POLITECNICO DI TORINO

Master Degree in Aerospace Engineering

Master Degree Thesis

Modelling and Control of a Skid-Steering Mobile Robot for Indoor Trajectory Tracking Applications

Supervisors
Prof. Giorgio Guglieri
Assistant Prof. Elisa Capello

Candidate
Filippo Arbinolo
251378

Tutor
Assistant Prof. Daniele Sartori
Shanghai Jiao Tong University

April 2020

This thesis was written under the supervision of Assistant Prof. Daniele Sartori from the Department of Electrical Engineering and Computer Science (EECS) of the Shanghai Jiao Tong University, China.

The experimental tests were conducted in collaboration with the Shanghai BeiDou Research Institute by the indoor testing facility located in West Hongqiao, Shanghai, China.

Abstract

The present MSc thesis addresses the problem of modelling and controlling a four-wheel Skid-Steering Mobile Robot (SSMR) used for indoor trajectory tracking applications. The industries in need of a simple and robust mobile robot have long been drawn to SSMR solutions, but their underlying dynamic complexity has repeatedly hampered their true potential. As this category of robots is forced to skid in order to turn, they are characterised by complex wheel-ground interactions, which primarily exhibit as non-linear time-varying frictional forces. Such phenomena profoundly affect the behaviour of an SSMR and are responsible for unreliable dynamic models and inaccurate motion control systems.

This research work builds on the literature resources currently available and advances a new modelling scheme containing an innovative friction model. Two trajectory tracking control systems are then presented: a Proportional-Integral-Derivative (PID) and a Linear Quadratic Regulator (LQR) optimal control. Furthermore, this research is characterised by the novel use of Artificial Intelligence (AI), namely Differential Evolution (DE), for optimisation purposes. The self-adapting jDE/best/2 mutation scheme with gradient descent proved to be an undoubtedly effective alternative to system identification techniques generally employed for unknown physical parameters estimation. Moreover, DE was applied to get around the traditional empirical controller tuning methods and, instead, to find a global optimum to the control problem. Despite the control systems still required a minimum degree of manual tuning, the performances delivered by the DE results outmatched the best measurements achieved with manual tuning alone.

Ultimately, the promising experimental results acquired at the BeiDou Research Institute - in Shanghai, China - are presented. They validate the modelling assumptions and serve as a proof of concept for the use of DE algorithms for controllers tuning.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

CAGR Compound Annual Growth Rate

CoG Center of Gravity

DARPA Defence Advanced Research Projects Agency

DE Differential Evolution

DoD Departement of Defense

EA Evolutionary Algorithm

EKF Extended Kalman Filter

EP Evolutionary Programming

ES Evolutionary Strategy

GA Genetic Algorithm

GP Genetic Programming

GPS Global Positioning System

ICR Instantaneous Center of Rotation

IMU Inertial Measurement Unit

IR Infrared

LQR Linear-Quadratic Regulator

LTI Linear and Time-Invariant

MCU Micro Controller Unit

MIMO Multiple-Input and Multiple-Output

PID Proportional-Integral-Derivative

PSO Particle Swarm Optimisation

RMS Root Mean Square

ROS Robot Operating System

SAR Search and Rescue

SISO Single-Input and Single-Output

SSMR Skid-Steering Mobile Robot

UAV Unamanned Aerial Vehicle

UGV Unamanned Ground Vehicle

UMV Unamanned Maritime Vehicle

UVS Unmanned Vehicles Systems

List of Symbols

$\boldsymbol{A}$ Constraint Vector

a, b, c Mobile robot geometric constants

$delay$ Commands delay

CR Crossover rate

df Objective function relative error threshold

d_{haus} Hausdorff distance

$\dot{\boldsymbol{q}}$ UGV global velocity vector

d_{ratio} Distance ratio

d_{rms} RMS of minimum distances

$\boldsymbol{E}$ Input transformation matrix

e_{bend} Total bending energy

$e_{bend-ratio}$ Total bending energy ratio

E Kinetic energy

$e(t)$ Generic signal error

$\boldsymbol{\eta}$ Kinematic control input vector

ϕ, θ, ψ Euler angles [rad]

$F(X, Y, Z)$ Global Coordinate System

$f(x, y, z)$ Local Coordinate System

F_{act} Active force/torque applied to the system

$\boldsymbol{F}$ Resultant vector of the active forces and torques

FDJ Dinamic friction force/torque

FF Actual friction force/torque

F_f Generic friction force

F Mutation scale factor

FSJ Static friction force/torque

F_x Resulting active force

F_y Resulting resistive force

g Gravitational acceleration: 9.81 m/s^2

Γ Active torque and force vector

I Mobile robot moment of inertia $[kgm^2]$

K_D PID derivative gain

$K_{P_{FF}}$ PID feedforward proportional gain

K_I PID integral gain

K_P PID proportional gain

k_s Signum approximation accuracy parameter

$\boldsymbol{\lambda}$ Vector of Lagrange multipliers

$\boldsymbol{M}$ Inertia matrix

m Vehicle mass

M_r Resulting resistive torque

M Resulting active torque

μ_c Coulomb friction coefficient

μ_v Viscous friction coefficient

N Normal force

NG Number of generations

NP Number of parameter vectors per population

nr Number of vectors used for mutation

$\boldsymbol{q}$ UGV state vector

$\boldsymbol{R}$ Vector of resultant reactive forces and torque

$\boldsymbol{R_{lg}}$ Local-global rotation matrix

r Effective rolling radius

R_x Resulting rolling resistance

$\boldsymbol{S}$ General coordinates transformation matrix

τ_1 F adaptation parameter

τ_2 CR adaptation parameter

$\boldsymbol{\tau}$ Torque control input vector

$\tau_{wheel_{max}}$ Torque limits

θ Heading angle

U Potential energy

u_x Longitudinal rolling coefficient

$\boldsymbol{V}$ UGV velocity vector

Acknowledgements

I wish to thank Prof. Giorgio Guglieri from the Politecnico di Torino and Assistant Prof. Daniele Sartori from the Shanghai Jiao Tong University, who supervised the realisation of this work. Their infinite availability and invaluable direction supported me throughout the entire research project. I thank Davide Locardi, with whom I shared the unique experience in Shanghai, and also Gabriele Ermacora and Manfredi Di Rovasenda for the unhesitating assistance they provided me whenever needed. Together, they helped to recreate a sense of belonging in a place as far from home as China.

The past five years would not have been possible without the trust, fellowship and perseverance brought by Alberto, Enrico, Federica, Federico and Francesco, who I am lucky to call friends. We strived together for what we are now achieving, and we certainly will for what is to come. Lastly, I would like to thank my parents, Alberto and Anna Maria, my sister, Costanza, and my grandparents, Filippo and Rosa Maria, who nourished my passions and inspired me throughout my life.

Preface

The present Master thesis is realised within the Department of Mechanical and Aerospace Engineering (DIMEAS) of the Politecnico di Torino for the fulfilment of the Master degree in Aerospace Engineering. The research project was conducted at the Shanghai Jiao Tong University (SJTU) Insitute for Sensing and Navigation (ISN) and by the BeiDou Research Institute of Shanghai over four months and with the help of another student from the Politecnico di Torino. The planned stay in China was abruptly shortened by a month due to an outbreak of COVID-19, which forced all the institutions to close. Therefore, as clearly evident from the last chapter, the results here presented do not portray the desired and planned outcome, as it is believed an extra month of experimental tests would have produced substantially better results, both in accuracy and meaningfulness.

This project aimed at designing and implementing three control laws used to track an indoor trajectory with an Unamanned Ground Vehicle (UGV), namely a Skid-Steering Mobile Robot (SSMR): a traditional Proportional-Integral-Derivative (PID) control, and two modern controls, namely a Linear-Quadratic Regulator (LQR) and Sliding Mode Control (SMC). The first was meant to serve as a reference to the other two, more interesting, controllers. However, before their design was possible, it was necessary to acquire enough knowledge of the vehicle analysed. Such knowledge was, in the first place, the dynamic behaviour of the SSMR. In the lack of a mathematical description, part of the project was, in fact, its eduction, a necessary step for faithful computer simulations and theoretical analyses of the control systems.

This thesis, in particular, focuses on the rover mathematical modelling, and the design and testing of the PID and LQR controls. The theoretical foundation for each one of them was derived from existing textbooks and research papers, but some personal contributions were added. The main ones are, undoubtedly, the introduction of a novel friction model and the use of Artificial Intelligence (AI) for both system identification and controllers tuning, in the form of a Differential Evolution (DE) algorithm. As explained in the introduction, friction is not only a complex phenomenon but also the underlying reason why SSMRs can traverse curved paths. It is therefore not possible to neglect it and, on the contrary, it is essential to simulate it with the highest possible degree of reality. The model here

introduced appears to solve common numerical problems and achieve the desired faithfulness. The DE algorithms, on the other hand, proved to be an invaluable tool when working with multivariable problems. Furthermore, as already highlighted by some recent research papers, the controllers tuning capabilities it provides make it a fascinating topic to study.

The thesis is structured as follows: Chapter 1 introduces the UGV technology, the principles of DE and examines the theoretical and practical problems addressed during this research; Chapter 2 defines the three mathematical models - two non-linear and one linear - conceived starting from the existing publications; Chapter 3 describes the experimental set-up employed, exposes the optimisation process which let the determination of the unknown parameters present in the mathematical models, and compares the three to explain which one is better, one last section analyses Gazebo simulator and shows why it was not used for this research; Chapter 4 presents the mathematical description of the control systems, their tuning process and the simulation results, which are then compared to the experimental tests; finally, Chapter 5 draws the conclusions and suggests possible improvements for future works.

Chapter 1

Introduction

1.1 Overview of Unmanned Ground Vehicles

Today's demand for automated, robotic and unmanned systems is primarily driven by applications that are inherently repetitive, unpleasant or dangerous. Despite their high cost of acquisition and integration, robotic systems are being employed by an increasing number of military and commercial realities. One notable example is provided by robotic arms, or *manipulators*, which are now widely used in the world of industrial manufacturing. First introduced in 1961, they now comprise a market share of 16,2 bln $ with an expected CAGR[1] of 14% over the period 2019-2021 [1]. Their speed and superhuman precision perfectly fit the realm of assembly lines, but they are characterised by a fundamental disadvantage: lack of mobility. Mobile robots, on the other hand, address this problem and can flexibly apply their talents whenever it is most effective. At the same time, the possibility to reduce casualties and increase combat effectiveness has always attracted the attention of the military, which in recent years has shown numerous successful applications of Unamanned Aerial Vehicle (UAV).

Unmanned Vehicles Systems (UVS) exist in three types (Fig.1.1): the UAV mentioned above, operating in the air, Unamanned Maritime Vehicle (UMV), operating both over and under the water surface, and lastly UGV, operating on land, both indoors and outdoors. Like many inventions, the earliest known descriptions of UVS appear to pre-date their development by many centuries, but it was not until 1898 that Nikola Tesla publicly showed the first remote control vehicle. It was a small radio-controlled boat exhibited in Madison Square Garden, New York. A few years later, in 1904, the Englishman, Jack Kitchen, experimented with similar technology and a steam-powered launch along the shores of Lake Windermere, and in 1906 a Spanish mathematician, Leonardo Torres y Quevedo, demonstrated the

[1] Compound Annual Growth Rate

remote control of a ship before the King of Spain.

Figure 1.1. Examples of modern UVS - From left to right: Northrop Grumman RQ-4 Global Hawk (UAV); DARPA ACTUV (UMV); Foster-Miller TALON (UGV)

Although themselves were slowed down by the cancellation of many R&D programs due to the lack of funds, UAVs have always received the greatest attention from both the scientific community and the financial backers, which were often the military. To better understand it, it is sufficient to study their development throughout history. The first remotely controlled aircraft is believed to be the Ruston Proctor AT developed in 1916 by Archibald Low. It was never considered a success, as it was flown only a handful of times and often crashed. More interesting applications of UAVs were designed during World War II. On the one hand, the USA funded the programs Operation Aphrodite and Anvil, which gave life to the "Weary Willies": modified B-17 bombers, renamed BQ-7, remotely controlled from a nearby manned aircraft. They were used to target bomb-resistant German fortifications, but in the final analysis, they were probably more dangerous to the pilot flying than they were to the Germans. On the other hand, the Germans were able to exploit the potential of UAVs to a much greater extent. The V-1 and V-2 flying bombs were in fact highly reliable due to their well-designed engines and guidance systems and were used effectively for the whole duration of WWII. [2].

Meanwhile, research and development of UGVs lagged behind presumably for the same reasons of today: the complexity of the terrain. As in the case of UAVs, this technology was initially intended for military use. In 1915, Victor Villar and Stafford Talbot, both from the United Kingdom, patented the Land Torpedo. It was meant to clear a channel through obstacles, such as barber wires found on the battlefield. In the 1930s the Soviet Army deployed the TT-26 tanks, commonly known as "Teletank". They were the first radio-controlled ground vehicles and were used both during WWII and the Winter War of 1939-1940 against Finland. Each teletank had to be controlled from another tank, following from a maximum

distance of 1500m. In 1940, the French engineer, Adolphe Kegresse, built the Beetle Tank. Its designs fell into the hands of the invading Germans, and the production of the Goliath Tracked Mine - an alternative name for the same vehicle - was commissioned to the Carl Borgward Corporation in Bremmen. The company produced a total of 7'500 units, sacrificially used as anti-tank, anti-infrastructure or bridge-demolition devices. Its design was never considered a success, as it embarked unreliable electric motors - or diesel engines in some cases - travelled very slowly (9 km/h), had a thin armour and reduced ground clearance (11 cm). Furthermore, the central unit was connected to a small control box via a triple-strand of telephone wires, which were vulnerable to entanglement and damage [3].

Despite being often overlooked, UGVs were the first unmanned system to achieve full autonomy, as accomplished around 1949 by William Grey Walter, from the Burden Neurological Institute in Bristol, England [4]. He designed Elmer and Elsie (Fig. 1.1), known as *Machina Speculatrix*, two robots able to respond to contact with other objects and to sense light via phototubes. They relied upon these to autonomously navigate using pure analogue components: two vacuum tube amplifiers that drove relays to control steering and drive motors. The two independent motors acted as two nerve cells, allowing the robots to exhibit simple intelligence. Indeed, Walter developed them to study the complex behaviour of networks of simple organisms [5]. A decade later, the Johns Hopkins University Applied Physics Laboratory built the Johns Hopkins Beast (Fig. 1.1), a sophisticated autonomous vehicle which could roam the University corridors centring itself using sonar to navigate. It could also locate wall sockets - painted in black over white walls - and plug itself in whenever it was low on battery [5].

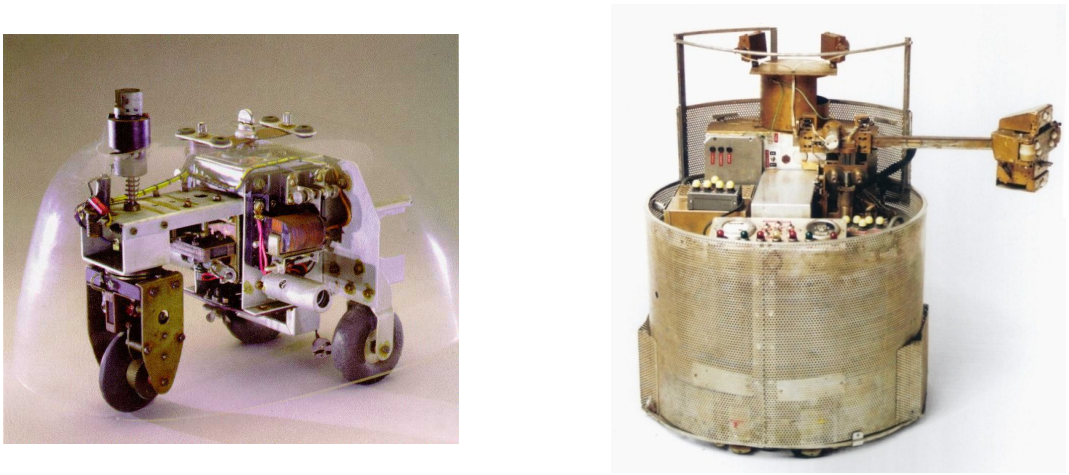

Figure 1.2. Left: William Walter's Elsie - 1950; Right: Johns Hopkins Beast Model II - 1962

During the late 1960s, Stanford University started to emerge as a pioneer of

robotics. From 1962 to 1972 the Defence Advanced Research Projects Agency (DARPA) funded the Stanford Research Institute - now reconstituted into SRI International - to design a mobile robot system named Shakey [6]. While other robots would have to be instructed on each individual step for completing a task, Shakey could receive some general predefined commands which were then automatically broken down and executed depending on the world surrounding it. Such commands were: travelling from one location to another, turning light switches on and off, climbing up and down from rigid objects, and pushing movable bodies around. Each one of them required the ability to perceive and model the environment and use its information for tasks like planning and route finding. The project led to numerous advancements in Artificial Intelligence (AI) techniques, such as the development of STRIPS, an automatic planning and scheduling software useful to find solutions in multidimensional spaces [7], and also in the fields of path search and motion planning, with the invention of the A* search algorithm and the visibility graph method. Shakey carried a television camera, an optical range-finder in a movable "head" and some bump sensors (Fig.1.3). The whole sequence of data processing and command definition was carried out on an external mainframe computer, at the time almost as big as a room.

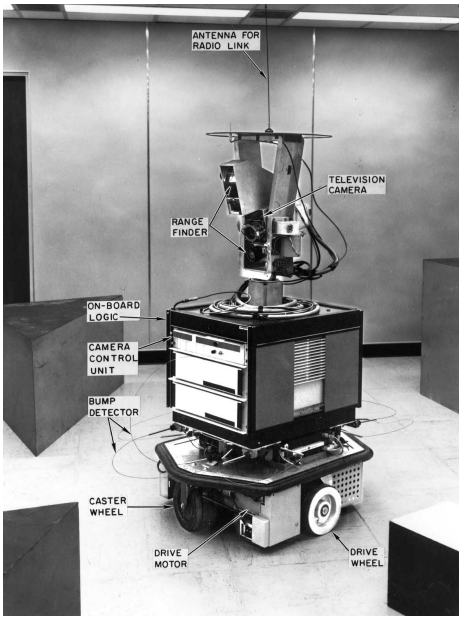

Figure 1.3. Stanford Shakey - Stanford Research Institute (SRI International), 1960 ca.

Figure 1.4. Stanford Cart - Stanford Advanced Artificial Intelligence Laboratory (SAIL), 1970 ca.

From around 1960 to 1980, the Stanford Artificial Intelligence Laboratory (SAIL)

worked on a long-term project known as Stanford Cart (Fig.1.4) [8]. The four-wheeled robot took many shapes, as it started as a moon rover and ended up being an autonomous vehicle. Similarly to Shakey, a computer program drove Cart through cluttered spaces, gaining its knowledge of the world entirely from images broadcasted by the on-board TV system. It planned an obstacle-avoiding path to the desired destination on the basis of a model built with this information. The plan changed as the Cart perceived new obstacles on its journey. The system was reliable for short runs, but slow: it moved 1 m every 10 to 15 minutes, in lurches. It is interesting to point out that the rover system was designed with maximum mechanical and control system flexibility to support a wide range of research in perception and control: it was a precursor of today's robots for scientific research.

Meanwhile, the field of mobile robotics came to be split into two branches. Up to that point, wheels were the only feasible means of locomotions, but with Space General Corporation walking machines of the early 1960s, and the General Electric quadruped of 1968, it became clear that legged designs were possible [9]. Since then, the desire to experiment on increasing levels of autonomy prompted a number of institutions to finance research programs in the field of legged robotics with a focus on artificial intelligence and behaviour-based designs. Among these, it is worth mentioning the Massachusetts Institute of Technology (MIT) which in 1989 developed Genghis [10], a six-legged, insect-like robot, and the Waseda University in Japan, which in 1984 created Wabot-2, one of the earliest anthropomorphic - or humanoid - robots, able to interact with a human, read a musical score and play tunes on keyboard instruments [11]. Nowadays, the development gap between wheeled and legged mobile robotics is still huge, for legged robots still suffer from huge limitations: they only achieve slow speeds, are challenging to build and require complex control algorithms. Wheeled robotics, on the other hand, has kept delivering significant results since the early achievements of Cart and Shakey. It is currently the most popular locomotion mechanism.

Picking up from their technical legacy, in 1977 the LAAS robotics group in Toulouse, France, started working on HILARE 1 (Fig.1.5), a modular robot used for experimental support [12]. It was a blend of AI, computer science, control theory and instrumentation. It served as a test-bed for a variety of problems in the domains of perception, learning, decision-making and communication. Its modularity allowed for incremental changes, which then stemmed into HILARE 2 and HILARE 2bis, a trailer-pulling wheeled robot eventually built in 1992 [13]. The works based on HILARE helped to consolidate what are today's subsystems of any mobile wheeled robot [14]:

- Power
- Communication
- Human-Robot Interaction

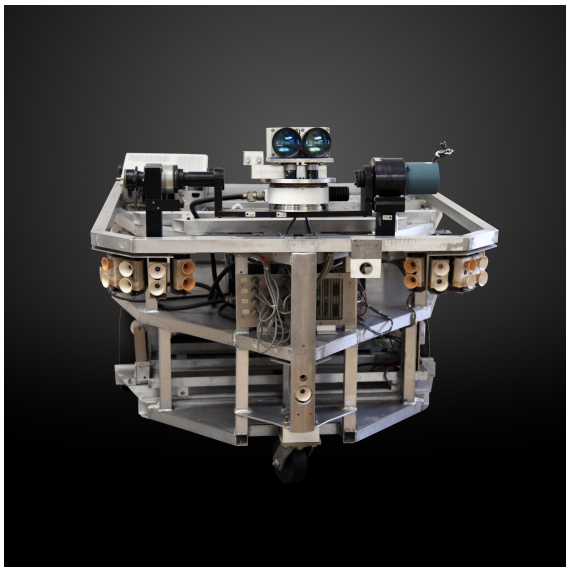

Figure 1.5. HILARE 1 - LAAS Robotics Group, 1977

- Health Maintenance
- Locomotion
- Autonomous Behaviour
  - Planning
  - Perception
  - Behaviours and Skills
  - Navigation
  - Learning and Adaptation

Clearly, mobile robots are intended for much more than just moving from one point to another, but when it comes to governing their motion, the standard control scheme is the one shown in Fig. 1.6, which is an exemplified version of what is presented in [15]. Motion control is the combination of path execution and actuation. In the case of wheeled robots, research focuses on three main areas: manoeuvrability, controllability and stability.

They are all affected by the combination of two physical properties: wheel types and geometry. Unlike automobiles, which are primarily designed for a highly standardised environment (the road network), mobile robots are designed for applications in a wide variety of situations. Automobiles all share similar wheel configurations because there is only one region in the design space that maximises manoeuvrability, controllability and stability for their standard environment: the

paved roadway. However, there is no single wheel configuration that maximises these qualities for the variety of environments faced by different mobile robots. This is the reason why there is a significant number of wheel configurations for mobile robots. In fact, few robots use the Ackermann wheel configuration of the automobile because of its poor manoeuvrability, with the exception of the ones designed for road systems.

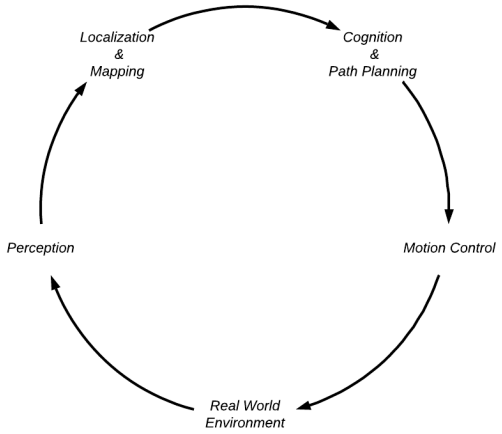

Figure 1.6. Wheeled Mobile Robot Control Scheme

Wheels are generally of four types: standard, castor, Swedish and spherical. The last two offer more degrees of freedom but are more difficult to produce. The total number of wheels employed directly affects the overall configuration. Conventionally, static stability is achieved with a minimum of three wheels and the robot's centre of gravity within the triangle described by them. However, most designs employ two, three, four or six wheels. Furthermore, taking into account the presence, or lack thereof, of a steering system, it is possible to obtain the designs depicted in Fig. 1.7.

When it comes to manoeuvrability, there are robots that can move in any direction (x, y) at any time. Clearly, it requires the use of either Swedish or spherical wheels. Other robots can do the same but first rotating on their vertical axis, sometimes without even changing their ground footprint, as in the case of two-wheel differential drive vehicles employing standard wheels. The less manoeuvrable designs are those using the Ackermann steering, which has a turning diameter higher than the vehicle itself.

Generally, there is a reverse correlation between manoeuvrability and controllability: omnidirectional robots are by far the most difficult to control. In order to understand the statement above, it is useful to imagine what is required to drive a robot in a straight line: in the case of an Ackermann system it is sufficient to

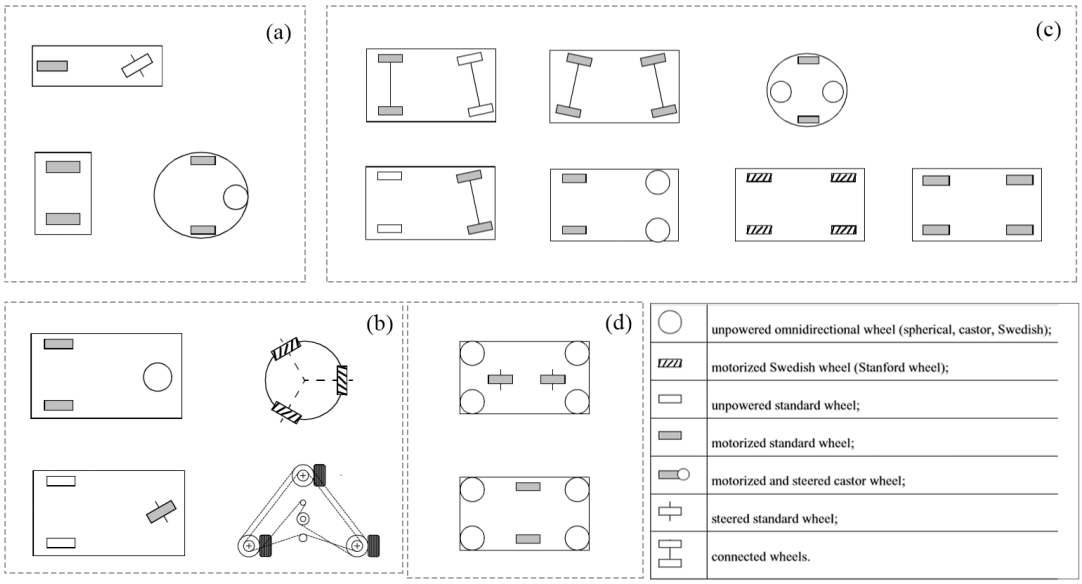

Figure 1.7. Possible wheel configurations: a) 2 wheels b) 3 wheels c) 4 wheels d) 6 wheels

centre and lock the wheels in position; differential drive systems[2] require that the driving motors follow the same velocity profile, a difficult feat due to differences in the motors, in the wheels and in the underlying terrain. The differential drive is used in the absence of a steering system, but they are often constrained to a type of steering known as skid-steering. Skid-steering is analogous to the tracked slip used by tanks, and when it is used in loose terrains, it dramatically increases manoeuvrability, with a detrimental effect on controllability. This type of design will be further analysed in Section 1.2.

For the purpose of this work, it is necessary to introduce another block from the control scheme depicted in Figure 1.6: perception. The perception of the real-world environment takes place in the following order:

- Sensing
- Information Extraction and Interpretation

As the name suggests, the sensory part of a robot depends on its sensors, which in turn depend on the robot applications. The most common are: wheel encoders, Inertial Measurement Unit (IMU), Global Positioning System (GPS), radar, LIDAR

[2]Differential drive systems have mechanisms that can drive each wheel independently.

and sonar; an in-depth review of such systems is provided by [16]. Statistical sensor fusion techniques, such as Kalam filtering, are then employed to merge multiple sensor readings and obtain estimates more accurate than one single measurement alone. Without sensor fusion, Simultaneous Localisation and Mapping (SLAM) techniques would not be possible. SLAM, in particular, is a challenging area of application of high-dimensional nonlinear filtering problems [17], but it is also one of the building blocks of autonomous navigation. On the other hand, dead reckoning is used when there is no need for autonomy, that is when map building and path planning can be performed offline, prior to the deployment of the robot. Dead reckoning is the process of determining one's position, uniquely using information about its initial conditions, speed and acceleration. It does not correct its errors using external world observations - for instance, via radar, LIDAR, GPS - but with well-tuned sensor fusion algorithms, it can be a good starting point for experimental purposes.

From what has been said about mobile robotics, it is easy to realise that the design process involves the integration of many bodies of knowledge. No mean feat, this makes mobile robotics a field as interdisciplinary as there can be. Up to the early 2000s, however, it was extremely difficult to find experts who were also willing to deal with the complex software infrastructures used by robots. In 2007, Willow Garage funded a project of the University of Stanford which was born from the desire to create a universal robotic framework that aimed at standardising and simplifying the complex interface with robots. It came to be known as ROS, and it now a vast collection of tools, libraries and conventions which are natively supported by more than 80 commercial robots. Without ROS, its counterparts, cheap technology and advances in miniaturisation, sensors, computer processing, signal and image processing, communications techniques and material science, robotics would not have progressed as much as it has done in the past five decades. Today's applications of Unmanned Ground Vehicles span several industries; some notable examples are:

- Self-driving Cars: Waymo, Uber, BMW
- Industrial Automated Guided Vehicles (AGVs): Siemens SIMATIC, Comau Agile1500
- Mine exploration: Carnegie Mellon Groundhog, Komatsu 930E-AT
- Outdoor exploration: Clearpath Warthog, Leo Rover
- Search and Rescue (SAR): VIEW-FINDER Project, ICARUS SUGV, ICARUS LUGV
- Fire fighting: Thermite RS-1, Shark Robotics COLOSSUS
- Volcanology: Carnegie Mellon Dante II, NASA Volcano-bot

- Home appliances: iRobot Roomba, Husqvarna Automower.
- Hospital and nursing: HelpMate, Care-O-bot
- Space exploration: NASA Curiosity (Fig.1.8), CNSA Yutu
- Military: USMC Gladiator TUGV, Israel Defense Forces Guardium, Foster-Miller TALON

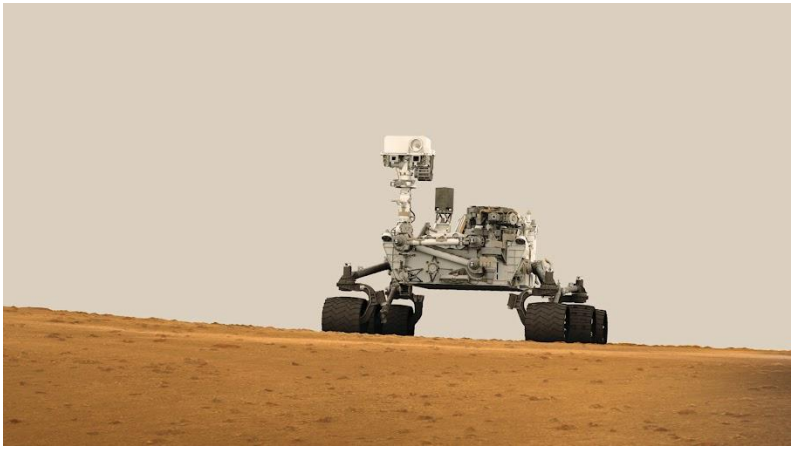

Figure 1.8. NASA Curiosity rover

1.2 Skid-Steering Control Problem and Contribution

Many of today's UGVs employ skid-steering mechanisms due to their simplicity, robustness and manoeuvrability, even in the roughest terrains. SAR, defence, outdoors exploration and household services are only some of its fields of application, but most of the systems deployed are teleoperated and have limited or no autonomous functions [18]. This choice can be explained analysing the mechanical behaviour of a SSMR: SSMRs, as their name suggests, do not have a steering mechanism and are thus able to follow curved paths by driving their wheels at different speeds, resulting in a sliding motion. Sliding, in turn, has some significant implications on both modelling and control.

Since skidding is the underlying physical effect that makes a SSMR steer, modelling this type of vehicle is extremely difficult. Despite being widely adopted in the field of robotics, pure kinematic models with pure-rolling and no-slip hypotheses are not sufficient to determine the response of the mobile robot to a set of inputs [19]. Such models have been extensively studied over the past two decades, particularly for two-wheel differential-drive mobile robots, which allow for a simplifying assumption without loss of generality: a nonholonomic constraint of the zero-later velocity of the wheel contact points. For SSMRs, however, nonzero wheel lateral velocity is allowed, and the zero-velocity constraint is no longer valid. A faithful dynamic model, however, assumes the perfect knowledge of any wheel-ground interaction to compute the exact value of all friction forces. The time-varying friction forces depend extensively on the underlying soil, on the wear state of the rubber wheels and on a significant number of parameters which can not be accurately predicted beforehand.

The degree of uncertainty about the wheel-ground friction forces, makes the prediction of the vehicle ICR nearly impossible. Knowing the exact position of the ICR is, in turn, essential to design robust control schemes and to obtain accurate dead reckoning estimates, which would otherwise deteriorate when wheel slip in non-negligible. Many present SSMR applications rely on external localisation measurements (GPS, LIDAR, radar, etc.) to correct dead recking readings. There are cases, however, that do not allow for correction. One example is provided by Mars Exploration Rovers, which exhibited considerable wheel slips [20][21].

In recent years some research studies tried to address the aforementioned problems. [22] proposed to limit the longitudinal position of the ICR - that is the x coordinate with respect to the local coordinate system, see Chap. 2 - to the wheelbase, in order to rule out all the cases of uncontrolled skidding. The obtained model was then used to design a nonlinear controller following the dynamic feedback linearization paradigm; robustness what only demonstrated for straight paths. [23] built on such results and extended them to a time differentiable and time-varying

control scheme based on Dixon's kinematic controller [24] and the strategy of forcing some transformed states to track an exogenous exponentially decaying signal produced by a tunable oscillator. They also described friction as a superposition of Coulomb and viscous friction but left the ICR to be supposed beforehand. [25] advanced a kinematic model based on a new set of parameters which can be obtained experimentally by employing a simple laser scanner. This method allows to evaluate the position of the ICR but may result too costly for real-time motion control and dead reckoning. Indeed, most advanced control solutions are often deemed unfeasible for real-time applications, like the slip-aware model predictive optimal control of [26]. Some simpler solutions came in the form of adaptive controllers [27], PI controllers [28][29], Proportional-Integral-Derivative (PID) controllers and LQR control laws [30], all of which were limited to slow-speed applications.

This thesis thus proposes a new modelling scheme based on the friction model presented by Borello e Dalla Vedova [31], which is ideally suited for accurate model-based simulations. The problem related to the exact knowledge of the ICR position is addressed by applying some simplifying assumptions to the model, expecting the control system to compensate for unpredicted dynamical behaviours. Furthermore, the nonlinear mathematical model is simplified assuming low-speed bi-dimensional applications - in other words, the SSMR will be constrained to planar motion only - and its unknown parameters are determined by means of a DE algorithm. Two reliable and straightforward trajectory tracking control schemes are then proposed: a PID controller and an optimal Linear-Quadratic Regulator (LQR) controller, both tuned using the same DE algorithm employed for system identification.

1.3 PID Control Related Work and Contribution

PID controllers are still widely used for mobile robots motor control and for industrial systems due to their good performance in a wide range of operating conditions, their straightforward architecture and the overall familiarity in the field of control systems [32]. However, its limited capability in scenarios which present nonlinearities, external disturbances and time-varying parameters, forces the control system engineer to choose a compromise between system robustness and performance.

