

Then, the total trajectory duration T can be computed as:

$$T = \sqrt{\frac{q_0 - q_{ref}}{\ddot{q}_{max}} C_a}, \qquad \text{or} \qquad T = \frac{q_0 - q_{ref}}{\dot{q}_{max}} C_v \tag{6.20}$$

When implementing a profile generator for industrial manipulator, the time T in which the robot performs its task, is computed choosing the formula of (6.20) that gives the greater result in order to consider the more limiting constraint between velocity and acceleration ones.

Since, in our case we have only velocity constraints for STR control, we choose the second definition of T reported in (6.20). Once parameter T is known, one shall compute the maximum acceleration as

$$\ddot{q}_{max} = \frac{q_0 - q_{ref}}{T^2} C_a \tag{6.21}$$

Moreover, we introduce the normalized time variable

$$r = \frac{t - t_0}{T} \tag{6.22}$$

where t_0 is the starting time of the profile.

The pseudo-code of the algorithm for the computation of $\dot{q}(t)$ is here reported:

```
if (r<0) % phase_0
    q_d=0
elseif (r>1) % phase_4
    q_d=0
elseif (r<=lambda) % phase_1
    q_d=q_dd_max(t-t0)
elseif (r>=1-lambda) % phase_2
    q_d=q_d_max
else % phase_3
    q_d=q_d_max-q_dd_max((t-t0)-(T-lambda*T))
```

The algorithm was converted in the *Activate* model reported in Figure 6.16, 6.17, 6.18.

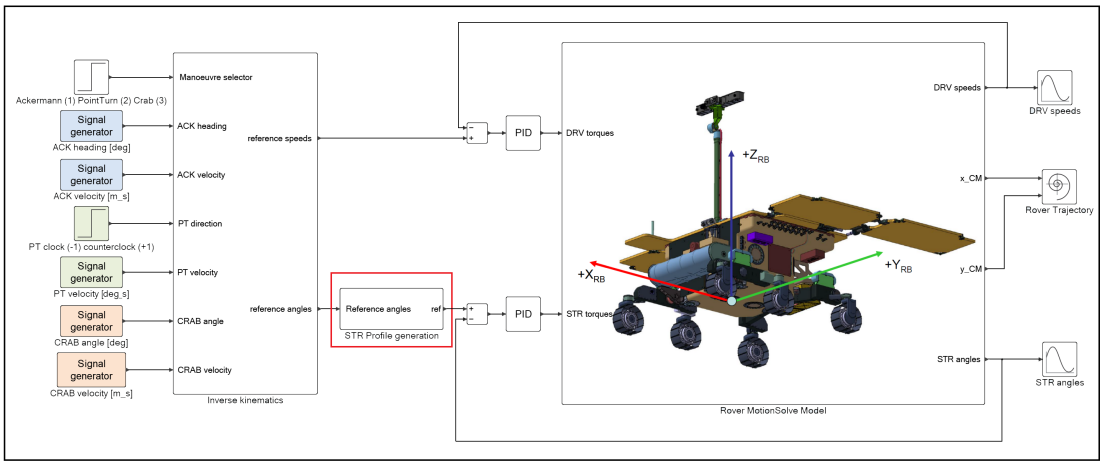


Figure 6.16: Complete Activate model. The *Profile Generator* (Path Planner) is highlighted in red

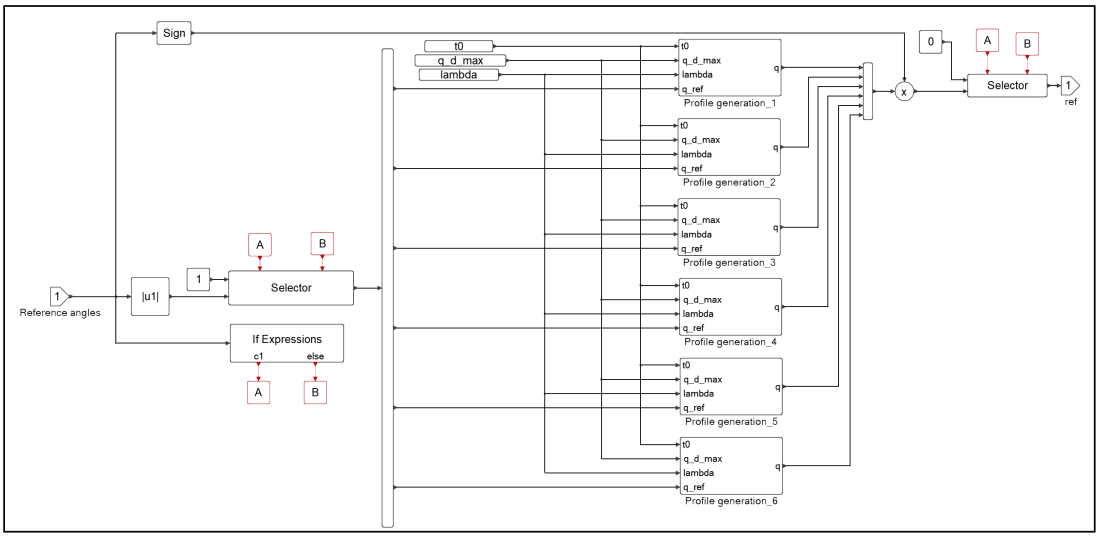


Figure 6.17: General Activate model of the *Profile Generator*

Figure 6.16 illustrates the general Activate model, where the *Profile Generator* is highlighted; looking inside that component one will find the block scheme reported in Figure 6.17.

It comprises six equal blocks each one corresponding to a rover wheel. The reason why it was implemented in this way is that when certain manoeuvre are commanded, e.g. Ackerman steering and Point Turn, the two blocks corresponding to the two middle wheels can be bypassed, since they must not be steered, and as consequence the value of the reference steering angle is null. Null values of reference will lead the solver to

not converge, indeed

$$Reference\ angle = 0 \quad \Longrightarrow \quad T = 0 \quad \Longrightarrow \quad s = \infty$$

However, by bypassing the two blocks when needed this unwanted situation will be avoided. Anyway, the two blocks are necessary when Crab manoeuvre is commanded.

Moreover, other conditions could lead to have null reference values also for front and rear wheels. In particular, when the user commands a straight motion, all wheels have to remain aligned with the longitudinal axis of the rover, i.e. their steering angles must remains at zero. In order to manage this error, an IF EXPRESSION block is inserted in the model.

According to he user manual [25], this block implements the conditional construct IF-THEN-ELSEIF-...-ELSE-END; one and only one of the output activation ports is fired synchronously when the block is activated. In our case, block A is activated if and only if $Reference\ angle = 0$.

Block A is called SETACTIVATIONSIGNAL; it sends output activations to a corresponding GETACTIVATIONSIGNAL block by connecting the event ports of the blocks without using a link. The activation signals A and B are used to control the so called SELECTINPUT, which selects one of its inputs and copies it into its output, when activated. The block determines the selected port depending on the way by which it has been activated. If the activation is received on its i-th input activation port (the top left port being numbered one), the i-th regular input is copied to the output (top most port being numbered one). In case the block is activated synchronously through more than one input activation ports, the lowest numbered port is taken into account, others are ignored.

In our case, *Reference angle* signal is sent to the *Profile generator* block if and only if it is non null. By contrast, when the reference is zero, an auxiliary non-null value is sent to the *Profile Generator*. This one always works producing an output profile, based on either the real reference value or the auxiliary variable. For this reason, a selector is required before the output port. It is controlled by the same activation signals A and B of the input selector; when B is active the output is connected to the profile computed by the profile generator; by contrast, when A is active, the profile generators are disconnected and the selector set the output to zero.

Once the general scheme is understood, let us enter inside the *Profile Generator* block implementing the algorithm previously described. It is shown in Figure 6.18.

The inputs of this block are:

- Starting time t_0

- Reference steering angle q_{ref}
- Maximum velocity $\dot{q}_{max} = 0.4\ [rad/s]$
- Variable λ

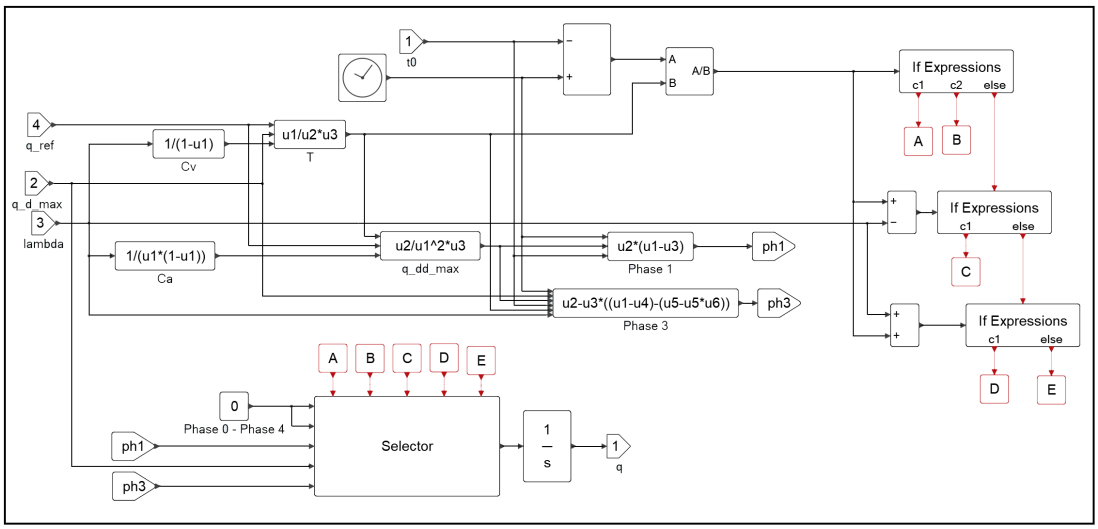


Figure 6.18: Activate implementation of the *Profile Generator* algorithm

Also in this case a SELECTOR block is needed. It is controlled by five control signals: A, B, C, D, E, each one corresponding to one phase of the profile.

A = phase0

B = phase4

C = phase1

D = phase2

E = phase3

Note that even if the name of the set signal variables are the same as before, they do not cause any problem since these are local variables and the tow block schemes are created at two different levels.

Anyway, the variables A,B,C,D and E are activated by three IF EXPRESSION blocks implementing the algorithm previously described. The first IF EXPRESSION (on the top) receives as input the normalised time variable $r = \dfrac{t - t_0}{T}$ and according to its value sends a proper output signal through one and only one among its output ports. In particular, if $r < 0$ we are in *phase0*, thus it activates the first output signal A; as a consequence, the SELECTOR connects the output port q to its first input, i.e. 0. In similar way, when $r > 1$ (*phase4*) the SELECTOR is activated by signal B, thus it copies the value of its second input (0) into its output, which is connected to q.

Else, in all other cases, more operations are required. The third output signal of the top IF EXPRESSION block activates a further IF EXPRESSION block, whose input is $r-\lambda$. When $r-\lambda<0$ (phase1) it activates the signal connected to its first output port, i.e. signal C. As a consequence, the SELECTOR connects q to its third input. This last is represented by a GETSIGNAL, which is defined in the user manual [25] as a block which connects bus ports from one block to another without connecting them physically. It is used in combination with a SETSIGNAL block, which transports its input data to its corresponding GETSIGNAL block. In particular, the *ph1* SETSIGNAL is connected to a block named *Phase1* which implements the expression

$$\ddot{q}_{max}(t-t_0)$$

which is the value output q must assume in phase1, i.e. when C is active.

By contrast, the second output of the second IF EXPRESSION block activates a further block with $r+\lambda$ as input; when $r+\lambda==1$ (phase2) signal D is activated and sent to the SELECTOR, and thus the output q is connected to its fourth input; all other cases (phase3) are handled by signal E and the SELECTOR connects its last input to the output port.

An example of the PROFILE GENERATOR outputs is depicted in Figure 6.19. This profile is obtained setting:

$$q_{ref}=\frac{\pi}{3} \qquad t_0=0; \qquad \dot{q}_{max}=0.4\ [rad/s] \qquad \lambda=\frac{1}{3}$$

thus the computed variables are:

$$C_v=1.5; \qquad C_a=4.5; \qquad T=3.927; \qquad \ddot{q}_{max}=0.3056$$

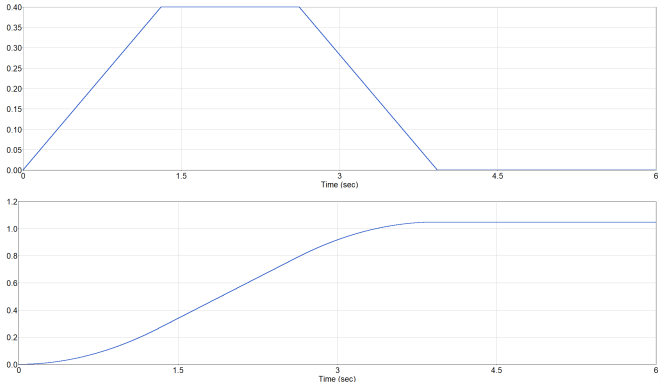


Figure 6.19: *Profile Generator* output esample

Chapter 7

THE INVERSE KINEMATICS

7.1 Introduction

Control will make the Rover follow the desired path that has been determined by the path planner. As for any real system, when the rover drives over the Mars surface, which consists of loose soil (a combinations of rocks and sand) it is subject to different disturbances. These cause deviations from the commanded path; when they are sufficiently large, the Rover curvature and/or heading have to be changed in a way that corrects the deviation. This task is accomplished by the *Trajectory Control*, which makes use of two types of manoeuvres: *Generic Ackermann* where the rover keeps driving along the path and *Generic Point Turn* where the rover stops driving along the path to perform a heading change.

As already mentioned the commands issued by Trajectory Control are *vehicle level* commands, since they deal with vehicle speed and heading, but the actual mobility actuators are the wheels with their driving and steering motors; thus, it is necessary to adequately transforming them into *wheel level* commands, i.e. drive velocity and steering angle. The *Locomotion Manoeuvre Control*, subject of this chapter, is in charge of acting this conversion. The transformation should also aim at ensuring that at any point in time all actuators are in a compatible configuration, i.e. not fighting each other, which would induce slippage, higher energy consumption, and decreased drivability.

In Figure 7.1 the control cycle is reported. The Locomotion Manoeuvre Control block inputs are the *Rover Manoeuvre Command* consisting in command type (type of manoeuvre) and command parameters (e.g. rover speed command, point turn centre of rotation position), while the module outputs are the *Wheels Commands*, i.e. the locomotion true steering and driving axes demands including, for example, the axes control target angle and the associated speed limit (for position control).

In this chapter, it will be explained how Locomotion Manoeuvre Control has been implemented in Activate. This required the kinematic modelling of the rover and the solution of the so called *Inverse Kinematics* problem. Firstly an overview of the

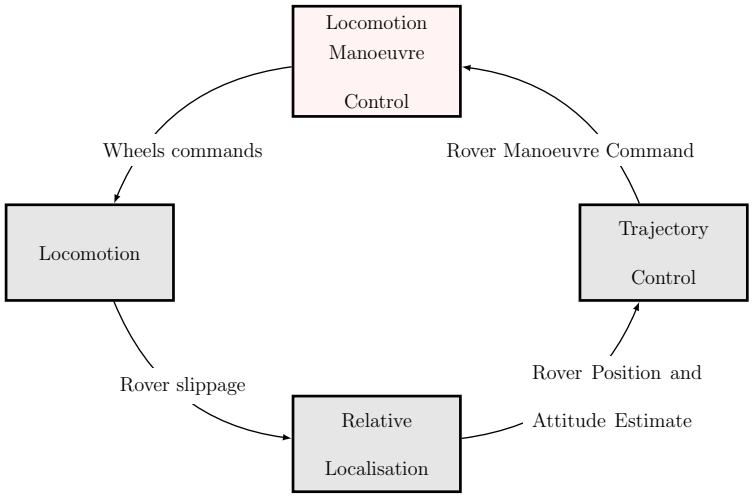


Figure 7.1: General cycle of the ExoMars mobility control

ExoMars rover locomotion capability is presented and a kinematic model is created starting from these informations; later, the Activate block scheme implementing that model is analysed describing in details the blocks required for the construction. The block definitions are taken from the Altair help guide [25]

7.2 Locomotion capability of ExoMars rover

The design of the ExoMars rover features six independent drive axes that can be driven simultaneously and six independent steering axes that can be driven simultaneously. This provides the capability of performing the following manoeuvres [2]

Conventional Ackerman manoeuvres : this type of manoeuvre allows to place the centre of rotation anywhere on the line binding the two middle wheels, outside of the wheel base. In order to achieve such a manoeuvre the corner wheels are steered such that the line binding the wheel centre to the rover centre of rotation is perpendicular to the wheel driving direction. The drive rate of each wheel is set proportionally to the distance between the wheel centre and the rover centre of rotation such that all wheels will describe a circular trajectory consistent with each other. Note that to achieve a conventional Ackerman manoeuvre the middle wheels do not need to steer. To drive in a straight line, the centre of rotation is placed infinitely far away from the rover on the line binding the two middle wheels.

Crab manoeuvres : this type of manoeuvre allows the rover to drive in a straight line with an angle with respect to the rover xRB direction. The centre of rotation is placed infinitely far away from the rover centre. In order to achieve such a manoeuver all wheels have the same steering angle amplitude, equal to the crab

angle, and all drive rates are equal.

Conventional point turn manoeuvres : this type of manoeuvre allows rotation around the rover centre. The centre of rotation cannot be moved.

Generic Ackerman manoeuvres which combine both Ackerman turn and crabbing capability: this type of manoeuvre allows placement of the centre of rotation anywhere on the plane to the left of the left wheels or to the right of the right wheels. In order to achieve such a manoeuvre the wheels are steered such that the line binding the wheel centre to the rover centre of rotation is perpendicular to the wheel driving direction. The drive rate is set proportionally to the distance between the wheel centre and the rover centre of rotation such that all wheels will describe a circular trajectory consistent with each other (i.e. each wheel travels around the same fraction of its circular trajectory in the same time).

Generic point turn manoeuvres which allow to move the point turn centre away from the traction geometric centre: this type of manoeuvre allows placement of the centre of rotation anywhere between the wheels. In order to achieve such a manoeuvre the wheels are steered and drive rates are set in a similar manner as for a Generic Ackerman manoeuvre, however in order to achieve a rotational motion, the left wheels and right wheels have opposite drive rate direction.

7.3 Rover Inverse Kinematics

The Locomotion Manoeuvre Control has to solve the kinematic problem of the wheeled mobile robot. It must translate the desired velocity of the vehicle into the velocities and steer angles of the wheels which allow to move it as desired. This problem is called *Inverse Kinematics*. The goal of this section is to establish the wheel speeds, steering angles and speeds as a function of the rover speed and heading, given the geometric parameters of the rover.

The first thing to do in order to define such function is to create a simple model of the rover. It has been made on the base of the ExoMars mechanical design. It comprises six wheels drivable and steerable independently. The distance between the center of the left and right wheels is

$$w = 1.44\ m$$

while the distance front-middle wheel and middle-rear wheels are respectively

$$a = 0.65\ m \qquad \text{and} \qquad b = 0.805\ m$$

The center of mass of the rover body is considered located at the center of the middle axis

$$Rover_{CM} = \begin{bmatrix} -650 \\ 0 \\ 800 \end{bmatrix}$$

the values are in millimiters as in MotionView environment. Note that the Global Reference frame is located in the center of the two front wheels.
In this thesis three manoeuvres are considered: Ackerman steering (straight forward motion obtained in the particular case of heading $\varphi = 0$), Point Turn and Crab. All these manoeuvres are implemented in the Activate *Inverse Kinematic* block. It must recognize the manoeuvre type set by the user, choose the corresponding parameters and compute the correct reference variables. In the following sections the rover kinematic models developed for each manoeuvre are presented.

7.3.1 Conventional Ackerman Kinematic Model

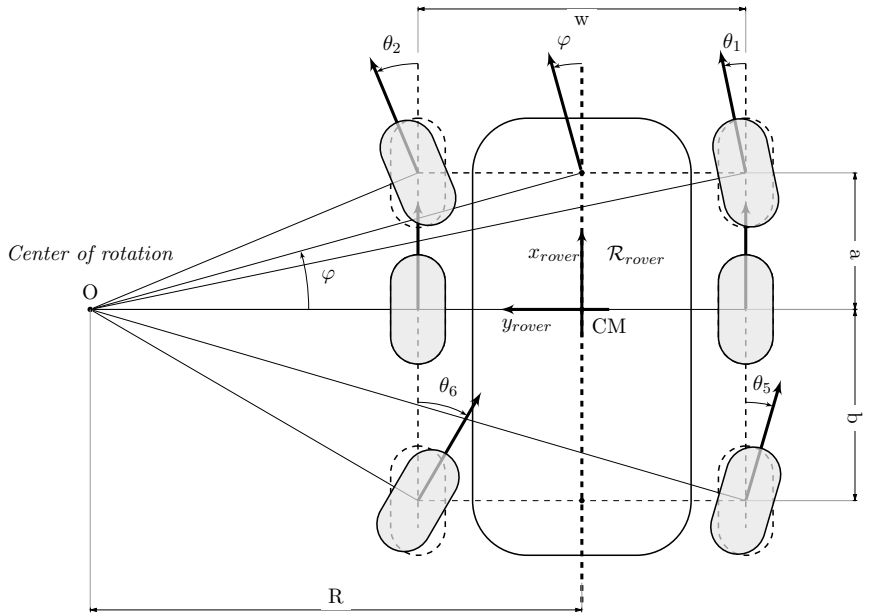


Figure 7.2: Kinematic model for Ackerman manoeuvre

The Conventional Ackerman manoeuvre commands are:

- v_{rover}: the rover CM velocity
- φ: the heading angle

The heading angle is considered referring to a virtual wheel set in the middle of the front axis computed between the x *axis* of the global reference frame and the

mobile x *axis* of the rover reference frame. As shown in Figure 7.2, the rover reference frame is attached to the rover CM; the x *axis* is aligned with the driving direction, the z *axis* points upwards and the y *axis* completes the right-handed coordinate system. The parameters have some limitation due to the actuator limits, but can assume both positive and negative values. According to the rover reference frame, negative values of speed cause the rover to reverse; positive angles correspond to left (anticlockwise) turns and negative angles correspond to right (clockwise) turns, as viewed by the rover, when travelling in the forward direction, while the opposite is true when travelling in the reverse direction. All angles are measured relative to the x *axis* of the rover RF, so 0° corresponds to straight motion.

Moreover, one have to take into account that the centre of rotation can be place anywhere on the line binding the two middle wheels. As described in Chapter 3 the non slippage condition is verified if the wheel longitudinal axis is perpendicular to the line joining the wheel center and the center of rotation; thus, each wheel must be steered in order to make it perpendicular to the radius of its trajectory R_i.

Starting from these informations, given the kinematic model presented in Figure 7.2 the vehicle turning radius R and the rover turn velocity are easily computed:

$$R = a\cot(\varphi) \tag{7.1}$$

$$\dot{\varphi} = \frac{v_{rover}}{R} \tag{7.2}$$

Then, in order to compute the wheel drive velocities, it is needed to compute the turning radius R_i of each wheel. This is done by:

$$R_1 = \sqrt{\left(R \pm \frac{w}{2}\right)^2 + a^2} \tag{7.3}$$

$$R_2 = \sqrt{\left(R \mp \frac{w}{2}\right)^2 + a^2} \tag{7.4}$$

$$R_3 = R \pm \frac{w}{2} \tag{7.5}$$

$$R_4 = R \mp \frac{w}{2} \tag{7.6}$$

$$R_5 = \sqrt{\left(R \pm \frac{w}{2}\right)^2 + b^2} \tag{7.7}$$

$$R_6 = \sqrt{\left(R \mp \frac{w}{2}\right)^2 + b^2} \tag{7.8}$$

The addition or subtraction of the $w/2$ term is depending on the sign of the command heading: upper sign for positive heading, lower sign for negative heading. This is due to the fact that positive heading places the center of rotation on the left side while negative heading places it on the right, always remaining on the line passing through the middle wheel centres. Then, the wheel rotational speeds ω_i are obtained computing the wheel center velocities $v_i = \dot{\varphi} R_i$ and dividing them by the wheel radius r_w:

$$\omega_i = \pm \frac{\dot{\varphi} R_i}{r_w} \tag{7.9}$$

where the direction is depending on the sign of the commanded rover speed, being concordant to it.

Finally, the steering angles θ_i are:

$$\theta_1 = \pm atan \frac{a}{R + \frac{w}{2}} \tag{7.10}$$

$$\theta_2 = \pm atan \frac{a}{R + \frac{w}{2}} \tag{7.11}$$

$$\theta_3 = \theta_4 = 0 \tag{7.12}$$

$$\theta_5 = \mp atan \frac{b}{R + \frac{w}{2}} \tag{7.13}$$

$$\theta_6 = \mp atan \frac{b}{R + \frac{w}{2}} \tag{7.14}$$

Also in this case, the sign of θ_i is depending on the the sign of heading angle: upper sign for positive heading, lower sign for negative heading.

7.3.2 Conventional Point Turn Kinematic Model

This manoeuvre allows rotation around the $Rover_{CM}$.

The Point Turn manoeuvre commands are:

- $\dot{\varphi}$: rover turn speed $[^\circ/s]$
- direction of rotation around the $z\ axis$ of the rover reference frame

In this case, the center of rotation in placed in the rover center of mass, thus the wheels must be steered accordingly. From these data, considering the kinematic model presented in Figure 7.3 the wheel rotational speeds ω_i are obtained computing the wheel center velocities $v_i = \dot{\varphi} R_i$, where R_i is the distance between the wheel center and the center of rotation, i.e. the rover CM, and dividing them by the wheel radius r_w:

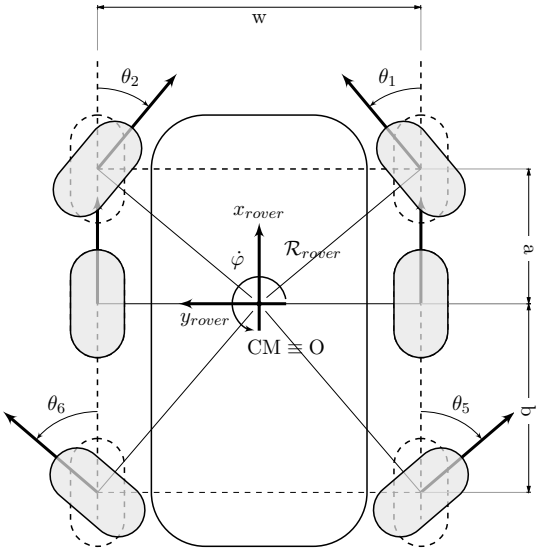


Figure 7.3: Kinematic model for Point Turn manoeuvre

$$R_1 = R_2 = \sqrt{a^2 + \left(\frac{w}{2}\right)^2} \tag{7.15}$$

$$R_3 = R_4 = \frac{w}{2} \tag{7.16}$$

$$R_5 = R_6 = \sqrt{b^2 + \left(\frac{w}{2}\right)^2} \tag{7.17}$$

$$\omega_i = \pm\frac{\dot{\varphi} R_i}{r_w} \tag{7.18}$$

where the sign of ω_i is depending on the wheel: it is concordant with the sign of rotation for right wheels, and opposite to the sign of rotation for left wheels.

Finally, the steering angles θ_i shall be calculated as follows:

$$\theta_1 = \pm atan\frac{a}{\frac{w}{2}} \tag{7.19}$$

$$\theta_2 = \pm atan\frac{a}{\frac{w}{2}} \tag{7.20}$$

$$\theta_3 = \theta_4 = 0 \tag{7.21}$$

$$\theta_5 = \mp atan\frac{b}{\frac{w}{2}} \tag{7.22}$$

$$\theta_6 = \mp atan\frac{b}{\frac{w}{2}} \tag{7.23}$$

where the upper sign holds for positive rotations, while the lower sign stands for negative ones.