Over the years various design methods[3] have been proposed to determine the PID controller parameters. Some methods use information about the open-loop step response, e.g. the Coon-Cohen reaction curve method [33], e.g. the Ziegler-Nichols frequency response method [34] or the gain-phase margin approach [35]. These simple tuning laws are often employed by commercial auto-tuner algorithms.

[3]With the term *design method* it is meant the determination of the three parameters of the controller which are the proportional, integral, and derivative gains.

However, since they all rely on very little information about the dynamic behaviour of the system, the results achieved which such methods are often unsatisfactory. For example, the Ziegler-Nichols method may give high overshoots, highly oscillatory dynamics, and long settling times for high-order systems; the Coon-Cohen method, on the other hand, is only valid for systems having S-shaped step responses of the plant, although it has been widely used as a heuristic method. Evolutions of these methods have been proposed with the promise of better results. It is the case of the Åström-Hägglund phase margin method [36] and the refined Ziegler-Nichols method [37]. However, in some circumstances, these methods do not produce adequate closed-loop responses, as in the presence of large time delays which produce oscillatory results. The major limit of these methods is the necessity to express the plant using a transfer function. To overcome these limitations, various methods have been developed to obtain optimal PID parameters, ranging from conventional pole placement [32] to a variety of AI techniques such as Artificial Bee Colony (ABC) optimisation algorithms [38], Simulated Annealing (SA), Population-Based Incremental Learning (PBIL), Particle Swarm Optimisation (PSO), Genetic Algorithm (GA) and Differential Evolution (DE) [39].

Recently, GA has been widely adopted in the search for optimal PID parameters thanks to its ease of implementation and its high adaptability, as demonstrated in [40]. [41] applied GA to derive the optimal gains of a PID controller used in a trajectory tracking application, thus demonstrating its improved performance when compared to conventional tuning methods when it comes to controller precision and convergence speed. Still, some studies have highlighted shortcomings in the performance of GAs, which often show premature convergence, loss of best solution and no assurance of global optimum [42].

GAs limitations are overcome by DE algorithms, which are designed to meet the requirement of the practical minimisation technique [43]. [44] studied the performance of DE, GA and PSO in the optimisation of a PID controller used to control the position of a manipulator, concluding that DE is generally more robust than other methods. [45] applied DE to tune the PID trajectory tracking controller used on a quadrotor. The simulation results show that the controlled system has a satisfactory response and that it is able to deal with the aircraft coupled dynamics, thus demonstrating the effectiveness of the proposed optimisation method.

The method illustrated in this text relies on a cascade multi-loop structure: an inner loop controls the velocity of the vehicle, an outer loop its position. The inner loop also presents two feedforward proportional gains which improve the velocity tracking. A total of ten gains will be tuned using a jDE algorithm with gradient descent, whose search boundaries were set after an initial trial and error analysis of the system to rule out any combination of values which yields unstable behaviours.

1.4 LQR Control Related Work and Contribution

In the realm of state variable feedback systems, and in particular in the field of optimal control, LQR controllers are among the most known and used. The hallmark of the LQR is its simplicity and ease of use, only possible, however, when all the system state variables can be fed to the controller, and the plant dynamics can be linearised. As all optimal controls, in contrast to the classic PID, its design depends on the definition of a range of weights - or performance indices - affecting the cost function the controller is meant to minimise. The use of such parameters makes the controller tuning much more intuitive and understandable, as each one of them is relatable to a physically measurable parameter.

In the field of robotics, many research papers focus on the long-standing problem of the two-wheeled self-balancing robot, a modern version of the inverted pendulum: a nonlinear strongly coupled problem. [46] and later [47] proved the effectiveness of an LQR control system, by linearising the plant around an operating point. The latter, in particular, presents a comparison between LQR and PID, concluding that the optimal control is not only significantly simpler to implement, but also produces better results. Further applications range from humanoid control to manipulators operation.

When it comes to trajectory tracking for nonholonomic systems, [48] uses an iterative LQR (ILQR) by iteratively linearising the plant of a two-wheeled robot around a nominal trajectory and then computing a local optimal feedback control law. [49] applied an LQR to solve the trajectory tracking problem of a simple two-wheeled robot, while [50] achieved similar results with a car-like robot. Most of the research work, however, focuses on aerial drones, as in the case of [51] and [52] who worked on trajectory tracking with quadrotors.

Despite being easier to tune, LQRs require the control system engineer to choose some weights which affect the performance of the system. In recent year, with the surge of AI, many pieces of research tried to apply its methods to tune LQRs. Vishal and Ohri [53] applied a GA to successfully tune the pitch control system of an aircraft, while Assahubulkahfi *et al.* [54] researched the use of PSO for the same matter.

Due to the well-known advantages of LQR control systems and the superior optimisation performances of DE algorithms, it is here proposed a trajectory tracking control system based on the former and tuned with the latter. Integral action is also added to reduce any residual steady sate error on the longitudinal coordinate of the rover. In order to apply the linear controller to the remarkably nonlinear vehicle here analysed, its dynamics is linearised around an equilibrium point, and the result is used to derive a state-space model. Once the optimal gain matrix is obtained by solving the Riccati equation, the control system is tested on the nonlinear description of the SSMR with successful results.

1.5 Optimisation by Differential Evolution

The following paragraphs will introduce the topic of DE, as it is a useful optimisation tool which was extensively used throughout the whole research work here discussed. Indeed, its applications will range from unknown model parameters estimation to controllers tuning.

To solve complex computational problems, researchers have long been looking into nature for inspiration. Optimisation is at the heart of many natural processes, very much like the Darwinian evolution itself. Through millions of years, every species had to adapt its physical structure to survive the environments it inhabited. The observation of the underlying relation between optimisation and biological evolution led to the development of an important paradigm of AI for performing complex search and optimisation: the evolutionary computing techniques. Evolutionary computation uses iterative progress, such as growth or development in a population. Such population is then selected in a guided random search using parallel processing to achieve the desired end. From when they were first conceived, Evolutionary Algorithms (EAs) stemmed into a number of different representatives of the field of nature-inspired metaheuristics. Nowadays, the most common evolutionary methods are: GAs, Evolutionary Programming (EP), Evolutionary Strategies (ESs), Genetic Programming (GP) and DE.

The DE algorithms emerged as a very competitive form of evolutionary algorithms after the first technical report on the matter was written in 1995 by [43]. DE were able to differentiate from other EAs for the following reasons:

- Compared to most EAs, DE is much more simple and straightforward to implement. The simplicity of the code is important for users who are not expert programmers and who are looking for ease of use. It is necessary to point out that when compared to the easy-to-code optimisation algorithm named Particle Swarm Optimisation (PSO), DE has better performances over a variety of problems [55].

- The performance of DE in terms of accuracy, convergence speed, and robustness makes it appealing to various real-world optimisation problems, where the end goal is to find an approximate solution in a reasonable amount of computational time.

- The number of control parameters is minimal: Cr (cross-over rate), F (difference vector scaling factor) and NP (population size) in classical DE. The effects of these parameters are well known, and simple rules exist to improve the performance of the algorithm without any serious computational burden.

1.5.1 Basic Concepts and Formulation

Before dealing with the mathematical formulation of DE algorithms, it is necessary to understand their purpose.

Many scientific and engineering disciplines often face search and optimisation problems. Optimisation, in this sense, means finding the best solution of a problem within the given constraints. The aim is to find a set of system parameters values for which the system performance will be the best under some given conditions. Usually, the parameters governing the system are represented by a vector $\mathbf{X} = \left[x_1, x_2, \ldots, x_D^T\right]$; in the case of real parameters optimization each parameter x_i is a real number. To measure how good is the system performance, an objective function - or fitness function - is designed. The task of the optimisation is therefore to search for the parameter vector $\mathbf{X}^*$ which minimises an objective function $f(\mathbf{X})(f : \Omega \subseteq \mathbb{R}^D \to \mathbb{R})$, i.e. $f(\mathbf{X}^*) < f(\mathbf{X})$ for all $\mathbf{X} \in \Omega$, where Ω is a non-empty large finite set serving as the domain of the search. For unconstrained optimization problems $\Omega = \mathbb{R}^D$. Typically, optimisation probelms are complicated by the existance of non-linear objective functions with multiple local minima. A local minimum $f_l = f(\mathbf{X_l})$ may be defined as $\exists \epsilon > 0 \forall \mathbf{X} \in \Omega : ||\mathbf{X} - \mathbf{X}_l|| < \epsilon \Rightarrow f_l \leq f(\mathbf{X})$, where $||.||$ is a p-norm distance measure.

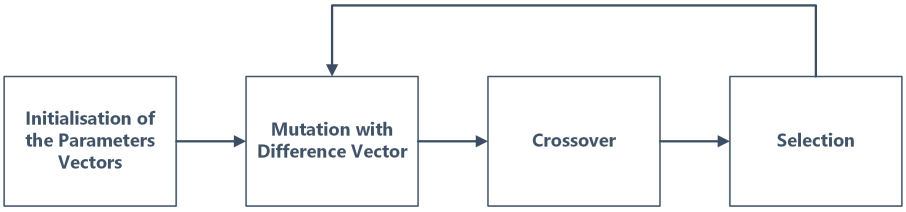

Figure 1.9. DE functioning scheme

DE in particular is a real parameters optimisation algorithm which is composed by four steps, some of which are iteratively repeated throughout the overall optimisation process (Fig. 1.9):

1. Initialisation of the Parameter Vectors
2. Mutation with Difference Vectors
3. Crossover
4. Selection

Each step is discussed in the upcoming paragraphs. Lastly, for the sake of clarity, it is necessary to introduce some notation; in the field of DE in particular, the following terms are commonly accepted:

- **Target vector**: a parent vector from the current generation.
- **Donor vector**: a mutant vector obtained through differential mutation.
- **Trial vector**: an offspring formed by recombining the donor with the target vector.

Initialisation of the Parameters Vectors

As DE searches for a global optimum point in a D-dimensional real parameter space $\mathbb{R}^D$, it starts with a randomly initiated population of NP D-dimensional parameter vectors. Each vector, also known as genome or chromosome, forms a candidate solution to the multidimensional optimization problem. Subsequent generations are denoted by $G = 0,1,\ldots,G_{max}$ and, since the paramters vectors are likely to change over different generations, the following notation is used to represent the $i-$th vector of the population at the current generation:

$$\mathbf{X}_{i,G} = \left[x_{1,i,G}, x_{2,i,G}, \ldots, x_{D,i,G}\right]$$

For each parameter of the problem, there may exists limitations to its magnitude, often related to physical components that have real-life constraints. For instance, if a parameters expresses a length, it can not be negative. The initial population - $G = 0$ - should then cover this range as evenly as possible. It can be achieved by uniformly randomizing individuals within the search space constrained by the prescibed minimum and maximum bound: $\mathbf{X}_{min} = \left[x_{1,min}, x_{2,min}, \ldots, x_{D,min}\right]$ and $\mathbf{X}_{max} = \left[x_{1,max}, x_{2,max}, \ldots, x_{D,max}\right]$. Consequently, the j-th component of the i-th vector can be written as:

$$x_{j,i,0} = x_{j,min} + rand_{i,j}[0,1] \cdot (x_{j,max} - x_{j,min})$$

where $rand_{i,j}[0,1]$ is a randomly generated number between 0 and 1, independently evaluated for each component of the i-th vector.

Mutation with Difference Vectors

While in biological terms mutation denotes a sudden change in the gene characteristics of a chromosome, in the context of evolutionary algorithms, it is also regarded as a change or perturbation induced by a random element.

Considering one of the simplest forms of DE-mutation, to create the donor vector for each i-th target vector from the current population, three distinct parameter vectors - e.g. $\mathbf{X}_{r_1^i}$, $\mathbf{X}_{r_2^i}$ and $\mathbf{X}_{r_3^i}$ - are sampled randomly from the current population. The indices r_1^i, r_2^i, and r_3^i are mutually exclusive integers randomly chosen from the range $[1, NP]$. These indices are randomly generated for each mutant vector. After the three vectors are chosen, the difference between any two of them

is evaluated and scaled by a scalar quantity F. The scaled difference is then added to the unused third vector, thus obtaining the donor vector $\mathbf{V}_{i,G}$ (Fig. 1.10). In mathematical terms:

$$\mathbf{V}_{i,G} = \mathbf{X}_{r_1^i,G} + F \cdot (\mathbf{X}_{r_2^i,G} - \mathbf{X}_{r_3^i,G})$$

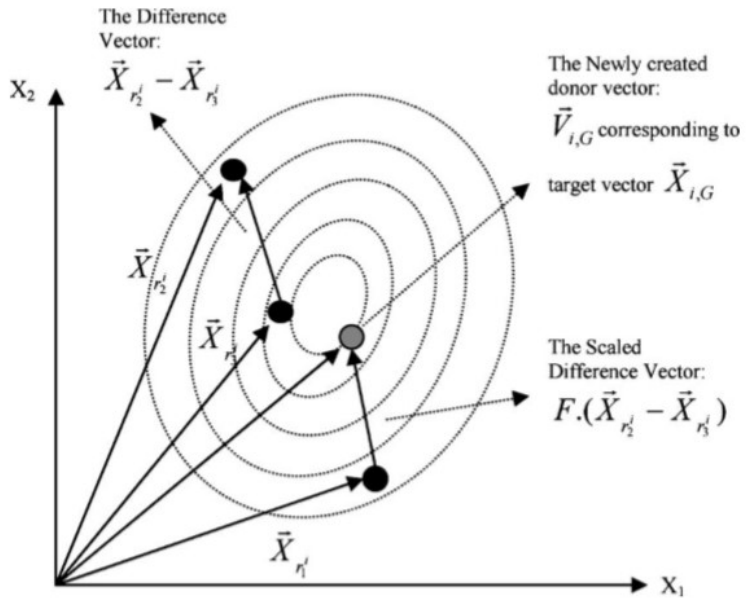

Figure 1.10. DE mutation scheme - Image from S. Das and P.N. Suganthan, *Differential Evolution: a Survey of the State-of-the-Art*, IEEE Transactions on Evolutionary Computation, Vol.15, N.1, February 2011

Crossover

After generating the donor vector via mutation, crossover takes care of enhancing the diversity of the population. Notably, the donor vector exchanges its components with the target vector $\mathbf{X}_{i,G}$ to form the trial vector $\mathbf{U}_{i,G=[u_{1,i,G},u_{2,i,G},...,u_{D,i,G}]}$. DE algorithms usually employ one of two kinds of crossover methods: exponential, or two-point modulo, and binomial, or uniform.

Exponential crossover requires an integer n to be chosen among the numbers $[1, D]$. It serves as a starting point in the target vector, from where the crossover starts. Likewise, the parameter L is chosen from the same range of values. L denotes the number of components the donor vector actually exchanges with the target vector. The trial vector is then obtained as:

$$u_{j,i,G} = v_{j,i,G} \text{ for } j = \langle n \rangle_D, \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D$$
$$x_{j,i,G} \text{ for all other} j \in [1, D]$$

where $\langle\rangle_D$ denotes a modulo function with modulus D. The integer L, on the other hand, is chosen from the range $[1, D]$ according to the following pseudo-code:

```
1 L = 0;
2 while (rand(0,1) ≤ Cr) AND (L ≤ D) do
3 |  L = L+1;
4 end
```

Cr is known as *crossover rate* and it is a control parameter of a DE algorithm, like the aforementioned F.

Binomial crossover is performed on each of the D variable whenever a randomly generated number between 0 and 1 is less than or equal to the crossover rate Cr. In this case, the number of parameters inherited from the donor vector has a nearly binomial distribution. In mathematical terms:

$$u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if } (rand_{i,j}[0,1] \leq Cr \text{ or } j = j_{rand}) \\ x_{j,i,G} & \text{otherwise} \end{cases}$$

where $rand_{i,j}[0,1]$ is, once again, an evenly distributed random number, which is evaluated for each $j-$th component of the $i-$th parameter vector. $j_{rand} \in [1,2,\ldots,D]$ is a randomly chosen index, which ensures that $\mathbf{U}_{i,G}$ gets at least one component from $\mathbf{V}_{i,G}$.

Selection

To keep the population size constant over subsequent generations, the next final step of the algorithm is given by a selection process which determines whether the trial vector is carried over to the next generation. The selection operation can be written as:

$$\mathbf{X}_{i,G+1} = \mathbf{U}_{i,G} \text{ if } f(\mathbf{U}_{i,G}) \leq f(\mathbf{X}_{i,G})$$
$$= \mathbf{X}_{i,G} \text{ if } f(\mathbf{U}_{i,G}) > f(\mathbf{X}_{i,G})$$

where $f(\mathbf{X})$ is the objective function to be minimised. In other words, if the new trial vector yields an equal or lower value of the objective function, it replaces the corresponding target vector in the next generation. In the other case, the target is kept in the population. Hence, the population either improves with respect to the objective function, or it remains the same, but it never worsens.

After selection, and with the new target vectors defined, the process is iterated until any of the following cases is satisfied:

- The number of generation reaches the predefined maximum number of iterations G_{max}
- The best fitting element of the population does not significantly improve its *fitness* over successive iterations. In other words, two subsequent generations produce a realtive difference in the objective function which is smaller than a given threshold.
- The objective function reaches a predefined target value.

The overall DE process is summarised in Fig. 1.11

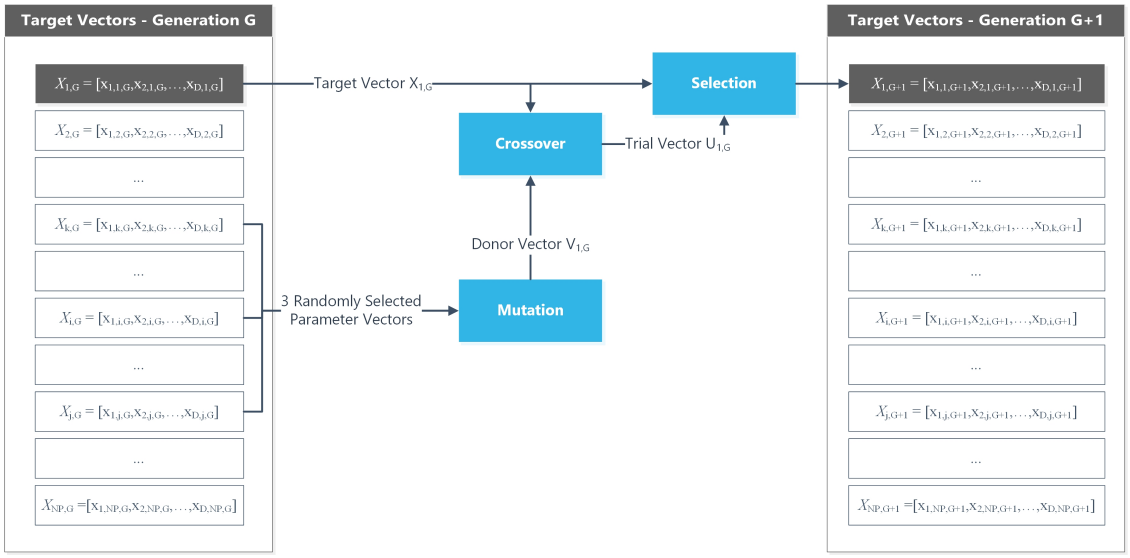

Figure 1.11. Example of DE applied to the first parameter vector of the first generation of a D-dimensional optimisation problem. The depicted process is repeated for each element of the initial population in order to obtain the second generation.

1.5.2 Comparison Between DE and Contemporary EAs

Once the functioning scheme of DE is clear, it is interesting to see how it compares to other contemporary EAs for real parameter optimisation. Structuring the comparison like the algorithm functioning scheme:

- **Mutation:** DE significantly differs from algorithms like ES and EP for the fact that it mutates the base vector with scaled population-derived difference vectors. As the generations pass, these differences tend to adapt to the natural scaling of the problem. For example, if the population becomes compact in one variable but remains widely dispersed in another, the difference vectors

sampled will be small in the former variable and large in the latter. Such automatic adaption significantly improves the convergence of the algorithm [56].

- **Crossover:** both DE and ES employ crossover to create a single trial vector, while most GAs recombine two vectors to produce two trial vectors.
- **Selection:** Unlike GAs that select parents based on their fitness, both ES and DE treat all individuals equally. In other words, each has an equal chance of being selected as a parent. In ES, each individual has the same chance of being selected when it comes to mutation, while DE randomly picks the base vectors, too. During the selection process, when an algorithm keeps the best-so-far solution, it is possible to talk about *elitism*, a characteristic that plays a major role in the convergence to a global optimum within some given computational time constraints. DE has a survivor scheme which differs from any other, because instead of ranking the combined population and choosing the first best fitting trial vectors, it employes a one-to-one competition scheme, where each parent vector only competes against its offspring. Parent-offspring competition has a superior ability to maintain population diversity when compared with ranking or tournament selection, where elites and their offspring may dominate the population rapidly.

1.5.3 Self-Adapting DE Algorithms

Recently, the field of DE witnessed some major breakthroughs that allowed it to emerge among other optimisation strategies. One of them was brought by the introduction of *self-adapting* algorithms. In the first paragraphs of this section, it was remarked how simple it is to set-up DE due to the presence of only three control parameters, which are kept fixed throughout the entire evolutionary process. Finding their optimal values, however, is not an easy task. Moreover, as evolution proceeds, the population of DE may move through different regions in the search space, where certain strategies associated with a specific parameter setting may be more effective than others. To overcome such inconvenience, researchers have actively investigated the adaptation of parameters and operators used in DE. The two most relevant solutions for this matter are the SaDE algorithm [57] and jDE [58]. The latter will be used throughout this text due to its good convergence performances [57] and due to the associated extensive work already conducted by the hosting University, the Shanghai Jiao Tong University.

1.6 Trajectory Evaluation Metrics

The last section introduced the concept of the objective function as a mean to measure the performance of a system and to assess the result of an optimisation

process. It is now necessary to illustrate its most fundamental constituents to understand its structure.

Since this research focuses on trajectory tracking problems, it follows that the objective function applied throughout this text will be purely based on the geometrical and physical properties of a trajectory. Each of its constituing metrics is the result of a comparison between any two trajectories, which most of the time are an ideal trajectory - the one the rover should follow - and a real trajectory - the one actually traversed. The proposed metrics are also a valid tool to compare the performance of different trajectory tracking controllers. Thanks to the work of [59], the employed metrics referred to two sample trajectories $traj_A$ and $traj_B$ are:

- **Distance ratio:** the distance ratio d_{ratio} is defined as the ratio between the length of two different trajectory:

$$d_{ratio} = \frac{d_B}{d_A}$$

- **Hausdorff distance:** the Hausdorff distance d_{haus} is a metric that measures the maximum of all the distances from a point in one set to the closest point in the other set [60]. In this case a set is a trajectory. It is often employed in image matching and handwriting recognition applications. In mathematical terms:

$$d_{haus}(A, B) = \max\{\min d(a, b)\}\,,\ a \in A\,, b \in B$$

  where d in the Euclidean distance, a are all the point in the subset A, and b are all the points in the subset B. A graphic example is provided in Fig. 1.12.

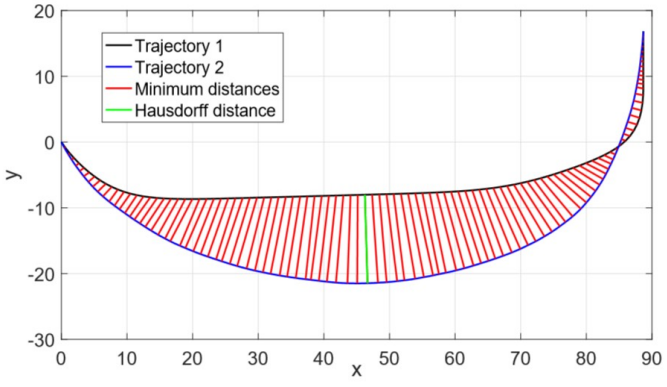

Figure 1.12. Minimum - or Euclidean - and Hausdorff distances between two trajectories - Image from Ermacora *et Al.*, *An Evaluation Framework for the Deployment of Mobile Robots Performing Real-World Indoor Autonomous Navigation.*

- **RMS of the minimum distances:** the Root Mean Square (RMS) of the minimum distances d_{rms} is a metric that evaluates the overall separation of two trajectories. It differs from the Hausdorff distance for it does not consider the maximum separation but the root mean square of all the available measurements, that is:

$$d_{rms} = \mathrm{RMS}\{\min d(a,b)\}\,,\ a \in A\,, b \in B$$

  where d in the Euclidean distance, a are all the point in the trajectory A, and b are all the points in the trajectory B.

- **Total bending energy:** the total bending energy e_{bend} measures the efficiency of a trajectory, in terms od how much "curvy" it is. If c_i is the local curvature along the descrete trajectory of n elements, the total bending energy is defined as [61]:

$$e_{bend} = \sum_{i=0}^{n} c_i^2 d_{T_i}$$

- **Total bending energy ratio:** the total bending energy radio $e_{bend-ratio}$ derives from the metric above, and for the two sample trajectories $traj_A$ and $traj_B$ it is:

$$e_{bend-ratio} = \frac{e_{bend_B}}{e_{bend_A}}$$

Chapter 2

Mathematical Model

The mathematical model on a SSMR is defined by three sets of differential equations that govern its dynamics. They describe the forces and moment acting on the robot and its consequential orientation with respect to an inertial reference frame. In this chapter, a 2D nonlinear model is presented along with its linearization and state-space representation. As any SSMR model, the one here presented is decoupled into an independent kinematic model and a dynamic model. The nonholonomic[1] constraint of the vehicle is accounted to obtain the kinematic model. It is derived using the chassis geometrical properties and a transformation between the coordinate systems. The dynamical properties are derived using the Lagrange-Euler equations. As already mentioned in the previous chapter, one of the main goals of the model is to represent the friction forces acting on the robot wheels faithfully. In this regard, the model is simplified to better fit the test vehicle used for experimental tests - the Clearpath Husky A200 - and an alternative friction model is presented.

2.1 Mathematical Premises

2.1.1 Reference Frames

A reference frame is a set of axes employed as a coordinate system to represent the position and orientation of a dynamic system; in this case, the robot. In mobile robotics, in particular, it is common to use two sets of reference frames: the local - or body - reference frame $f(x, y, z)$ and the global - or fixed - reference frame

[1] A nonholonomic system is a system whose state depends on the path taken in order to achieve it. In other words, it is a system in which there is a continuous closed circuit of the governing parameters, by which the system may be transformed from any given state to any other state. Such a system is described by a set of parameters subject to differential constraints [62].

$F(X, Y, Z)$.

Local Reference Frame

The local reference frame $f(x, y, z)$ is a non-inertial Cartesian reference frame originating from the robot Center of Gravity (CoG) (Fig. 2.1). Its axes are defined as follows: the x axis lies on the robot plane of symmetry, and it is directed towards its nose; the y axis is parallel to the ground and points left; the z axis points upward to complete the right-handed coordinate system. The robot moments of inertia calculated in this reference frame do not change during its motion.

Global Reference Frame

The global reference frame $F(X, Y, Z)$ is an inertial Cartesian reference frame originating from a fixed point in space (Fig. 2.1). Its axes are defined as follows: the X and Y axes lie on a plane parallel to the ground, and the Z axis points upward to complete the right-handed coordinate system. The robot moments of inertia calculated in this reference frame do change during its motion.

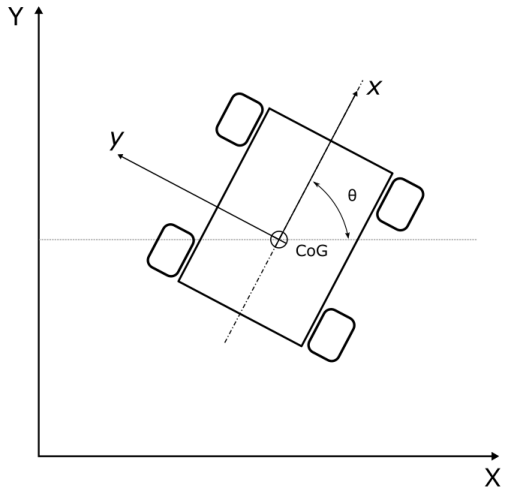

Figure 2.1. Local and Global reference frames

2.1.2 Euler Angles

The Euler angles are three angles - Φ, Θ, Ψ - used to represent the orientation of a rigid body, and consequently its mobile frame of reference, with respect to a fixed coordinate system. The composition of rotations expressed by this set of angles is

always sufficient to reach any target frame. Considering for instance two coordinate systems $F_1(X_1, Y_1, Z_1)$ and $F_2(X_2, Y_2, Z_2)$ in Fig. 2.2, the sequential rotations of magnitude Φ_1, Θ_1, Ψ_1 will align F_2 to F_1.

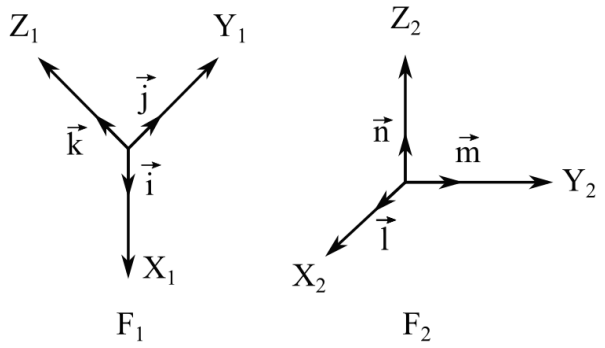

Figure 2.2. Generic Coordinate System Rotation

Similarly, Euler angles can also be used to transform the components of a generic vector between two reference frames by means of simple rotations. Assuming that $[F_{X_1}, F_{Y_1}, F_{Z_1}]^T$ is a generic vector in the F_1 coordinate system , the mathematical relationship with the corresponding vector $[F_{X_2}, F_{Y_2}, F_{Z_2}]^T$ in the F_2 coordinate system is:

$$\begin{bmatrix} F_{X_2} \\ F_{Y_2} \\ F_{Z_2} \end{bmatrix} = \boldsymbol{\Phi\Theta\Psi} \begin{bmatrix} F_{X_1} \\ F_{Y_1} \\ F_{Z_1} \end{bmatrix} = \boldsymbol{R_{21}} \begin{bmatrix} F_{X_1} \\ F_{Y_1} \\ F_{Z_1} \end{bmatrix}$$

where $\mathbf{R_{21}}$ is the complete transformation matrix and $\boldsymbol{\Phi}$,$\boldsymbol{\Theta}$, $\boldsymbol{\Psi}$ are the elementary rotation matrices:

$$\boldsymbol{\Phi} = \begin{bmatrix} cos\Phi & -sin\Phi & 0 \\ sin\Phi & cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{\Theta} = \begin{bmatrix} cos\Theta & 0 & sin\Theta \\ 0 & 1 & 0 \\ -sin\Theta & 0 & cos\Theta \end{bmatrix}, \boldsymbol{\Psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\Psi & -sin\Psi \\ 0 & sin\Psi & cos\Psi \end{bmatrix}$$

Since the elementary matrices are orthogonal, also $\mathbf{R_{21}}$ is orthgonal, so $\mathbf{R_{21}}^{-1} = \mathbf{R_{21}} = \mathbf{R_{21}}^T$. This allows to define the inverse transformation as:

$$\begin{bmatrix} F_{X_1} \\ F_{Y_1} \\ F_{Z_1} \end{bmatrix} = \boldsymbol{R}_{\mathbf{21}}^T \begin{bmatrix} F_{X_2} \\ F_{Y_2} \\ F_{Z_2} \end{bmatrix}$$

Global-Local Transformation

Based on the theoretical aspects highlighted above, it is possible to project the local reference frame $f(x, y, z)$ onto the global reference frame $F(X, Y, Z)$ by means of a single rotation of magnitude θ about the z axis. In terms of Euler angles:

$$\Phi = \theta$$

thus the rotation matrix tying the local and the global reference frame is:

$$\boldsymbol{R_{lg}} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

2.2 Nonlinear Mathematical Model

The following section will analyse the generalised nonlinear mathematical model of a four-wheel SSMR, building on the premises exposed in Chap. 1.2. Further simplifying assumptions are introduced in the last part of the chapter. As already mentioned, the model is decoupled into kinematics and dynamics. According to the conclusions expressed in Section 1.2, the following assumptions are considered without loss of generality:

1. The CoG of the robot is fixed and lies on its longitudinal plane of symmetry xz.
2. The vehicle is rigid and moves on a horizontal plane.
3. The robot four wheels are always in contact with the ground surface.
4. There is only one point of contact between the wheels and the ground.
5. The maximum vehicle speed is below 1 m/s.
6. The longitudinal wheel skidding is neglected.
7. The tire lateral force is a function of its vertical load.

2.2.1 Kinematic Model

Allowing that the SSMR is constrained to planar motion, it is possible to define the state vector describing the generalised coordinated of the robot[2] and the body velocity vector, namely $\boldsymbol{q} = [X, Y, \theta]$ and $\boldsymbol{V} = [v_x, v_y, \omega_z]^T$, with v_x, v_y and ω_z the

[2]I.e. the CoG position, X and Y, and the orientation θ with respsect to the inertial frame.

longitudinal, lateral and angular velocity, respectively (Fig.2.3). From the local velocity vector, it is possible to derive the global velocity vector[3] $\dot{\boldsymbol{q}} = [\dot{X}, \dot{Y}, \dot{\theta}]^T$ by means of a simple rotation, obtained with the rotation matrix introduced in Eq. 2.1:

$$\dot{\boldsymbol{q}} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{V} = \boldsymbol{R_{lg}} \boldsymbol{V} \tag{2.2}$$

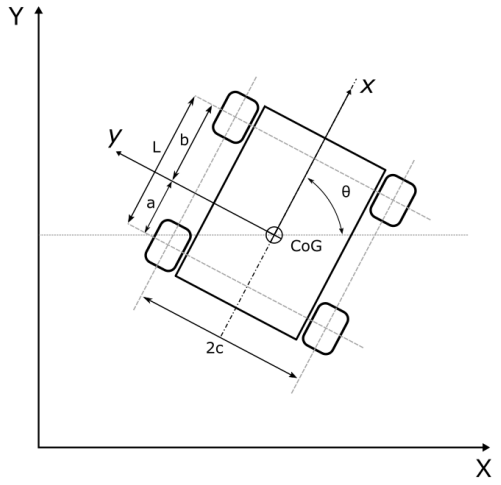

Figure 2.3. Free Body Kinematics

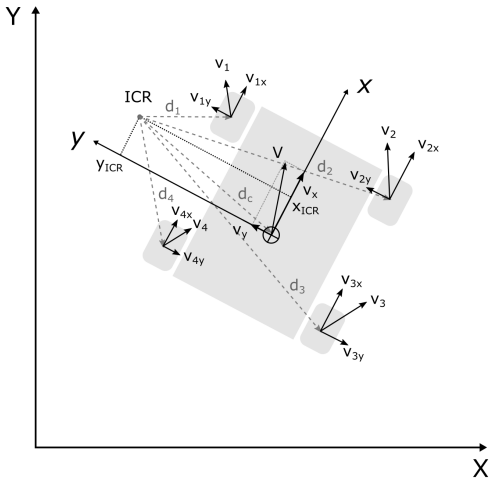

Figure 2.4. Wheel velocities and ICR

Eq. 2.3 does not impose any restriction on the robot movement since it describes free-body kinematics only. Hence, it is necessary to analyse the relationship between the speed of the wheels and the overall local velocity of the robot, neglecting, for the sake of simplicity, the thickness of the wheels, assuming they are in contact with the ground at points P_i (Fig.2.5), and assuming that they are rotating with angular velocity $\omega_i(t)$, where $i = 1, \ldots, 4$ represents the i-th wheel. Since skid-steering robots generally move at non-zero lateral velocity v_{iy}, wheels are tangent to the path only when $\omega = 0$, i.e., when the robot moves on a straight line.

As we neglect any longitudinal slip between the wheels and the surface, according to this assumption based on the work of [63], the following relation can be developed:

$$v_{ix} = r_i \omega_i$$

[3]Assumin planar motion, the relationship $\dot{\theta} = \omega_z$ is verified.

where v_{ix} is the longitudinal component of the total velocity v_i of the i-th wheel expressed in the local frame f and r_i denotes the so-called effective rolling radius of the wheel[4].

Considering the scheme in Fig. 2.4, it is possible to express the following relationship between the position of the ICR, that lies within the range $(-a; b)$, and the local velocity vector components with respect to the local frame:

$$\frac{||\boldsymbol{v_i}||}{||\boldsymbol{d_i}||} = \frac{||\boldsymbol{V}||}{||\boldsymbol{d_c}||} = |\omega_z| \tag{2.3}$$

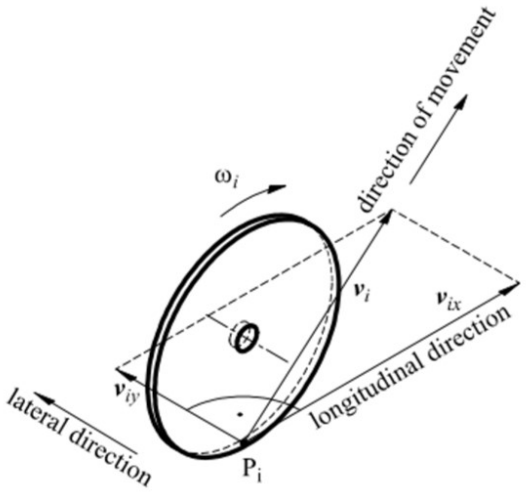

Figure 2.5. Velocity components of a wheel - Image from Kozłowski and Pazderski, *Modeling and Control of a 4-wheel skid-steering mobile robot*, Int. J. Appl. Math Comput. Sci., 2004, 477-496

where $||\cdot||$ represents the Euclidean norm. Defining the ICR coordinates in the local frame as

$$ICR(x_{ICR}, y_{ICR}) = (-d_{xc}, -d_{yc})$$

it is possible to rewrite Eq. 2.3 as

$$\frac{v_x}{y_{ICR}} = -\frac{v_y}{x_{ICR}} = \omega_z \tag{2.4}$$

Furthermore, the coordinates between the ICR and the i-th wheel satisfy the following relationships:

[4]It corresponds to the ratio between the linear velocity of the wheel centre and its angular velocity

$$\begin{aligned} d_{1y} &= d_{4y} = d_{cy} + c \\ d_{2y} &= d_{3y} = d_{cy} - c \\ d_{1x} &= d_{2x} = d_{cx} + b \\ d_{3x} &= d_{4x} = d_{cx} - a \end{aligned} \tag{2.5}$$

The relationship among wheel velocities can be obtained from Eqs. 2.3 and 2.5:

$$\begin{aligned} v_L &= v_{1x} = v_{4x} \\ v_R &= v_{2x} = v_{3x} \\ v_F &= v_{1y} = v_{2y} \\ v_B &= v_{3x} = v_{4y} \end{aligned} \tag{2.6}$$

where:

- v_L and v_R represent the longitudinal velocities of the left and right wheels
- v_F and v_B represent the lateral velocities of the front and rear wheels.

Combining Eqs. 2.3 - 2.6, it is possible to obtain the relationship between the wheel and the robot velocities:

$$\begin{bmatrix} v_L \\ v_R \\ v_F \\ v_B \end{bmatrix} = \begin{bmatrix} 1 & -c \\ 1 & +c \\ 0 & -x_{ICR} + b \\ 0 & -x_{ICR} - a \end{bmatrix} \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} \tag{2.7}$$

Moreover, considering for each wheel the same effective radius $r_i = r$, from Eqs. 2.2.1 and 2.6, it is possible to write:

$$\boldsymbol{\omega_w} = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \frac{1}{r} \begin{bmatrix} v_L \\ v_R \end{bmatrix} \tag{2.8}$$

where ω_L and ω_R are the left and right wheels velocity, respectively. Finally, it is possible to derive a relationship between the angular velocity of the wheels and the robot velocity vector; using Eqs. 2.7 and 2.8 it is possible to introduce a new control input at kinematic level $\boldsymbol{\eta} \in \mathbb{R}^2$:

$$\boldsymbol{\eta} = \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} = r \begin{bmatrix} \frac{\omega_l + \omega_r}{2} \\ \frac{-\omega_l + \omega_r}{2c} \end{bmatrix} \tag{2.9}$$

In order to complete the kinematic model of the SSMR, a velocity constraint must be introduced:

$$v_y + x_{ICR}\omega_z = 0 \tag{2.10}$$

This constraint was first intrduced by [22] and it is a nonintegrable equation that imposes the null motion along the local y direction. In other words it represents a non-holonomic constraint which can be rewritten in the Pfaffian form. Introducing the constraint vector $\boldsymbol{A} \in \mathbb{R}^{3\times 2}$ we can write:

$$\begin{bmatrix} -sin\theta & cos\theta & x_{ICR} \end{bmatrix} \begin{bmatrix} \dot{X} & \dot{Y} & \dot{\theta} \end{bmatrix}^T = \mathbf{A}(\mathbf{q})\dot{\mathbf{q}} = 0$$

Since the generalised velocity vector $\dot{\mathbf{q}}$ is always in the null space of $\mathbf{A}$, introducing a general coordinate transformation matrix $\mathbf{S} \in \mathbb{R}^{3\times 2}$ we can write:

$$\dot{\boldsymbol{q}} = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{\eta} \tag{2.11}$$

where

$$\boldsymbol{S^T}(\boldsymbol{q})\boldsymbol{A^T}(\boldsymbol{q}) = \mathbf{0} \tag{2.12}$$

and

$$\boldsymbol{S}(\boldsymbol{q}) = \begin{bmatrix} cos\theta & x_{ICR}sin\theta \\ sin(\theta) & -x_{ICR}cos\theta \\ 0 & 1 \end{bmatrix} \tag{2.13}$$

Since $dim(\boldsymbol{\eta}) = 2 < dim(\boldsymbol{q}) = 3$, eq. 2.11 describes the kinematics of an underactuated robot.

The adoption of the angular velocity ω as a control input instead of v_y as initially proposed by [22], happens to be more convenient as it does not require the knowledge of the y_{ICR} variable, which can not be evaluated but experimentally.

2.2.2 Dynamic Model

As highlighted by [22], the dynamic properties of a SSMR play a significant role in its motion. When on curved paths, due to the interactions between its wheels and the ground, reactive forces are much greater than inertial forces. When compared to a vehicle with pure rolling assumptions, a SSMR is thus much more affected by dynamic effects. In this chapter, it is therefore presented a dynamic model useful for control and simulation purposes.

The dynamic model of a common SSMR can be derived using the Euler-Lagrage principle with Lagrange multipliers to take the non-holonomic constraint into account. The Euler-Lagrange equation is:

$$\boldsymbol{\Gamma} = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}} - \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}$$

where Γ denotes the active torques and forces vector without considering any external force. $\mathcal{L} = E - U$ is the Lagrangian of the system, which is defined as the sum of its kinetic and potential energy. Since the robot considered is constrained to

planar motion, the total potential energy U is equal to zero; hence the Lagrangian is equal to the kinetic energy of the robot:

$$\mathcal{L}(\dot{\mathbf{q}}, \mathbf{q}) = E(\dot{q}, q)$$

Neglecting the kinetic energy of the wheels for sake of simplicity, the following equation describes the kinetic energy of the vehicle:

$$E = \frac{1}{2} m \boldsymbol{v}^{\boldsymbol{T}} \boldsymbol{v} + \frac{1}{2} I \omega_z^2 \tag{2.14}$$

where I is the moment of inertia of the robot about its CoG. Since

$$\boldsymbol{v}^{\boldsymbol{T}} \boldsymbol{v} = v_x^2 + v_y^2 = \dot{X}^2 + \dot{Y}^2$$

equation 2.14 can be rewritten as

$$E = \frac{1}{2} m (\dot{X}^2 + \dot{Y}^2) + \frac{1}{2} I \dot{\theta}^2$$

The inertial forces can be obtained from the time derivative of the partial derivative of the kinetic energy:

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial E}{\partial \ddot{\mathbf{q}}} = \begin{bmatrix} m\ddot{X} \\ m\ddot{Y} \\ I\ddot{\theta} \end{bmatrix} = \boldsymbol{M} \ddot{\boldsymbol{q}} \tag{2.15}$$

where $\mathbf{M} \in \mathbb{R}^{3\times3}$ denotes the constant, diagonal, positive definite inertia matrix:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix}$$

Before deriving the resistive forces, it is necessary to introduce the wheel-ground interaction, with reference to Fig. 2.6. The active and reactive forces, respectively F_{xi} and N_i are functions of the wheel torque and the gravity load; in addition F_{xi} is linearly depended on the wheel control input τ_i:

$$F_{xi} = \frac{\tau_i}{r} \tag{2.16}$$

According to [63], it is assumed that the vertical load N_i acts from the surface of the wheels. Neglecting additional dynamic properties like the kinematic energy of the wheels, refering to Fig. 2.7 it is possible to obtain the following equilibrium equations:

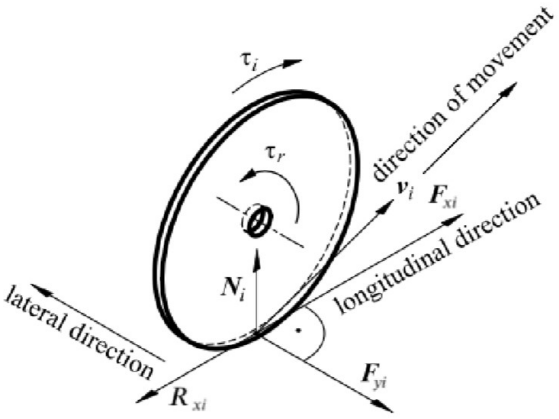

Figure 2.6. Forces acting on a wheel - Image from Kozłowski and Pazderski, *Modeling and Control of a 4-wheel skid-steering mobile robot*, Int. J. Appl. Math Comput. Sci., 2004, 477-496

$$\begin{aligned} N_4 a &= N_1 b \\ N_2 b &= N_3 a \\ \sum_{i=1}^{4} N_i &= mg \end{aligned}$$

where m and g are, respectively, the vehicle mass and the gravitational acceleration. If the SSMR has longitudinal symmetry, then

$$\begin{aligned} N_1 = N_2 &= \frac{a}{2(a+b)} mg \\ N_3 = N_4 &= \frac{b}{2(a+b)} mg \end{aligned} \tag{2.17}$$

Assuming that R_{xi} derives from the rolling resistant moment τ_{ri}, it can be regarded as pure rolling resistance. F_{yi}, on the other hand, is the lateral reactive force, which, based on [22], can be regarded as a friction force. Due to the fact that friction is a highly nonlinear phenomenon which depends on a wide range of variables, modelling its behaviour is considered an extremely daunting task. Therefore, friction F_f is often simplified as a superposition of Coulomb and viscous friction:

$$F_f(v) = \mu_c N \, sgn(v) + \mu_v v$$

where N is the force perpendicular to the surface, and μ_c and μ_v are the Coulomb and viscous coefficients, respectively. Since the mobile robot maximum velocity v is

relatively low, especially during lateral skidding, it is possible to neglect the viscous term $\mu_v v$ due to the relation $\mu_c N >> \mu_v v$, thus simplifying the model.

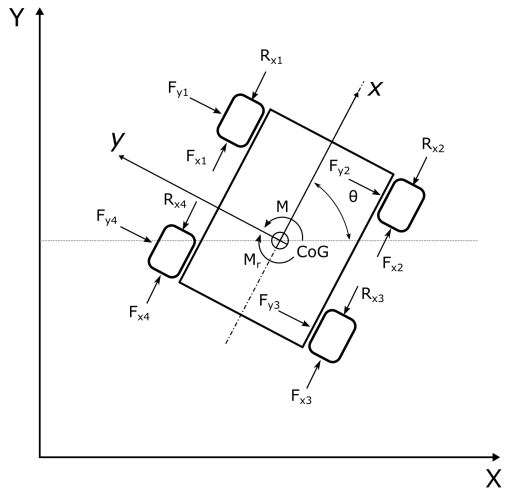

Figure 2.7. Active and resistive forces acting on the vehicle

Based on the previous considerations, the longitudinal rolling resistance and the lateral friction force can be rewritten as:

$$\begin{aligned} R_{xi} &= u_x mg\, sgn(v_{xi}) \\ F_{yi} &= \mu_c mg\, sgn(v_{yi}) \end{aligned} \tag{2.18}$$

where u_x is the longitudinal rolling coefficient. It is now possible to write the resulting rolling resistance R_x, the resulting friction force F_y and the resistive moment M_r around the CoG with respect to the local reference frame $f(x, y, z)$:

$$\begin{aligned} R_x(\dot{\mathbf{q}}) &= \sum_{i=1}^{4} R_{xi}(v_{xi}) \\ F_y(\dot{\mathbf{q}}) &= \sum_{i=1}^{4} F_{yi}(v_{yi}) \\ M_r(\dot{\mathbf{q}}) &= b \sum_{i=1,2} F_{yi}(v_{yi}) - a \sum_{i=3,4} F_{yi}(v_y i) + c\left(\sum_{i=2,3} R_{xi}(v_{xi}) - \sum_{i=1,4} R_{xi}(v_{xi}) \right) \end{aligned} \tag{2.19}$$

The resistive forces can then be merged in the resultant resistive forces vector $\mathbf{R} \in \mathbb{R}^3$:

$$\mathbf{R}(\dot{\mathbf{q}}) = \begin{bmatrix} R_x(\dot{\mathbf{q}})cos\theta - F_y(\dot{\mathbf{q}})sin\theta \\ R_x(\dot{\mathbf{q}})sin\theta + F_y(\dot{\mathbf{q}})cos\theta \\ M_r(\dot{\mathbf{q}}) \end{bmatrix} \tag{2.20}$$

Moving to the forces generated by the robot actuators, it is possible to write their resultants as active force F_x and active torque M:

$$\begin{aligned} F_x &= \sum_{i=1}^{4} F_{xi} \\ M &= c\left(\sum_{i=2,3} F_{xi} - \sum_{i=1,4} F_{xi} \right) \end{aligned} \tag{2.21}$$

which can be rewritten in the global reference frame as the vector of the active forces $\mathbf{F} \in \mathbb{R}^3$

$$\mathbf{F} = \begin{bmatrix} F_x cos\theta \\ F_x sin\theta \\ M \end{bmatrix} \tag{2.22}$$

Introducing a new control input vecotr $\tau \in \mathbb{R}^2$, which now depends on the active torques:

$$\tau = \begin{bmatrix} \tau_L \\ \tau_R \end{bmatrix} = \begin{bmatrix} \tau_1 + \tau_4 \\ \tau_2 + \tau_3 \end{bmatrix} \tag{2.23}$$

and combining eq. 2.23 with Eqs. 2.16, 2.21 and 2.22, it is possible to write the active forces vector as:

$$\mathbf{F} = \mathbf{E}(\mathbf{q})\tau \tag{2.24}$$

where $\mathbf{E} \in \mathbb{R}^{3\times2}$ is the input transformation matrix defined as

$$\mathbf{E}(\mathbf{q}) = \frac{1}{r} \begin{bmatrix} cos\theta & cos\theta \\ sin\theta & sin\theta \\ -c & +c \end{bmatrix}$$

A dynamic model can be obtained from Eqs. 2.15, 2.20 and 2.24:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{R}(\dot{\mathbf{q}}) = \mathbf{E}(\mathbf{q})\boldsymbol{\tau}$$

The system has yet to include the kinematic constraint introduced in the previous section. It is thus necessary to impose a Lagrange multiplier $\boldsymbol{\lambda}$ [22], obtaining:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{R}(\dot{\mathbf{q}}) = \mathbf{E}(\mathbf{q})\boldsymbol{\tau} + \mathbf{A}^{\boldsymbol{T}}(\mathbf{q})\boldsymbol{\lambda} \tag{2.25}$$

To eliminate the unknown Langrange multiplier, it is possible to left multiply 2.25 by the matrix $\mathbf{S^T(q)}$ defined in eq. 2.13 and use the kinematic constraint in eq. 2.12 to rewrite the dyanmic model in generalised coordinates $\mathbf{q}$:

$$\mathbf{S^T(q)M(q)\ddot{q} + S^T(q)R(\dot{q}) = S^T(q)E(q)\tau} \tag{2.26}$$

For control purposes, it is more convenient to express 2.26 in terms of the local velocity vector $\boldsymbol{\eta}$. After taking the time derivative of 2.11, it is possible to write:

$$\mathbf{\ddot{q} = \dot{S}(q)\eta + S(q)\dot{\eta}}$$

and replacing it in 2.26, the dynamic equations finally become:

$$\mathbf{M\dot{\eta} + C\eta + R = E\tau} \tag{2.27}$$

where:

$$\begin{aligned} \boldsymbol{M} &= \boldsymbol{S^T(q)M(q)S(q)} \\ \boldsymbol{C} &= \boldsymbol{S^T(q)M(q)\dot{S}(q)} \\ \boldsymbol{R} &= \boldsymbol{S^T(q)R(\dot{q})} \\ \boldsymbol{E} &= \boldsymbol{S^T(q)E(q)} \end{aligned} \tag{2.28}$$

The dynamic model here obtained will be regarded, from now on, as the *nonlinear model with Coulomb friction.*

2.2.3 Further Simplifications

In order to lower the complexity of the mathematical model without significant impacts on its accuracy, some further assumptions were applied to the nonlinear description:

1. The SSMR has a total of two planes of symmetry: the xy plane and the xz plane. Its CoG consequently lies by their intersection. Such an assumption is valid, considering the geometry of the mobile robot introduced in the next chapter. It follows that:
   - In terms of geometry it is possible to write that $a = b$ (Fig.2.3).
   - Considering Eq. 2.17, it is possible to write $N_1 = N_2 = N_3 = N_4$.
2. The position of the longitudinal coordinate of the ICR - x_{ICR} - is known. In particular $x_{ICR} = 0$, which means it lies on the CoG. Such hypothesis is reasonable if applied contextually to the previous one and the assumptions introduced in Sec. 2.2. It implies that:

- Eq. 2.10 becomes $v_y = 0$, i.e. the vehicle has zero lateral acceleration and zero net force along its local y axis.

3. The resistive torque generated by the rolling resistance in Eq.2.19 is negligible.
4. The wheels on each side of the robot move at the same speed. Such assumption is particularly valid for the SSMR used during the experimental tests, as it has a mechanical link bewtween the wheels lying on each of its sides. Therefore:
   - $\tau_1 = \tau_4 = \tau_L$ and $\tau_2 = \tau_3 = \tau_R$, hence, referring to Eq. 2.16, $F_{x1} = F_{x4}$ and $F_{x2} = F_{x3}$.
   - It is possible to introduce two new variabiles: $F_{xL} = F_{x1} + F_{x4}$ and $F_{xR} = F_{x2} + F_{x3}$.

Another significant effect of these assumptions is that it is no longer required to compute the forces acting on each wheel, but it is possible to evaluate them two at a time. The simplifications here listed lead to a more straightforward and computationally lighter formulation of the nonlinear model. Later in this text, it is compared to the initial model with outstanding results. The quality of the simplified formulation led to it being used as the starting point for the linearization process.

2.2.4 Improved Friction Model

Due to the nature of the movement of a generic SSMR, it was decided to implement an innovative dry friction model which compared to the classical static models, like Karnopp or Quinn, significantly improves both the accuracy and the stability of any numerical simulation. Besides, faithful modelling is useful to avoid steady-state errors, limit cycles and unsafe behaviours of a control system built upon it. Indeed, when friction is the only nonlinear phenomenon, its knowledge allows for system linearization by compensation methods [64].

The model employed, which is based on the work of [31], offers the following advantages:

- It estimates the sign of the resistive force, based on the sign of the body velocity.
- It distinguishes between any adherence and dynamic condition. It allows for two distinct friction forces to be used: a stiction, or static friction force FSJ, and a dynamic friction force FSD.
- It evaluates the stopping of a body previously in motion.
- It correctly estimates the breakaway condition a body which is initially in a standstill position.

- A body in a standstill position is correctly kept in such condition.
- It takes into account the presence of any end-stops, considering a completely inelastic collision.

Its mathematical formulation is:

$$FF = \begin{cases} -F_{act} & \text{if } v = 0 \wedge |F_{att}| \leq FSJ \\ -sgn(F_{act}) \cdot FSJ & \text{if } v = 0 \wedge |F_{att}| > FSJ \\ -FDJ & \text{if } v \neq 0 \end{cases} \tag{2.29}$$

where F_{act} is the active force to which friction opposes, while FSJ and FDJ are the static and dynamic friction forces, respectively. They are supposed to only depend on the weight of the robot, and can be written as:

$$\begin{aligned} FSJ &= \mu_s \cdot N \\ FDJ &= \mu_d \cdot N \end{aligned} \tag{2.30}$$

where μ_s is the static friction coefficient, μ_d is the dynamic friction coefficient, and N is the normal force acting between the two sliding bodies. In this case, it would be the wheel load as expressed in eq. 2.17.

In order to avoid the numerical instability of a Karnopp-like model, the simulation block which implements the friction model includes a zero crossing detection algorithm which sets the body velocity to zero whenever it senses a sign change in the linear or angular velocity of the body during two consecutive integration steps:

$$\begin{aligned} v(t_{i+1}) &= 0 \text{ if } v(t_{i+1})v(t_i) \leq 0 \\ \dot{\theta}(t_{i+1}) &= 0 \text{ if } \dot{\theta}(t_{i+1})\dot{\theta}(t_i) \leq 0 \end{aligned} \tag{2.31}$$

Whenever the imposed stop happens to be incorrect, the disequilibrium of the active and resistive forces - or torques - would cause the system to overcome the breakaway force and restore its motion within one timestep. Such control represents the major distinction between the more common friction models listed at the beginning of this section. A block diagram of the model is represented in Fig. 2.8

The model obtained combining the assumptions in Sec. 2.2.3 with the friction model here presented will be regarded, from now on, as the *nonlinear model with the improved friction formulation.*

2.2.5 Wheel Torque Controller

To fully simulate the behaviour of a SSMR, it is necessary to know the relationship between the kinematic control input - Eq. 2.9 - and the wheel torque control input

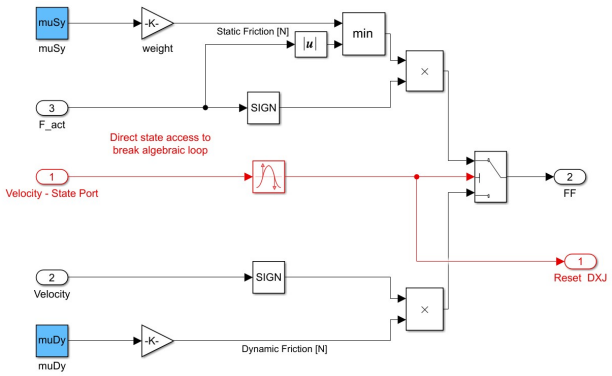

Figure 2.8. Block diagram of the improved dry friction model with a zero crossing detection control scheme

- Eq. 2.23. In fact, as later confirmed in Sec. 3.1, such type of mobile robot only accepts kinematic inputs which are then translated into wheel torque commands by its own torque control system loaded in an onboard Micro Controller Unit (MCU). For sake of simplificty, and lacking further information, the robot was supposed to possess a basic proportional controller which operates by the following contro law:

$$\begin{bmatrix} \tau_l \\ \tau_r \end{bmatrix} = K_{p_\tau} \cdot \mathbf{e} \tag{2.32}$$

where K_{p_τ} is the proportional gain of the control system and $\mathbf{e}$ is the error vector obtain comparing the actual wheel speeds to the commanded speeds, which are both obtained using Eq. 2.9:

$$\begin{bmatrix} e_l \\ e_r \end{bmatrix} = \begin{bmatrix} \omega_{l_{cmd}} - \omega_{l_{act}} \\ \omega_{r_{cmd}} - \omega_{r_{act}} \end{bmatrix}$$

where $\omega_{l/r_{cmd}}$ are the commanded wheel velocities and $\omega_{l/r_{act}}$ are the actual velocities.

2.3 Linear Mathematical Model

The nonlinear equations of motion have general validity within all operating conditions. They provide a powerful tool for studying the performance and the dynamic behaviour of a SSMR. In some occasions, however, the accuracy provided by the nonlinear model might not be necessary, and a more straightforward mathematical formulation would be preferable. Such simplification is achievable linearising the governing equations. For this purpose, the linearisation was carried out on the nonlinear model with the simplifying assumptions introduced in Sec. 2.2.3 but without the improved friction model presented in Sec. 2.2.4, as it is not suitable

for the process later presented. In the model chosen, friction forces are the only nonlinearities of the system. However, due to the presence of a signum function, which has a discontinuous first derivative, the Jacobian linearization is not possible about a specific operating point near zero. As a solution, a signum function can be approximated using an arctangent function as proposed by [23]:

$$sgn(x) = \frac{2}{\pi} arctan(k_s x) \tag{2.33}$$

where $k_s \gg 1$ is a constant which determines the approximation accuracy according to the relation

$$\lim_{k_s \to \infty} \frac{2}{\pi} arctan(k_s x) = sgn(x) \tag{2.34}$$

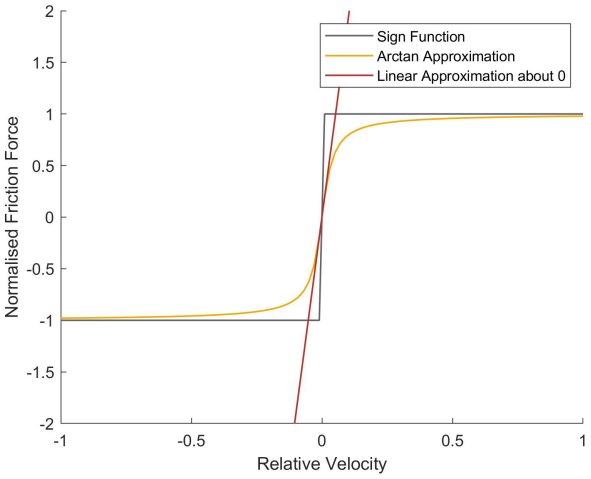

Figure 2.9. Hyper viscous friction model - Friction with *sign* function; friction with the *arctan* approximation and $k_s = 30$; linear appossimation about 0

Substituting 2.33 into 2.18, it is possible to write:

$$\begin{aligned} R_{xi} &= u_x N_i \frac{2}{\pi} arctan(k_{sx} v_{xi}) \\ F_{yi} &= \mu_c N_i \frac{2}{\pi} arctan(k_{sy} v_{yi}) \end{aligned} \tag{2.35}$$

and their Taylor expansion about an equilibrium point given by $\overline{v_{xi}}$ and $\overline{v_{yi}}$:

$$\begin{aligned} R_{xi} &\approx R_{xi}(\overline{v_{xi}}) + \frac{\partial R}{\partial v_{xi}}\bigg|_{v_{xi}=\overline{v_{xi}}} \delta v_{xi} + \text{higher order terms} \\ F_{yi} &\approx F_{yi}(\overline{v_{yi}}) + \frac{\partial F}{\partial v_{yi}}\bigg|_{v_{yi}=\overline{v_{yi}}} \delta v_{yi} + \text{higher order terms} \end{aligned} \tag{2.36}$$

knowing that the deviation variables are:

$$\begin{aligned} \delta v_{xi} &= v_{xi} - \overline{v_{xi}} \\ \delta v_{yi} &= v_{yi} - \overline{v_{yi}} \end{aligned} \tag{2.37}$$

Neglecting higher order terms and evaluating the partial derivatives:

$$\begin{aligned} R_{xi} &= R_{xi}(\overline{v_{xi}}) + u_x N_i \frac{2k_{sx}}{\pi(1 + (k_{sx}\overline{v_{xi}})^2)} \delta v_{xi} \\ F_{yi} &= F_{yi}(\overline{v_{yi}}) + \mu_c N_i \frac{2k_{sy}}{\pi(1 + (k_{sy}\overline{v_{yi}})^2)} \delta v_{yi} \end{aligned} \tag{2.38}$$

Choosing as equilibrium point $\overline{v_{xi}} = \overline{v_{yi}} = 0$, then $R_{xi}(\overline{v_{xi}}) = F_{yi}(\overline{v_{yi}}) = 0$ as there is no friction at zero velocity. Furthermore it is possible to rewrite the deviation variables as $\delta v_{xi} = v_{xi}$ and $\delta v_{yi} = v_{yi}$. Substituting into eq. 2.37 and rearranging the terms:

$$\begin{aligned} R_{xi} &= \frac{2}{\pi} k_{sx} u_x N_i v_{xi} \\ F_{yi} &= \frac{2}{\pi} k_{sy} \mu_c N_i v_{yi} \end{aligned} \tag{2.39}$$

Renaming $K_{ui} = \frac{2}{\pi} k_{sx} u_x$ and $K_{yi} = \frac{2}{\pi} k_{sy} \mu_c$, it is possible to rewrite eq. 2.39 in a simpler form:

$$\begin{aligned} R_{xi} &= N_i \cdot K_{ui} \cdot v_{xi} \\ F_{yi} &= N_i \cdot K_{yi} \cdot v_{yi} \end{aligned} \tag{2.40}$$

The mathematical formulation just obtained is analogous to what is generally known as hyperviscous friction model [31]. It presumes that, within the range of application, viscous effects dominate on any other form of resistance. The deriving resistive forces are thus linearly dependent on the vehicle speed. K_{ui} and K_{yi} are, therefore, two viscous friction coefficients, which refer to the robot forward and lateral motion, respectively. The graphical representation of such a model is depicted in Fig. 2.9.

2.3.1 State-Space Representation

Two approaches are available for the analysis and the design of a feedback control system. The first is known as classical, or frequency-domain technique, which is based on converting a system's differential equations to transfer functions, thus generating a mathematical model of the system that algebraically relates a representation of the output to a representation of the input. The second, more modern approach is known as the state-space approach, which provides the dynamics as a set of coupled first-order differential equations in a set of internal variables known as *state variables*, together with a set of algebraic equations that combine the state variables into physical output variables. It is a unified method for modelling, analysing, and designing a wide range of systems, including nonlinear, time-varying, Multiple-Input and Multiple-Output (MIMO) systems. The state-space approach was therefore chosen to represent the linear model of a SSMR with multiple inputs. It also proves to be necessary for the design of the LQR proposed later in this text.

Definition of System State

The concept of *state* of a dynamic system refers to a minimum set of variables, known as *state variables*, that fully describes the system and its response at any given set of inputs. In particular, a state-determined system model has the characteristic that, citing [65], "A mathematical description of the system in terms of a minimum set of variables $x_i(t)$, $i = 1, \ldots, n$, together with knowledge of those variables at the initial time t_0 and the system inputs for time $t \geq t_0$, are sufficient to predict the future system state and outputs for all time $t \geq t_0$".

The State Equations

In the standard form of state equations, the mathematical description of the system is expressed as a set of n coupled first-order ordinary differential equations, known as *state equations*, in which the time derivative of each state variable is expressed in terms of state variables $x_1(t), \ldots, x_n(t)$ and the system inputs $u_1(t), \ldots, u_n(t)$. In the general case the form of the n state equations is:

$$\begin{aligned} \dot{x}_1 &= f_1(\mathbf{x}, \mathbf{u}, t) \\ \dot{x}_2 &= f_2(\mathbf{x}, \mathbf{u}, t) \\ \vdots &= \vdots \\ \dot{x}_n &= f_n(\mathbf{x}, \mathbf{u}, t) \end{aligned}$$

where $\dot{x}_i = dx_i/dt$ and each of the functions $f_i(\mathbf{x}, \mathbf{u}, t)$, $(i = 1, \ldots, n)$ may be a general nonlinear, time varying function of the state variables, the system input, and time.

Limiting the attention to systems that are Linear and Time-Invariant (LTI), the previous equations become a set of n coupled first-order linear differential equations with constant coefficients. For a LTI system of order n and with r inputs:

$$\begin{aligned}
\dot{x}_1 &= a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + b_{11}u_1 + \ldots + b_{1r}u_r \\
\dot{x}_2 &= a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + b_{21}u_1 + \ldots + b_{2r}u_r \\
&\vdots \\
\dot{x}_n &= a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n + b_{n1}u_1 + \ldots + b_{nr}u_r
\end{aligned}$$

where a_{ij} and b_{ij} are constant that describe the system. The previous equation can be written compactly in a matrix form:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \ldots & b_{1r} \\ b_{21} & \ldots & b_{2r} \\ \vdots & & \vdots \\ b_{n1} & \ldots & b_{nr} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}$$

which can be summarised as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2.41}$$

where the state vector $\mathbf{x}$ is a vector of length n, the input vector $\mathbf{u}$ is a column vector of length r, $\mathbf{A}$ is $n \times n$ square matrix of the constant coefficients a_{ij}, and $\mathbf{B}$ is an $n \times r$ matrix of the coefficients b_{ij} that weight the inputs.

Output Equations

A system *output* is defined to be any system variable of interest. A description the system in terms of a set of state variables does not necessarily include all of the variables of direct engineering interest. An important property of the linear state equation description is that all system variables may be represented by a linear combination of the state variables x_i and the system inputs u_i. An arbitrary ouput variable in a system of order n with r inputs may be written:

$$y(t) = c_1x_1 + c_2x_2 + \ldots + c_nx_n + d_1u_1 + \ldots + d_ru_r$$

where the c_i and the d_i are constant. If a total of m system variables are defined as outputs, then:

$$\begin{aligned}
\dot{y}_1 &= c_{11}x_1 + c_{12}x_2 + \ldots + c_{1n}x_n + d_{11}u_1 + \ldots + d_{1r}u_r \\
\dot{y}_2 &= c_{21}x_1 + c_{22}x_2 + \ldots + c_{2n}x_n + d_{21}u_1 + \ldots + d_{2r}u_r \\
&\vdots \\
\dot{y}_n &= c_{m1}x_1 + c_{m2}x_2 + \ldots + x_{mn}x_n + d_{m1}u_1 + \ldots + d_{mr}u_r
\end{aligned}$$

or in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & & & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} d_{11} & \dots & d_{1r} \\ d_{21} & \dots & d_{2r} \\ \vdots & & \vdots \\ d_{m1} & \dots & d_{mr} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix}$$

which can be summarised as:

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

where $\mathbf{y}$ is a column vector of the output variables $y_i(t)$, $\mathbf{C}$ is an $m \times n$ matrix of the constant coefficients c_{ij} that weight the state variables, and $\mathbf{D}$ is an $m \times r$ matrix of the constant coeffient d_{ij}, that weight the system inputs. For many physical system, including the one here analysed, $\mathbf{D}$ is a null matrix, and the output equation reduces to a simple weighted combination of the state variables:

$$\mathbf{y} = \mathbf{C}\mathbf{x} \tag{2.42}$$

State Space Model for a SSMR

From Eqs. 2.41 and 2.42, it is cleat that for an LTI system, the goal is to to write a set of equations in the form:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} \end{aligned} \tag{2.43}$$

In order to do so it is useful to follow the notation just introduced. Eq. 2.9 can be rewritten as:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} = r \begin{bmatrix} \frac{\omega_l + \omega_r}{2} \\ \frac{-\omega_l + \omega_r}{2c} \end{bmatrix}$$

Solving for ω_l and ω_r, the wheel speeds are then:

$$\begin{aligned} \omega_l &= u_1 - c\,u_2 \\ \omega_r &= u_1 + c\,u_2 \end{aligned}$$

Using Eq. 2.32 and 2.9, which is a relationship valid for SSMR kinematics as well, it is possible to write:

$$\begin{aligned} \tau_l &= K_{p_\tau}(u_1 - c\,u_2 - \dot{x} + c\,\dot{\theta}) \\ \tau_r &= K_{p_\tau}(u_1 + c\,u_2 - \dot{x} - c\,\dot{\theta}) \end{aligned} \tag{2.44}$$

Building from the assumptions and the linearization proposed in Sec. 2.3, the generalised equations of the rover become:

$$\begin{aligned}\ddot{x} &= \frac{1}{m}(F_{xR} + F_{xL} - R_x)\\ \ddot{\theta} &= \frac{1}{I}[(F_{xR} - F_{xL})\, c - M_r]\end{aligned}$$

or rather:

$$\begin{aligned}\ddot{x} &= \frac{1}{m}\left[\frac{1}{r}(\tau_R + \tau_L) - \dot{x} m g K_u\right]\\ \ddot{\theta} &= \frac{1}{I}\left[\frac{c}{r}(\tau_R - \tau_L)\, c - \dot{\theta} m g K_y c\right]\end{aligned}$$

Finally, substituting Eq. 2.44:

$$\begin{aligned}\ddot{x} &= \dot{x}\left(-2\frac{K_{p_\tau}}{rm} - mgK_u\right) + 2\frac{K_{p_\tau}}{rm}u_1\\ \ddot{\theta} &= \frac{1}{I}\left[\dot{\theta}\left(-\frac{K_{p_\tau}t^2}{2r} - mgK_y c\right) + \frac{K_{p_\tau}t^2}{2r}u_2\right]\end{aligned} \tag{2.45}$$

It is now possible to select a set of state variables called *phase variables*, where each subsequent variable is defined as the derivative of the previous state variable. In this case:

$$\begin{aligned}x_1 = x &\rightarrow \dot{x}_1 = x_2\\ x_2 = \dot{x} &\rightarrow \dot{x}_2 = \ddot{x}\\ x_3 = \theta &\rightarrow \dot{x}_3 = x_4\\ x_4 = \dot{\theta} &\rightarrow \dot{x_4} = \ddot{\theta}\end{aligned}$$

Substituting in Eq. 2.45 and converting to matrix form:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & -2\frac{K_{p_\tau}}{rm} - mgK_u & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 0 & \frac{1}{I}\left(-\frac{K_{p_\tau}t^2}{2r} - mgK_y c\right)\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix} + \begin{bmatrix}0 & 0\\ \frac{2K_{p_\tau}}{rm} & 0\\ 0 & 0\\ 0 & \frac{K_{p_\tau}t^2}{2Ir}\end{bmatrix} \tag{2.46}$$

Since the output variables required by the control system are the same state variables included in the vector $\mathbf{x}$, the output equations can be simply written as:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{2.47}$$

Eq. 2.46 and 2.47 can be finally written in a more compact form that is equivalent to the one presented in Eq. 2.43:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x}$$

From Eq. 2.46 it is possible to see that the effects of the two control inputs are decoupled: the input u_1 only affects the linear motion of the rover, while the input u_2 only affects its angular response. In other words, the assumptions made in Sec. 2.2.3, upon which the whole linear formulation is constructed, allowed to shift from a MIMO system to two decoupled Single-Input and Single-Output (SISO) systems. Such property is essential for a more straightforward analysis of the control system.