7.3.3 Crab Kinematic Model

This manoeuvre allows to drive the rover along a straight line with an angle φ with respect to the rover $x\ axis$. Therefore, the Crab manoeuvre commands are:

- v_{rover}: rover velocity [m/s]
- φ: rover angle [rad]

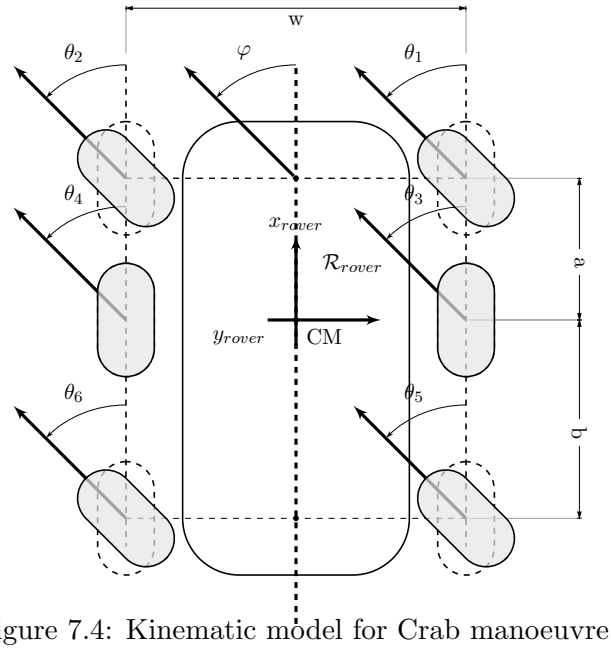


Figure 7.4: Kinematic model for Crab manoeuvre

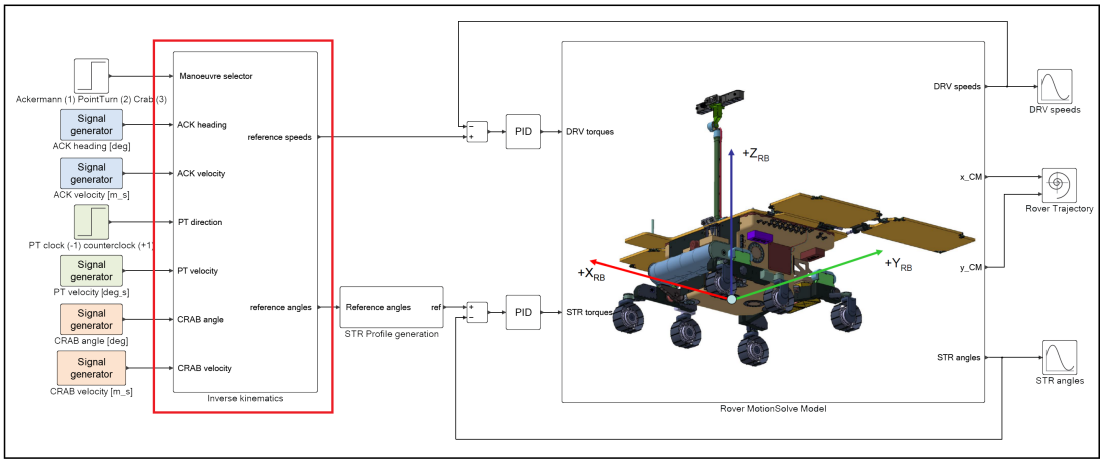
The computation of the wheel rotational speeds and steering angles are trivial, since all wheels must be steered of the same angle φ commanded for the rover and the wheel speed are computed dividing the rover speed v_{rover} by the wheel radius r_w:

$$\theta_i = \varphi \qquad \forall i = 1, ..., 6 \tag{7.24}$$

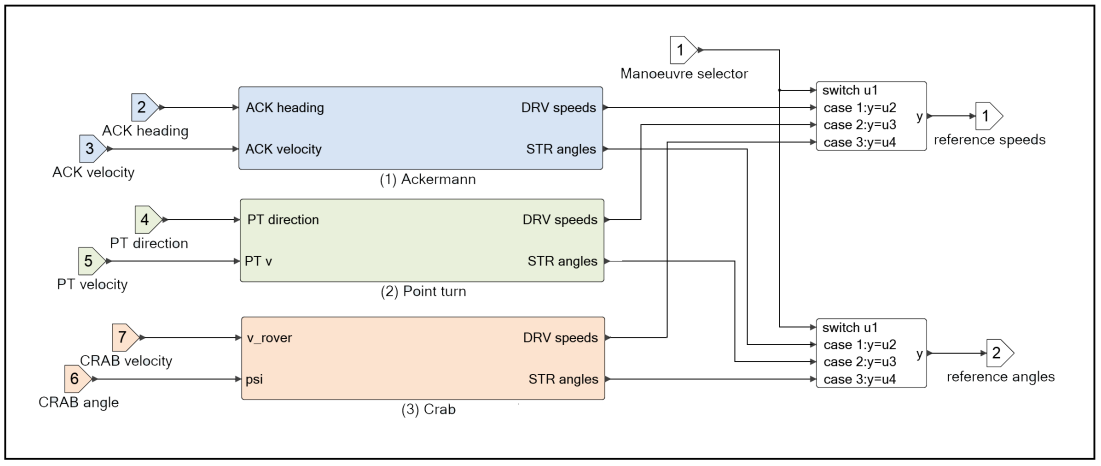
$$\omega_i = v_{rover} \qquad \forall i = 1, ..., 6 \tag{7.25}$$

7.4 Algorithm implementation in Activate

The previously defined formulas have been translated in a block diagram implemented in Activate, constituting the *Locomotion Control* block. The overall block scheme is reported in Figure 7.5a, while Figure 7.5b shows what is inside the *Inverse Kinematics* block



(a) Complete Activate model; the *Inverse Kinematic* block is highlighted in red



(b) *Inverse Kinematic* block scheme. The Ackerman, Point Turn and Crab inverse kinematic algorithms are implemented in the blue, green and orange blocks respectively

Figure 7.5: *Inverse Kinematics* block overview

As already stated, the model has been implemented in a way that it is possible to choose among 3 manoeuvre (actually 4 considering the particular case of Ackerman steering with $\varphi = 0$, i.e. straight forward motion) setting only the manoeuvre commands without caring of wheel rotational velocities and steering angles. Indeed this is exactly the purpose of *Inverse Kinematics* block.

The inputs are:

0. Manoeuvre type

(1) Ackerman steering

(2) Point Turn

(3) Crab

1. Ackerman commands:
   - Rover heading φ
   - Rover CM velocity v_{rover}
2. Point Turn command
   - Direction of rotation: (+1) counter clock wise, (-1) clock wise
   - Rover turn speed $\dot{\varphi}$
3. Crab commands
   - Rover crab angle φ
   - Rover CM speed v_{rover}

As illustrated in Figure 7.5b, three blocks were created, each one implementing the kinematic models defined in the previous sections, corresponding to the three available manoeuvres. Each block receives the associated input commands and computes the reference values for wheel rotational speeds and steering angles, required to perform the desired manoeuvre. These outputs are sent to a ConditionalNSelect block. As defined in the user manual [25] this block copies one of its inputs to its output. The first (top) input is the control input, in our case the manoeuvre type; the remaining inputs are data inputs: they're labelled as case 1, case 2, case 3. The value of the control input determines which data input is copied. For example if the control signal is 2, the "case 2" input is copied to the output. In our particular case, for each output (wheel rotational speeds, front wheels steering angles, middle wheels steering angles and rear wheels steering angles) there are three data inputs, each one corresponding to a manoeuvre type.

In the following section each one of the three block is described.

7.4.1 Ackerman manoeuvre

The *Ackerman manoeuvre* block should compute the reference rotational speeds and steering angles of the wheel according to the input rover heading and linear velocity. The inverse kinematic algorithm is illustrated in Figure 7.6. Three SELECTOR blocks are needed in order to handle the particular case of straight trajectory which corresponds to have $\varphi = 0$. This conditions leads the solve to not converge, because of the trajectory radius is computed as $a \cot \varphi$, therefore $\varphi = 0 \Longrightarrow R = \infty$.

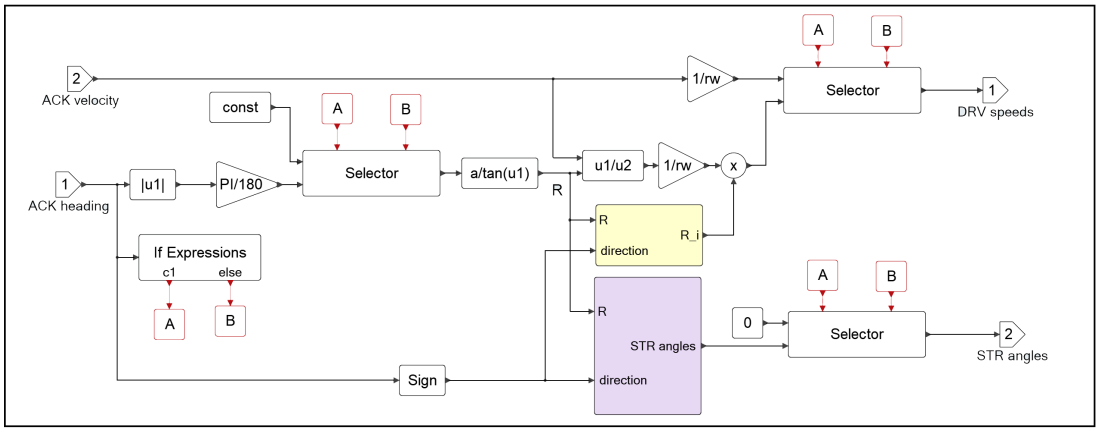
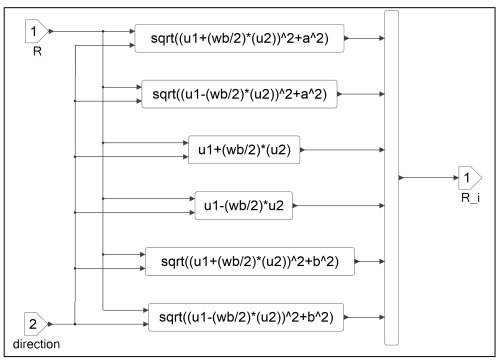


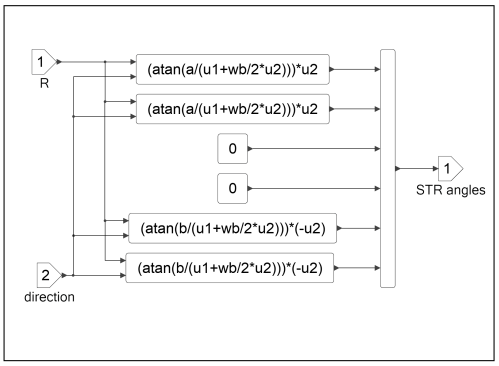
Figure 7.6: Ackerman *Inverse Kinematics* algorithm implementation in Activate. The yellow block is in charge of computing the distance R_i of each wheel from the center of rotation; whereas, the purple block is in charge of computing the wheel steering angles θ_i

The selectors are activated by two signals A and B, which are set by the IF EXPRESSION block, being A the signal corresponding to the particular case $\varphi = 0$.

The computation of the radius R_i of each wheel is implemented as shown in Figure 7.7a, while Figure 7.7b illustrate how the wheel steering angles are computed.



(a) Computation of the distance R_i of each wheel from the center of rotation



(b) Computation of the wheel steering angles θ_i

Figure 7.7: Description of how R_i and θ_i are computed

7.4.2 Point Turn and Crab manoeuvre

The implementation of the kinematic model for Point Turn and Crab manoeuvres are much more trivial with respect to Ackerman one, since it is just a matter of algebraic equations, without any particular case to take care of.

The block schemes are reported in te following figures.

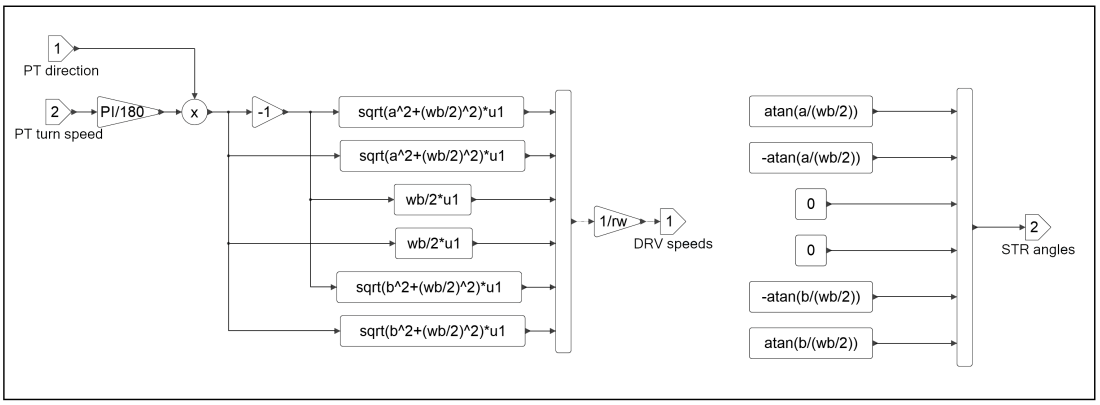


Figure 7.8: Point Turn inverse kinematic algorithm implementation in Activate.

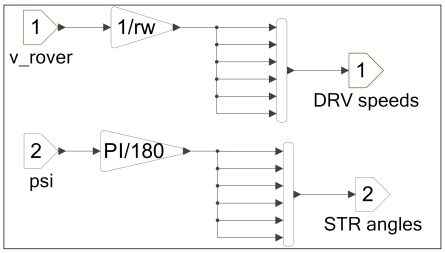


Figure 7.9: Crab inverse kinematic algorithm implementation in Activate.

Chapter 8

MODEL VALIDATION

In this chapter the validation of the Activate-MotionSolve model is described. The outputs obtained from the model during co-simulation is compared with the behaviour of a real rover. Since the ExoMars rover was not available for the test, the system used for the real simulation was the *Integrated Vehicle Breadboard* (IVBB). It is representative of ExoMars Rover and consists in the fusion of the following breadboards: the Chassis and Locomotion BB; the Navigation BB and the Avionics BB. Useful informations are drawn from [8].

The systems composing the IVBB rover are here described:

Chassis and Locomotion Breadboard As for the ExoMars rover, it provides the mobility by means of a chassis with 6 flexible wheels that can be driven and steered independently. These are designed as three pairs, each one connected to a single bogie: two lateral bogies attached to the chassis aligned with the x *axis* of the rover reference frame, and one bogie perpendicular to this axis, attached to the rear. All bogies are connected to the chassis through passive joints, so they are free to rotate about them. The locomotion system is operated by six electric drives located in the hubs of the wheels (DRV motors), whereas similar drives are located at the joint between the bogie and leg to which the wheel is connected . These lasts make the rover performs a variety of steering movements. All motors are locally controlled.