Chapter 3

Experimental Set-up and Model Tuning

This chapter introduces the experimental set-up and describes the model tuning process necessary to faithfully simulate the behaviour of the testing platform: the Cleapath Husky A200. An overview of the testing facility of the BeiDou Research Institute in Shanghai is presented as well. A brief explanation is dedicated to ROS, to better understand how the robot was controlled during the experimental tests. Furthermore, this chapter features an innovative parameter estimation method based on a class of evolutionary algorithms named Differential Evolution Algorithms. The last pages are dedicated to the accuracy analysis of one of the most common robotics simulators: Gazebo Simulator. Its conclusions explain why it was decided not to use it for this research.

3.1 Clearpath Husky

Husky A200, by Clearpath Robotics Inc., is the SSMR used as the testing platform. It is a four-wheel differential drive unmanned ground vehicle customizable with a wide variety of research payloads. Due to its rugged construction and high torque drivetrain, it is often used for outdoor exploration. Moreover, it is fully supported in ROS thanks to community-driven Open Source code. It is therefore able to receive the user control commands either via remote or via an onboard computer.

3.1.1 Physical Properties

The robot geometry is as represented in Fig. 3.1. With a length of 0,99 m, a width of 0,67 m, a height of 0,39 m and a 20 Ah lead-acid battery pack (VRLA), the robots weights little over 73 kg (Tab. 3.1). Its right and left pairs of wheels are

mechanically linked, and each pair is actuated by an electric motor. The motor specifications, however, were not made available by the producer.

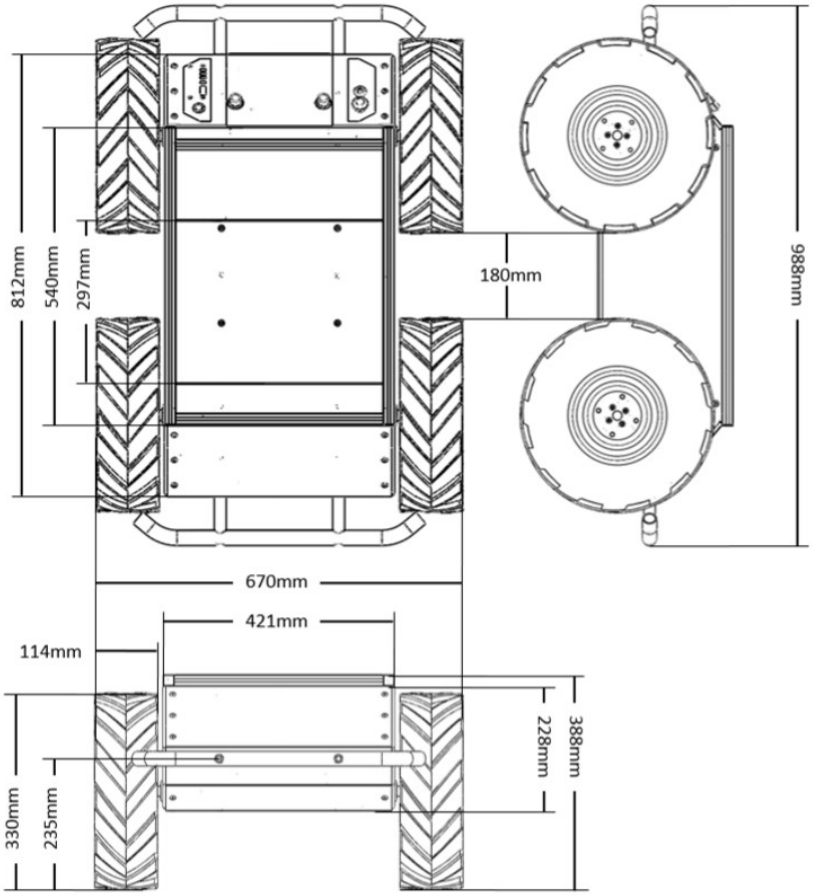

Figure 3.1. Clearpath Husky A200 geometry - Image from the robot user manual

mass	kg	73,30
length	m	0,99
width	m	0,67
height	m	0,39
track width	m	0,55
I moment of inertia around z	$kg\,m^2$	2,16
max linear speed	m/s	1,00
max angular velocity	rad/s	2,00

Table 3.1. ClearPath Husky A200 Specifications

3.1.2 Onboard Computer and Accepted Commands

The platform can be controlled in two ways:

- Remote: via a Logitech controller which can output linear and angular velocity commands.
- Onboard Computer: it is possible to use the ROS middleware to give commands at different levels of autonomy.

The onboard computer used during the experimental tests ran on Linux 18.04 LTS and used ROS Melodic Morenia as middleware. The joystick and the computer combined can output two native command types plus a third one which was implemented as part of the research work:

1. **Linear and angular velocity**: in this case, the robot has no autonomy and uses a control system to regulate the wheel speeds. This command is the only type of command which can be generated by a remote.
2. **Target pose**: it is possible to communicate a target pose[1] and let the robot navigate autonomously in a known environment. If the sensors allow it, the robot can also navigate in an unknown environment as well, by mapping its surrounding area. Once the robot localises itself, it autogenerates sequential intermediate target poses and the corresponding speed commands necessary to reach them. The commands are eventually fed to the low-level wheel control system.
3. **Reference trajectory**: not supported natively, this type of input can be interpreted only thanks to the control systems developed during this research activity. Once the trajectory has been read, the robot is then able to parse it into successive poses and feed them to an underlying control system. This enhancement is better analysed during the following chapters.

Each input enters the system at different stages of the control flow. Starting from the first one, each control grants the robot an increasing degree of autonomy. A schematic representation of the enhanced control flow is available in Figure. 3.2.

It is important to point out that the trajectory control systems presented later, only affect the position and velocity control layers. Indeed, a built-in MCU is in charge of:

- Converting the robot reference velocities into wheel speeds

[1] A pose differs from a "position" for, apart from indicating a point in space, it also contains information about its orientation. Following the conventions of Chap. 2, a pose can be expressed as a vector of three elements. E.g. pose = $\begin{bmatrix} X & Y & \theta \end{bmatrix}$

- Actuating the wheels using the electric motors
- Controlling the wheel speeds

Since the MCU internal functioning and parameters were not made available to the public, a system identification method is proposed in the next chapters.

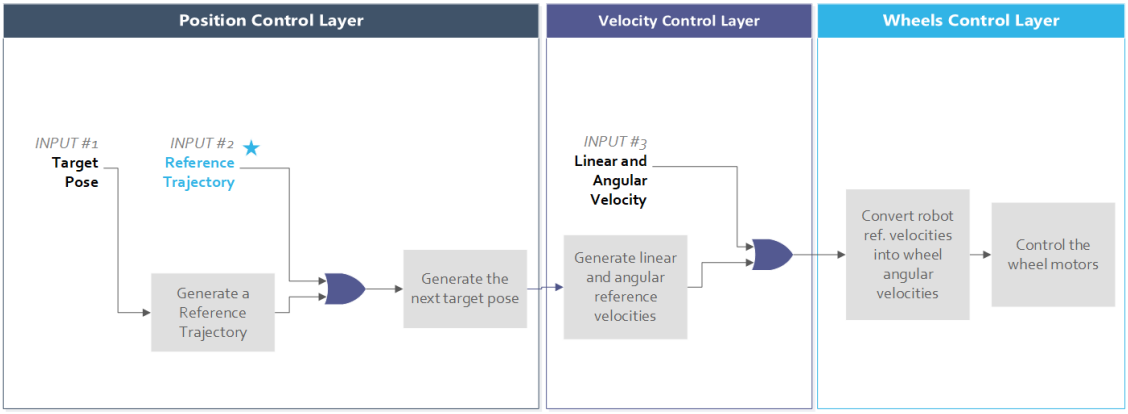

Figure 3.2. Husky logic diagram from a generic control input to wheel motion. Input 2: a new type of control input introduced by this research work.

3.1.3 Sensor Types and Sensor Fusion

The Husky used for the experimental tests had only two sensors:

- **UM6 Orientation Sensor** (CH Robotics LLC): an IMU sensor that provides information about the robot spacial orientation expressed in quaternions, its angular velocity and its linear acceleration.
- **Wheel Encoders**: they provide information about the robot velocity and incremental movement. Each encoder has a precision of 78'000 ticks/m.

The data generated by the two sensors is then fed to an Extended Kalman Filter (EKF), the nonlinear variant of the Kalman filter. It is used to estimate the position of the robot by knowing the previous state, the applied input and the sensor measurements. Without it, the dead-reckoning measurements would be severely compromised by the skid-steering phenomenon. The EKF relies on the *robot localization packagae*, a ROS stack specifically developed by Clearpath Robotics. For the purpose of this work, the EKF parameters were left as default.

3.2 Indoor Testing Facility

The indoor testing facility at the West Hongqiao division of the BeiDou Research Institute of Shanghai is a 500 m^2 space dedicated to robotics applications (Fig. 3.3). The facility is overlooked by 62 Infrared (IR) motion tracking cameras which measure the movement of any set of markers, allowing to asses the response of a robot with sub-millimetric precision. Fig. 3.4 shows the underlying functioning of any IR tracking system: the cameras emit beams in the IR spectrum and measure what is bounced back by the markers. The information measured by every camera is then processed to output the final result. In the case of the BeiDou Research Institute, the cameras were the RTS 1000 produced by Realis, a Chinese company specialised in tracking systems (Tab. 3.2) while the markers were four retro-reflectors mounted on the four corners of the Husky rover.

Figure 3.3. BeiDou Research Institute - Robotics laboratory by the West Hongquiao facility

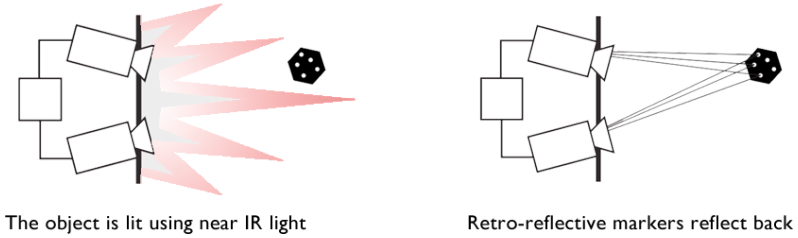

Figure 3.4. Infrared (IR) tracking scheme

Image Resolution	1280x1024 @ 210 FPS
AOV	58° x 48°
LED	14 high-power LEDs RTS1000
Measurement Frequency	4.8 ms
Maximum Distance of the Object	13 m

Table 3.2. Specifications of the IR motion tracking camera RTS1000 produced by Realis

3.3 Robot Operating System (ROS)

ROS, as introduced in Chap. 1, is a tool-based peer-to-peer framework that helps develop and deploy robotics software solutions. The following paragraphs introduce the most basic concepts of ROS in order to enable the reader to better understand how the proposed control systems interact with the Husky robotic platform.

Excluding more advanced elements like services and actions, the most fundamental building blocks of ROS are:

- **Nodes**: nodes are executables written in Python or C++. Thanks to ROS they can communicate with other nodes. This feature allows them to produce complex behaviours starting from simple functionalities.
- **Topics**: topics are named buses over which nodes exchange data. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of whom they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic. Lastly, topics are intended for unidirectional, streaming communication.
- **Messages**: a message is a simple data structure, comprising typed fields. Nodes communicate with each other by publishing messages on topics.

It is thus possible to imagine a robot as a graph comprised of nodes exchanging messages over predefined topics. The example in Fig. 3.5 shows an example of an elementary node interaction. The uppermost element, the ROS Master, can be simply described as a software process that gives different nodes all the information required to exchange information between each other. Once the nodes have located themselves through the Master, the connection switches to peer-to-peer. In other words, nodes directly communicate with each other without relying on any external piece of software.

One more ROS feature to introduce is called *tf*. tf is a package that lets the user keep track of multiple coordinate frames over time. It maintains the relationship

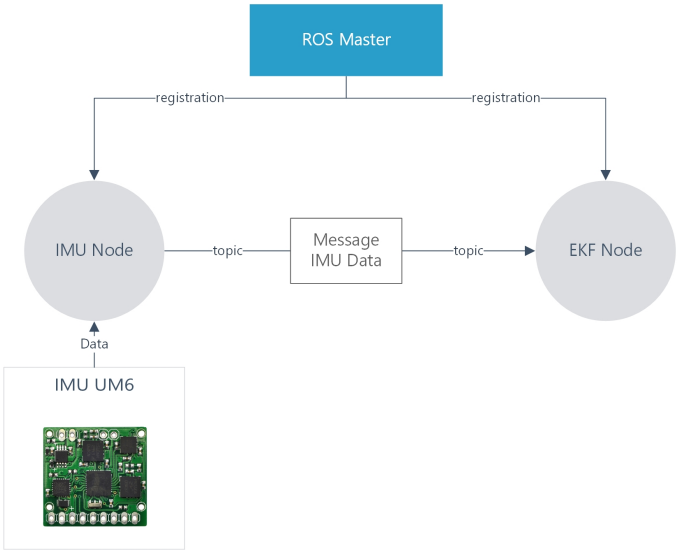

Figure 3.5. Example of the basic functioning of ROS: two nodes registered to the Master exhanging data over a topic.

between coordinate frames in a tree structure buffered in time and lets the user transform points and vectors between any two coordinate frames at any desired point in time. It is therefore essential to correctly read the data coming from sensors located at different points of the robots. Fig. 3.6 shows the complete ROS graph of the Husky robot used for experimental purposes. It is possible to notice numerous processes that stream their data to a tf node in order to express any physical quantity with respect to the local body frame of the robot.

Focusing on the software running the Husky itself, the odometry and control functions are implemented in the same fashion of any other piece of software within ROS: they are based on a number of interconnected nodes. The understanding of their structure is a prerequisite for appreciating of the proposed control system, later in this thesis. The odometry stack functions as shown in Fig. 3.7, that is:

1. Two different nodes read the encoders and IMU data, respectively. They are named `husky_node` and `imu_filter_madgwick`.

2. Their outputs are sent over the topics `husky_velocity_controller/odom` and `imu/data` and read by the node `ekf_localization_node`, which is in charge of fusing the sensors data.

3. The EKF node outputs are then transmitted over the `odometry/filtered` topic.

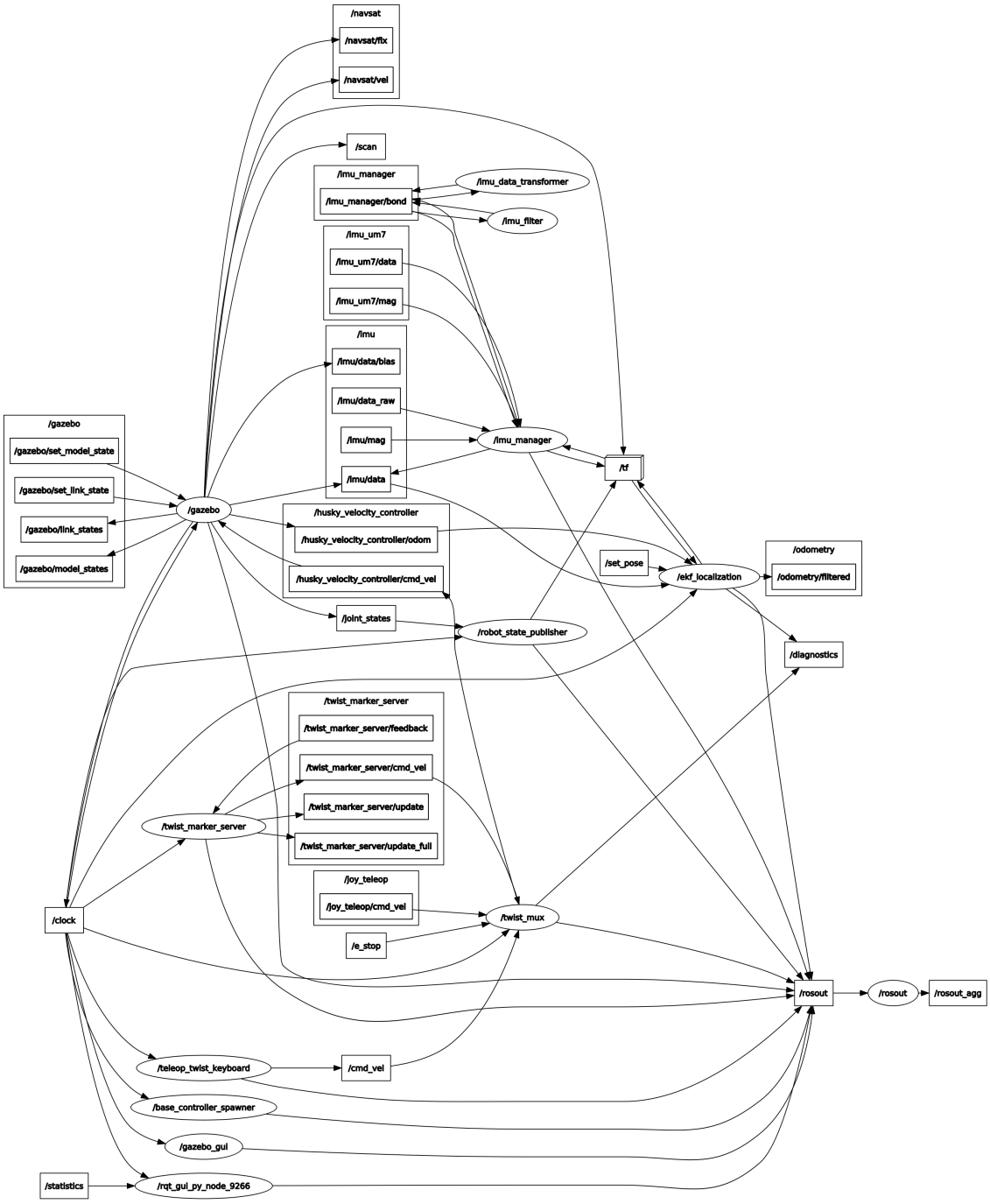

Figure 3.6. ROS graph of the Husky robot used during tests. The image was taken while using the Gazebo simulator, hence the presence of a node named gazebo.

On the other hand, the command stack in Fig. 3.8 follows this scheme:

- The commands - linear and angular velocity - can be publishished on four different topics:
  - `joy_teleop/cmd_vel` - Joystick teleoperation input - Priority 10
  - `twist_marker_server/cmd_vel` - Interactive marker (pose) input through RViz[2] - Priority 8
  - `move_base/cmd_vel` - Autonomous movement input from the `husky_navigation` package. - Priority 2
  - `cmd_vel` - Miscellaneous external input. It it the topic that will be later used to control the Husky - Priority 1
- The node `twist_mux` receives the commands and, if more than one source is available, it will output the one with the highest priority.
- `twist_mux` outputs the final command on the topic `husky_velocity_controller/cmd_vel` which is then fed to the wheel controller on the integrated MCU.

The actual movement of the rover measured by the IR tracking system is published on the `/vrpn\_client\_node/husky/pose` topic.

[2]RViz, abbreviation for ROS visualization, is a 3D visualization tool for ROS. It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information.

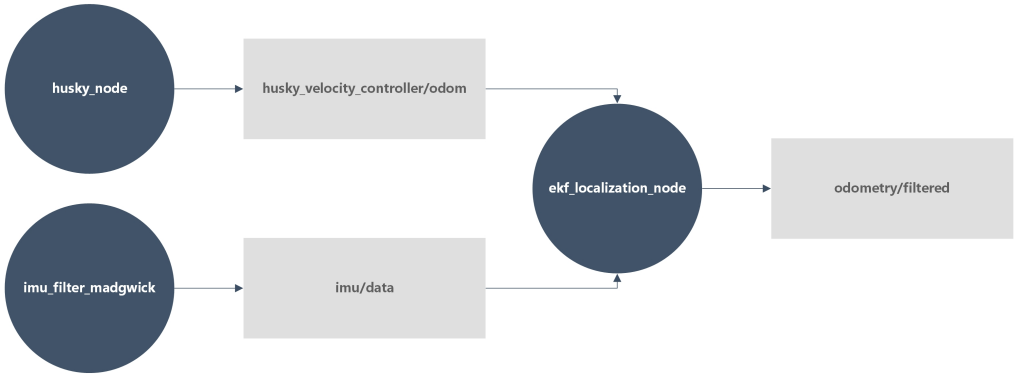

Figure 3.7. ROS topics and nodes in a Husky robot - Odometry functions

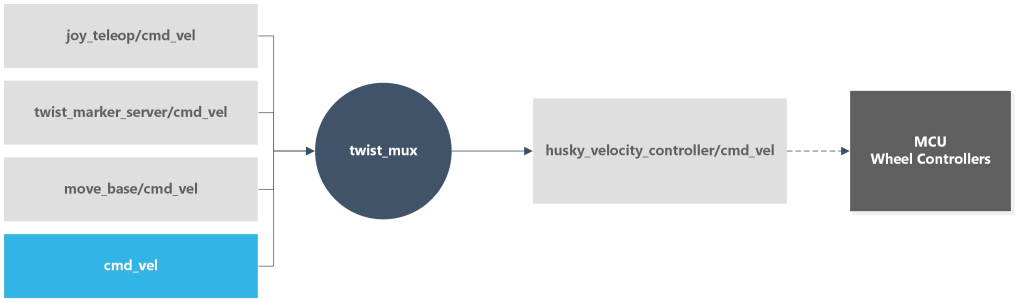

Figure 3.8. ROS topics and nodes in a Husky robot - Control functions - The highlighted topic will be used to control the Husky platform in the subsequent chapters.

3.3.1 MATLAB Integration

It is worth mentioning that despite the ROS integration with MATLAB 2019a with the Robotics System Toolbox, it is not possible to use the MATLAB console to run a Linux script that executes ROS commands, a beneficial functionality indeed. In particular, scripts that function flawlessly when called from the Linux shell fail to execute when called using the common `![script path]` syntax within MATLAB. It was concluded that the cause of this issue is MATLAB itself, that changes the Linux environment variables whenever it is running, therefore altering the correct functioning of any script. To overcome such problem, it was decided to modify the affected scripts to include a series of `export` instances to reset the environment variables to the desired values. The pseudo-code of a generic script of that kind is provided for future references; the correct values of the environment variables can be determined by running the `env` command within the Linux shell.

```
#!/bin/bash
export LD_LIBRARY_PATH=...;
export ROS_ETC_DIR=...;
export CMAKE_PREFIX_PATH=...;
export ROS_ROOT=...;
export ROS_MASTER_URI=...;
export ROS_VERSION=...;
export ROS_PYTHON_VERSION=...;
export PYTHONPATH=...;
export ROS_PACKAGE_PATH=...;
export ROSLISP_PACKAGE_DIRECTORIES=...;
export PATH=...;
export PKG_CONFIG_PATH=...;
export ROS_DISTRO=...;
Script body;
```

3.4 Simulink Physical Models

In the chapter dedicated to the mathematical description of the SSMR - Chap. 2 - three variants were proposed:

1. The nonlinear model with the Coulomb friciton model.
2. The nonlinear model with the improved friction formulation.
3. The LTI state-space model with the hyperviscous friction formulation.

In the following paragraphs, each one of them is converted into a Simulink model, upon which the optimisation processes were based. The same models are used to simulate the behaviour of the controllers later proposed. The possibility to rely on faithful simulations proves to be essential to avoid many trial and error operations and, consequently, for a swift deployment of the control systems.

All models, independently from their underlying mathematical formulation, share some commonalities. The first is the Input-Output scheme shown in Fig. 3.9, notably:

- **Inputs**
  - u_cmd - linear velocity command
  - $omega\_cmd$ - angular velocity command
- **Outputs**
  - $q[X, Y, theta]$ - robot pose in the global reference frame
  - $dq[dX, dY, dtheta]$ - robot velocity vector in the global reference frame
  - $eta[dx, dtheta]$ - robot velocity vector in the local reference frame

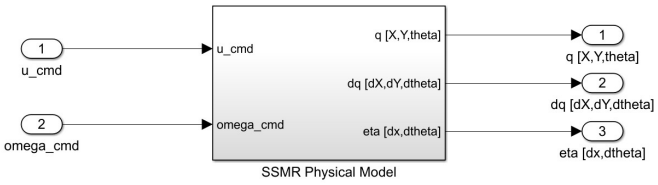

Figure 3.9. Inputs and outputs of a Husky rover - Simulink model

Fig. 3.10 depicts the inside structure of the block shown in Fig. 3.9. The physical models later introduced, only differ for the *Husky Plant* block. Instead, the common blocks are:

- **Saturation Blocks** - one for each command, they limit the upper and lower values of u_cmd and $omega\_cmd$. For the Husky rover these values are: $u\_cmd \in [-1, +1]\,\mathrm{m/s}$ and $omega\_cmd \in [-2, +2]\,\mathrm{rad/s}$.
- **Delay Blocks** - one for each command, they simulate the delay introduced by the internal processing units during the transmission and conversion of the inputs. Their entity will be discussed in Sec. 3.5.
- **Wheel Torque Controller** - as introduced in Sec. 2.2.5, this block will simulate the feedback control loop that converts the velocity inputs into wheel torque commands and regulates them, too.

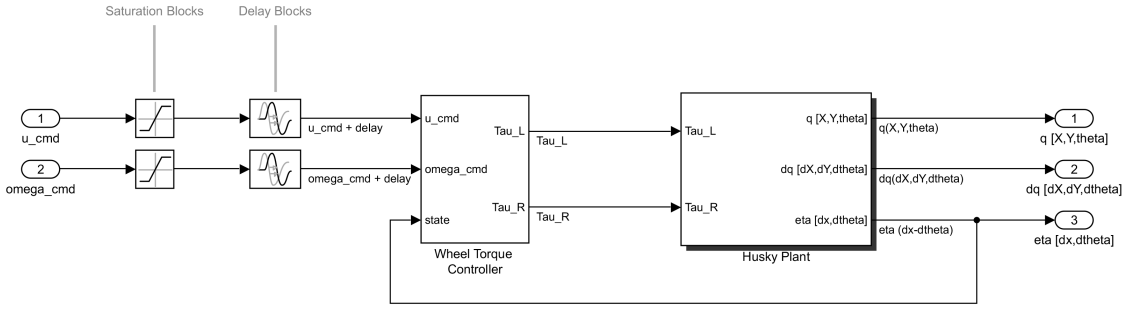

Figure 3.10. Commands flow of a Husky rover - Simulink model

The wheel torque controller in Fig. 3.11 is comprised of two branches, both based on the equations presented in Sec. 2.2.5. The top branch processes the reference velocities that are the command signals after they passed through the saturation and delay blocks. The lower branch converts the robot velocity vector in the local reference frame into actual wheel speeds. The difference between the two branches outputs an error value which is then fed to a pure proportional gain. Its output, in turn, is saturated to simulate the mechanical limits of the wheel electric motors.

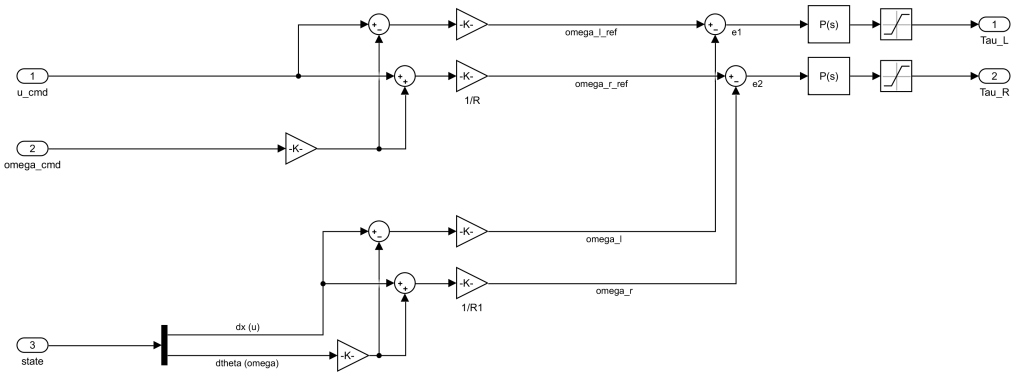

Figure 3.11. Wheel Torque Controller of a Husky rover - Simulink model

3.4.1 Nonlinear Model with Coulomb Friciton

The model in Fig. 3.12 corresponds to the block-diagram representation of Eq. 2.28. The structure is self-explanatory, as all the relevant matrices can be clearly seen.

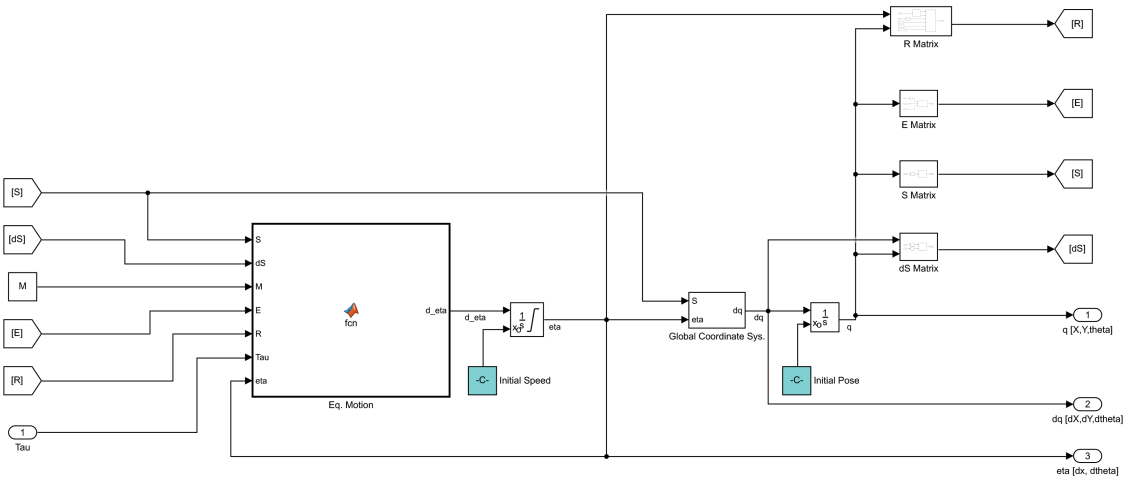

Figure 3.12. Husky Plant - Nonlinear model with Coulomb friciton - Simulink model

3.4.2 Nonlinear Model with the Improved Friction Formulation

In Fig. 3.13 it is possible to see the block-diagram of the nonlinear dynamic model based on the improved friction formulation. The upper branch, in green, evaluates the longitudinal dynamics of the vehicle, while the lower branch, in blue, computes the lateral-directional dynamics of the rover related to its rotation θ around the CoG.

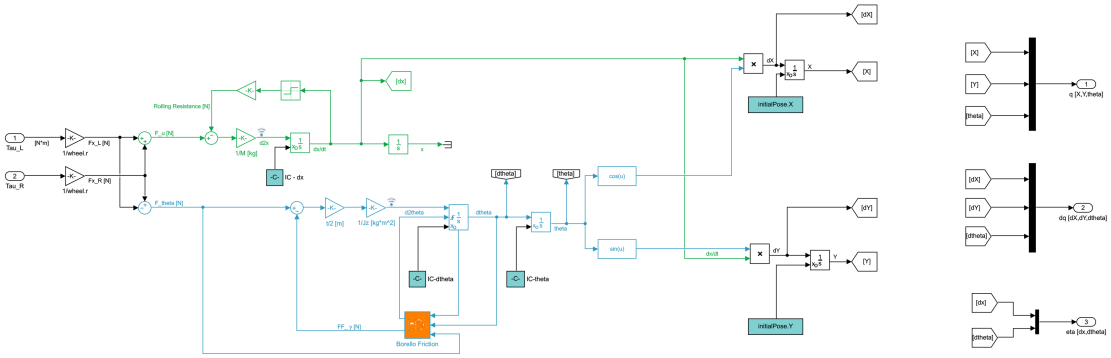

Figure 3.13. Husky Plant - Nonlinear model with the improved friction formulation - Simulink model

The orange block contains the improved friction model and has the same structure reported in Fig. 3.14. Its inputs are:

1. $Velocity - StatePort$: The velocity of the rover, in this case $\dot{\theta}$, computed by the state port of the integrator between $\ddot{\theta}$ and $\dot{\theta}$. Using the integrator standard output, the simulation would incur into an algebraic loop, a computation error that stops the program. Its sign change detector is used to set the value of $\dot{\theta}$ to zero.

2. $Velocity$: The standard output of the integrator described above.

3. F_act : The resultant of the active forces.

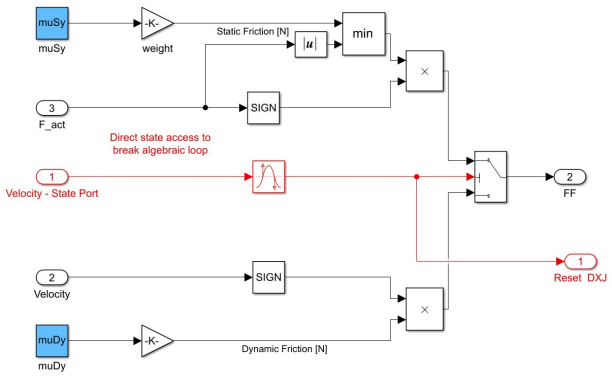

Figure 3.14. Block diagram of the improved dry friction model with a zero crossing detection control scheme

3.4.3 LTI State-Space Model with the Hyperviscous Friction Formulation

The block diagram in Fig. 3.15 represents the LTI state-space formulation expressed in Sec. 2.3. It relies on the default State-Space block. All that lies after the demultiplexer is needed to shift from the local reference frame to the global reference frame.

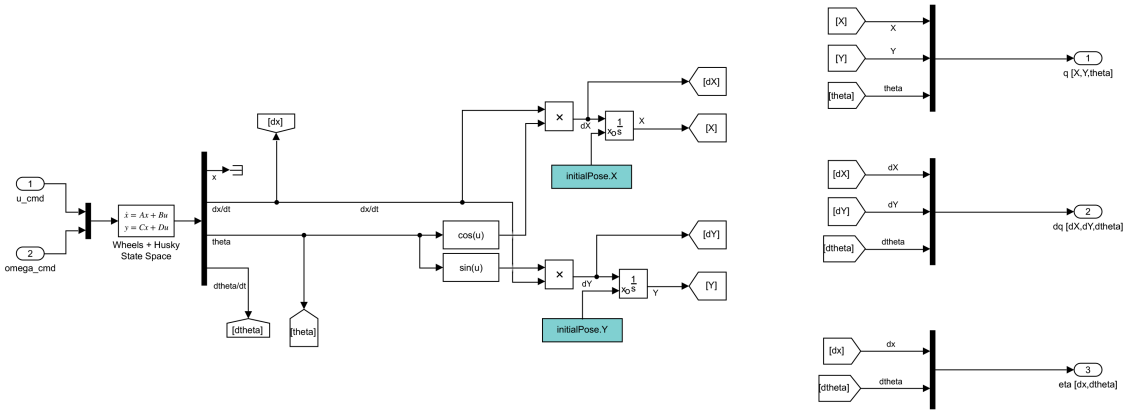

Figure 3.15. Husky Plant - LTI State Space formulation - Simulink model

3.4.4 Sensors Noise Model

As presented in Sec. 3.1, the Clearpath Husky used for experimental tests relied on five sensors - four wheel encoders and one IMU - whose data was interpreted and

merged by an EKF. The EKF output, like any other sensor's, is affected by noise. The analysis of the nature of the noise is a discipline by itself and is, therefore, outside of the scope of this thesis. It is useful, however, to address its modelling, as noise can severely impact the performance of a controller. Taking, for instance, a PID controller, the derivative gain, if not tuned accounting for noise, can bring the system to unstable behaviours when tested in a real scenario.

To understand the properties of the noise affecting the EKF of the Clearpath Husky employed, its output was measured in different scenarios and then replicated via Matlab and Simulink. The conditions the robot was tested at are listed in Tab. 3.3 and were used to determine the noise component affecting the quantities contained in the $q[X, Y, theta]$ and $eta[dx, dtheta]$ vectors. The dq vector was not considered as it is not used by the control system, and its modelling is therefore unrequired.

Test Case N.	**cmd_u**	**cmd_omega**
1	0.25	0
2	0.50	0
3	0	0.25
4	0	0.50

Table 3.3. Test cases employed for the EKF sensor noise analysis

Each test case was processed in MATLAB as follows:

- The measurements are converted into a *timeseries* objects
- Each timeseries is filtered with an highpass filter (*highpass* command) to retain only the noise component of the signal.
- The noise variance (*var* command) is evaluated.

The variances are then used to set up the Simulink *random number* blocks whose output is summed to the original signals to replicate their noise component, as in the example in Fig. 3.16.

Tab. 3.4 sums up the the variances obtained by applying, for each physical measure, a hipass filter with the following settings:

- Fpass - Passband frequency - 10 Hz
- Steepness - Transition band steepness - 0.85
- Stopband Attenuation - 60 dB

An example of the results achieved with the signal dx is presented in Fig. 3.17.

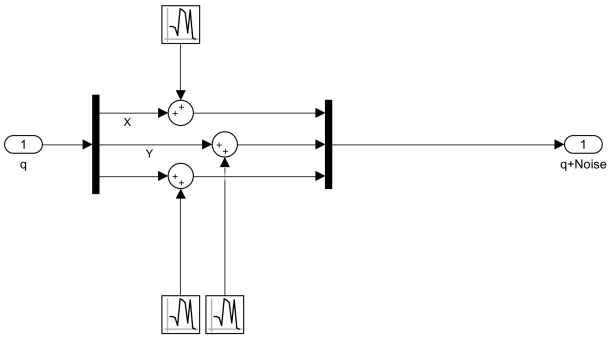

Figure 3.16. Sensor noise component added to the q vector within the Simulink model of the SSMR

Paramter	Variance
dx	5.0115e-05
dtheta	3.5298e-05
X	8.8786e-06
Y	7.9259e-12
theta	1.2941e-08

Table 3.4. Test cases employed for the EKF sensor noise analysis

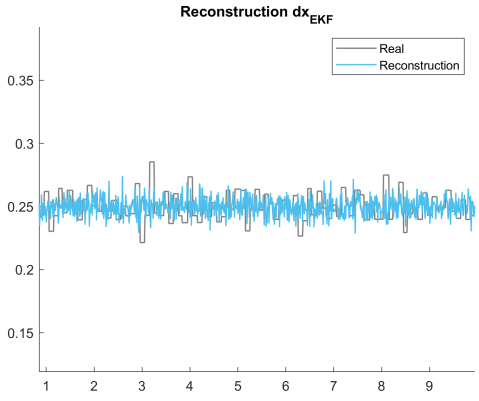

Figure 3.17. EKF dx output reconstruction (continuous time) compared to the real measured output (discrete time)

3.4.5 External Disturbances

The external disturbances acting on the Husky rover during experimental tests within a laboratory environment are minimal. Despite that, they were included in the Simulink models to test the control systems better. The disturbances were chosen so that they affected the angular accelerations of the rover. They were simulated including a random number block, whose output was summed to the $\ddot{\theta}$ acceleration (Fig. 3.18). The variance of the block was set to 25, a value determined by trial and error.

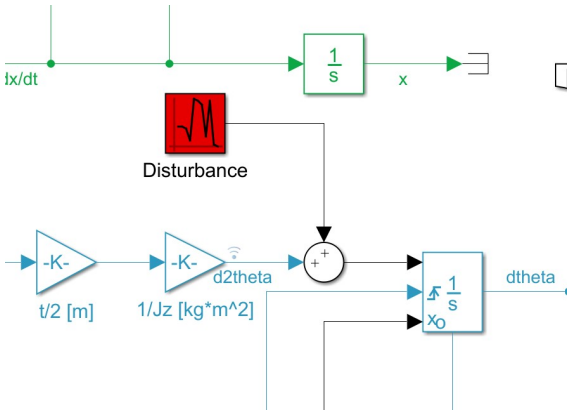

Figure 3.18. External disturbance added to the angular acceleration of the husky rover - Simulink

3.4.6 Models Discretisation

The mathematical models implemented in Simulink were discretised so that they could simulate the real behaviour of the Husky rover. In order to do that, *Zero-Order Hold* (ZOH) blocks were used. A ZOH holds its input for the sample period specified; in the case of the Husky, there were multiple signals which were transmitted or received at different frequencies, as shown in Tab. 3.5.

Signal	Frequency [Hz]	Sample Time [s]
RTS	120	0.008
EKF	50	0.020
Command Input	20	0.050

Table 3.5. Frequency and Sample Time for each signal interfacing with the Husky rover

3.5 Differential Evolution for Unknown Parameters Estimation

Section 1.5 introduces the topic of optimisation processes based on Differential Evolution algorithms, a class of Evolutionary Algorithms, which in turn represent one of the main branches of AI. The following subchapter presents how the *jDE/best/2* with gradient descent self-adapting variant of such algorithms were used to determine the unknown parameters present in the mathematical formulations introduces so far. The underlying idea behind this process is:

- A real Husky rover was controlled with a sequence of known commands. As it was tested in the laboratory environment presented in Sec. 3.2, its exact response was tracked and measured with the IR system.
- A DE algorithm was designed so that, by executing numerous Simulink simulations, it could experiment with different values of the unknown parameters until the simulated behaviour would mimic the real one.

By doing so, and by testing the results on more than one sample trajectory, it was possible to find the combination of unknown physical parameters which guarantees the most faithful simulation model, given the mathematical premises it is based upon. Certainly, the single parameters are not exact by themselves, but their combination is. Furthermore, this type of optimisation, which aims at replicating a real, measured, result, allows to include in the output parameters the effects of other nearly exact or totally unaccounted physical quantities, if not of entire dynamic effects.

3.5.1 Code Properties and Structure

The self-adapting DE algorithm employed, first proposed by [58], is known as *jDE* or, more specifically, *jDE/best/2* with gradient descent. It is used throughout this text due to its good convergence performances [57], its improved robustness [66] and due to the extensive work already conducted by the hosting University, the Shanghai Jiao Tong University.

jDE codes, in general, follow a self-adaptation scheme in which the control parameters F and CR are encoded into the individual members of a population and are adjusted by introducing two new parameters, τ_1 and τ_2. Initially, F and CR values of each individual are known and identical. A random number $rand$ is then uniformly generated in the range [0,1]. If $rand < \tau_1$, the F value is reinitialised to a new value in its predefined range of [0.1,1.0]; otherwise, it is kept unchanged. A similar process is valid for CR, which is compared to the parameter τ_2 and is, eventually, reinitialised to the range [0,1]. This research in particular, employed a variant of the classic jDE algortim, in which, if F or CR are not smaller than the

randomly generated number, they are evaluated using a gradient descent instead of being kept unchanged. In mathematical terms, the jDE scheme with gradient descent can be expressed as:

$$F_{i_{G+1}} = \begin{cases} F_{min} + F_{max} \cdot \text{rand}(2) & \text{if rand}(1) \leq \tau_1 \\ F_{i_G} - F_{i_G} \cdot slp \cdot (ng-1)/ng_{max} & \text{if rand}(1) > \tau_1 \text{ and } F_{i_{G+1}} \geq F_{min} \\ F_{min} & \text{in any other case} \end{cases}$$

$$CR_{i_{G+1}} = \begin{cases} \text{rand}(3) & \text{if rand}(4) \leq \tau_2 \\ CR_{i_G} - CR_{i_G} \cdot slp \cdot (ng-1)/ng_{max} & \text{if rand}(4) > \tau_2 \text{ and } CR_{i_{G+1}} \geq CR_{min} \\ CR_{min} & \text{in any other case} \end{cases}$$

where slp is the constant slope gradient, ng is the generation currently evaluated, and ng_{max} is the maximum number of generations the algorithm is limited to.

The notation *best/2*, on the other hand, refers to the mutation scheme presented by [67]:

$$V_{i,G} = X_{best,G} + F \cdot (X_{r_1^i,G} - X_{r_2^i,G}) + F \cdot (X_{r_3^i,G} - X_{r_4^i,G})$$

where the indices $r_1^i, r_2^i, r_3^i, r_4^i$ are mutually exclusive integers randomly generated within the range $[1, NP]$. That is to say that the mutation scheme creates a donor vector starting from four randomly selected vectors from the current population.

The code was written in MATLAB and divided into a main file and two subfunctions to allow faster riconfiguration:

- **main**: intialise all the required parameters
- **jDE_best_2_desc**: executes all the mutation, crossover and selection processes based on the parameters set in the main file.
- **eval_obj_fun**: receives the parameter vector from the jDE/best/2 function, uses it to run a simulation in Simulink, evaluates the resulting objective function and returns it to the jDE function for the selection process.

The main file has the following structure:

```
1  Import the reference trajectory and the commands used to generate it;
2  Set the SSMR known properties, the initial conditions, the Simulink solver
     parameters;
   /* Initialise the jDE algorithm                                    */
3  Set the unknown parameters as global variables and declare their search
     bounds;
4  Set the objective function weigths;
5  Set the population number parameter NP;
6  Set the maximum number of generations n_gen;
7  Set the intial scale factor F;
8  Set the initial cross-over rate CR;
9  Set the obj fun relative error threshold for optimisation stop df;
   /* Run the optimisation                                            */
10 Call the jDE_best_2_desc function to run the optimisation process;
11 Save and print the optimisation results;
```

Algorithm 1: Main File - Data Initialisation and Results Output

The MATLAB function eval_obj_fun, which receives the simulation parameters from the jDE optimiser, runs Simulink and computes the objective function, has the following structure:

```
1  Receive the parameter and trial vectors from the jDE_best_2_desc
     function;
2  Run the Simulink simulation using the new input parameters;
3  Parse the results;
4  for each point of the reference trajectory do
5  |   for each point of the simulated trajectory do
6  |   |   Evaluate the distance between each point dist_traj(i);
7  |   end
8  end
9  Evaluate the RMS of dist_traj(i);
10 Evaluate di maximum of dist_traj(i) - i.e. Housdorff distance ;
11 Evaluate the bending energy ;                          /* if needed */
12 Evaluate the objective function;
13 Return the objective function result;
```

Algorithm 2: eval_obj_fun - Simulink simulations and objective function evaluation

On the other hand, the jDE function called at line 10 of the main is:

```
1  Receive the input parameters from the main;
2  Initialise the jDE parameters: τ1, τ2, F_bound, CR_bound, slp, nr, nX;
3  Initialize the first parameter vector X_init;
4  for each parameter to be optimised do
5  |  X_init(i,:) is a randomly selected value within the search boundaries;
6  end
   /* Run the optimisation process                                    */
7  while number of generations ≤ n_gen AND relative difference between the
     obj function of two subsequent generations ≤ df do
8  |  for each element of the population do
9  |  |  Find four random indexes within NP;
10 |  |  Evaluate randF : two random numbers in the range F_bound;
11 |  |  if randF(1) ≤ τ1 then
12 |  |  |  Evaluate F using randF(2) else
13 |  |  |  |  Evaluate F using the slope descent with gradient slp;
14 |  |  |  end
15 |  |  end
16 |  |  Define the mutant vector using F and the four selected parameter
   |  |   vectors;
17 |  end
18 |  for each element of the population do
19 |  |  Evaluate randCR : two random numbers in the range CR_bound;
20 |  |  if randCR(1) ≤ τ2 then
21 |  |  |  Evaluate CR using randCR(2) else
22 |  |  |  |  Evaluate CR using the slope descent with gradient slp;
23 |  |  |  end
24 |  |  end
25 |  |  Build a trial vector;
26 |  end
27 |  for each element of the population do
28 |  |  Call the external function eval_obj_fun to test both the sample
   |  |   vector and the trial vector within Simulink ;
29 |  |  if the objective function improves using the trial vector then
30 |  |  |  Keep the trial vector;
31 |  |  |  Evaluate the relative improvement of objective function;
32 |  |  end
33 |  end
34 end
```

Algorithm 3: jDE_best_2_desc - Optimisation process with gradient descent

3.5.2 Sample Trajectories

Four reference trajectories were tested in the BeiDou Research Institute robotics laboratory by creating a simple ROS node in Python that executed a predefined series of commands. In particular, as the greatest interest regards the dynamic effects which arise during steering, it was decided to command constant linear velocity and, when the rover had reached its target speed, angular velocity, too. In particular, the angular velocity command followed a sinewave, of which amplitude and frequency were changed to obtain different sample trajectories. At higher frequencies or amplitudes, in fact, nonlinear effects become more evident, and the vehicle response changes significantly. The parameters of the four trajectories are listed in Tab. 3.6 and a graphical example of the commands provided for trajectory number 2 is present in Fig. 3.19.

The rover position was measured by the IR tracking system and its data, together with the rover internal measurements, were registered using the *rosbag* functionality provided by ROS. The rosbag was then processed in MATLAB: it was cropped and smoothed, to remove some sensors' noise. The smoothing process relied on the native `smoothdata` function, using a Gaussian-weighted moving average with a *SmoothingFactor* of 0.01. The SmoothingFactor is a parameter that determines the size of the window considered by the algorithm at each data point: a greater value produces a greater smoothing. In the cases here presented, a very low number was adopted to remove only the underlying sensor noise. It was concluded that more intense smoothing would have cut through some important parameters variations. An example of the low impact smoothing applied is visible in Fig. 3.20.

Trajectory	**u_cmd [m/s]**	**omega_cmd [rad/s]**	**f [Hz]**	**A [m]**
1	0.5	$A \cdot sin(\omega t)$	0.50	1.00
2	0.5	$A \cdot sin(\omega t)$	0.25	0.25
3	0.5	$A \cdot sin(\omega t)$	0.25	0.50
4	0.5	$A \cdot sin(\omega t)$	0.25	0.75

Table 3.6. Reference trajectories parameters

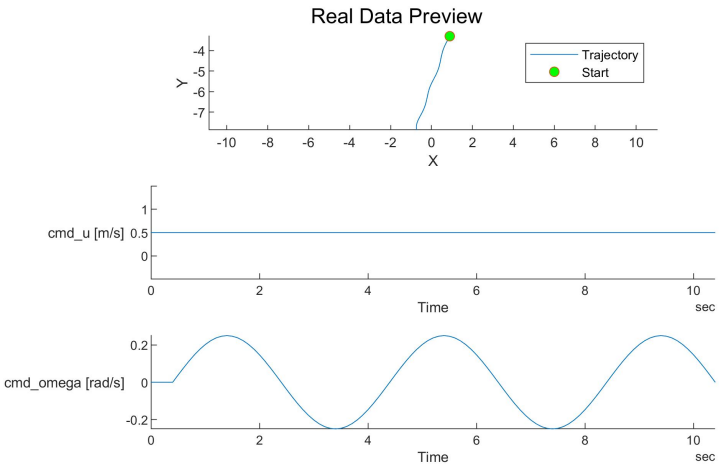

Figure 3.19. Sample trajectory n.2 commands and odometry measurements

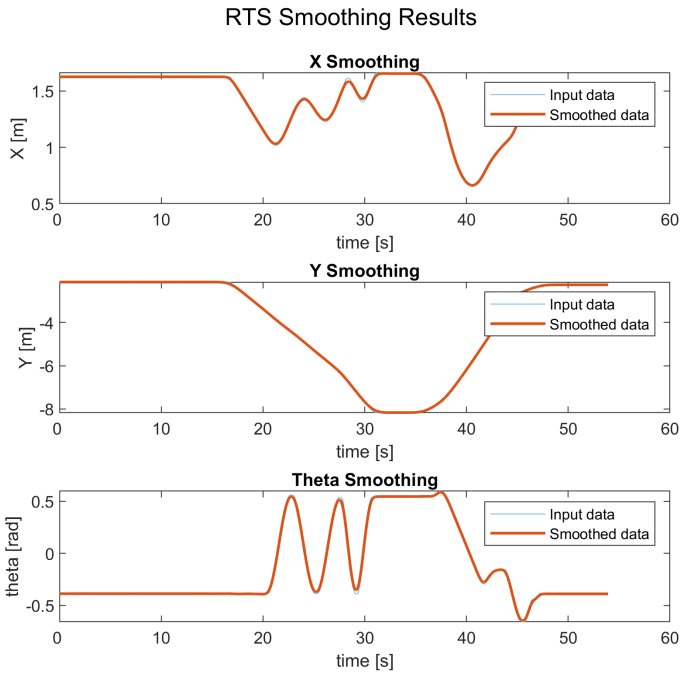

Figure 3.20. Example of the results achived by smoothing the RTS data

3.5.3 Objective Function

For the optimisation of the physical models of the Husky rover, it was chosen to employ the following quantities:

- d_{rms} - The RMS of the distances between the sample trajectory and the simulated trajectory. The more similar the two trajectories - i.e. the greater the accuracy of the mathematical model - the closer d_{rms} should be to zero.
- d_{haus} - The Hausdorff distance between the sample trajectory and the simulated trajectory. As in the case of the RMS, it should tend to zero.
- d_{ratio} - The ratio between the simulated trajectory and the sample trajectory. In a perfect simulation, it is equal to 1.

They were combined to form the following function:

$$f = K_{rms} \cdot d_{rms} + K_{haus} \cdot d_{haus} + K_{dratio} \cdot d_{ratio} \tag{3.1}$$

A rigorous description of the single elements of the function is provided in Sec. 1.6.

3.5.4 Optimisation of the Nonlinear Model with Coulomb Friction

In the nonlinear model with the Coulomb friction formulation the following parameters had to be determined:

- K_{P_τ} - Wheel torque controllers proportional gain
- u_x - Rolling resistance coefficient
- μ_c - Coulomb friction coeffient between the wheels and the ground along y.

The linear and angular velocity command delays - $delay_u$ and $delay_\omega$ respectively - were manually determined by observing the response curves of the rover. The maximum torque generated by the electric motors, instead, was found using the Husky specifications published by Clearpath Robotics Inc.

The configuration parameters in Tab.3.7 were used to run the optimisation cycles. The objective function weigths were adjusted until they guaranteed the best results, as well as the parameters search boundaries which were gradually reduced to ensure faster convergence. By instructing the DE algorithm to replicate the fourth trajectory described in Tab. 3.6, as it represented the most common case of application of the rover, the results in Tab. 3.8 were achieved after 1500 iterations, that is after 30 generations. The parameters were then tested on the other three sample trajectories - 1,2 and 3 - giving the results in Tab. 3.9 and depicted in Fig. 3.21, 3.22, 3.23 and 3.24.

Comparing the results, it appears that:

- The reference trajectory - trajectory number 4 - is simulated with a high degree of realism. Not only can be seen in Fig. 3.24 but also in its objective function coefficients. As a matter of fact, d_{rms} and d_{haus} have an order of magnitude of 10^{-2}, while d_{ratio} shows that the simulation length is only 3.12% longer that the real trajectory. It is deemed to be a satisfactory result.

- Trajectory number 1 is, by far, the one that most diverges from reality. The main reasons are two: first, it is the case with the higher frequency and amplitude of the angular velocity command, which means it is the case with the greatest nonlinearities; second, it was concluded that the motor engines output slightly different torque, leading to some unpredicted behaviours, almost invisible in the other, more linear trajectories. It will be more evident while analysing the trajectory controllers.

Parameter	**Description**	**Value**
F_{init}	F initial value	0.75
F_{bound}	F adaptation boundaries	[0.10,0.90]
CR_{init}	CR initial value	0.80
CR_{bound}	CR adaptation boundaries	[0.00,1.00]
τ_1	F adaptation parameter	0.10
τ_2	CR adaptation parameter	0.10
nr	Number of vectors used for mutation	4
slp	Gradient descent slope	0.20
df	Stop condition - relative improvement of the obj f.	0.001
NP	Population size	50
n_{gen}	Maximum number of generations	300
K_{rms}	Objective function weight for d_{rms}	20
K_{haus}	Objective function weight for d_{haus}	8
K_{dratio}	Objective function weight for d_{ratio}	5
$steptime$	Simulink fixed step time	0.001 [s]

Table 3.7. Nonlinear model optimisation - jDE configuration parameters

Parameter	Description	Search Boundaries	Result
$delay_u$	Linear vel. cmd. delay	-	0.22 [s]
$delay_\omega$	Angular vel. cmd. delay	-	0.22 [s]
$\tau_{max_{wheel}}$	Max and min wheel torque	-	$\pm$ 7 [Nm]
u_x	Rolling resistance coeff.	[0.00, 0.10]	0.0727 [m]
μ_c	Coulomb friction coeff. - y	[0.10, 0.20]	0.0254
K_{P_τ}	Wheel torque proportional gain	[20.00,100.00]	56.75

Table 3.8. Nonlinear model optimisation - Final values of the unknown parameters

Trajectory	$d_{rms}\,[m]$	$d_{haus}\,[m]$	d_{ratio}
1	0.1416	0.3726	0.9666
2	0.0217	0.1039	1.0408
3	0.0252	0.0557	0.9998
4*	0.0146	0.0250	1.0312
*reference trajectory used with jDE			

Table 3.9. Nonlinear model optimisation - Objective function terms obtained by simulating the sample trajectories using the parameters estimated setting trajectory n.4 as reference case during the optimisation process.

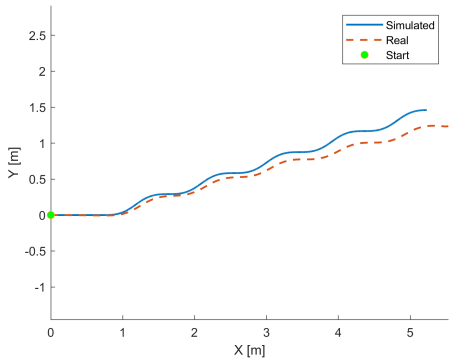

Figure 3.21. Trajectory 01 - Real vs Simulated - Nonlinear model with Coulomb friction optimisation run by setting traj 04 as reference.

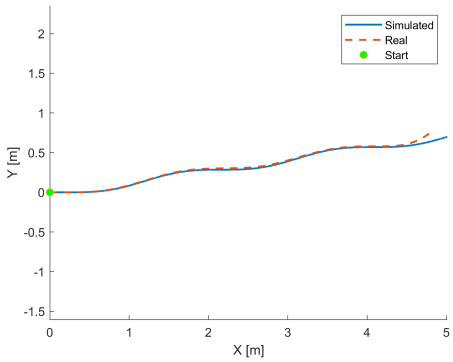

Figure 3.22. Trajectory 02 - Real vs Simulated - Nonlinear model with Coulomb friction optimisation run by setting traj 04 as reference.

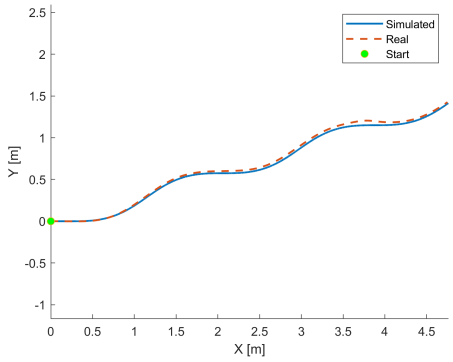

Figure 3.23. Trajectory 03 - Real vs Simulated - Nonlinear model with Coulomb friction optimisation run by setting traj 04 as reference.

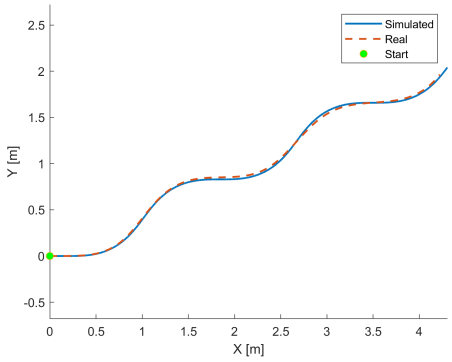

Figure 3.24. Trajectory 04 - Real vs Simulated - Nonlinear model with Coulomb friction optimisation run by setting traj 04 as reference.

3.5.5 Optimisation of the Nonlinear Model with the Improved Friction Formulation

In the nonlinear model with the improved friction formulation the following parameters had to be determined:

- K_{P_τ} - Wheel torque controllers proportional gain
- u_x - Rolling resistance coefficient
- μ_s - Static friction coeffient between the wheels and the ground along y.
- μ_d - Dinamic friction coeffient between the wheels and the ground along y.

The commands delays, as well as the maximum wheel torque, were kept identical to the previous case. The configuration parameters in Tab.3.10 were used to run the optimisation cycles. The objective function weights were adjusted until they guaranteed the best results, as well as the parameters search boundaries which were gradually reduced to ensure faster convergence. By instructing the DE algorithm to replicate the fourth trajectory described in Tab. 3.6, as it represented the most common case of application of the rover, the results in Tab. 3.11 were achieved after 2500 iterations, that is after 50 generations. The parameters were then tested on the other three sample trajectories - 1,2 and 3 - giving the results in Tab. 3.12 and depicted in Fig. 3.25, 3.26, 3.27 and 3.28.

The considerations made in the previous case are still valid. It is worth noting though that the search boundaries of μ_s and μ_d had to be chosen so that they didn't overlap, as it avoided numerical problems during the simulations.

Parameter	Description	Value
F_{init}	F initial value	0.75
F_{bound}	F adaptation boundaries	[0.10,0.90]
CR_{init}	CR initial value	0.80
CR_{bound}	CR adaptation boundaries	[0.00,1.00]
τ_1	F adaptation parameter	0.10
τ_2	CR adaptation parameter	0.10
nr	Number of vectors used for mutation	4
slp	Gradient descent slope	0.20
df	Stop condition - relative improvement of the obj f.	0.001
NP	Population size	50
n_{gen}	Maximum number of generations	300
K_{rms}	Objective function weight for d_{rms}	20
K_{haus}	Objective function weight for d_{haus}	8
K_{dratio}	Objective function weight for d_{ratio}	5
$steptime$	Simulink fixed step time	0.001 [s]

Table 3.10. Optimisation of the nonlinear model with the improved friciton formulation - jDE configuration parameters

Parameter	Description	Search Boundaries	Result
$delay_u$	Linear vel. cmd. delay	-	0.22 [s]
$delay_\omega$	Angular vel. cmd. delay	-	0.22 [s]
$\tau_{max_{wheel}}$	Max and min wheel torque	-	$\pm$ 7 [Nm]
u_x	Rolling resistance coeff.	[0.05, 0.20]	0.1282 [m]
μ_s	Static friction coeff. - y	[0.08, 0.15]	0.0972
μ_d	Dynamic friction coeff. - y	[0.00, 0.07]	0.0409
K_{P_τ}	Wheel torque proportional gain	[60.00,120.00]	118.04

Table 3.11. Optimisation of the nonlinear model with the improved friciton formulation - Final values of the unknown parameters

Trajectory	$d_{rms}\,[m]$	$d_{haus}\,[m]$	d_{ratio}
1	0.1420	0.3720	0.9668
2	0.0222	0.1049	1.0411
3	0.0252	0.0561	1.0000
4*	0.0145	0.0255	1.0403
*reference trajectory used with jDE			

Table 3.12. Optimisation of the nonlinear model with the improved friciton formulation - Objective function terms obtained by simulating the sample trajectories using the parameters estimated setting trajectory n.4 as reference case during the optimisation process.

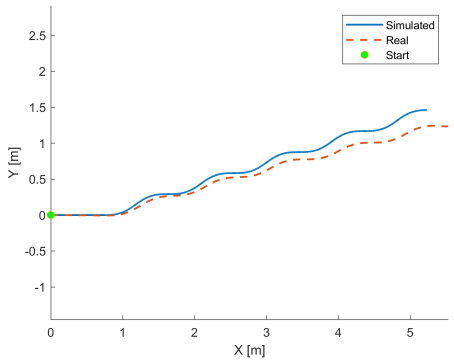

Figure 3.25. Trajectory 01 - Real vs Simulated - Optimisation of the nonlinear model with the improved friction fromulation run by setting traj 04 as reference.

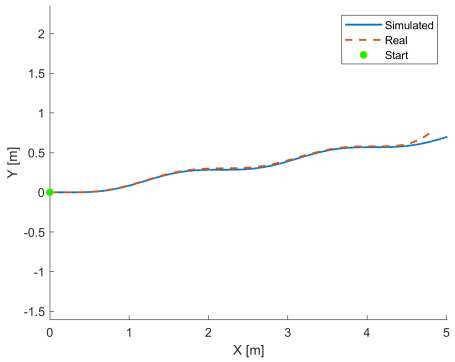

Figure 3.26. Trajectory 02 - Real vs Simulated - Optimisation of the nonlinear model with the improved friction fromulation run by setting traj 04 as reference.

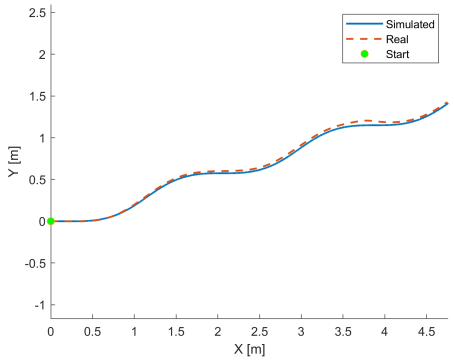

Figure 3.27. Trajectory 03 - Real vs Simulated - Optimisation of the nonlinear model with the improved friction fromulation run by setting traj 04 as reference.

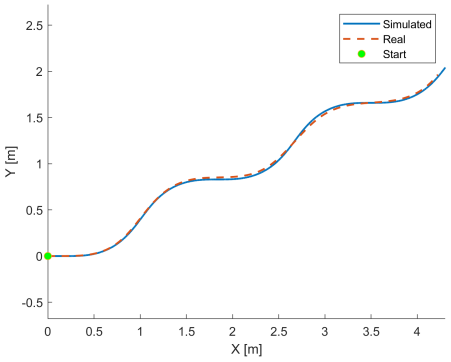

Figure 3.28. Trajectory 04 - Real vs Simulated - Optimisation of the nonlinear model with the improved friction fromulation run by setting traj 04 as reference.

3.5.6 Optimisation of the LTI State-Space Model with the Hyperviscous Friction Formulation

In the LTI state space formulation with the hyperviscous friction formulation the following parameters had to be determined:

- K_{P_τ} - Wheel torque controllers proportional gain
- K_u - Longitudinal motion viscous coefficient
- K_y - Lateral motion viscous coefficient

The commands delays, as well as the maximum wheel torque, were kept identical to the first case presented. The configuration parameters in Tab.3.13 were used to run the optimisation cycles. The objective function weights were adjusted until they guaranteed the best results, as well as the parameters search boundaries which were gradually reduced to ensure faster convergence. By instructing the DE algorithm to replicate the fourth trajectory described in Tab. 3.6, as it represented the most common case of application of the rover, the results in Tab. 3.14 were achieved after 5750 iterations, that is after 115 generations. The parameters were then tested on the other three sample trajectories - 1,2 and 3 - giving the results in Tab. 3.15 and depicted in Fig. 3.29, 3.30, 3.31 and 3.32.

The considerations made in the first case presented are still valid.

Parameter	**Description**	**Value**
F_{init}	F initial value	0.75
F_{bound}	F adaptation boundaries	[0.10,0.90]
CR_{init}	CR initial value	0.80
CR_{bound}	CR adaptation boundaries	[0.00,1.00]
τ_1	F adaptation parameter	0.10
τ_2	CR adaptation parameter	0.10
nr	Number of vectors used for mutation	4
slp	Gradient descent slope	0.20
df	Stop condition - relative improvement of the obj f.	0.001
NP	Population size	50
n_{gen}	Maximum number of generations	300
K_{rms}	Objective function weight for d_{rms}	20
K_{haus}	Objective function weight for d_{haus}	8
K_{dratio}	Objective function weight for d_{ratio}	5
$steptime$	Simulink fixed step time	0.001 [s]

Table 3.13. Optimisation of the state-space model with the hyperviscous friction model - jDE configuration parameters

Parameter	**Description**	**Search Boundaries**	**Result**
$delay_u$	Linear vel. cmd. delay	-	0.22 [s]
$delay_\omega$	Angular vel. cmd. delay	-	0.22 [s]
$\tau_{max_{wheel}}$	Max and min wheel torque	-	$\pm$ 7 [Nm]
K_u	Longitudinal motion viscous coeff.	[0.01, 0.03]	0.0277
K_y	Lateral motion viscous coeff.	[0.00, 0.03]	0.0298
K_{P_τ}	Wheel torque proportional gain	[80.00,300.00]	229.58

Table 3.14. Optimisation of the state-space model with the hyperviscous friction model - Final values of the unknown parameters

Trajectory	$d_{rms}\,[m]$	$d_{haus}\,[m]$	d_{ratio}
1	0.1427	0.3793	0.9641
2	0.0154	0.0677	1.0382
3	0.0176	0.0451	0.9972
4*	0.0150	0.0269	1.0285
*reference trajectory used with jDE			

Table 3.15. Optimisation of the state-space model with the hyperviscous friction model - Objective function terms obtained by simulating the sample trajectories using the parameters estimated setting trajectory n.4 as reference case during the optimisation process.

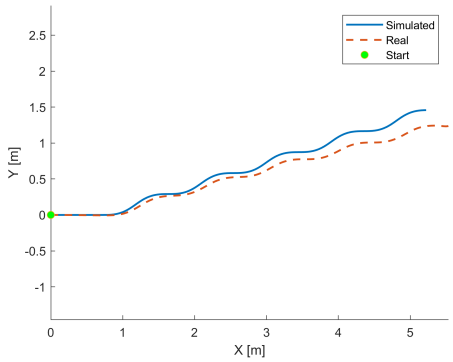

Figure 3.29. Trajectory 01 - Real vs Simulated - Optimisation of the state-space model with the hyperviscous friction model run by setting traj 04 as reference.