In order make the rover being in the same load condition it will see on Mars (motor torque, wheel/soil interaction), the system is designed to have a mass about 38% of the flight model. The subsystem is controlled by the locomotion SW, which allows the following modes: path following, straight line locomotion, Ackerman steering, rotation (point turn), and lateral motion (crab).

Navigation Software Breadboard It provides the capability of generating autonomously a navigable, safe and optimal path from the Rover current location to a predefined

target. It is based on stereo images processing for Digital Elevation Map (DEM) creation, Navigation Map computation and obstacle detection and avoidance. The two cameras are mounted on a Pan-Tilt unit at about $2m$ height

Integrated Breadboard Equipment Procurement Tt is based on a LEON2 / RTEMS platform for performance evaluation of the developed flight SW. COTS control and power systems are integrated to support testing. It also includes a development environment and an EGSE for testing support and remote control

8.1 Real simulation of IVBB rover

During the test, the IVBB Rover is asked to perform certain manoeuvres, each one executed singularly. The manoeuvre commands are set through a Graphical User Interface. The procedure required to operate the locomotion system is the following:

1. Launch the *Locomotion* sub-GUI from the IVBB GUI
2. Connect to the rover
3. Wait for the rover to complete its initialisation sequence.
4. Enter manoeuvre command values, e.g. rover speed $[cm/s]$ and heading $[^\circ]$ for Ackerman steering, in the appropriate text input fields. These parameters have some limitations: the rover speed cannot exceed ± 10 $[cm/s]$ and heading cannot exceed $\pm$ 35°. If these conditions are violated, a warning will be displayed on the window status bar and the text field defaulted to the maximum value.
5. Pass the values to the rover. This makes the rover steer its wheels to the appropriate orientations to achieve the submitted heading.
6. Upon completion of the steer operation, the rover starts moving at the requested speed
7. The rover will continue to move until the Stop signal is sent to it, or the requested heading has been reached (in point turn mode), or the requested distance has been travelled (in crab mode)
8. If the stop button is pressed, the wheels will realign back to the initial position
9. Disconnect the Rover

8.1.1 Locomotion Software GUI

The exact appearance of the GUI is presented in Figure 8.1.

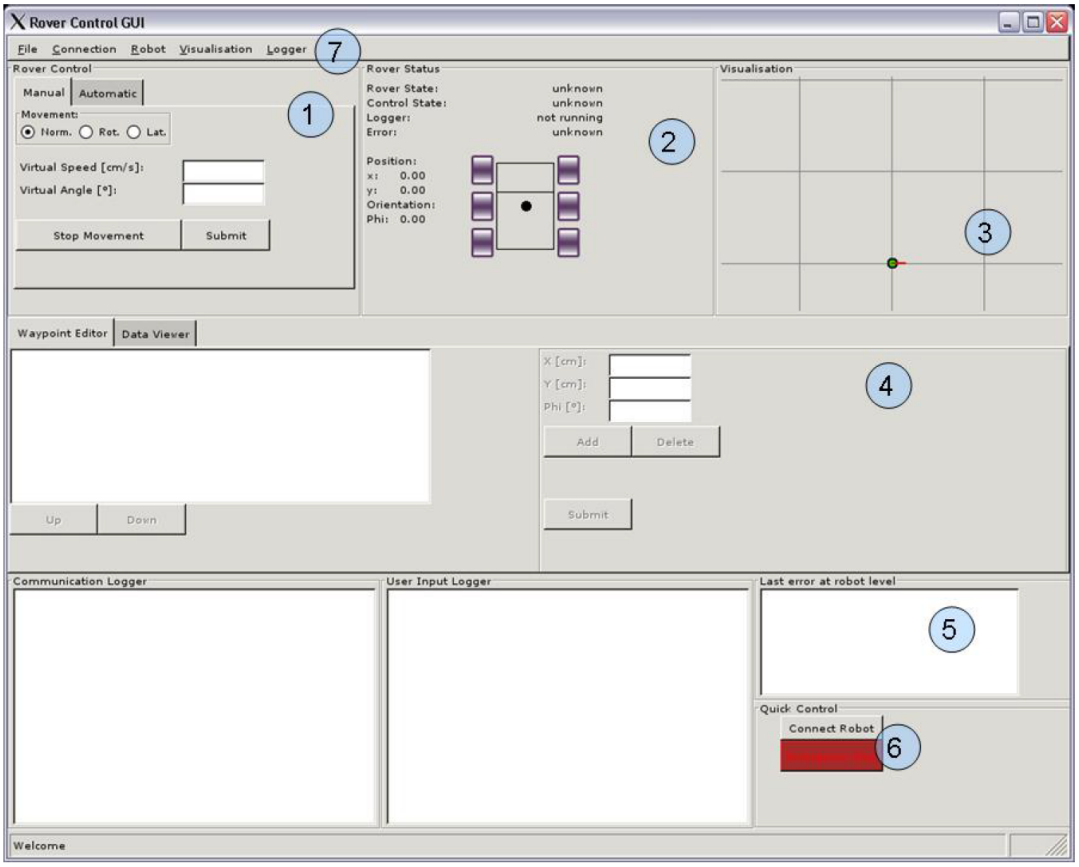


Figure 8.1: IVBB Locomotion Software GUI

It is composed of the following areas:

1. Rover Control Window: it allows selection of the different control modes of the rover (Section 8.1.1); each control mode changes the input fields that have to set to drive the rover. When switching between the different modes, the text field and description for entering the control values will change as well.

2. Rover Status Window: it displays the status, and control state of the connected rover. The field Rover State shows the state of the robot, e.g. init, stop, steer or move. Control State indicates the control mode which can be either automatic, normal, on spot or lateral mode. The field Logger indicates if the data logger is running and collecting data to write to the hard drive, or disabled. The Rover Status Window also displays the pose and a schematic of the rover to visualise the wheel angles, both current and intended. The wheels drawn in dark green show the theoretical wheel steering angle, the wheels in pink/purple show the steering angle measured on the rover.

3. Visualisation Window: it shows the position and orientation of the robot within a global coordinate frame. The robot is represented by a bright green circle, with its orientation shown by the red line that emanates from the centre of the green circle. The window also shows the trajectory of the robot

4. Waypoint Editor/Data Viewer: it is only enabled for use in Automatic control mode, and allows the user to define a path of way-points for the rover to follow.

5. Information fields: there are three information fields across the bottom of the GUI: the Communication Logger, which shows information about the connection and the telecommands sent to the rover, the User Input Logger, which displays information about the user actions, and the Error Logger.

6. Quick Control Buttons: they are the *Connect Robot* button, which connects the GUI to the rover, and the *Emergency Stop*.

7. Menu bar

Control modes

Four different control modes are available, which can be grouped in three *Manual modes* and one *Automatic mode*. The default mode the rover enters on start up is Manual *Normal*. The two other Manual modes are special motion modes: Manual *TurnOnSpot* (point turn), and Manual *Lateral* (crab). *Automatic* mode aims at following a given set of waypoints.

All control modes allow to set individual manoeuvre commands with the exception of Automatic mode, which depends on coordinate inputs in the form of waypoints. These are then converted to manoeuvre commands automatically.

Each of the four modes has the following *internal states*:

init This refers to the rover initialisation process for each control mode, which mainly consists of steering

Stop The rover is stopped

Turn it The rover orientates its wheels for interim turn on the spot (Automatic mode only)

Turn move The rover performs the interim turn on the spot (Automatic mode only)

Turn reset The rover straightens the wheels (Automatic mode only)

Steer The rover performs a steer motion to reach the desired steer wheels angle

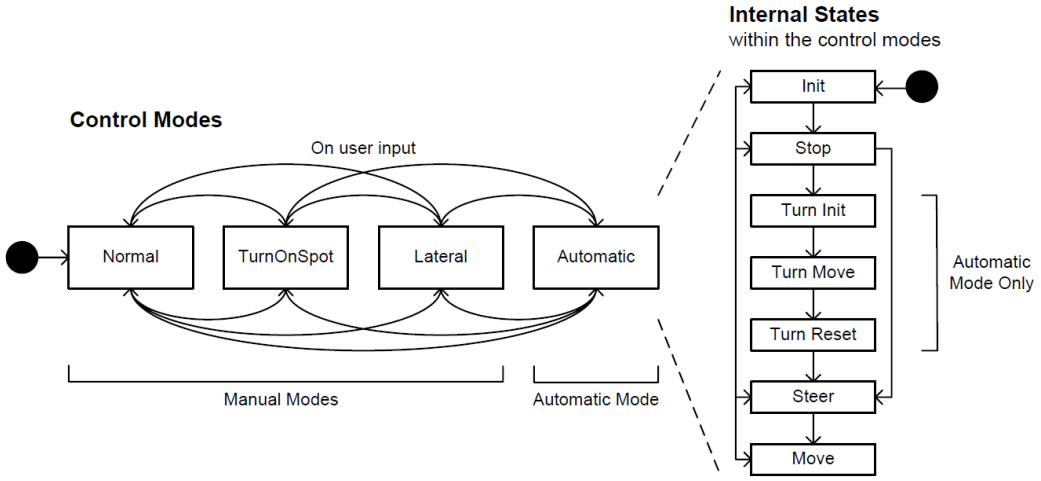


Figure 8.2: Overview of the control modes and their internal states

Move The rover performs a driving motion

An overview of the control modes and their internal states is given in Figure 8.2. The change between the different control modes is controlled by the user, while the change between the different internal states is controlled automatically by the software (e.g. when the wheels have been steering and have finished orientating themselves correctly, the state changes from Steer to Move). The conditions to change between the different internals states depend on the control mode.

Manual - Normal mode This mode allows to perform Conventional Ackerman manoeuvre. The parameters characterising it are the speed and heading angle of a virtual wheel set in the middle of the front axis. The manoeuvre parameters are subject to the following limitations:

$$\begin{aligned} \varphi_{max} &= \pm 35^{\circ} \\ \dot{\varphi} &= \pm 0.1^{\circ} \\ v_{max} &= \pm 10[cm/s] \\ \dot{v}_{max} &= \pm 0.2[cm/s] \end{aligned}$$

Manual – TurnOnSpot Mode When this mode is entered, the wheels are turned automatically towards the centre of the rover for circular motion. The parameter that can be set are the rotational speed of the rover and its rotational displacement, i.e. the variation in the orientation of the robot computed as the difference between the x *axis* of the rover RF and the global one. As soon as the values are submitted, the rover begins to rotate about its centre until the specified rotational displacement has been reached.

With regard to this manoeuvre, the commands are partially different with respect to the Activate model ones. When performing software simulation, the rover does not stop when a target position is reached, but it continues its motion till the end of the simulation time. This is because the input parameters is only the rotational speed of the rover, and an additional input defining the direction of rotation ((+) counterclock wise or (-) clock wise), but no target angular position is defined. Moreover, in IVBB locomotion SW clockwise rotation can be only achieved by entering a negative rotational speed since no negative values of rotational displacement can be entered.

Manual – Lateral Mode In this mode, the rover will turn all six wheels to the input steering angle and, upon reaching the correct position, the rover will move the distance specified. In this control mode the rover can only move in the forward direction, up to a heading of 60°. Positive angles correspond to motion with a component in the positive y direction, negative angles in the negative y direction. The speed cannot be manually controlled and is hard-coded as 3 cm/s.

Automatic mode This control mode takes a list of waypoints as the input for the rover; a waypoint represents a point on the xy plane, the origin and axes of which corresponds to the initial position and pointing of rover as displayed in the GUI. The movement of the robot will be along the path defined by the waypoints list. Anyway, the autonomous navigation is not argument of this thesis.

The test was performed asking the rover to perform the following individual manoeuvres:

1. Manoeuvre1 - Normal mode (Ackerman): $v_{rover} = 3\ cm/s;\ \varphi = -35°$
2. Manoeuvre2 - Normal TurnOnSpot mode (Point turn): $v_{rover} = 3\ °/s$ performed for $t = 200\ s$
3. Manoeuvre3 - Normal Lateral mode (crab): $v : rover = 3\ cm/s$ performed for $t = 200\ s$
4. Manoeuvre4 - Normal mode (Straight forward): $v_{rover} = 3\ cm/s$ performed for $t = 100\ s$

8.2 Model adaptation

In this chapter we want to compare the results coming from simulation with those given by real system movement. In order to have correspondence between input/output signals of the two systems we have to modify a little the Activate model.

First of all the geometric dimensions of the mechanical structure have to be change, since the two rovers are of different dimensions. In particular, the distance between the center of the left and right wheels (track) is

$$w = 1.2\ m$$

while the distance front-middle wheel and middle-rear wheels are equal

$$a = b = 0.7\ m$$

The center of mass of the rover body is considered located at the center of the middle axis, therefore MotionSolve will consider the rover CM position at $(-700, 0)$. This is due to the fact that in MotionView the Global origin with respect to which the CM position is defined, is located at the center point between the front wheels.

Finally the wheel radius value must be change to

$$r_w = 0.125m$$

Moreover, in Chapter 7 we said that the inverse kinematic block computes the reference values for steering angles and rotational velocities of the wheels, on the basis of manoeuvre commands. These inputs were considered as the rover CM velocity and heading (defined as the angles between the x *axis* of the rover reference frame and the global one). These hypothesis have been done in order to be consistent with the actual behaviour of ExoMars rover.

By contrast, IVBB manoeuvre commands are not exactly the same; in particular, Ackerman manoeuvre commands are the velocity and heading relative to a *virtual wheel* centred in the middle of the front axis. This means that the *Inverse Kinematics* block must be modified as explained in the following section.

8.2.1 Modified Ackerman Kinematic Model

Considering the Ackerman manoeuvre the computation of the turn rate $\dot{\varphi}$ made in Chapter 7 is no more valid. Looking at Figure 8.3 the new turn rate can be computed as

$$\dot{\varphi} = \frac{v_{virtual}}{R_{virtual}} \tag{8.1}$$

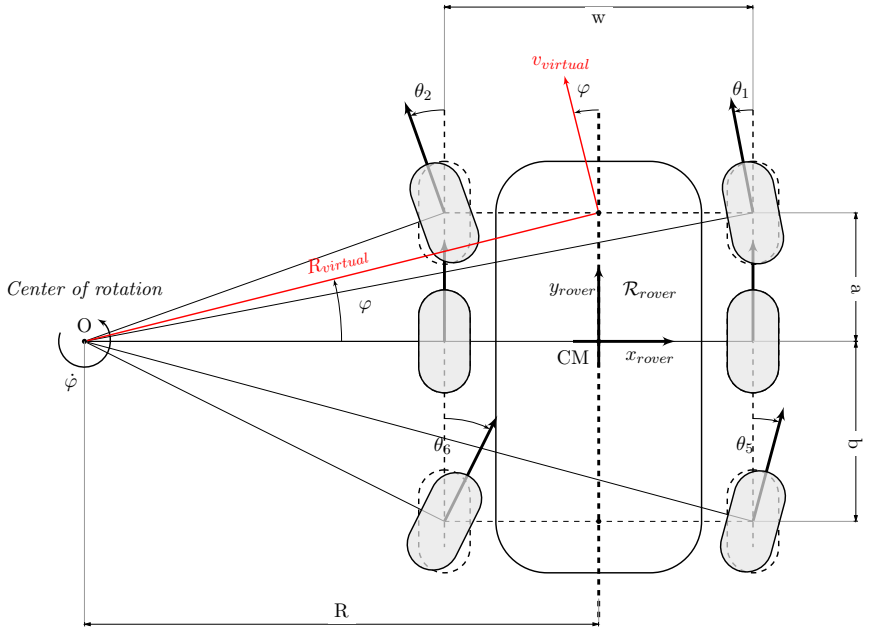


Figure 8.3: Modified kinematic model for Ackerman manoeuvre when using IVBB

Thus the block scheme is modified as illustrated in Figure 8.4

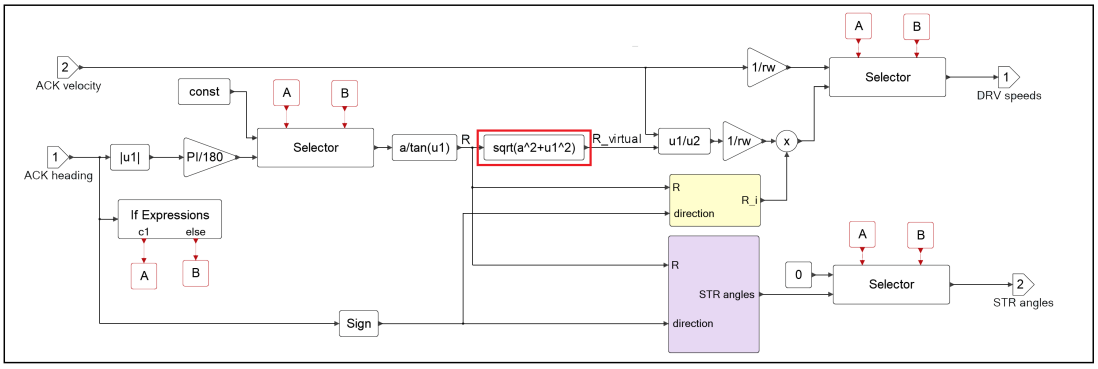


Figure 8.4: Activate block scheme of the modified *Inverse Kinematics* block

8.3 IVBB simulation results

8.3.1 Manoeuvre1 - Ackerman

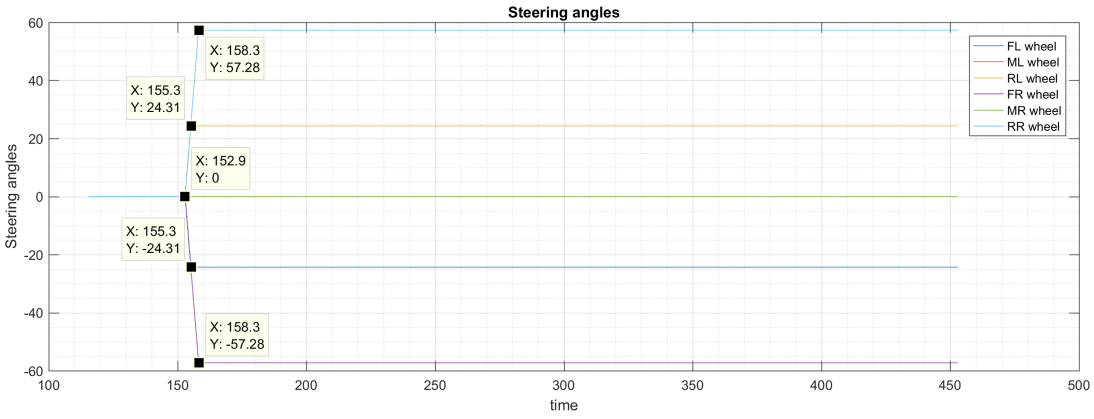
The first manoeuvre performed was a Conventional Ackerman with

$$v_{rover} = 3\ cm/s; \quad \varphi = -35^{\circ}$$

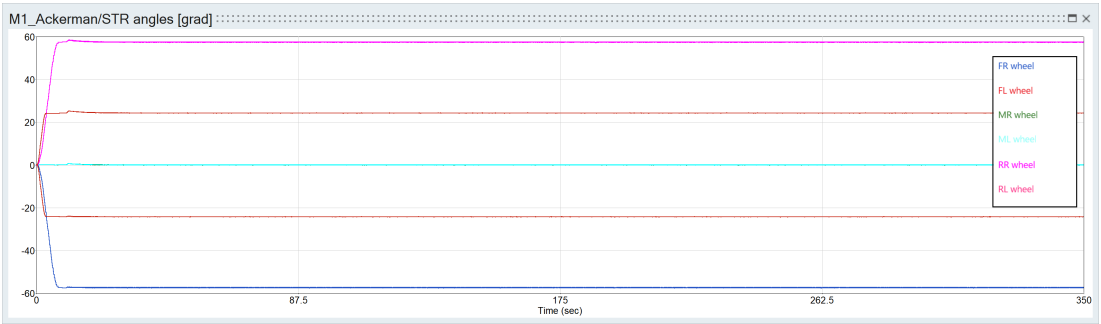
The results are reported in figures 8.5, 8.6 and 8.7.

As can be seen, the first action performed by IVBB rover was to steer the wheels of angles computed accordingly to the input rover heading. Only when all steering axes have reached the reference angular position, the drive actuators start their actions. Indeed, the rover position starts increasing at $t = 158\ s$, exactly when the FR and RR wheels reach their references.

The same manoeuvre, with the same parameters, was commanded to the Activate model. As can be seen, the steering angles and rotational velocities of the wheels are equal to those resulting from real system. It must only specified that the trajectory seems different, but the relevant parameter is the trajectory diameter, that is equal in both simulations. The trajectory computed by co-simulation is simply shifted along the $x\ axis$; the reason why this happens is that the multi-body model was created in a way that leads to have the center of the rover shifted backward along the global $x\ axis$. Since the solver variable identifying the trajectory defines it with respect the global origin, the trajectory plotted in Activate starts at $(-700, 0)$. By contrast, the global reference frame considered by IVBB software is created as soon as the rover is powered on, making it coincident with the rover center. Thus, the real trajectory starts at $(0, 0)$.

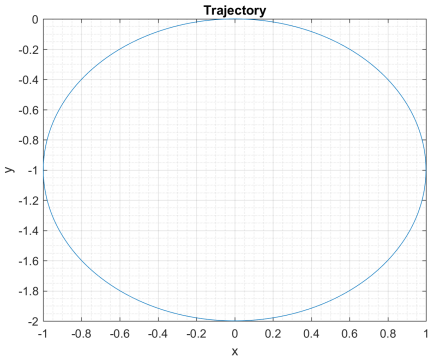


(a) IVBB aangles

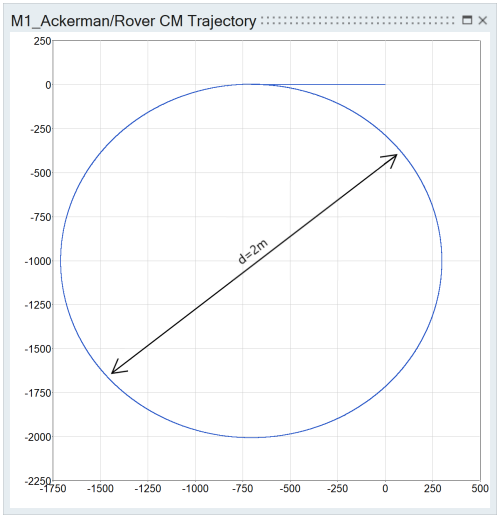


(b) Activate angles

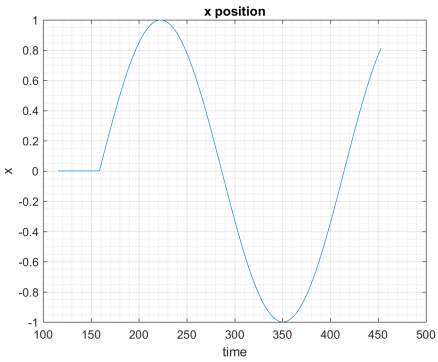
Figure 8.5: M1 Ackerman manoeuvre: wheel steering positions



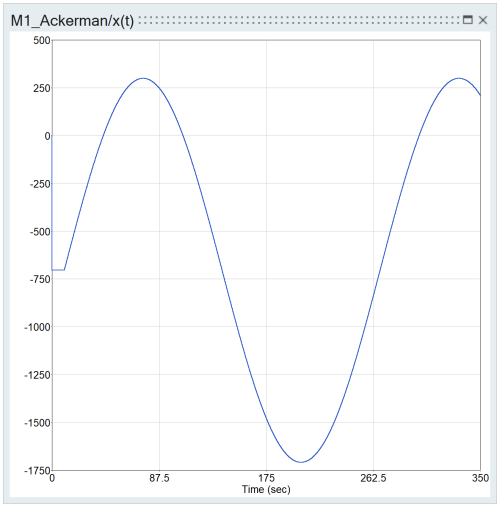
(a) IVBB Rover CM trajectory



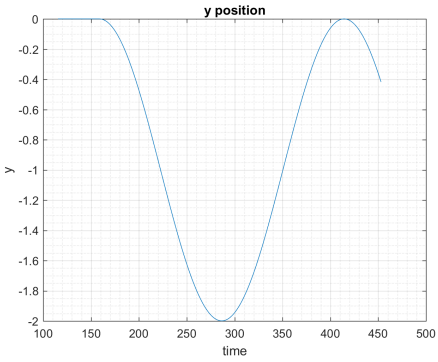
(b) Activate Rover CM trajectory



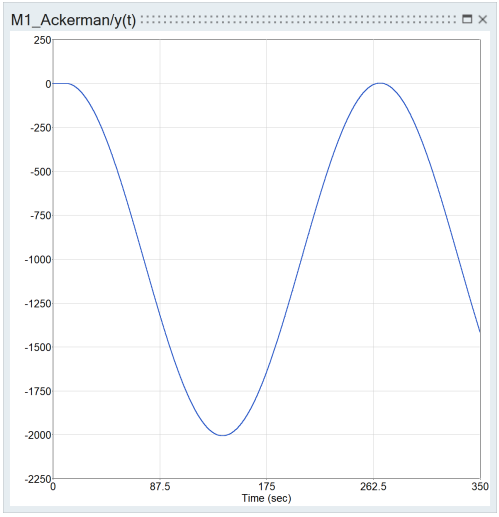
(c) IVBB Rover CM x position



(d) Activate Rover CM x position

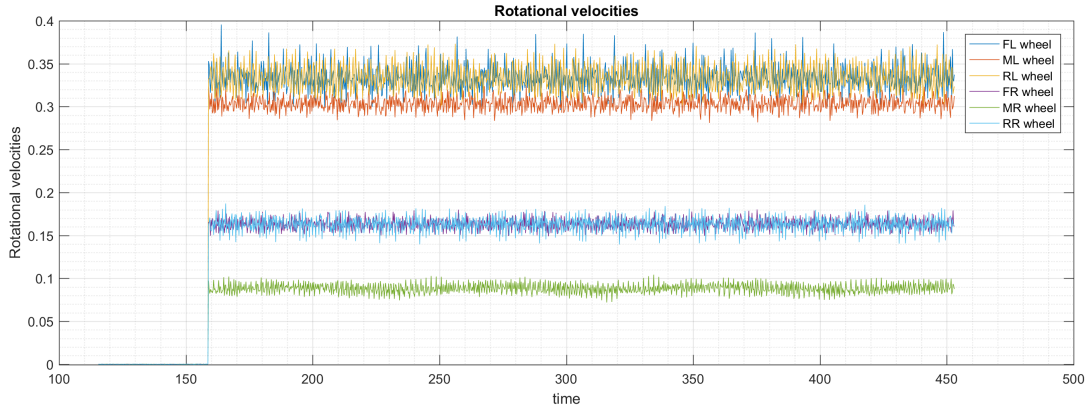


(e) IVBB Rover CM y position

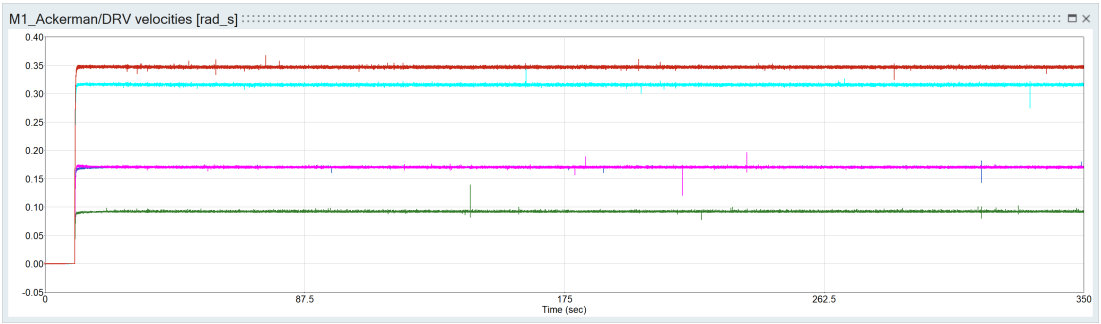


(f) Activate Rover CM y position

Figure 8.6: M1 Ackerman manoeuvre: rover CM x, y position time evolution and xy trajectory



(a) IVBB velocities



(b) Activate velocities

Figure 8.7: M1 Ackerman manoeuvre: wheel rotational velocities

8.3.2 Manoeuvre2 - Point turn

The second manoeuvre performed was a point turn with the following input command:

$$v_{rover} = 3^{\circ}/s \qquad t = 100s$$

The results can be compared looking at figures 8.8, 8.9 and 8.10. It is evident that the wheel rotational velocities resulting from Activate model do not correspond to those reported in the IVBB .log files. Analysing the situation in more details, it came out that the IVBB software does not correctly computes the reference values of the rotational velocities when in Point Turn mode. This is demonstrated by examine the rover heading time evolution, i.e. the rotation of the rover around the vertical z *axis*, computed as the difference between the rover x *axis* and the global one. Figure 8.11a reports the rover heading change during the manoeuvre. In order to check easily if the IVBB velocities are correct, one shall consider a time interval of $T = t_2 - t_1 = 100\ s$; since the commanded turn speed was $\dot{\varphi} = 3^{\circ}/s$ it is expected the rover heading change to be $\Delta = \varphi(t_2) - \varphi(t_1) = 300\ ^{\circ}$.