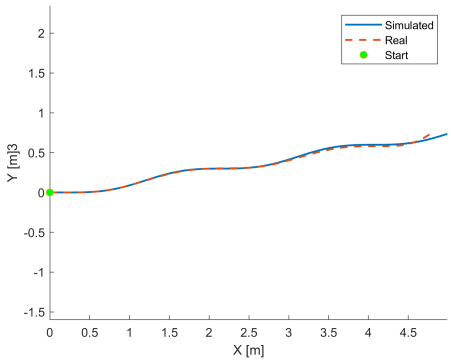

Figure 3.30. Trajectory 02 - Real vs Simulated - Optimisation of the state-space model with the hyperviscous friction model run by setting traj 04 as reference.

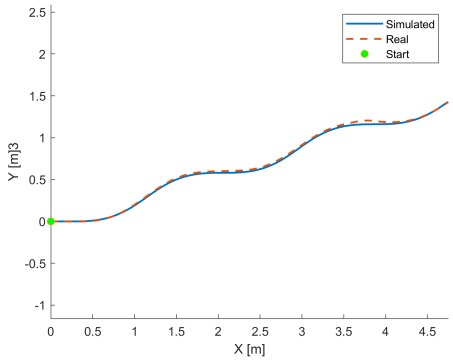

Figure 3.31. Trajectory 03 - Real vs Simulated - Optimisation of the state-space model with the hyperviscous friction model run by setting traj 04 as reference.

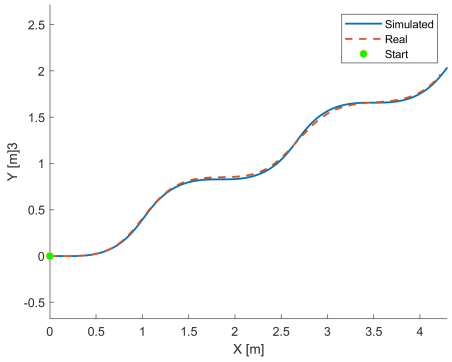

Figure 3.32. Trajectory 04 - Real vs Simulated - Optimisation of the state-space model with the hyperviscous friction model run by setting traj 04 as reference.

3.6 Mathematical Models Comparison

Comparing the results obtained with the three mathematical models, it is clear that they are very similar to each other. None of the models can correctly represent the first trajectory but, as already explained, that is due to unaccounted nonlinearities and wheel motor imperfections. As expected, despite having unknown parameters in common - e.g. the wheel torque proportional gain - their optimised value changes from case to case, as the DE algorithm looks for the best suitable combination of parameters, instead of their isolated exact values.

The linear model appears to be a fitting simplification of the nonlinear counterparts. It was therefore used to assess the initial response of the proposed controllers and to design the LQR controller in particular, whose mathematical description requires the system to be expressed in state-space. The nonlinear models, on the other hand, are required for accurate simulations. Comparing their outputs in terms of XY coordinates is not enough to asses which one is better: the most considerable differences can be appreciated by looking at the acceleration curves and the simulation time. Referring to Fig. 3.33, it possible to notice that the $d\theta/dt$ acceleration curves significantly differ from one model to the other: the Coulomb friction model exhibits numerical artefacts, while the improved friction model is much more faithful and includes the stiction phenomenon, too. Regarding the simulation time, since the Coulomb friction model is significantly more complex, it requires more time to be executed. Taking trajectory number four as reference, the model took on average it takes 2,75 seconds to simulate, while its counterpart, with all its simplifying assumptions, only took about 0,70 seconds. A reduction of 74% is of great importance when the simulation has to be repeated numerous times, as in the case of the trajectory tracking controllers optimisation via jDE.

Given all the considerations above, only the LTI and the nonlinear model with the improved friction formulation were used to design and test the controllers.

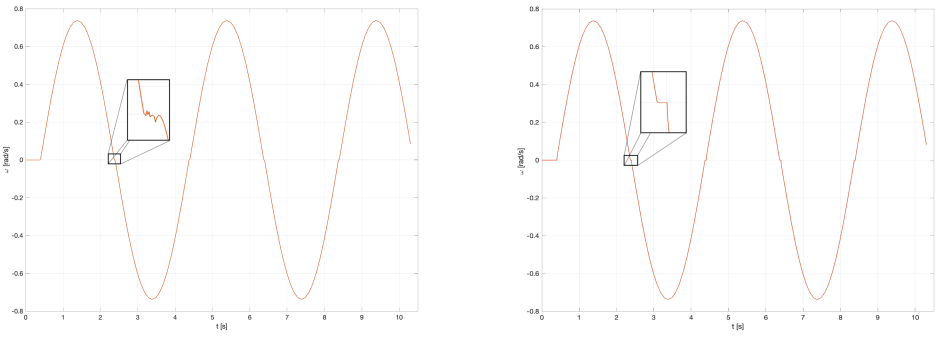

Figure 3.33. $d\theta/dt$ acceleration curves - Left: Coulomb friction with numerical instabilities; Right: improved friction formulation

3.7 Gazebo Simulator

Gazebo simulator, frequently used by the robotics community, was initially thought to be the best simulation environment to test the trajectory controllers with. In fact, it relies on popular physics engines, natively supports the Husky model developed by Clearpath Robotics, and can interface with Simulink via ROS. After thorough analyses and repeated attempts, however, it was concluded that its outputs were far off from reality, an issue not easily mended. This section offers a brief overview of the tests and consequent unsatisfactory results, which contributed to the final decision to replicate the model within Simulink and abandon Gazebo fully. The considerations here presented are only valid for Gazebo 9 and ROS Melodic.

Gazebo 9 supports four physics engine: ODE, Bullet, Simbody and DART. However, only one of them is compatible with ROS: ODE, which, not so coincidentally, is the default engine and also the most outdated. The creator's website (www.ode.org) defines ODE as "an open-source, high-performance library for simulating rigid body dynamics" with applications ranging from games to 3D animation tools and industrial simulators. At first, the only configuration deemed necessary was related to the ground-wheel friction coefficients, that had to replicate the testing facility conditions. Initial tests highlighted deeper problems, only solvable by thoroughly studying the user manual. This resulted in the selection of the parameters whose tuning would have impacted the robot dynamics the most:

- `iters` - The number of iterations for each simulation step. A higher number produces greater accuracy at a performance cost.
- `max_step_size` - Maximum time step size at which every system in the simulation can interact with the states of the world.
- `contact_surface_layer` - The depth of the surface layer around all geometry objects. Contacts are allowed to sink into the surface layer up to the given depth before coming to rest.
- `contact_max_correcting_velocity` - The maximum correcting velocities allowed when resolving contacts.
- `max_contacts`- Maximum number of contacts allowed between two entities. In this case, wheels and ground.
- `friction_model` - The type of friction model used by ODE; it can be a pyramid, box or cone model.
- `cfm` - Constraint force mixing parameter; used to smooth the contact between two surfaces.

- Friction Coefficients - namely `mu` and `mu_2`. The first determines the friction force along the first direction of the pyramid model, determined by the parameter `fdir1`, while the second along the direction perpendicular to `fdir1`.

As the detailed explanation of each parameter would be outside the scope of this thesis, the reader is advised to refer to the online documentation for further information.

The parameters were tested by means of MATLAB, which allowed to effortlessly reconfigure Gazebo, to output a predefined set of commands and to post-process the data recorder by ROS. The code was structured, as shown below:

```
1  Run a Linux script that opens Gazebo and executes all the launch file
     required;
2  Run a Linux script that resets the Husky pose and velocity;
3  Get from the user the commands to test (cmd_u and cmd_omega);
4  for i = 1:number of test cases do
5  |  Execute the script that loads the parameters of the i-th case;
6  |  Create a ROS publisher;
7  |  Create a ROS Service Client;
8  |  Execute the script that starts a rosbag;
9  |  for j = 1:(cmd duration × cmd frequency) do
10 |  |  Create a ROS message with the required cmd_u;
11 |  |  Create a ROS message with the required cmd_omega;
12 |  |  Publish the commands via the ROS publisher;
13 |  |  Wait for the duration of the commands (1/cmd frequency) ;
14 |  end
15 |  Create a ROS message with cmd_u = cmd_omega = 0;
16 |  Stop the rover by publishing the previous cmd;
17 |  Call the ROS Service Client that calls the
   |    /gazebo/get_physics_properties service;
18 |  Parse its output to read the Gazebo configuration parameters;
19 |  Stop and save the rosbag;
20 end
21 Shutdown ROS;
22 Close Gazebo;
```

Algorithm 4: MATLAB algorithm to test Gazebo parameters

The scripts used to switch between physics profile relied on the command:

```
gz physics --profile [preset_name]
```

The field `preset_name` is related to the preset coded into the `world` file launched by `husky_empty_world.launch` or any other launch file within the `husky_gazebo` package. The XML world file, in turn, was structured as follows:

```
<?xml version="1.0" ?>
<sdf version='1.6'>
  <world name='default'>
    <model name='ground_plane'>
      <link name='link'>
        <collision name='collision'>
          <surface>
            <friction>
              <ode>
                <mu>0.250000</mu>
                <mu2>0.150000</mu2>
              </ode>
            </friction>
          </surface>
        </collision>
    </model>

    <physics name='preset1' default='1' type='ode'>
      <max_step_size>...</max_step_size>
      <real_time_update_rate>...</real_time_update_rate>
      <max_contacts>...</max_contacts>
      <ode>
        <solver>
          <type>quick</type>
          <iters>...</iters>
          <sor>...</sor>
          <friction_model>...</friction_model>
        </solver>
        <constraints>
          <contact_surface_layer>...</contact_surface_layer>
          <contact_max_correcting_vel>...</contact_max_correcting_vel>
          <cfm>...</cfm>
          <erp>...</erp>
        </constraints>
      </ode>
    </physics>

    <physics name='preset2' default='0' type='ode'>...</physics>
    <!--other physics presets-->
  </world>
</sdf>
```

Tab. 3.16 shows the combination of parameters that better represented the experimental observations.

Despite the attempts, it was not possible to achieve satisfactory results, as the

Parameter	Value
iters	400
max_step_size	0.001 (default)
contact_surface_layer	0.001 (default)
contact_max_correcting_velocity	100
max_contacts	20 (default)
friction_model	cone
cfm	0 (default)
mu	0.25
mu2	0.15

Table 3.16. ODE physics engine main parameters - Gazibo Simulator

examples provided in Fig. 3.34 to 3.35. They show the measured response of the rover to the same inputs used to test trajectory number 3 (Tab. 3.6), which should produce a sinewave in the XY plane, preceded by a straight path.

The tests were conducted using the parameters presented in Tab. 3.16 and the three runs differ for the friction model employed: pyramid (default), box and cone model. The time axes shouldn't be taken as reference as they present some problems which couldn't be addressed at the time of the experiments.

Analysing the responses, it is evident that:

- Despite being approximations of the same physical description, the three friction model give very different results.
- The box friction model is, by far, the worst. The other two models are far off from reality, too, and produce very unreliable results.
- Even though the weel speeds are identical in modulus and opposite in sign, at some point during run 1 and 2, the simulated robot drives in a straight line. A behaviour which is not physically possible unless some external disturbances had arisen. This is clearly not the case of the simulation here reported.
- The angular velocity fluctuations with respect to the reference signal (Fig. 3.37) follow an unusual pattern and are accompanied by sudden linear accelerations. It is possible to see that in this cases the linear velocity is greater than the reference speed of about 40-50%.

These phenomena are far from the real response witnessed and measured in the laboratory environment. Not being able to solve such type of issues, it was concluded that replicating the mathematical model of the vehicle within the MATLAB-Simulink suite would have been best.

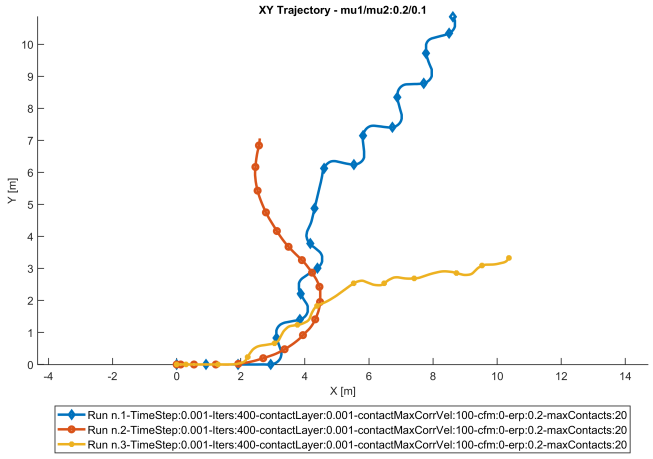

Figure 3.34. Trajectory in the XY frame - Run1: pyramid model; Run2: box model; Run3: cone model - Gazebo Simulator ODE physics engine

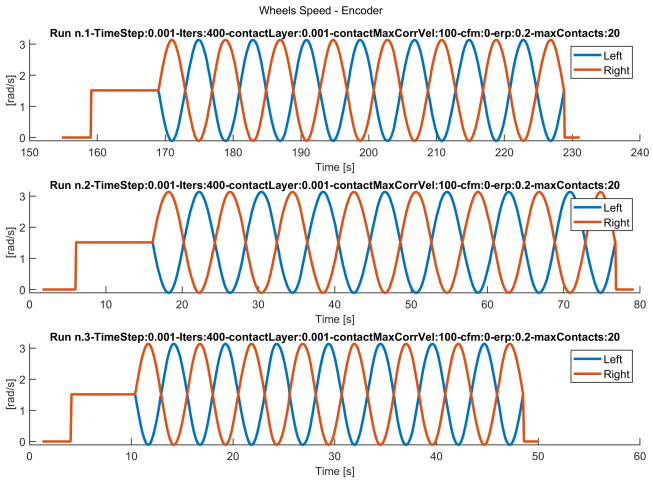

Figure 3.35. Wheels Speed - Run1: pyramid model; Run2: box model; Run3: cone model - Gazebo Simulator ODE physics engine

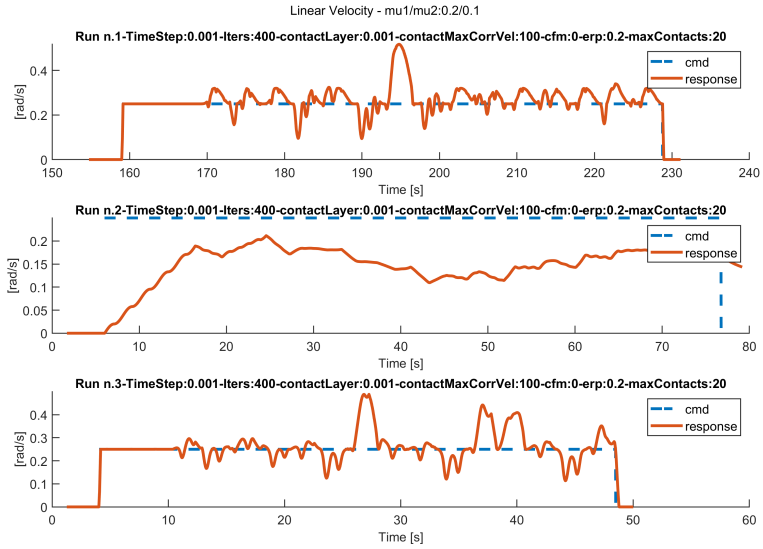

Figure 3.36. Linear Velocity - Run1: pyramid model; Run2: box model; Run3: cone model - Gazebo Simulator ODE physics engine

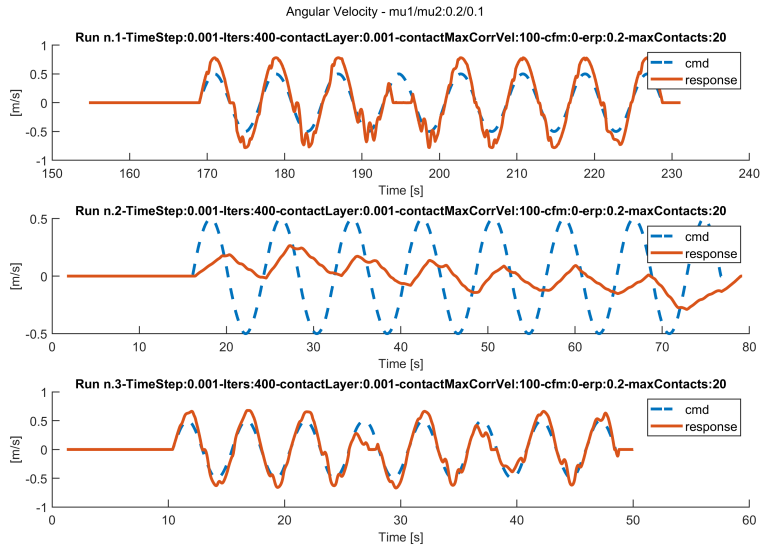

Figure 3.37. Angular Velocity - Run1: pyramid model; Run2: box model; Run3: cone model - Gazebo Simulator ODE physics engine

Chapter 4

Trajectory Tracking Controllers

This chapter presents the achievements related to the design and testing of two trajectory tracking controllers: one based on PID components and the other on a single LQR. The first sections introduce the topic of trajectory tracking, comparing it to its closest relative, path-following, and showing how a trajectory can be fed to a control system. A brief overview of the control system integration with ROS is then presented. The final section is dedicated to the PID and LQR controller, respectively, describing their mathematical properties, the tuning process based on DE, the Simulink results and, finally, the experimental measurements. It is worth remembering that the experimental results here presented are incomplete, for the same reasons highlighted at the beginning of this text: this research was abruptly stopped by an outbreak of COVID-19 in mainland China at the end of January 2020. Therefore, the last and most promising computer simulations related to the LQR couldn't be tested on the real rover.

4.1 Introduction to Trajectory Tracking

As the terms "trajectory" and "path" are often misused, it is deemed necessary to cast some light on the topic. A path is a series of subsequent points in space, also known as waypoints. A trajectory, on the other hand, is identical to a path but for the presence of time in its definition: on a plane, every point of a trajectory is determined by three coordinates, two spatial (x, y) and one temporal (t). Path-following is therefore employed in scenarios where time is irrelevant to the mission, and precise spacial movements are favoured. Generally, a path-following control system works with one waypoint at a time and aims at the subsequent waypoint when the vehicle is at a certain range from the target. Similarly, in the case of a trajectory, the waypoints are fed singularly, but the switch from one to the other

doesn't occur when certain proximity is achieved, but when the mission time is greater than the target temporal coordinate.

4.1.1 Waypoint Selector Architecture

The waypoint selection scheme just introduced was first developed in MATLAB - see Algorithm 5 - and then implemented in Simulink. Instead of solely feeding the known trajectory waypoints imported from a xlsx or CSV file, it was decided to linearly interpolate the X, Y and t coordinates each time the control systems operates. Doing so, the control system acquires a different target for each control cycle, thus preventing the position error from oscillating excessively. This solution proved to smooth the overall behaviour of the controllers.

The proposed algorithm can be easily used in ROS by converting it in C++ or Python. However, having the possibility of using Simulink, the latter was used to implement the interpolation, selection and stopping functionalities. The first two rely on the native *Table Lookup* blocks; the latter on a custom function block which contains lines 13 to 16 of the pseudo-code presented. Fig. 4.1 shows the Simulink block diagram as described.

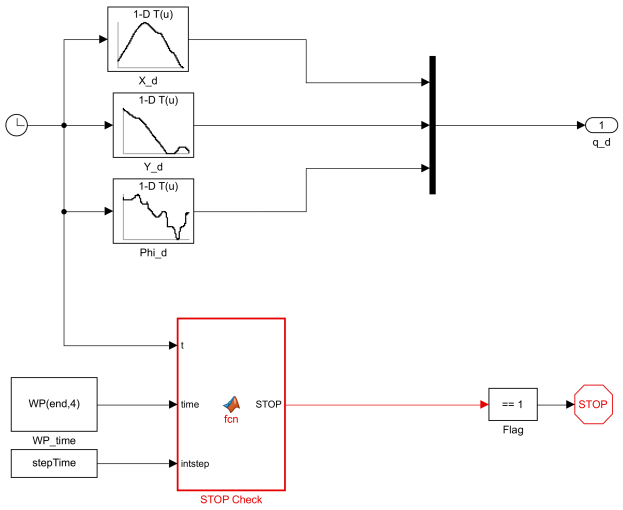

Figure 4.1. Simulink waypoint selector based on the the Table Lookup blocks

```
Import a series of waypoints from an xlsx or CSV file;
/* Waypoints parsing                                          */
if the waypoints don't include a time coordinate then
    Ask the user a reference velocity;
    for each waypoint do
        Evaluate the time coordinate so that the average constant velocity
         required to move from the previous waypoint to the one considered
         is equal to the reference velocity input by the user;
    end
end
for each waypoint do
    Evaluate the angle φ of the segment that connects the current waypoint
     to the next one;
end
/* Select the waypoint to feed to the control system          */
while stopCondition != 1 do
    t = mission time;
    if t == t_lastWaypoint then
        Select the last waypoint as target;
        stopCondition = 1;
    end
    else
        Find two waypoints WP_n(X_n, Y_n, t_n) and WP_{n+1}(X_{n+1}, Y_{n+1}, t_{n+1})
         so that t_n ≤ t < t_{n+1};
        Linearly interpolate the X,Y and t coordinate between the two
         selected waypoints;
    end
    Feed the result to the control system;
end
```

Algorithm 5: Waypoint selection algorithm

4.1.2 Position Error Evaluation Scheme

Downstream of the waypoint selector lies another block shared by the two control system: a position error evaluation block. Refferring to Fig. 4.2, five geometric quantities are introduced:

- ρ - The distance vector between the vehicle CoG to the next waypoint WP_n. In particular
$$|\rho| = \sqrt{\Delta X^2 + \Delta Y^2}$$
where ΔX is the distance between the vehicle and WP_n along the X axis and ΔY is the equivalent along the Y axis.

- β - The angle between the ρ vector and the local x axis.

- α - The angle between the x axis and the segment connecting the target waypoint WP_n to the one after it WP_{n+1}.

- ϕ - The angle between the global X axis and the segment connecting the target waypoint WP_n to the one after it WP_{n+1}.

- Δy - The distance between the vehicle CoG and the next waypoint WP_n along the y local axis.

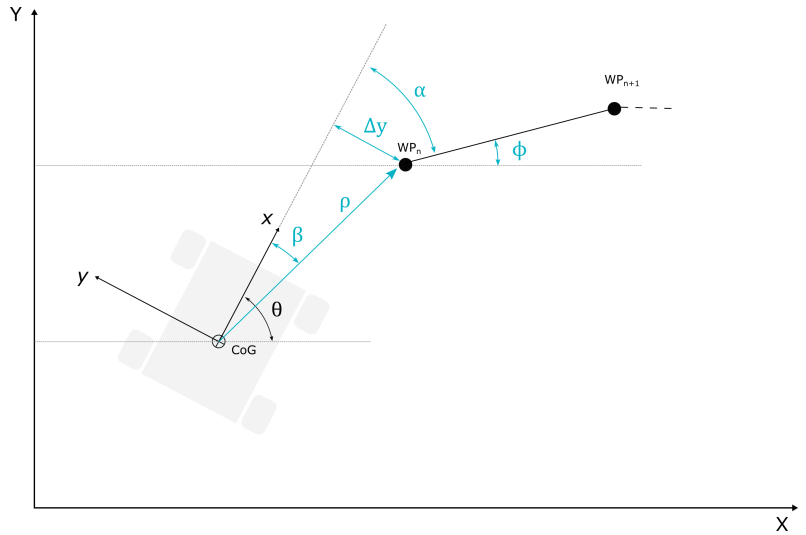

Figure 4.2. Parameters measured for each waypoint WP_n in relation to its subsequent waypoint WP_{n+1}

The following relationships are valid:

$$\begin{aligned} \beta &= tan^{-}1\left(\frac{\Delta Y}{\Delta X}\right) - \theta \\ \alpha &= \theta - \phi \\ \Delta y &= \rho sin\beta \end{aligned} \tag{4.1}$$

These definitions are now useful to define the three errors the control systems work with:

$$\begin{aligned} e_\rho &= \rho \\ e_y &= \Delta y \\ e_\alpha &= \alpha \end{aligned} \tag{4.2}$$

where e_y is used to orient the vehicle so that it reaches the next target in the shortest time possible, while e_α is used to align the vehicle to the trajectory. By only using the latter, the robot risks to travel parallel to the trajectory without never reaching it, while just by only using e_y the control becomes very unresponsive when the distance to WP_n is minimal. It is worth mentioning that it was decided to use Δy instead of an angular parameter because the latter, on the contrary, oscillates excessively as ρ lessen in magnitude.

Fig. 4.3 presents the Simulink implementation of the position error scheme, whose outputs are used differently depending on the type of controller employed. The scheme has two peculiarities:

- The *yawToTarget* block contains a custom function that evaluates β (Eq. 4.1) relying on the four-quadrant inverse tangent *atan2* function that avoids incurring in a discontinuity when evaluating the angle. In particular, while the *atan* function evaluates the angle in the range $[-\frac{\pi}{2}, +\frac{\pi}{2}]$, *atan2* does it in the range $[-\pi, +\pi]$, as shown in Fig. 4.4.

- e_α is controlled by a switch which zeroes its output whenever the robot is further that 0.4 meters from its next waypoint. This solution was implemented to help the robot approach the trajectory in the shortest time possible whenever it is disaligned.

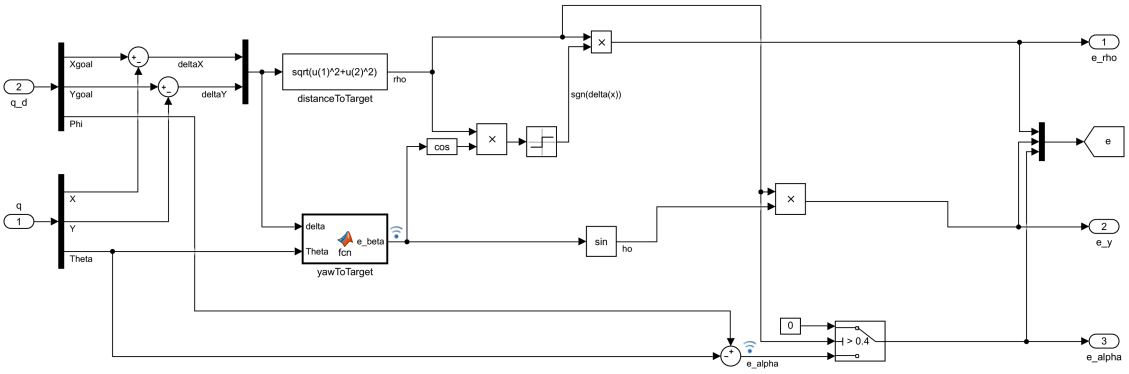

Figure 4.3. Simulink position error estimation scheme

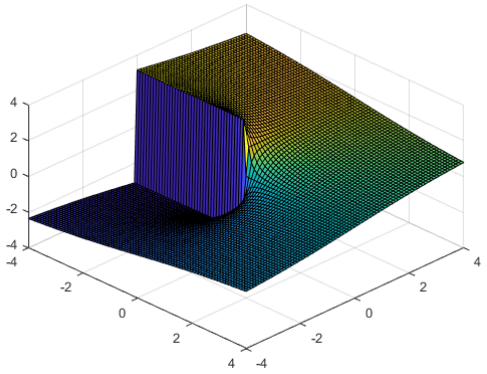

Figure 4.4. Matlab *atan2* function - Image from the MathWorks official website (https://www.mathworks.com/)

4.1.3 Sample Trajectories

The control systems later introduced were tested on four types of trajectories:

1. Straight line
2. Circumference
3. Sinusoidal trajectory
4. Generic trajectory (Fig. 4.5)

The generic trajectory, provided by the research team at the Shanghai Jiao Tong University, was empirically obtained by measuring the movement of a real Husky

rover in the same laboratory environment employed for this research. Its curves are therefore known to be practicable by the analysed rover.

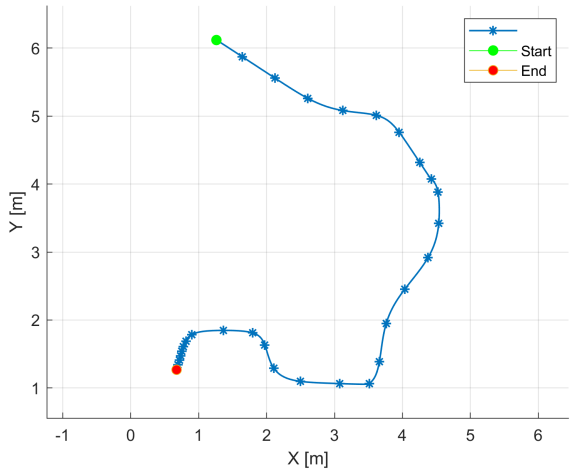

Figure 4.5. Generic trajectory used to test the trajectory tracking controllers

4.2 Controllers Integration with ROS

Whenever the control systems have to be tested on the real Husky rover, the mathematical model of the SSMR has to be replaced by the real vehicle. Therefore, the control inputs have to be commanded to the rover, and its sensors feedback has to be received and parsed. As introduced in Sec. 3.3, it is all possible thanks to ROS and the Matlab-Simulink-ROS integration via the MathWorks ROS Toolbox. The interface is based on the simplest ROS components: publishers and subscribers. In particular, the following were implemented:

- **Command velocity publisher**: It generates a message of type *geometry_msgs/Twist* sampled at 20 Hz. The message is then compiled via the *Bus Assignment* block which assigns the commands u_cmd and omega_cmd to the arguments *Linear.X* and *Angular.Z* . The message is then published on the */cmd_vel* topic (Fig. 4.6).

- **Odometry subscriber**: It received the output of the EKF via a message of type *nav_msgs/Odometry* sampled at 120Hz. The subscriber picks the message from the */odometry_filtered* topic and parses the results via a bus selector (4.7). The angular information, provided in quaternions, is transformed to Euler angles via the *quat2eul* function.

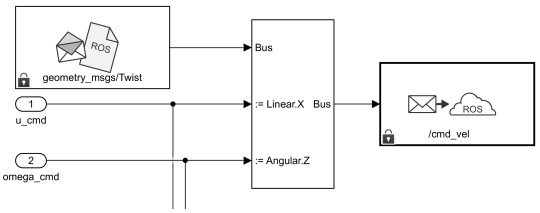

Figure 4.6. ROS Commands Publisher - Simulink

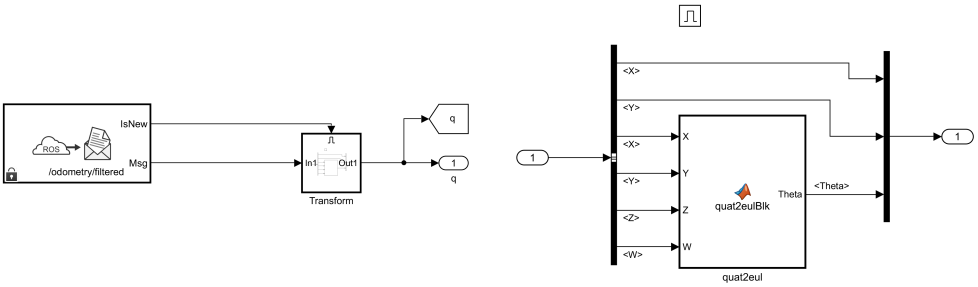

Figure 4.7. ROS Odometry Subscriber and Trasnformation Block - Simulink

- **RTF subscriber:** It receives the IR measurements in the global reference frame. Similarly to the odometry subscriber, it receives a *geometry_ msgs/PoseStamped* message sampled at 120Hz. The message is picked from the */vrpn_ client_ node/husky/pose* topic. Its angular information has to be converted from quaternions to Euler angles in the same way of the EKF output.

The correct functioning of the interface requires three expedients:

- **Real time implementation**: a *Simulation Pace Block* had to be added in order to run Simulink in real-time and output the commands at the correct frequency. The result was nearly exact, as the pace error over a 30 seconds test was in the order of 10^{-2} seconds.
- **RTS and EKF initialisation**: the RTS and EKF reference frames had to be aligned at the beginning of every test in order to compare the results easily. The RTS initial condition was subtracted to each subsequent reading, while the EKF was manually set to zero using the *set_ pose* service available in ROS.
- **RTS angular continuity**: the RTS has a 2π discontinuity between $-\pi$ and π. The discontinuity was removed by means of an algorithm developed within MATLAB.

4.3 PID Control

4.3.1 Introduction to PID Controllers

The PID control technique was first developed in the 20$^{\text{th}}$ century for automatic ship steering. The first theoretical formulation by [68] was published in 1922 and was successfully applied to U.S. Navy ships. The early mechanical implementation soon led to electronic analogue controllers that determined the success of PIDs in industrial applications. Their simplicity and straightforward implementation contributed to their popularity in many different fields, including aviation and robotics.

The underlying principle of PIDs is the regulation of a feedback signal, in particular of the error between the measured and the desired state. The PID gains adjust the control action $u_c(t)$ according to the properties of the error signal $e(t)$. The most basic form of PID used a simple proportional gain K_P which determines the control action as

$$u_c(t) = K_P e(t)$$

The adoption of such a simple control is limited by the possibility of incurring in steady-state error in response to a constant reference and a constant disturbance the controller is, by definition, unable to reject. Furthermore, an excessively large value of K_P might lead to instability, in particular with high-order systems.

In order to eliminate the steady state error the integral gain K_I is commonly adopted to support the proportional action. Its contribution is proportional - by a K_I factor - to the integral of the error from the initial time to the considered instant of time t:

$$u_c(t) = K_I \cdot \int_0^t e(t)dt$$

The integral contribution has the advantage of taking into account the past values of the error. For this reason, it is possible to have a control action different from zero, also when the error is zero. The addition of the integral term to a simple proportional controller improves the tracking of the steady-state response. The drawback of increasing K_I is a slower response for equal overshoot or a rise in overshoot for an unchanged response velocity. Furthermore, when an integral controller outputs a control action which is greater than the upper or lower saturation bounds of the actuator or motor that has to execute it, the system incurs in a phenomenon known as *windup*: the integral of the error keeps rising although the system is limited by its physical properties, resulting in a disproportional control action output by the integral control, thus altering the correct functioning of the control system. Such drawback can be easily overcome employing and *anti-windup*

device that limits the integral output whenever the controlled system reaches a saturation point.

Finally, a derivative term is added to improve the stability of the system by increasing the damping of the response thanks to an anticipatory behaviour. The rate of change of the signal is used so that its contribution to the total control action is proportional to the gain K_D:

$$u_c(t) = K_D \cdot \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

As previously mentioned in Sec. 3.4, the derivative contribution is susceptible to sensor noise which transfers to the error signal. Due to its nature, high-frequency noise, when derived, yields significantly high values, which, when multiplied by K_D, produce a disproportionate control signal that could offset the overall system stability. For this reason, derivative control is often coupled with a lowpass filter.

The sum of the three contibutrions yields the proportional-integral-derivative controller, a linear combination of the error, its integral and its derivative in time:

$$u_c(t) = K_p \cdot e(t) + K_I \cdot \int_0^t e(t)dt + K_D \cdot \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{4.3}$$

or, in the frequency domain:

$$U_C(s) = K_P + \frac{K_I}{s} + K_D \cdot s \tag{4.4}$$

4.3.2 PID Control System Architecture

The PID control system shown in Fig. 4.9 is composed of two cascade loops: an outer position control loop, and an inner velocity control loop. The whole system was first conceived in continuous time and then discretised as presented in Sec. 3.4. The position control loop was set to operate at 120 Hz, like the EKF, while the inner loop was limited to 20 Hz, as the Husky discards any command at higher frequencies. Furthermore, it is worth mentioning that, whenever an integral gain is employed, a clamping anti-windup mechanism is applied. Its logic structure is shown in Fig. 4.8 , and can be summarised as: it stops integration when the sum of the PID components exceeds the output limits and the integrator output and PID input have the same sign; it resumes integration when the sum of the PID components exceeds the output limits and the integrator output and PID input have opposite sign.