Contrary to what was expected, the change of the heading according to the IVBB data is $\Delta = 246^{\circ}$. Therefore, the velocities of the wheel are not the correct ones required to perform the manoeuvre as desired by the user. They are smaller than what is necessary.

Overall, it is reasonable to consider the Activate model correct, even if its outputs do not corresponds to the one resulting from real system, since it was demonstrated to be incorrect.

A further analysis aims verifying the correct heading change according to the Activate model. The performed test validates it, indeed, Figure 8.11b shows the heading computed by MotionSolve when the wheel rotational speeds are:

$$\omega_{FR} = -0.3891;\ \omega_{FL} = 0.3891;$$

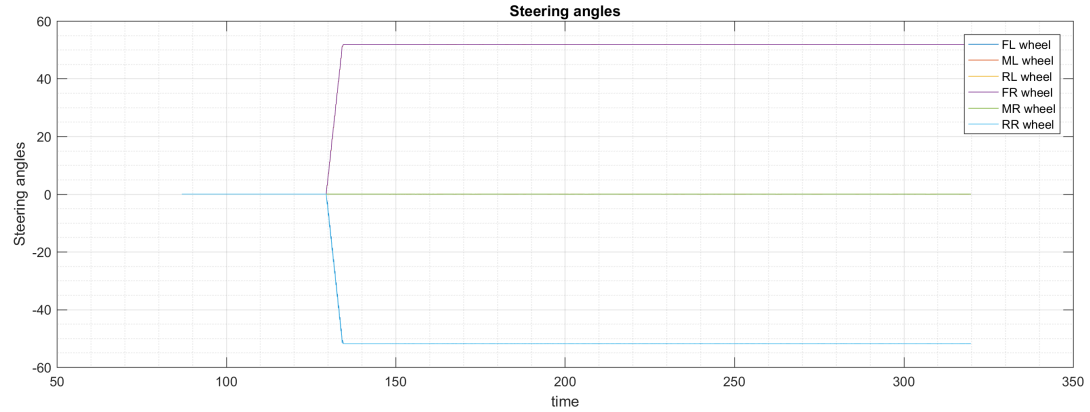
$$\omega_{MR} = -0.2412;\ \omega_{ML} = 0.2412;$$

$$\omega_{RR} = -0.3891;\ \omega_{RL} = 0.3891$$

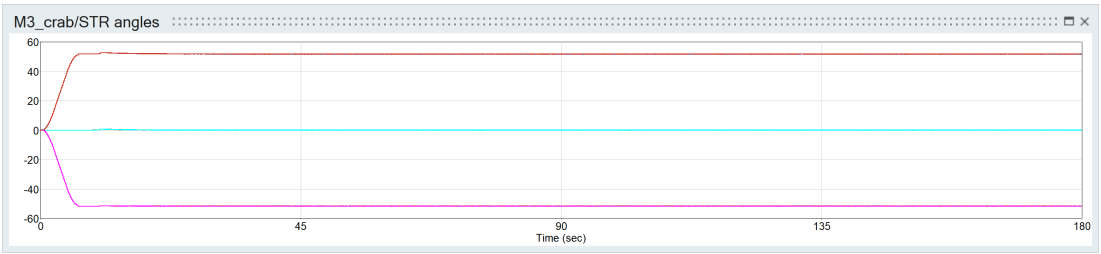
Considering two time instants t_1 and t_2 such that $t_2 - t_1 = 100\ s$ the corresponding change of the rover heading during that period is:

$$\Delta = \| -5.4329 + 0.2461 = 5.1865 \| \ rad = 297.18^{\circ}$$

that is exactly the expected one.

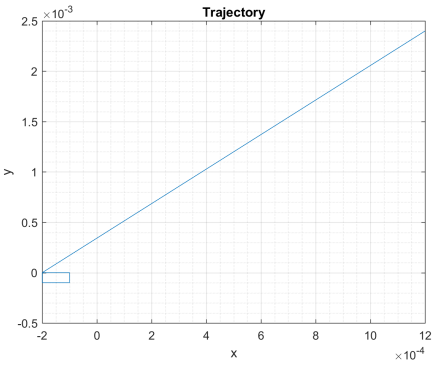


(a) IVBB aangles

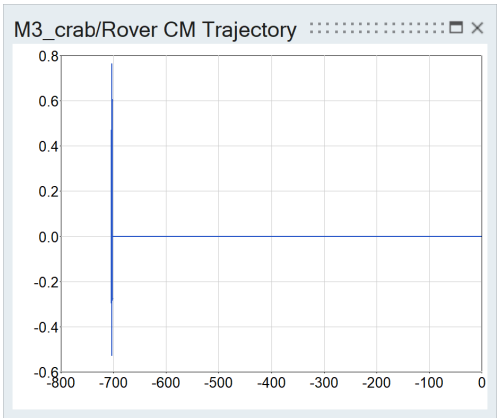


(b) Activate angles

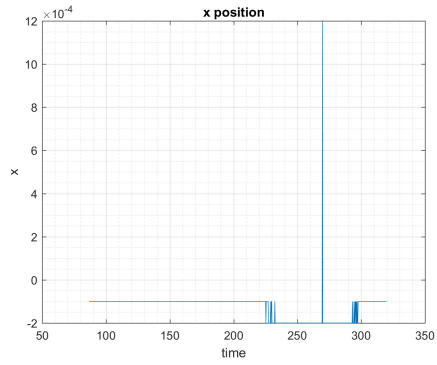
Figure 8.8: M2 Point Turn: wheel steering positions



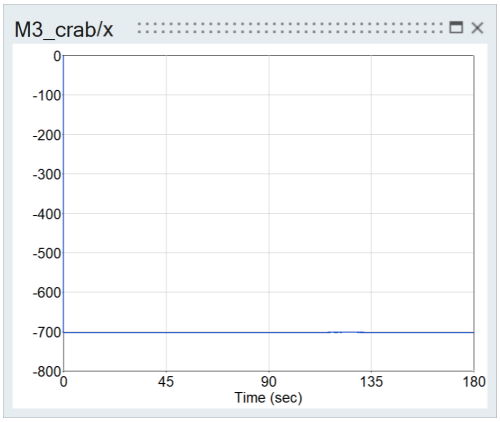
(a) IVBB Rover CM trajectory



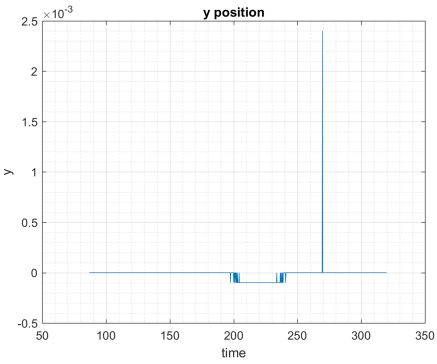
(b) Activate Rover CM trajectory



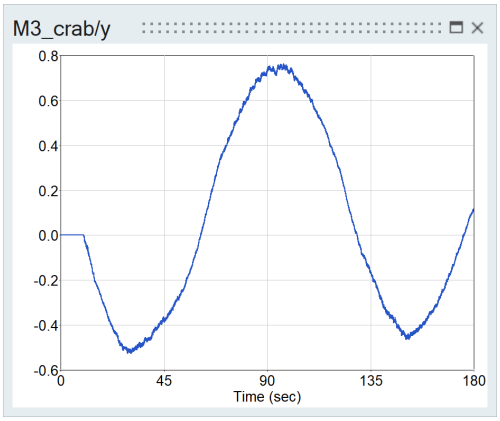
(c) IVBB Rover CM x position



(d) Activate Rover CM x position

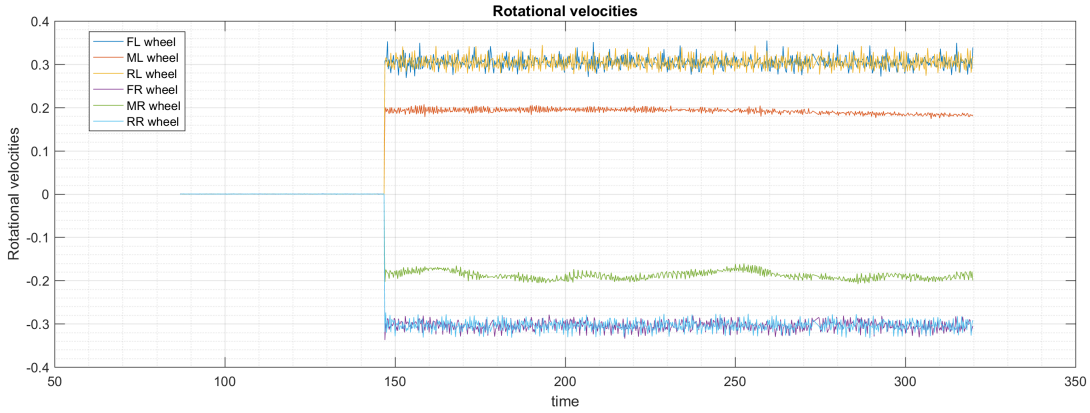


(e) IVBB Rover CM y position

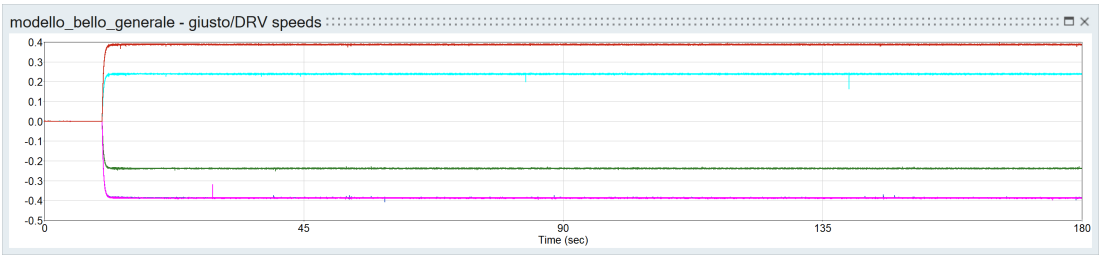


(f) Activate Rover CM y position

Figure 8.9: M2 Point Turn manoeuvre: rover CM x, y position time evolution and xy trajectory

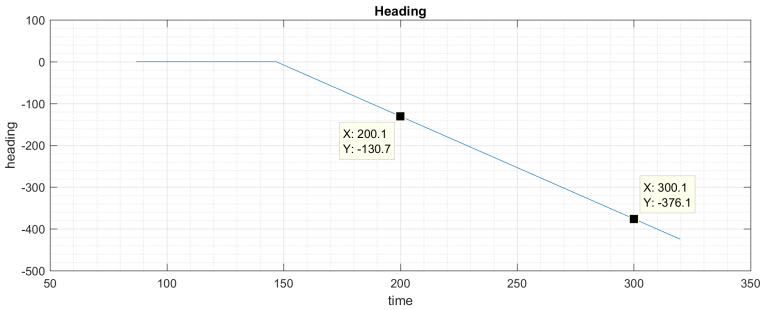


(a) IVBB velocities

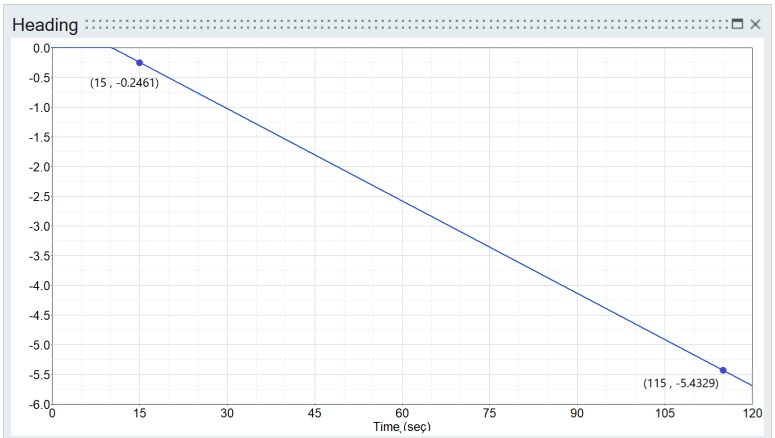


(b) Activate velocities

Figure 8.10: M2 Point turn manoeuvre: wheel rotational velocities



(a) IVBB heading



(b) Activate heading

Figure 8.11: M2 Point turn manoeuvre: rover heading

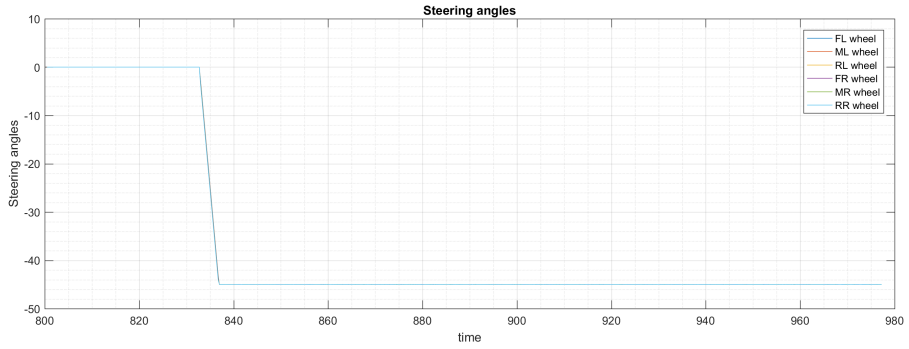
8.3.3 Manoeuvre3 - Crab

The third trajectory commanded to the IVBB rover consisted in a Crab manoeuvre with the following input commands:

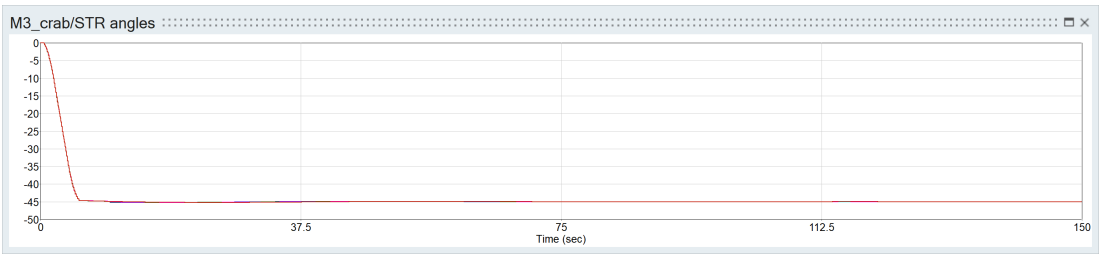
$$\varphi = -45°; \qquad v_{rover} = 3\ cm/s$$

The simulations in real time and in Activate environment were run for $150s$. The results so obtained are reported in the following figures. With these inputs the roves travelled a distance $d = 4.169\ m$ keeping all steering axes to $-45°$. From Figure **??** it is noticeable that the rotational velocities are the same for all wheels, $\omega_w = 0.25\ rad/s$, both in real system and in Activate.

The trajectory followed by the rover in the xy plane is reported in Figure 8.13 and also in this case the results are similar.

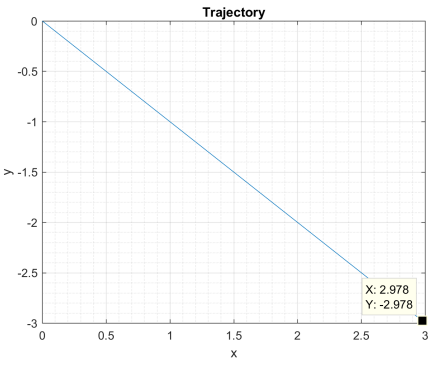


(a) IVBB aangles

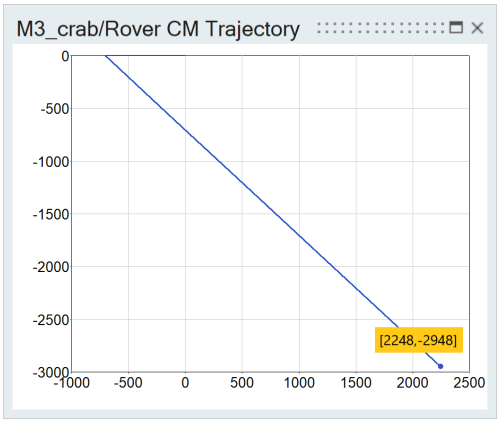


(b) Activate angles

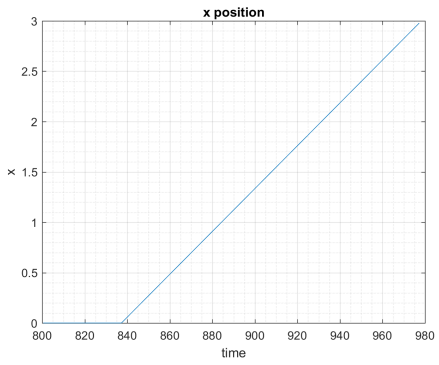
Figure 8.12: M3 Crab manoeuvre: wheel steering positions



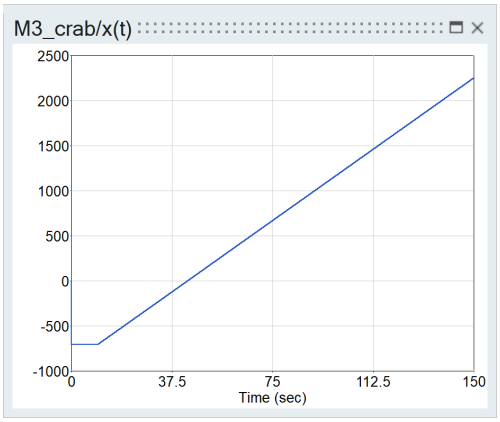
(a) IVBB Rover CM trajectory



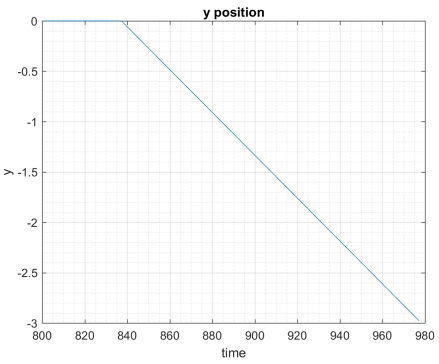
(b) Activate Rover CM trajectory



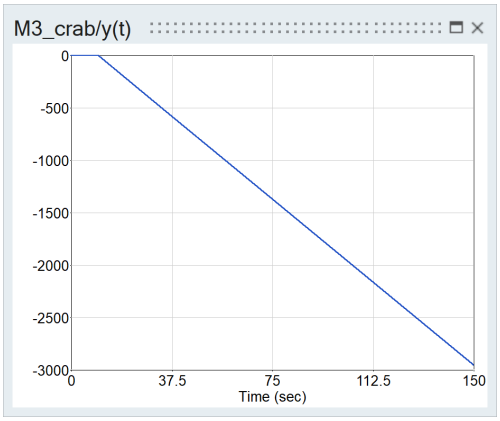
(c) IVBB Rover CM x position



(d) Activate Rover CM x position

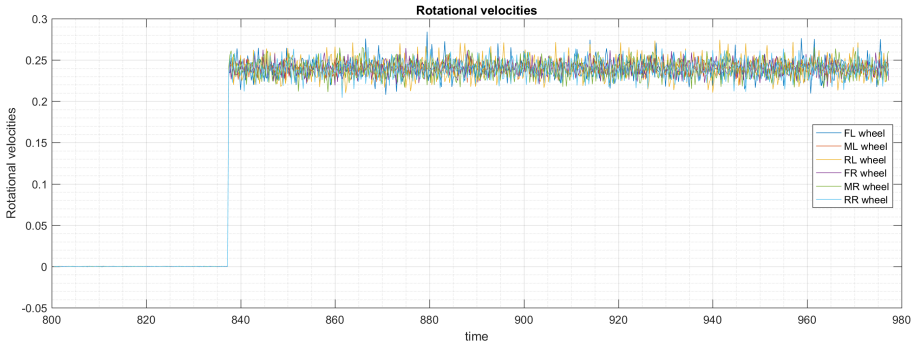


(e) IVBB Rover CM y position

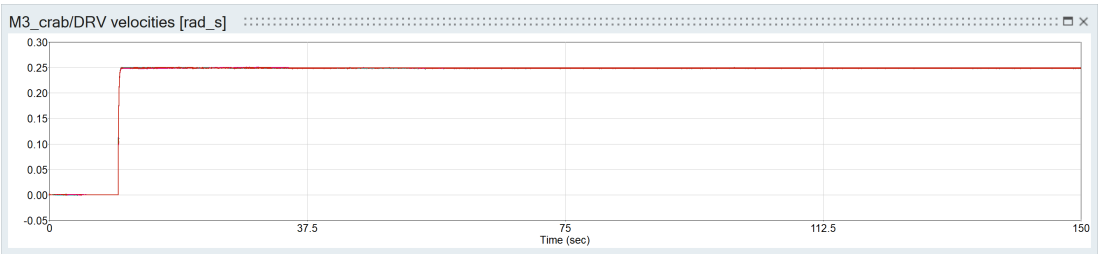


(f) Activate Rover CM y position

Figure 8.13: M3 Crab manoeuvre: rover CM x, y position time evolution and xy trajectory



(a) IVBB velocities



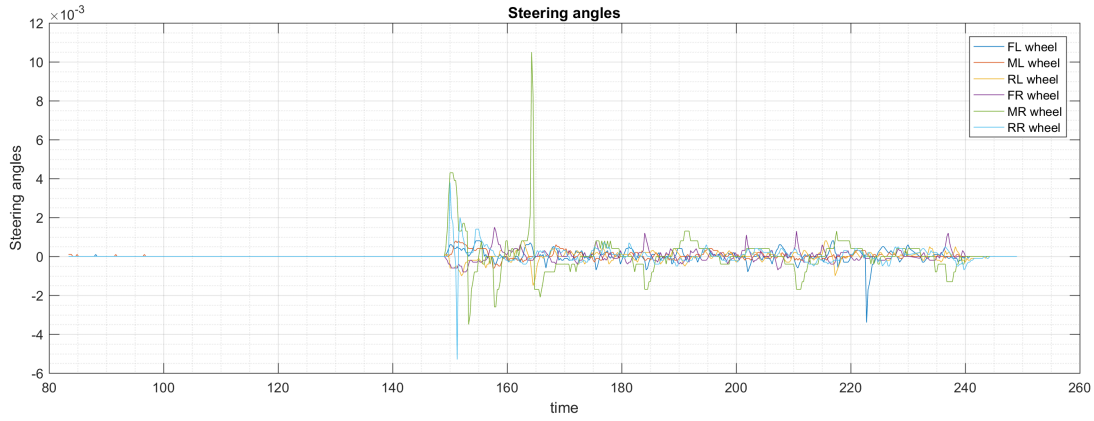
(b) Activate velocities

Figure 8.14: M3 Crab manoeuvre: wheel rotational velocities

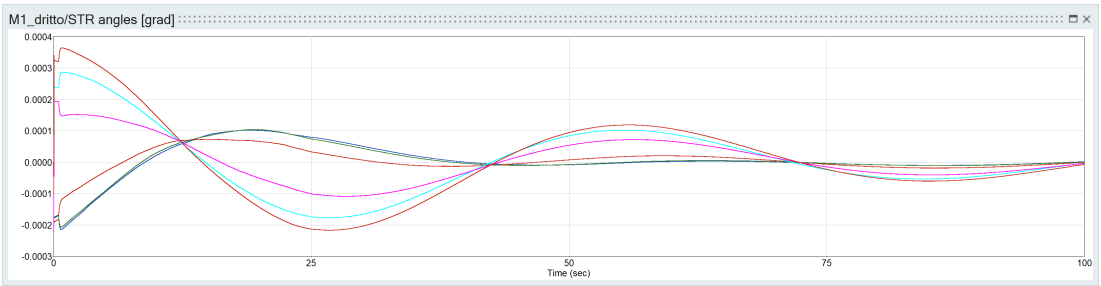
8.3.4 Manoeuvre4 - Straight forward motion

The fourth manoeuvre consists in driving the rover straight forward at $v_{rover} = 3\ cm/s$ for about $T = 100s$. The results obtained from IVBB real simulation are reported in the following figures. With these inputs the roves travelled a distance $d = 2.44\ m$ keeping the steering axes to $0°$.

The same manoeuvre was tested in Activate environment and the results are shown in the figures. The rotational velocity is the same for all wheels as shown in Figure 8.17 and it is $\omega_w = 0.25\ rad/s$ while the steering angle are all null, see Figure 8.15. The trajectory followed by the rover in the xy plane, obtained through co-simulation, is reported in Figure 8.16b; it is noticeable that it is practically a straight line, starting from $p_1 = (-700; 0)\ mm$ and ending at $p_2 = (2210; 0.00027)\ mm$. Point P_1 is not centred in the xy plane origin since it represents the rover center of mass as modelled in MotionView, i.e. shifted with respect the $(0, 0)$ point. From these data the travelled distance can be computed, resulting in $d = 2.91\ m$.

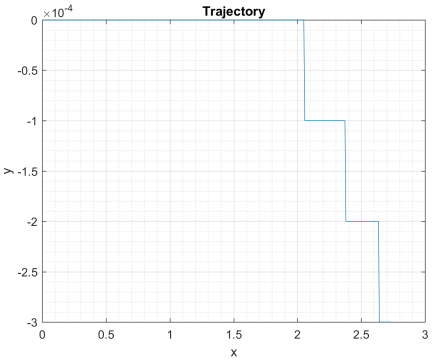


(a) IVBB aangles

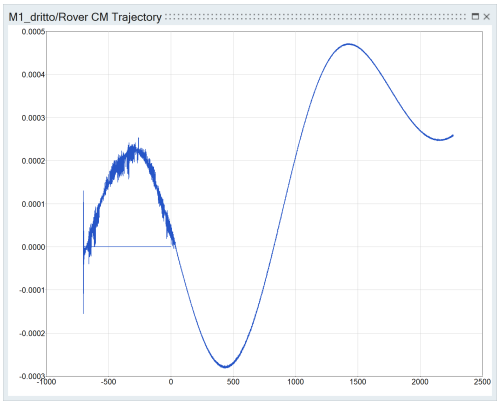


(b) Activate angles

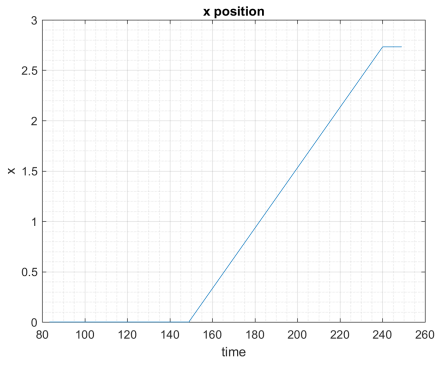
Figure 8.15: M4 Straight Forward motion: wheel steering positions