Position Control Loop

The position control loop converts position information to reference velocities, relying on the position feedback of the EKF, then processed by the position error

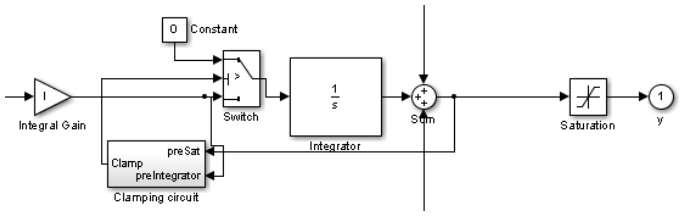

Figure 4.8. Integrator clamping logic - Simulink

block described in Sec. 4.1 and three PI controllers. The position error block takes as inputs the $q[X, Y, \theta]$ vector - the real position of the Husky - and the desired position vector $q_d[X_d, Y_d, \theta_d]$, which is output by the waypoint selector block. It then outputs the three errors e_ρ, e_y and e_α, which are fed to one proportional-integral block each, namely $\rho - PID$, $y - PID$ and $\alpha - PID$. In particular, the reference commands of the last two are summed to yield the reference command $\omega_{ref} = \omega_{ref1} + \omega_{ref2}$. Each PI block works in discrete time, includes a saturator, an anti-windup system and has the traditional parallel configuration (Tab. 4.1.

Controller	**Gains**	**Saturation Bounds**	**Input**	**Output**
$\rho - PID$	$K_{P\rho}, K_{I\rho}$	[+1, -1]	e_ρ	u_{ref}
$y - PID$	K_{Py}, K_{Iy}	[+inf, -inf]	e_y	ω_{ref1}
$\alpha - PID$	$K_{P\alpha}, K_{I\alpha}$	[+inf, -inf]	e_α	ω_{ref2}

Table 4.1. PID position control loop properties

The following equations are therefore valid:

$$u_{ref} = K_{P\rho} \cdot e_\rho + K_{I\rho} \cdot \int_0^t e_\rho \cdot dt$$
$$\omega_{ref} = \left[K_{Py} \cdot e_y + K_{Iy} \cdot \int_0^t e_y \cdot dt\right] + \left[K_{P\alpha} \cdot e_\alpha + K_{I\alpha} \cdot \int_0^t e_\alpha \cdot dt\right]$$

Velocity Control Loop

The velocity control loop receives as feedback the robot velocities in the local reference frame - $eta[dx, d\omega]$ - and subtracts it to the reference velocities generated by the position control loop. The resulting errors - e_u and e_ω - are processed by one PI block each, $u - PID$ and $\omega - PID$. Their outputs are then summed to two feedforward branches - one for u, FF_u, and one for *omega*, FF_ω - that multiply the reference signal by a static gain. This solution was adopted to increase

the responsiveness of the inner loop, which shall have a faster dynamic behaviour that the outer, and improve the overall performance. As in the previous case, the controllers properties are summarised in Tab. 4.2, where $u_{cmd} = u_{cmd1} + u_{cmd2}$ and $\omega_{cmd} = \omega_{cmd1} + \omega_{cmd2}$.

Controller	**Gains**	**Saturation Bounds**	**Input**	**Output**
$u - PID$	K_{Pu}, K_{Iu}	[+inf, -inf]	e_u	u_{cmd1}
$\omega - PID$	$K_{P\omega}, K_{I\omega}$	[+inf, -inf]	e_ω	ω_{cmd1}
FF_u	$K_{P_{FFu}}$	-	u_{ref}	u_{cmd2}
FF_ω	$K_{P_{FF\omega}}$	-	ω_{ref}	ω_{cmd2}

Table 4.2. PID velocity control loop properties

In mathematical terms:

$$u_{cmd} = [K_{Pu} \cdot e_u + K_{Iu} \cdot \int_0^t e_u \cdot dt] + K_{P_{FFu}} \cdot u_{ref}$$
$$\omega_{cmd} = [K_{P\omega} \cdot e_\omega + K_{I\omega} \cdot \int_0^t e_\omega \cdot dt] + K_{P_{FF\omega}} \cdot \omega_{ref}$$

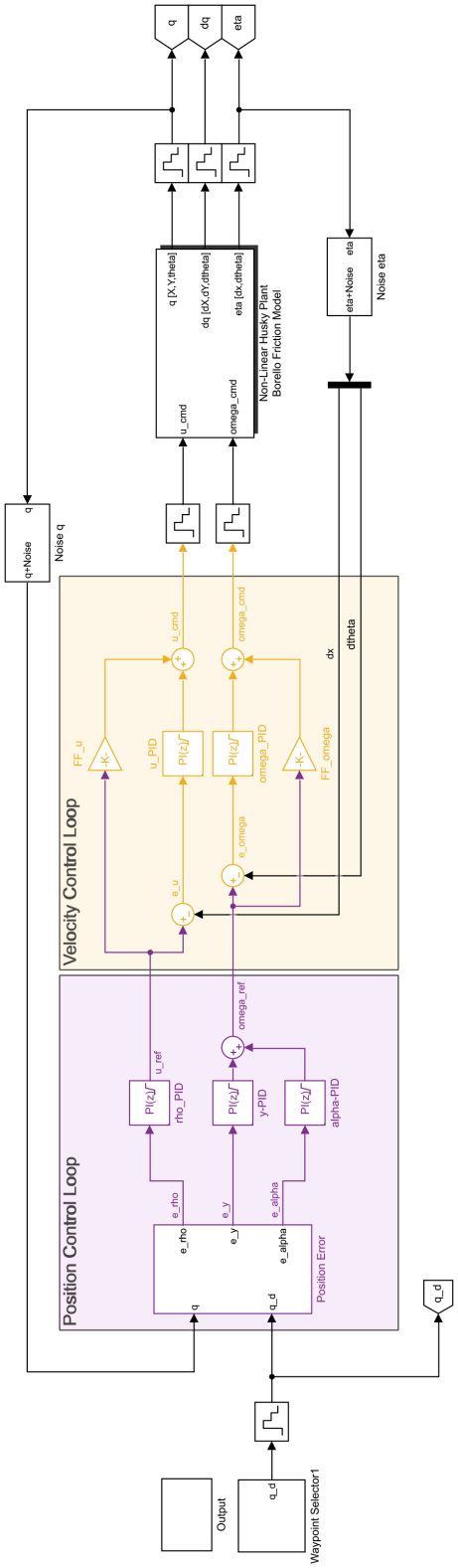

Figure 4.9. PID architecture for trajectory tracking - Simulation set-up

4.3.3 Tuning via Differential Evolution

As introduced in Chap. 1, the ten gains of the PID control system were tuned using the same DE algorithm employed for the determination of the unknown parameters of the mathematical models. Due to the increased complexity of the optimisation problem, the approach here employed varied slightly. In fact, before running the DE process, the linear model of the system was used to easily determine the search boundaries of each gain, therefore removing from the search space all the cases that would have led to system instability. Once such ranges were established, the optimisation was run on the nonlinear model with all the measured delays, sensors noise and external disturbances. The objective function had a similar structure to the one presented in Sec. 3.5 but for an extra parameter: the bending energy ratio introduced in Sec. 1.6. It was added to favour the results that better replicated the curve of the reference trajectory and that had the least oscillatory motion. The function therefore became:

$$f = K_{rms} \cdot d_{rms} + K_{haus} \cdot d_{haus} + K_{dratio} \cdot d_{ratio} + K_{ebratio} \cdot e_{bend-ratio}$$

where the gains were chosen for the balanced results they produced during the initial tests of the algorithm.

The DE optimiser compared the simulated trajectory to the reference trajectory, always using the generic trajectory in Sec. 4.1.3 as a test case. The other three trajectories were simply used to check the results in different scenarios. The most promising optimisation results were achieved using the parameters listed in Tab 4.3 and are presented in Tab. 4.4. It is worth noting that the feedforward gains were set manually before the optimisation process.

Parameter	Description	Value
F_{init}	F initial value	0.75
F_{bound}	F adaptation boundaries	[0.10,0.90]
CR_{init}	CR initial value	0.80
CR_{bound}	CR adaptation boundaries	[0.00,1.00]
τ_1	F adaptation parameter	0.10
τ_2	CR adaptation parameter	0.10
nr	Number of vectors used for mutation	4
slp	Gradient descent slope	0.20
df	Stop condition - relative improvement of the obj f.	0.001
NP	Population size	100
n_{gen}	Maximum number of generations	150
K_{rms}	Objective function weight for d_{rms}	100
K_{haus}	Objective function weight for d_{haus}	15
K_{dratio}	Objective function weight for d_{ratio}	10
$K_{ebratio}$	Objective function weight for $e_{bend-ratio}$	10
$steptime$	Simulink fixed step time	0.001 [s]

Table 4.3. PID tuning - jDE configuration parameters

Control Loop	Parameter	Search Range	Optimised Value	Final Value
Position	$K_{P\rho}$	[0,6]	2.836	=
	$K_{I\rho}$	[0,2]	0.115	=
	K_{Py}	[0,6]	5.983	=
	K_{Iy}	[0,2]	1.971	=
	$K_{P\alpha}$	[0,6]	4.507	=
	$K_{I\alpha}$	[0,2]	5.276e-04	=
Velocity	K_{Pu}	[0 2]	0.037	=
	K_{Iu}	[0 2]	0.477	=
	$K_{P\omega}$	[0 2]	0.650	=
	$K_{I\omega}$	[0 2]	0.548	=
	$K_{P_{FFu}}$	-	-	1
	$K_{P_{FF\omega}}$	-	-	0

Table 4.4. PID tuning- Gains obtained with DE plus the manually tuned feedforward gains

4.3.4 Simulink Results

Linear Trajectory - Case 1

Linear trajectory generated imposing an average linear velocity of 0.25 m/s.

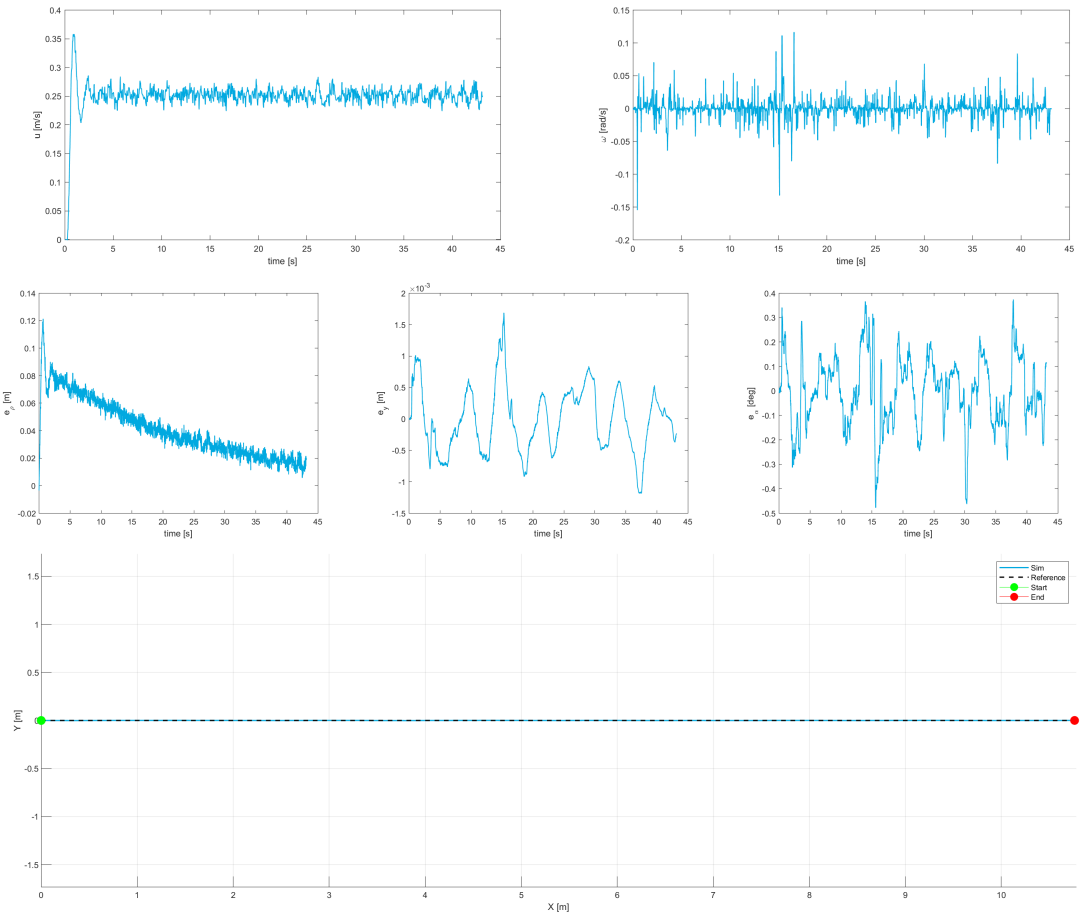

Figure 4.10. PID simulation results - Linear trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0167	0.0055	0.9985

Table 4.5. PID trajectory evaluation metrics - Linear trajectory at 0.25 m/s - Reference vs. Simulated

Linear Trajectory - Case 2

Linear trajectory generated imposing an average linear velocity of 0.50 m/s.

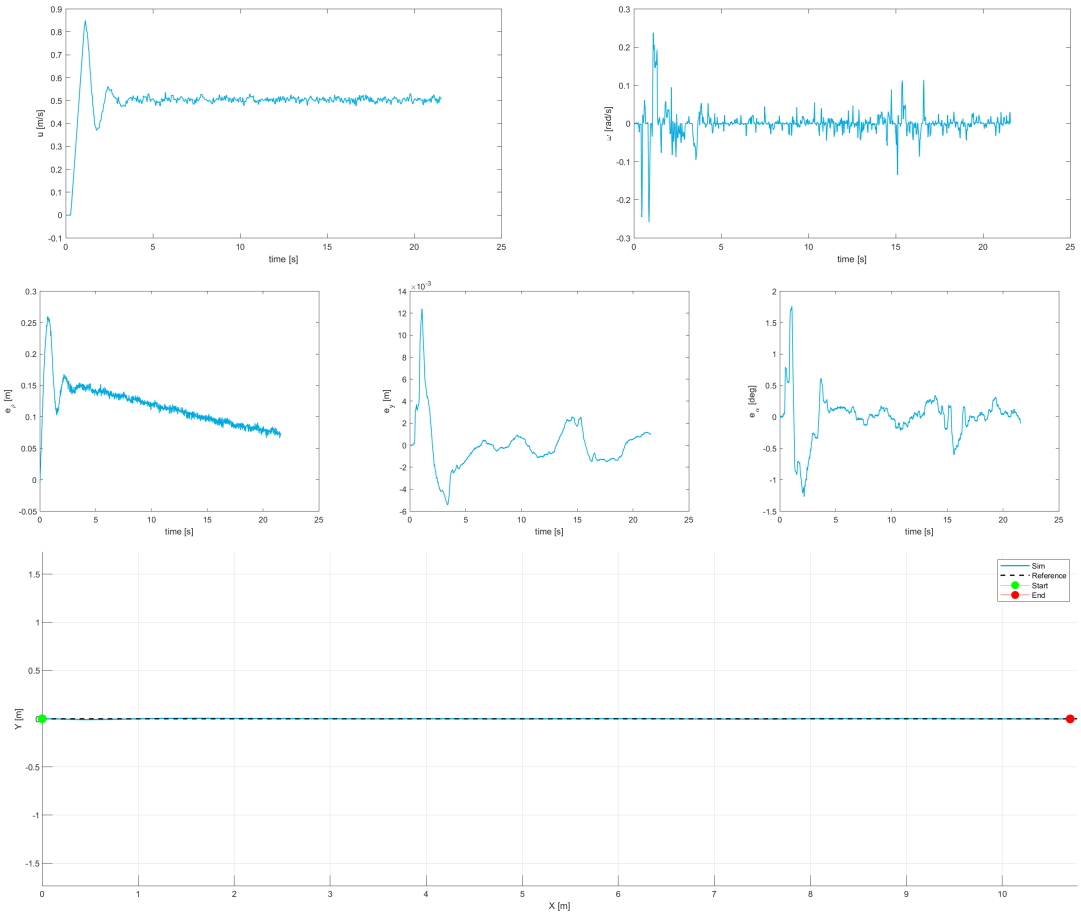

Figure 4.11. PID simulation results - Linear trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0729	0.0073	0.9933

Table 4.6. PID trajectory evaluation metrics - Linear trajectory at 0.25 m/s - Reference vs. Simulated

Circular Trajectory - Case 1

Circular trajectory generated imposing an average linear velocity of 0.25 m/s and a radius of 1.5 m.

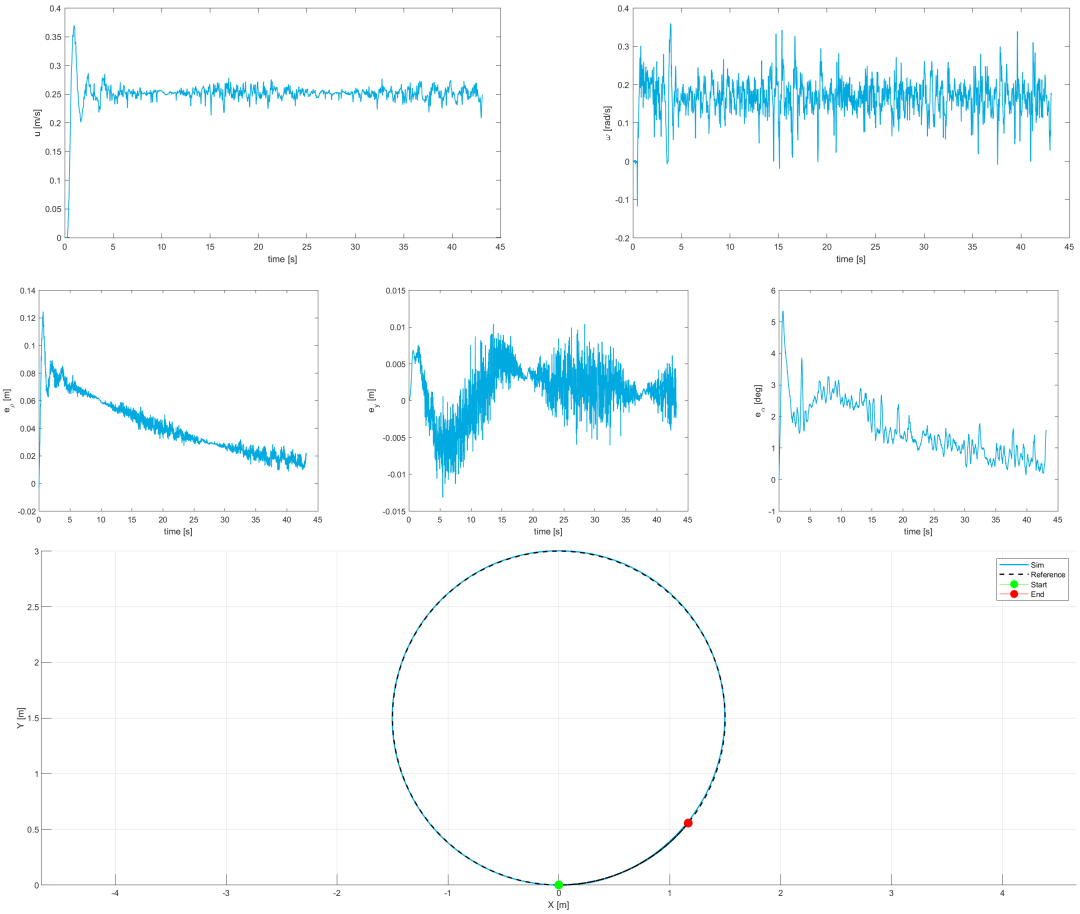

Figure 4.12. PID simulation results - Circular trajectory at 0.25 m/s and radius 1.5 m - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0130	0.0060	0.9987

Table 4.7. PID trajectory evaluation metrics - Circular trajectory at 0.25 m/s and radius 1.5 m - Reference vs. Simulated

Circular Trajectory - Case 2

Circular trajectory generated imposing an average linear velocity of 0.50 m/s and a radius of 1.5 m.

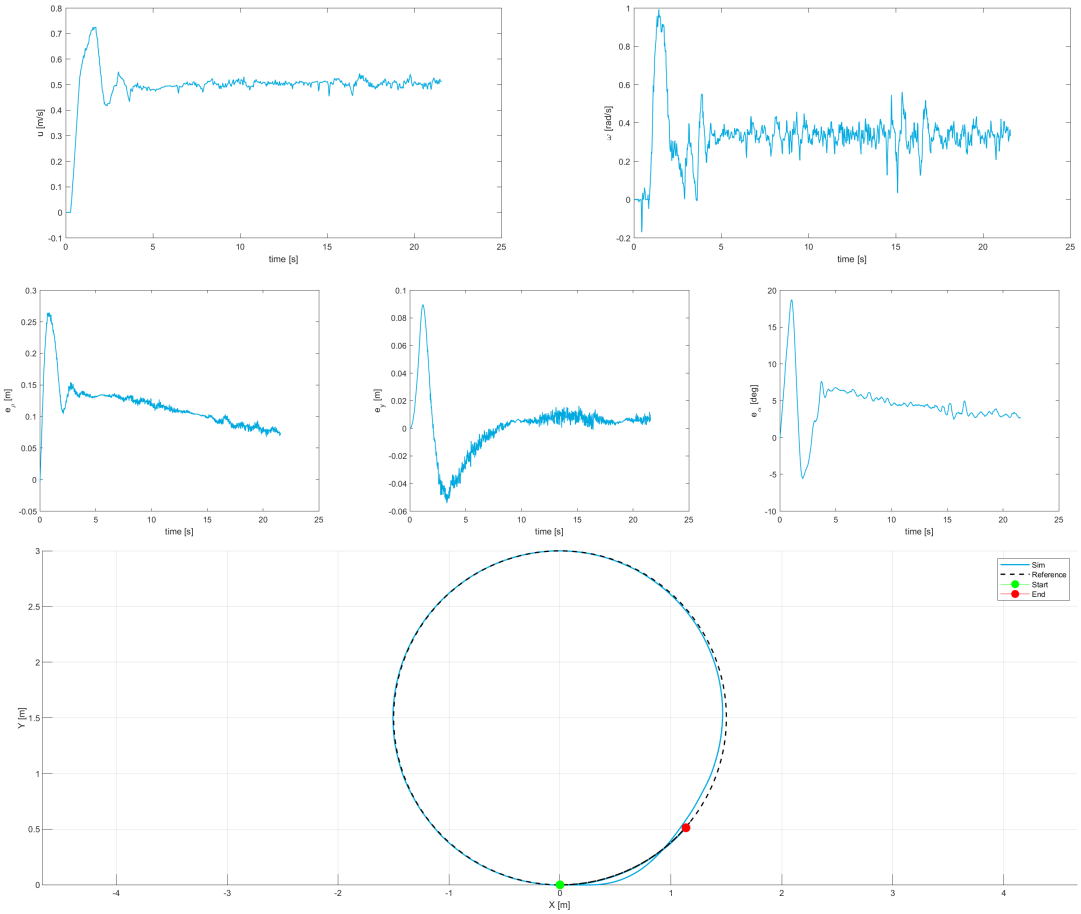

Figure 4.13. PID simulation results - Circular trajectory at 0.50 m/s and radius 1.5 m - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0130	0.0060	0.9987

Table 4.8. PID trajectory evaluation metrics - Circular trajectory at 0.50 m/s and radius 1.5 m - Reference vs. Simulated

Sinusoidal Trajectory - Case 1

Sinusoidal trajectory generated imposing an average linear velocity of 0.25 m/s.

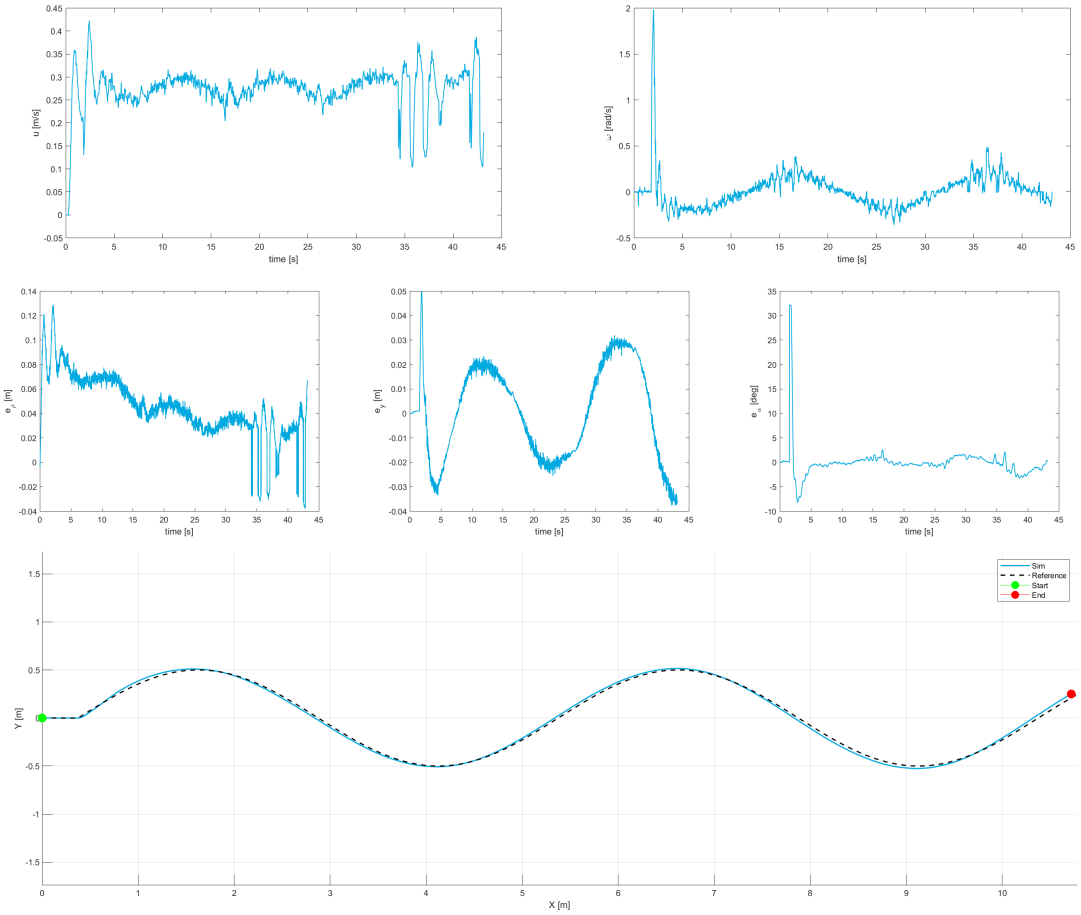

Figure 4.14. PID simulation results - Sinusoidal trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0620	0.0196	1.0003

Table 4.9. PID trajectory evaluation metrics - Sinusoidal trajectory at 0.25 m/s - Reference vs. Simulated

Sinusoidal Trajectory - Case 2

Sinusoidal trajectory generated imposing an average linear velocity of 0.50 m/s.

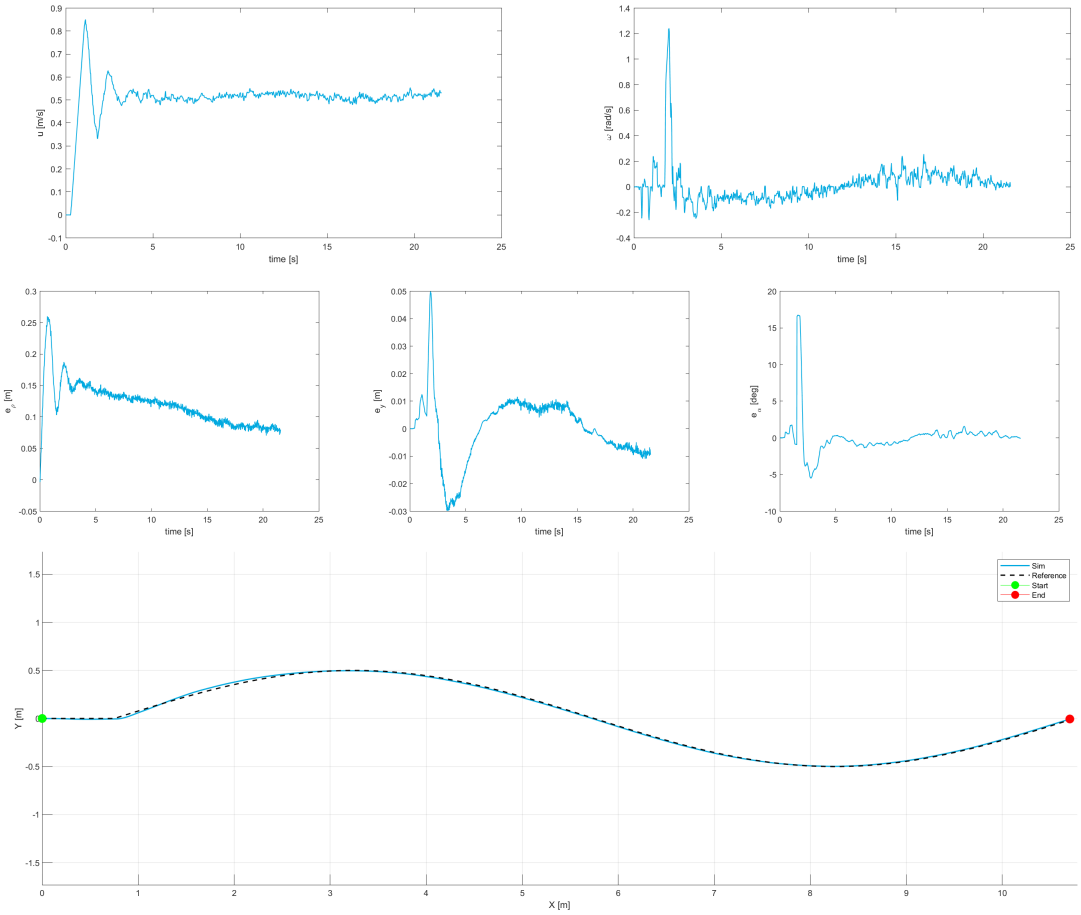

Figure 4.15. PID simulation results - Sinusoidal trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0782	0.0124	0.9935

Table 4.10. PID trajectory evaluation metrics - Sinusoidal trajectory at 0.50 m/s - Reference vs. Simulated

Generic Trajectory - Case 1

Generic trajectory generated imposing an average linear velocity of 0.25 m/s.

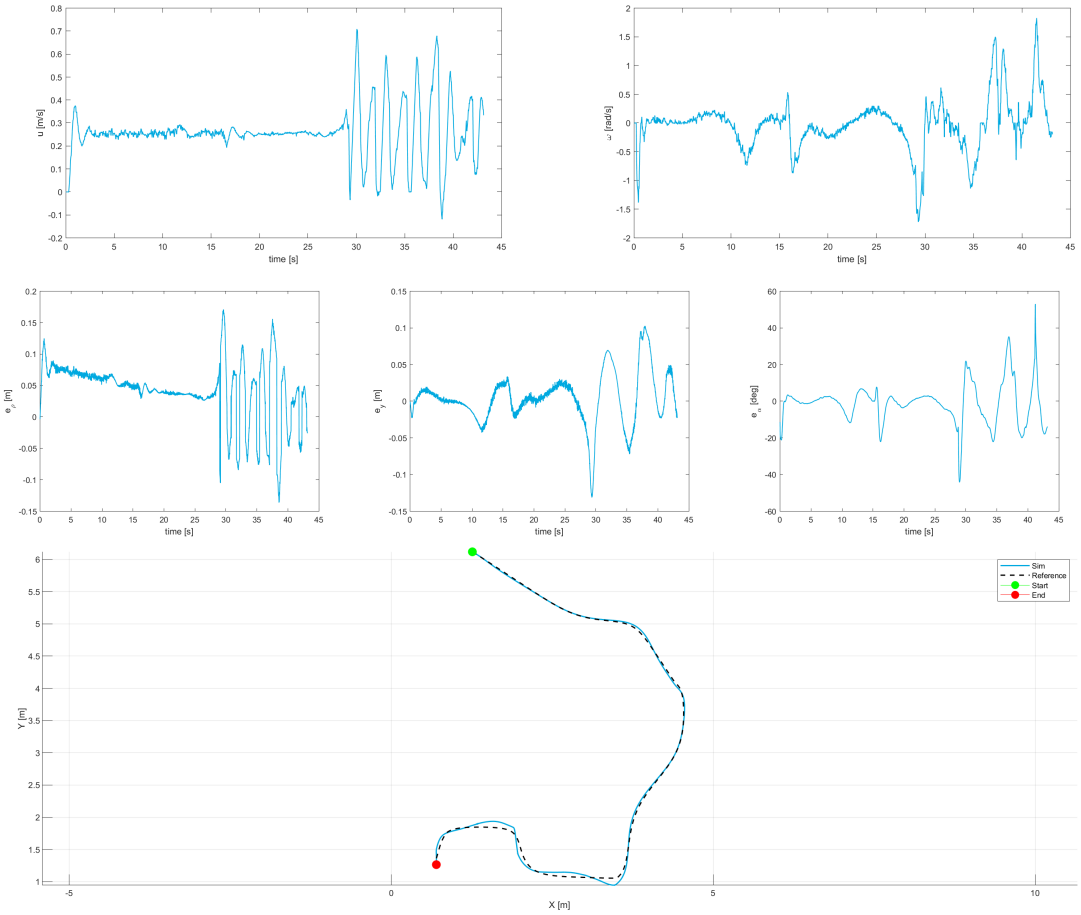

Figure 4.16. PID simulation results - Generic trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.1074	0.0331	1.0339

Table 4.11. PID trajectory evaluation metrics - Generic trajectory at 0.25 m/s - Reference vs. Simulated

Generic Trajectory - Case 2

Generic trajectory generated imposing an average linear velocity of 0.50 m/s.

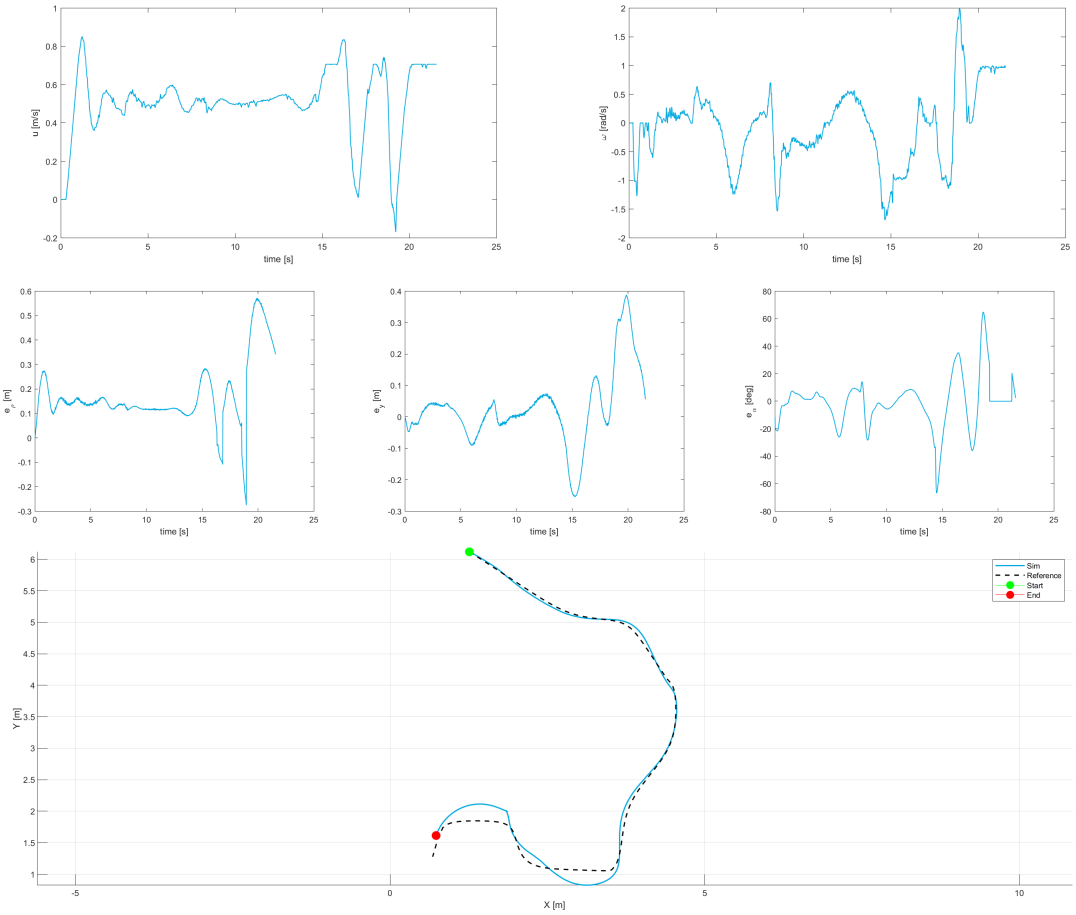

Figure 4.17. PID simulation results - Generic trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.3417	0.0959	1.0274

Table 4.12. PID trajectory evaluation metrics - Generic trajectory at 0.50 m/s - Reference vs. Simulated

4.3.5 Experimental Results

Linear Trajectory - Case 1

Linear trajectory generated imposing an average linear velocity of 0.25 m/s.