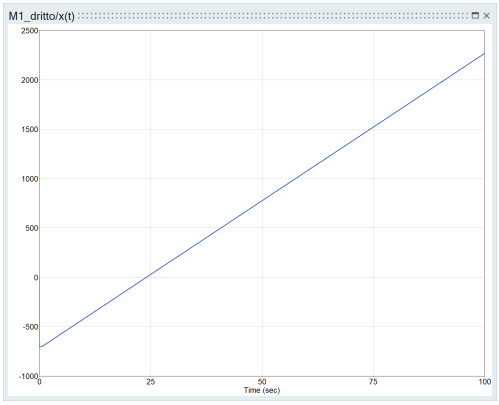
(a) IVBB Rover CM trajectory



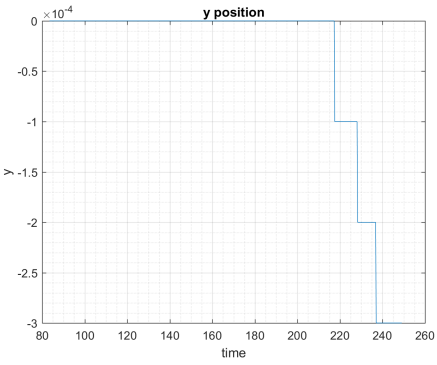
(b) Activate Rover CM trajectory



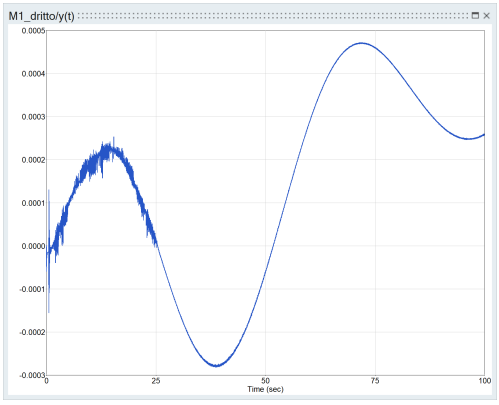
(c) IVBB Rover CM x position



(d) Activate Rover CM x position

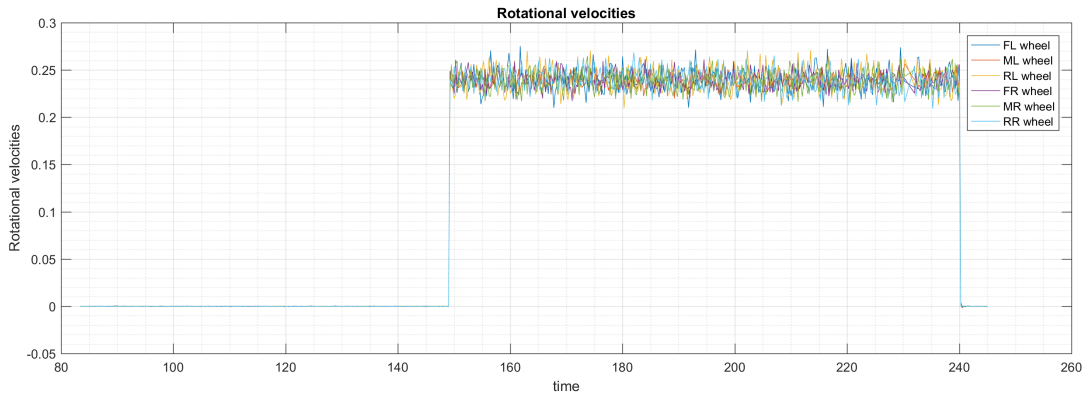


(e) IVBB Rover CM y position

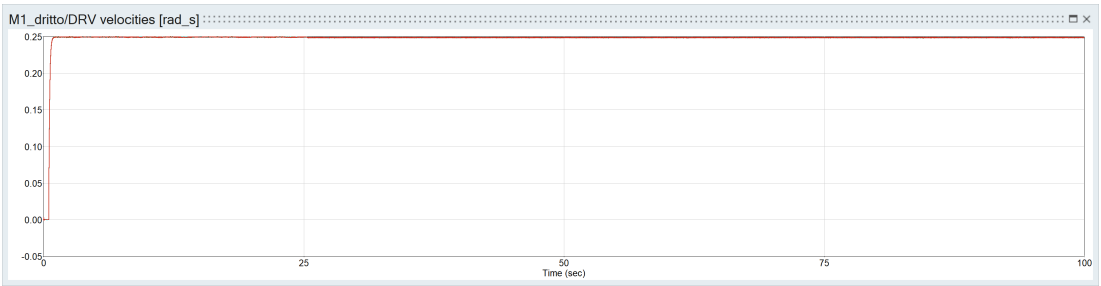


(f) Activate Rover CM y position

Figure 8.16: M4 Straight Forward motion: rover CM x, y position time evolution and xy trajectory



(a) IVBB velocities



(b) Activate velocities

Figure 8.17: M4 Straight Forward motion: wheel rotational velocities

Chapter 9

CONCLUSIONS

The work presented in this paper has the objective of develop a control system responsible of driving the Multi-Body System model of the ExoMars rover developed by the Mechanical Computer Aided Engineering group of Thales Alenia Space Italia. The main idea was to employ several software being part of Altair software house taking advantage of the co-simulation capabilities. Thanks to this it was possible to create a MotionSolve model as a Functional Mock-Up unit representing the ExoMars rover multi-body model, and couple this with the controller implemented in a dedicated environment (SolidThinking/Activate) in order to perform co-simulation. This will enable engineers to simulate and analyse the rover mechanical structure when performing manoeuvres according to user commands.

The first part of the work, concerning the modelling of the MBS ExoMars rover model was among the most challenging activities. The search for the right setting of model entity parameters has made it necessary to perform a long and accurate validation work, requiring lots of simulations and analysis aiming to verify the consistency of the model with the real system behaviour. In particular, a major effort was required to properly adapt the MotionSolve model in such a way that it can be correctly interfaced with Activate controller model.

Once the accuracy of the MBS model was verified, it was converted in a standalone modelling unit and imported in the 1-D block environment simulator solidThinking/Activate.

At this point, the focus shifted in the implementation of a proper control system able to drive the MotionSolve rover model, starting from the simple PIDs of the low-level control loop. These lasts are responsible of the the wheel drive and steering actuators control.

Once, the PIDs parameters has been correctly tuned, it was required to improve the usability of the overall model. This goal was achieved by adding a high-level locomotion control, in charge of interfacing with the user and performing necessary transformations on the commands. The great advantage of this function is that users

can easily manage the model by setting directly rover level manoeuvre commands, i.e. the desired manoeuvre type and rover velocity. In this way they do not act on the PID inputs, which are wheel level commands, thus they do not have to solve equations for determining wheel steering angles and velocities that will make the rover move as desired.

The most satisfying activity of this work was the model validation process. Thales Alenia Space Italia and Altec companies gave the opportunity to perform several test on a real rover. However, the ExoMars rover, whose design was considered in the thesis development was not available. Thus the test was performed on the first prototype of ExoMars rover, whose locomotion system is representative of the final one, thus it provides the same mobility functionalities, but the geometric dimensions of the chassis, the wheel radius and the overall mass are not the equal those used in the modelling phase of this thesis. Despite this situation was firstly regarded as an inconvenience, it gave the opportunity of demonstrate the great advantage of the produced work. Indeed, thanks to the model versatility it was easily to adapt it to work in those new conditions.

Once the MBS model was modified and the PIDs controlling it were re-tuned to the new working condition, the Activate model simulation was run in parallel with the real motion of IVBB. This made it possible to verify if the developed control system correctly drives the MBS rover model. By running the two systems, Activate model and real IVBB, with the same manoeuvre commands, it came out that the overall developed model has the same characteristics of the software controlling the real rover mobility function.

Nonetheless, this work have some limitation. Even if the MBS model was verified to have realistic behaviours, it does not represent real specific situations, since when modelling contacts between wheels and ground the parameter were only set in order to have forces respecting the physics of the problem. Therefore the magnitude of the forces acting on the wheels could not be assumed to be realistic and it would not make sense to study the stresses the wheels are subject to. Indeed, real wheel-ground contacts modelling require challenging characterisation activities that are not subject of this thesis.

Future studies could be focused exactly on this topic. Several research activities into solving this problem are already in progress; in particular a big effort is being done in modelling the non-linear contact of wheel on the soft-soil terrain characterising the Mars surface.

Overall, once the desired contact accuracy will be reached, the developed control system model can be used to drive the rover as required, so that it is possible to study real stesses acting on the wheels when the rover is performing specific manoeuvres.

Bibliography

[1] AIRBUS. *ExoMars Rover Vehicle, Absolute Localisation Description.* 2016.

[2] AIRBUS. *ExoMars Rover Vehicle Locomotion Control System Design Description.* 2016.

[3] AIRBUS. *ExoMars Rover Vehicle Trajectory Control System Design Description.* 2016.

[4] AIRBUS. *ExoMars Rover Vehicle Design Description.* 2017.

[5] AIRBUS. *ExoMars Rover Vehicle Mobility Design, Development and Verification Plan.* 2017.

[6] ASI. *L'Esplorazione Umana dello Spazio.* from course material.

[7] ASTRIUM. *ExoMars Rover Vehicle, Absolute Localisation Description.* 2018.

[8] EADS Astrium. *EExoMars Rover Vehicle. Integrated (Rover) Vehicle Breadboard User Manual and Commissioning Report.* 2009.

[9] K. Astrom and T. Hagglund. *PID Controllers:Theory, Design, and Tuning.* 1995.

[10] K. Astrom and R. M. Murray. *Feedback Systems, An Introduction for Scientists and Engineers.* 2016.

[11] L. Biagiotti and C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots.* 2008.

[12] B. Bona. *Robotics course material available at http://www.ladispe.polito.it/corsi/meccatronica/01PEEQW/2016-17/exams.html.* 2016.

[13] P. Corke. *Robotics, Vision and Control. Fundamental algorithms in Matlab.* 2013.

[14] MacDonald Dettwiler and Associates Inc. (MDA). *ADE Control Design Cumulative Summary Report.* 2017.

[15] MacDonald Dettwiler and Associates Inc. (MDA). *MDA BEMA Detailed Design Document*. 2017.

[16] ESA. *The Red Planet*. 2014, http://exploration.esa.int/mars/44997-the-red-planet/.

[17] ESA. *EXOMARS ROVER*. 2015, http://exploration.esa.int/mars/45084-exomars-rover/.

[18] ESA. *EXOMARS MISSION (2020)*. 2016, http://exploration.esa.int/mars/48088-mission-overview/.

[19] ESA. *EXOMARS TRACE GAS ORBITER AND SCHIAPARELLI MISSION (2016)*. 2017, http://exploration.esa.int/mars/46124-mission-overview/.

[20] Altair HyperWorks. *MotionSolve User's Guide*. 2017.

[21] J. P. Laumond. *Robot Motion Planning and Control*. 1997.

[22] NASA. *Why do we send robots to space?* 2017, https://spaceplace.nasa.gov/space-robots/en/.

[23] C. Secchi. *Robotica Mobile course material available at http://www.automazione.ingre.unimore.it/*. 2010.

[24] R. Siegwart and I. R Nourbakhsh. *Introduction to Autonomous Mobile Robot*. 2004.

[25] solidThinking. *Activate Reference Guide*. 2017.

[26] solidThinking. *Extended Definitions for solidThinking Activate 2017.3*. 2017.

[27] Thales Alenia Space. *EXOMARS ADE, System Technical Description*. 2018.

[28] Altair University. *Practical Aspects of Multi-Body Simulation with HyperWork*. 2015.

[29] J. Vago, O. Witasse, P. Baglioni, A. Haldemann, G. Gianfiglio, T. Blancquaert, Don McCoy, Rolf de Groot, and the ExoMars team. *ESA's next step in Mars exploration*. 2013.

Acknowledgements

Here I am, writing the last lines of this thesis that enshrine the end of a major chapter in my life. So many people contributed in one way or another in the achievement of this milestone, and even if it is not easy to resume all my grateful in few words, I want to thank them all, trying not to forget anyone who took part in this journey.

I would first like to thank the Company Thales Alenia Space Italia for giving me the opportunity of internship during which I had a great chance for learning and experiencing professional work. My gratitude goes to the Mechanical CAE group, in particular to my thesis advisor Dott. Gerlando Augello for allowing me to live this growth experience, for the support, advices and guidance he gave me throughout both the internship and thesis development periods.

I am also deeply grateful to Prof. Marcello Chiaberge, the door to its office was always open whenever I ran into a trouble or had a question about my research or writing.

Special thanks are due to Andrea Merlo e Ciro Napolitano for coming to my aid; their support was essential not only for the thesis goal achievement, but also for bettering the working atmosphere: with their constant good mood they have contributed to tone down my ever-present despair. Thanks to "boss" Andrea for being the teacher everyone would like to have and thanks to "slave" Ciro for making himself always available. Thanks for all the little fingers raised up.

Thanks to all my temporary fellows: to Antonio for the precious support, to Mauro for keeping me company during several hard-working evening, to Marco and Gianluca for their funny jokes, and obviously to Rocco for entertaining us with his endless tales and for reading my dissertation with surprising interest.

I also thank a special friend, Dario, without whom all this wouldn't be possible. Who knew we could become colleagues?

Then, I must express all my gratitude to my family, for being always present but never pressing, for bearing my huge irritability during exam periods and for joying with me of my goal achievements. Thanks to my dad for passing down to me his "engineering mind " but especially his stubbornness. Thanks to my little but special sister, because above all I know she loves me and I love her; by hoping to be always the right example to follow, I wish her to make all her dreams come true. Thanks to the strongest woman I know, my Mami. The achievement of this final goal is in great part due to her, because she is my point of reference, because she never gave up on me, because she always understood when something was wrong and always tried to help me, because she knew when not to talk to me and leave lose myself in my studies and at the same time, she knew when I needed to rest, go out and eat some uramaki together.

Thanks to Alberto for being always by my side. Thanks for being everything I need, because spending time together meant forgetting about all problems that swirl through my head. Thanks for giving me all your love never asking for anything in exchange. Thanks for being always there, with open arms, ready to hug me whenever I need. Thanks for teaching me how to dream ... and I won't stop.

Thanks to the BigBamboo Family, to Andrea Chiara Davide Federico Giulia Ilaria Marco Mario and Roberta, rigorously with full names, for being the best fellowship. Thanks for being the perfect distraction to escape from hard moments.

Thanks to my mechatronic fellow travellers, to Alessio Angelo Aurora Daniele Eva Federico Giulia Luca Massimo Ondina and Simone, for sharing with me all feelings university life implies. And thanks to my old fellows Danilo Federica and Nicola for keeping company during K classes.

Thanks to Edoardo for being the best friend ever, because even though we are not physically close I know I can always count on him.

Thanks to Giulia, for being my partner in misadventure; thanks because I know that if I say "see you later, alligator" she will reply "in a while, crocodile".

Thanks to my thesis fellows, to Carmine Francesco Gabriele Miriana Rocco and Stefano, for withstanding my long faces and for lightening the mood.

Thanks to the NothingWorks Team, to Bianca Demetrio Elisa Francesco Riccardo Stefano, for remembering me that if nothing works everything is ok.

And now, *may the force be with me.*