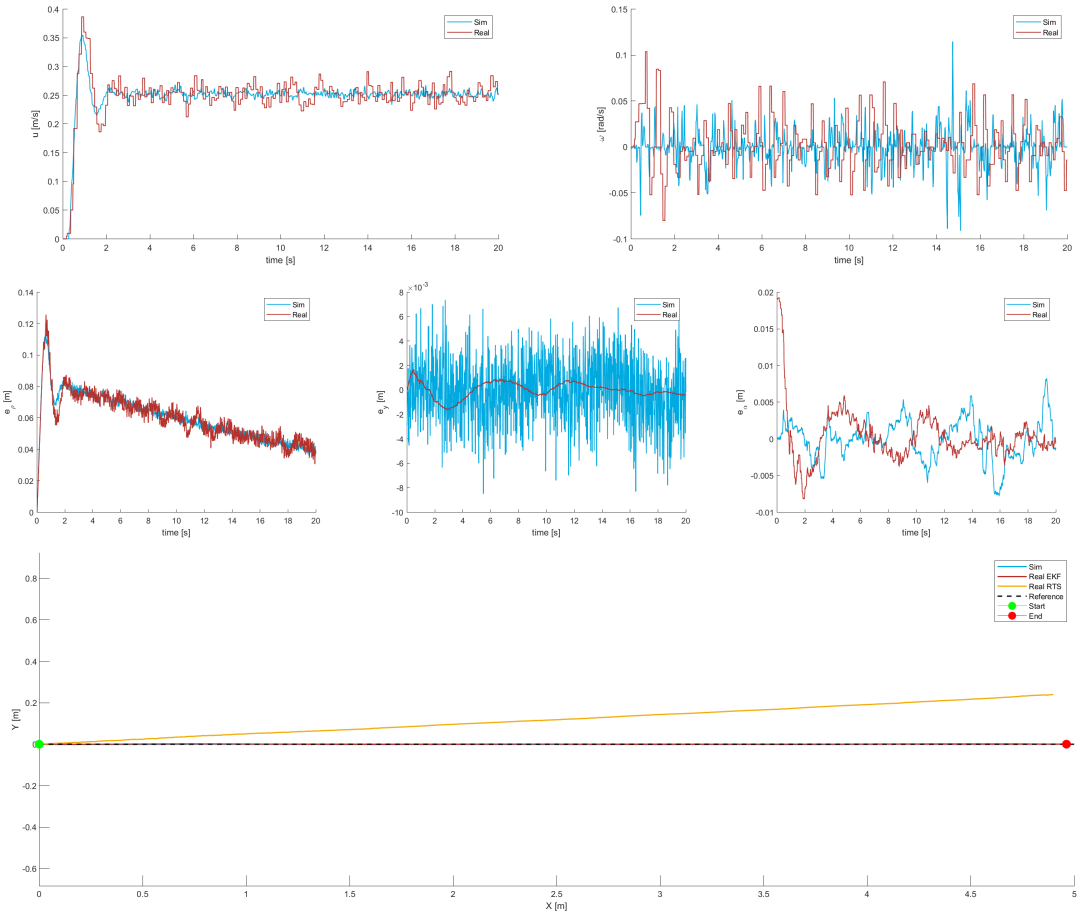

Figure 4.18. PID experimental results - Linear trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0033	0.0017	0.9995

Table 4.13. PID trajectory evaluation metrics - Linear trajectory at 0.25 m/s - Simulated vs. Real

Linear Trajectory - Case 2

Linear trajectory generated imposing an average linear velocity of 0.50 m/s.

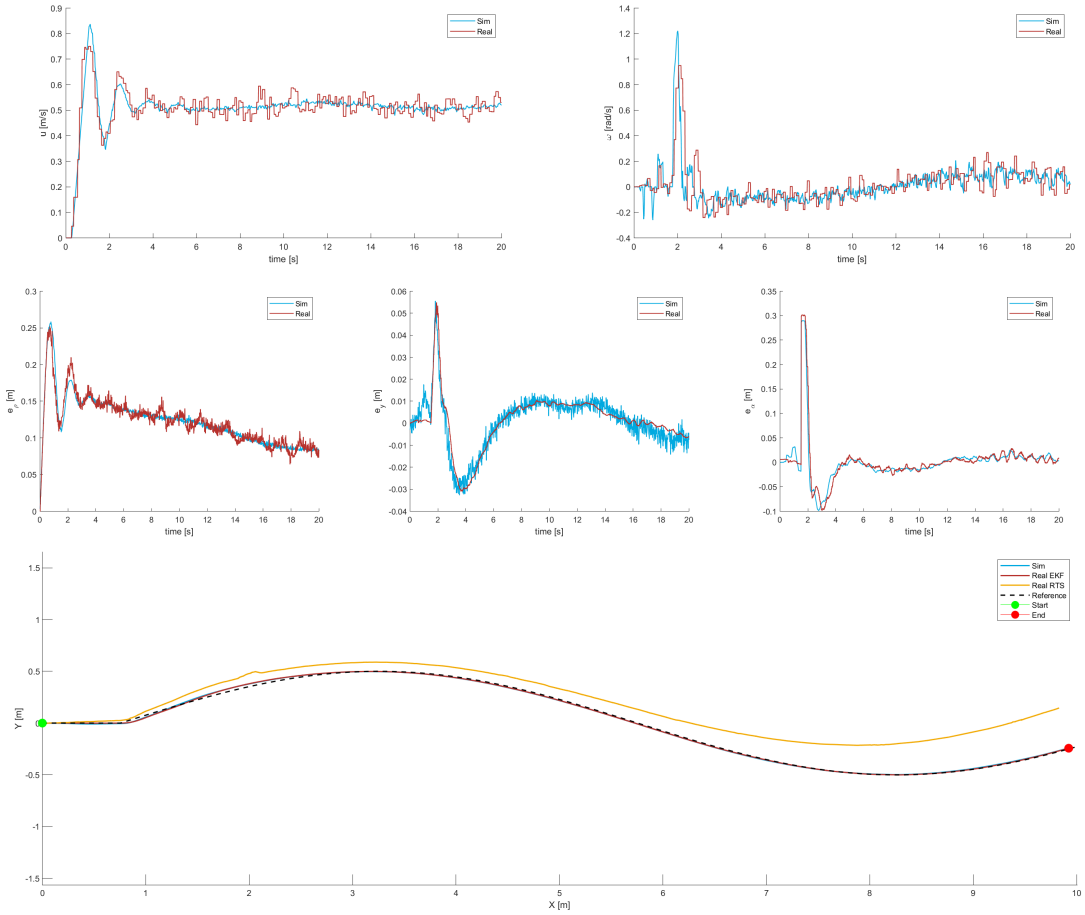

Figure 4.19. PID experimental results - Linear trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0108	0.0040	0.9994

Table 4.14. PID trajectory evaluation metrics - Linear trajectory at 0.50 m/s - Simulated vs. Real

Circular Trajectory - Case 1

Circular trajectory generated imposing an average linear velocity of 0.25 m/s and a radius of 1.5 m.

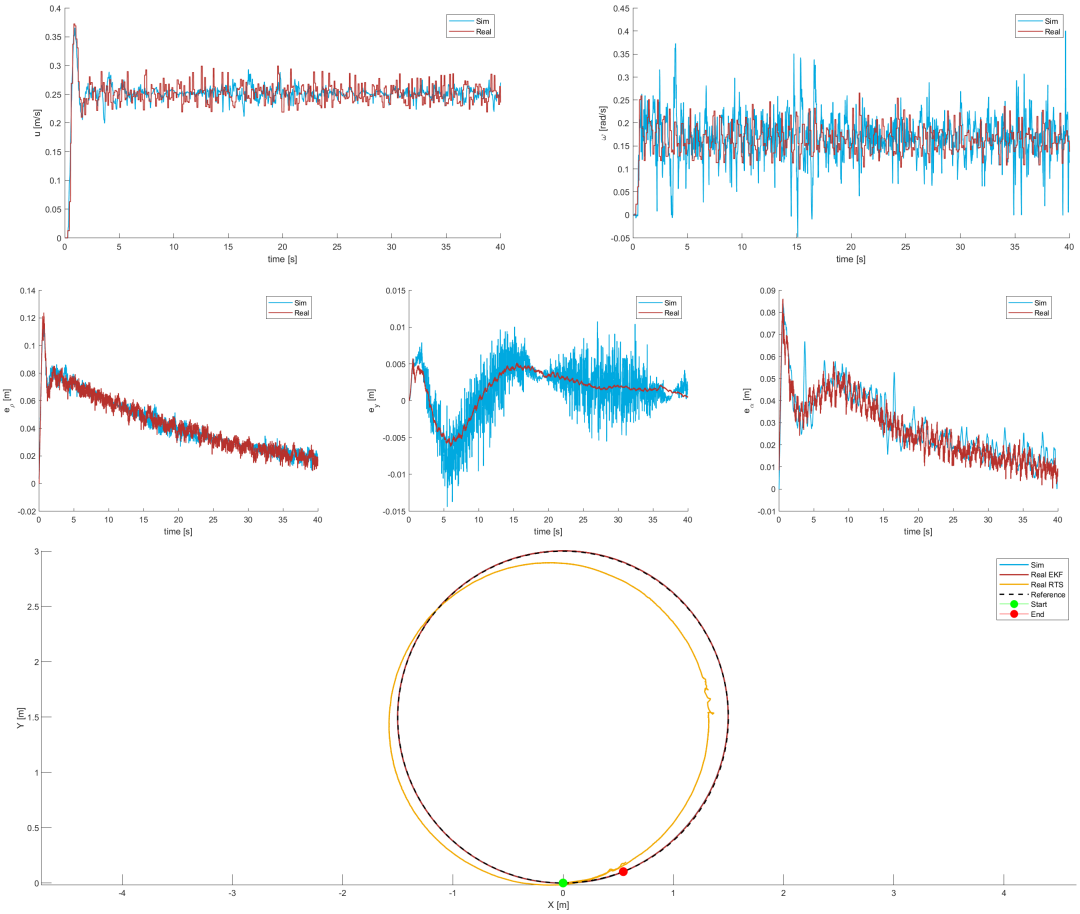

Figure 4.20. PID experimental results - Circular trajectory at 0.25 m/s and radius 1.5 m- From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0031	0.0016	0.9994

Table 4.15. PID trajectory evaluation metrics - Circular trajectory at 0.25 m/s and radius 1.5 m - Simulated vs. Real

Circular Trajectory - Case 2

Circular trajectory generated imposing an average linear velocity of 0.50 m/s and a radius of 1.5 m.

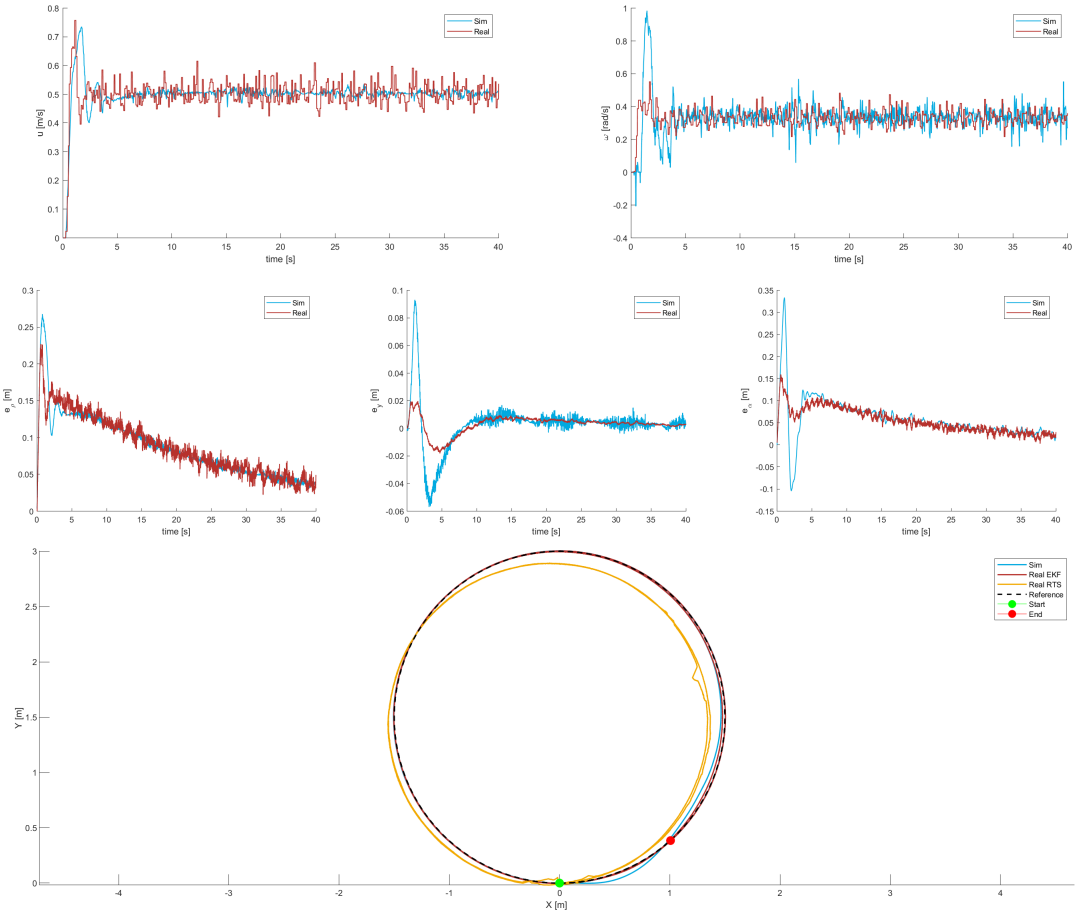

Figure 4.21. PID experimental results - Circular trajectory at 0.50 m/s and radius 1.5 m- From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0231	0.0056	1.0006

Table 4.16. PID trajectory evaluation metrics - Circular trajectory at 0.50 m/s and radius 1.5 m - Simulated vs. Real

Sinusoidal Trajectory - Case 1

Sinusoidal trajectory generated imposing an average linear velocity of 0.25 m/s.

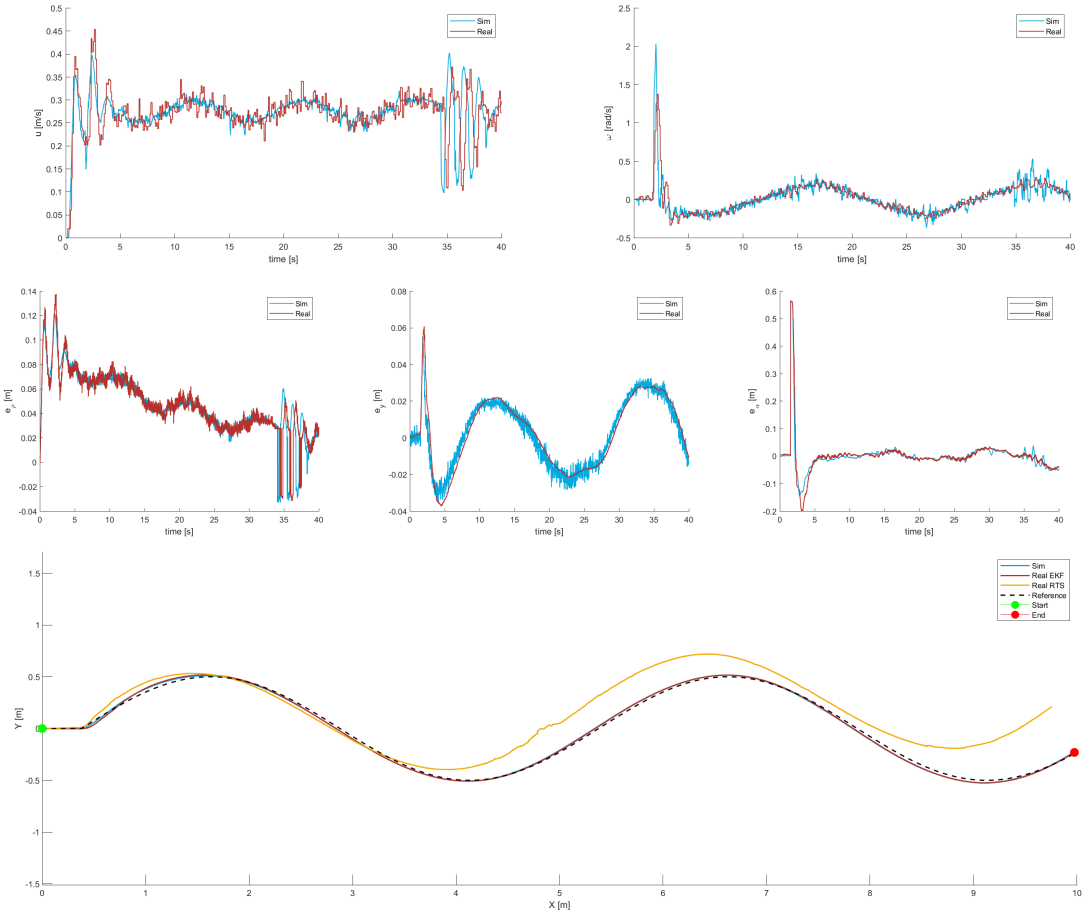

Figure 4.22. PID experimental results - Sinusoidal trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0192	0.0048	0.9988

Table 4.17. PID trajectory evaluation metrics - Sinusoidal trajectory at 0.25 m/s - Simulated vs. Real

Generic Trajectory - Case 1

Generic trajectory generated imposing an average linear velocity of 0.25 m/s.

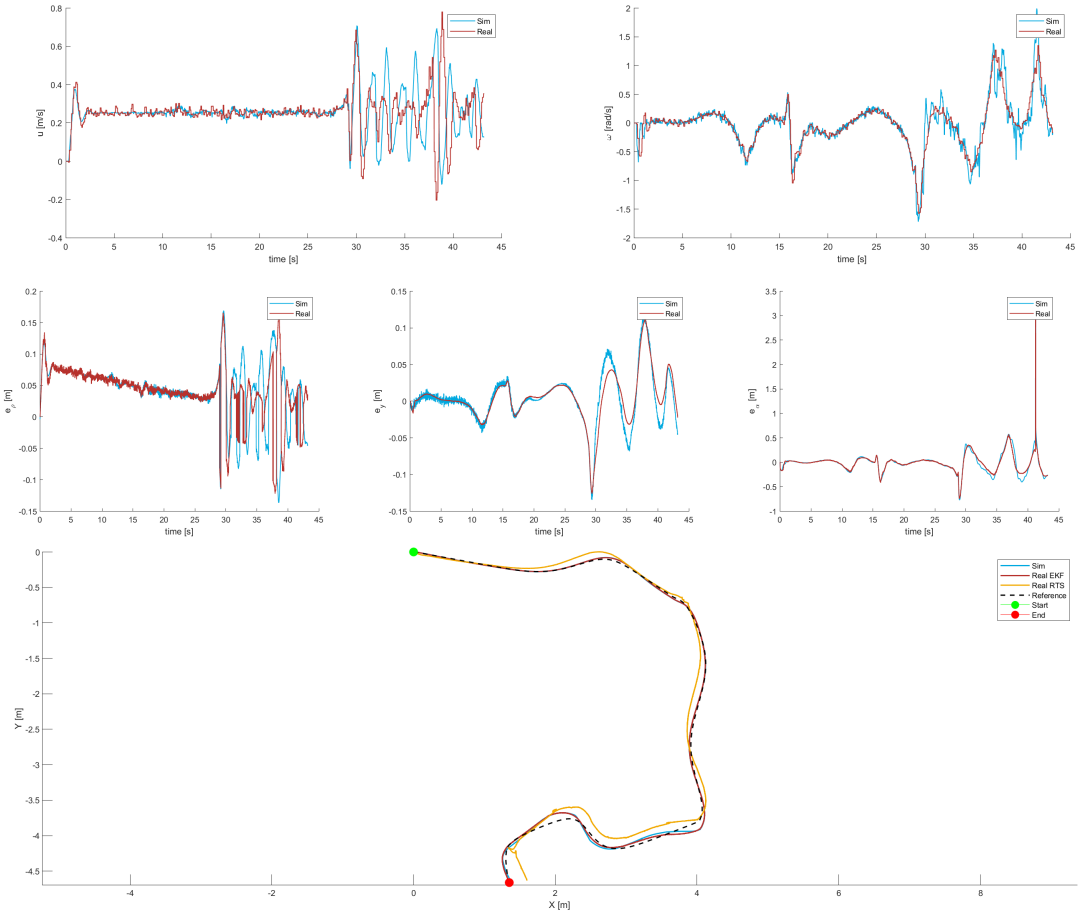

Figure 4.23. PID experimental results - Generic trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0392	0.0118	0.9832

Table 4.18. PID trajectory evaluation metrics - Generic trajectory at 0.25 m/s - Simulated vs. Real

Generic Trajectory - Case 2

Generic trajectory generated imposing an average linear velocity of 0.50 m/s.

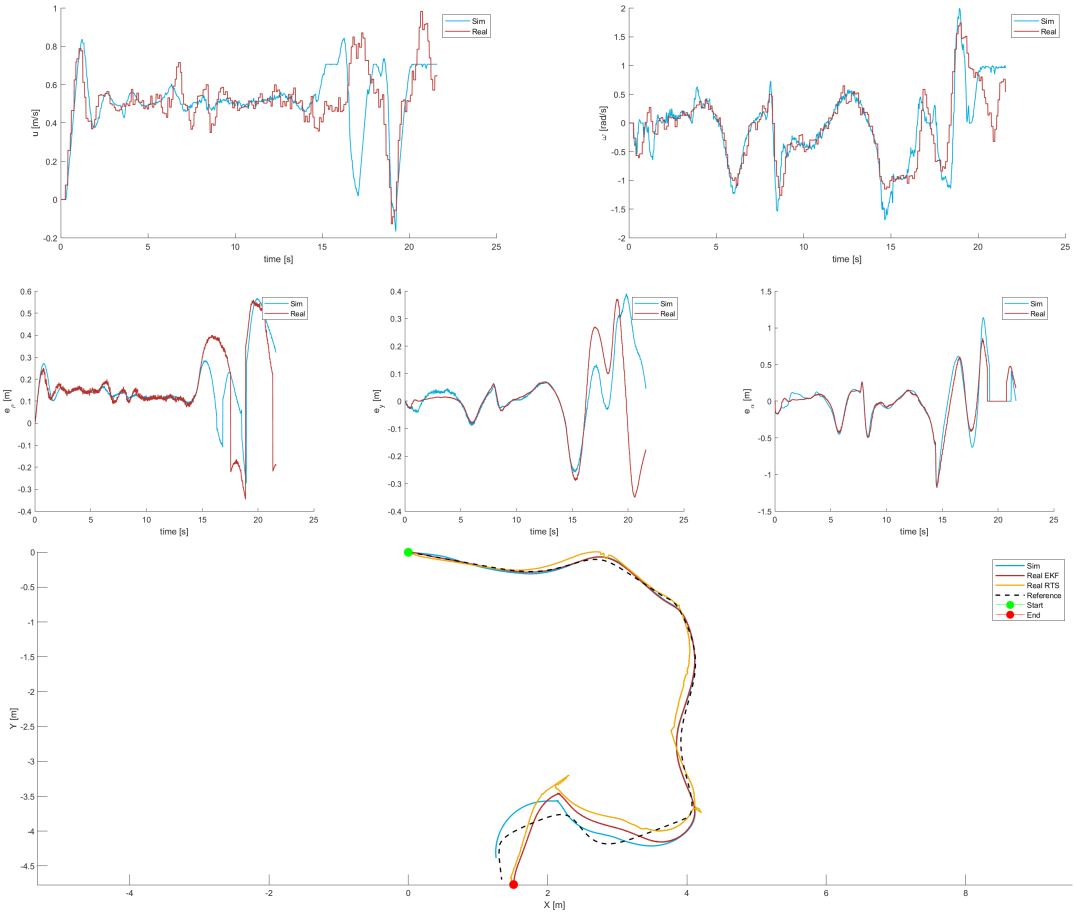

Figure 4.24. PID experimental results - Generic trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.4569	0.1014	0.9918

Table 4.19. PID trajectory evaluation metrics - Generic trajectory at 0.50 m/s - Simulated vs. Real

4.4 LQR Controller

4.4.1 Introduction to LQR Controllers

The Linear Quadratic Regulator (LQR), an important part of the solution of the Linear-Quadratic-Gaussian (LQG) problem, is a type of optimal control which operates a dynamic system at minimum cost whenever the system is described by a set of linear differential equations and the cost function is described by a quadratic function. As the definition suggests, the LQR, like optimal control systems in general, are designed to minimise the performance indices of the system. Therefore, the performance indices - or figures of merit - are the only design parameters which require to be set up. Unlike PIDs, the design of an optimal controller does not require to tune parameters which don't have any physical meaning, allowing for an automated design procedure.

For the derivation of the LQR, it is assumed the plant is written in state-space form $\dot{x} = Ax + Bu$, and that all of the n states are available for the controller, a case defined as *full-state* feedback. The feedback gain is a matrix K, implemented as $u = -K(x - x_{desired})$. The system dynamics are then written as:

$$\dot{x} = (A - BK)x + BKx_{desired}$$

where $x_{desired}$ represents the vector of desired states and serves as the external input to the closed-loop system. The A-matrix of the closed-loop system is $(A-BK)$ and the B-matrix of the closed-loop is BK. It is now necessary to introduce a general procedure for solving optimal control problems which will then be specialised for the LQR.

The problem for a fixed end time t_f is stated as:

$$\begin{gathered}\text{choose u(t) to minimise } J = \psi(x(t_f)) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt \\ \text{subject to } \dot{x} = f(x(t), u(t), t) \\ x(t_0) = x_0\end{gathered}$$

where $\psi(x(t_f), t_f)$ is the *terminal cost*; the total cost J is a sum of the terminal cost and an integral along the way. It is assumed that $L(x(t), u(t), t)$ is nonnegative. The first step is to augment the cost using a costate vector $\lambda(t)$:

$$\overline{J} = \psi(x(t_f)) + \int_{t_0}^{t_f} (L + \lambda^T(f - \dot{x}))dt$$

where $\lambda(t)$ can be anything desired, as it multiplies $f - \dot{x} = 0$. Along the optimum trajectory, variations in J and hence $\overline{J}$ should vanish. This follows from the fact that J is chosen to be continuous in x, u, and t. The variation can be written as:

$$\delta\overline{J} = \psi_x \delta x(t_f) + \int_{t_0}^{t_f} \left[L_x \delta x + L_u \delta u + \lambda^T f_x \delta x + \lambda^t f_u \delta u - \lambda^T \delta \dot{x} \right] dt$$

where subscripts denote partial derivatives. The last term can be evaluated using integration by parts as

$$-\int_{t_0}^{t_f} \lambda^T \delta \dot{x} dt = -\lambda^T(t_f)\delta x(t_f) + \lambda^T(t_0)\delta x(t_0) + \int_{t_0}^{t_f} \dot{\lambda^T} \delta x dt$$

and thus

$$\begin{aligned}\delta\overline{J} =& \psi_x(x(t_f)\delta x(t_f)) + \int_{t_0}^{t_f} (L_u + \lambda^T f_u)\delta u dt + \\ & \int_{t_0}^{t_f} (L_x + \lambda^T f_x + \dot{\lambda}^T)\delta x dt - \lambda^T(t_f)\delta x(t_f) + \lambda^T(t_0)\delta x(t_0)\end{aligned}$$

The last term is zero, since it is not possible to change the intial condition of the state by varying something later in time: it is a fixed value. This way of writing $\overline{J}$ makes it clear that there are three components of the variation that must independently be zero, since it possible to vary any of x, u or $x(t_f)$:

$$\begin{aligned} L_u + \lambda^T f_u &= 0 \\ L_x + \lambda^T f_x + \dot{\lambda}^T &= 0 \\ \psi_x(x(t_f)) - \lambda^T(t_f) &= 0 \end{aligned}$$

The second and third requirements are met by explicitly setting

$$\begin{aligned} \dot{\lambda}^T &= -L_x - \lambda^T f_x \\ \lambda_T(t_f) &= \psi_x(x(t_f)) \end{aligned}$$

The evolution of λ is given in *reverse time*, from a final state to the initial. Hence it is clear the primary difficulty of solving optimal control problems: the state propagates forward in time, while the costate propagates backwards. The state and the costate are coordinated through the above equations. Many solutions rely on numerical methods, where the simplest approach is the gradient method. In the case of the LQR with zero terminal cost, it is possible to set $\psi = 0$ and

$$L = \frac{1}{2} x^T Q x + \frac{1}{2} u^T R u \tag{4.5}$$

where the requirement that $L \geq 0$ implies that both Q and R are positive definite. In the case of linear plant dyamics, it is possible to write:

$$
\begin{aligned}
L_x &= x^T Q \\
L_u &= u^T R \\
f_x &= A \\
f_u &= B
\end{aligned}
$$

so that

$$
\begin{aligned}
\dot{x} &= Ax + Bu \\
x(t_0) &= x_0 \\
\dot{\lambda} &= -Qx - A^T \lambda \\
\lambda(t_f) &= 0 \\
Ru + B^T \lambda &= 0
\end{aligned}
$$

Since the system is linear, it is possible to say that $\lambda = Px$. Inserting this into the $\dot{\lambda}$ equation, and then using the $\dot{x}$ equation, and a substitution for u, then:

$$PAx + A^T Px + Qx - PBR^{-1}B^T Px + \dot{P} = 0$$

This has to hold for all x, in fact it a matrix equation, the *matrix Riccati equation.* The steady state solution satisfies:

$$PA + A^T P + Q - PBR^{-1}B^T P = 0$$

This equation is the *matrix algebraic Riccati Equation* (MARE), whose solution P is needed to compute the optimal feedback gain K. The MARE is easily solved by standard numerical tools in linear algebra. Furthermore, the equation $Ru+B^T\lambda = 0$ gives the feedback law

$$u = -R^{-1}B^T Px$$

Referring to Sec. 2.3, where the state vector $x = [x, \dot{x}, \theta, \dot{\theta}]^T$ and the command vector $u = [v_x, \omega_z]^T$ are introduced, it is possible to say that the matrices of the cost function shown in Eq. 4.5 are of the form:

$$
Q = \begin{bmatrix}
Q_{11} & 0 & 0 & 0 \\
0 & Q_{22} & 0 & 0 \\
0 & 0 & Q_{33} & 0 \\
0 & 0 & 0 & Q_{44}
\end{bmatrix}
$$

$$
R = \begin{bmatrix}
R_{11} & 0 \\
0 & R_{22}
\end{bmatrix}
$$

where the elements of Q represent the how much the error of the respective element of the state vector has to be reduced, and the elements of R represent the actuation effort: the higher the value, the more responsiveness - and therefore power - the system will require from the actuators.

4.4.2 LQR Control System Architecture

The LQR controller in Fig. 4.26, in contrast the PID here presented, is based on a single control loop. The system was first conceived in continuous time and then discretised as presented in Sec. 3.4. The position vector $q[X, Y, \theta]$ is fed to the already introduced position error block which, jointly, received the reference waypoint from the waypoint selector block. The position error block now only has two outputs: e_{rho} and e_θ. Its mode of operation is identical to what has already been introduced, but the y error and the α error are combined before the output:

$$e_\theta = K_\alpha \cdot e_\alpha + K_y \cdot e_y$$

K_y and K_α were introduced to allow for greater flexibility during the design process.

The error e_ρ follows two parallel branches, used to sum itself to the output of an integral block (with an anti-windup mechanism), necessary the eliminate any steady state error. The state variables related to the vehicle speed - $\dot{x}$ and $\dot{\theta}$ - are, instead, directly fed to the LQR block. Within the LQR block, the input state vector is multiplied by the optimal gain matrix K previously evaluated in Matlab, and its ouputs are saturated to avoid exceding the vehicle maximum input limits (Fig.4.25). Due to its structure, this LQR control system relies on the following parameters:

- K_α - e_α weight
- K_y - e_y error weight
- K_{I_ρ} - Integral action gain acting on e_ρ
- K optimal feedback gain matrix based on the Q and R matrices of the LQR

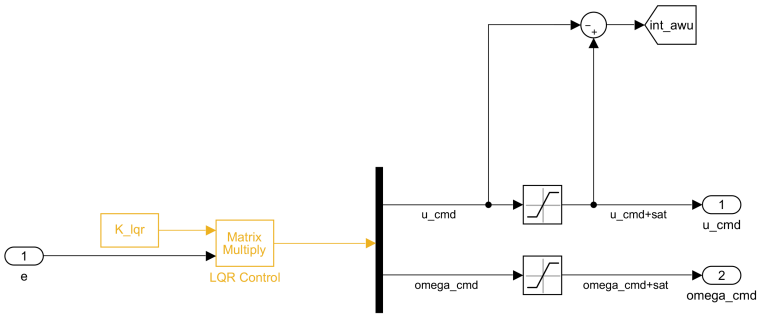

Figure 4.25. LQR controller inner structure - Simulink

4.4.3 Tuning via Differential Evolution

Despite being an optimal control, thus based on an array of physics-related parameters, LQR too needs to be tuned. As with the previously introduced PID, the same DE algorithms employed for unknown model parameters estimation were adapted to find an optimal configuration of the controller. In this case, the parameters which had to be found were the six matrix elements of Q and R; the other three gains shown in the previous section were set manually to allow for some fine-tuning. The optimisation algorithm was modified so that it used the state-space description of the Husky rover to find the optimal gain matrix K supposing a set of Q and R indices. The K matrix was then tested with the nonlinear model with the improved friction formulation. The objective function used to compare the results was:

$$f = K_{rms} \cdot d_{rms} + K_{haus} \cdot d_{haus} + K_{dratio} \cdot d_{ratio} + K_{ebratio} \cdot e_{bend-ratio} + K_{u_{RMS}} \cdot u_{RMS}$$

where $u_{RMS} = RMS(\dot{x})/u_{ref}$ and u_{ref} is the average speed necessary to complete the reference trajectory. This parameter was introduced to penalise results which showed oscillatory behaviours in the $\dot{x}$ variable. As for the PID, the gains were chosen for the balanced results they produced during the initial tests of the algorithm.

Tab. 4.20 sums up the parameters used to run the DE algorithm, while Tab. 4.21 shows all the parameters necessary to run the controller, including the ones found via DE. It is worth remembering that the results later presented were measured by employing the **manually tuned paramters**, as it was not possible to experimentally test the ones found via optimisation.

Parameter	Description	Value
F_{init}	F initial value	0.75
F_{bound}	F adaptation boundaries	[0.10,0.90]
CR_{init}	CR initial value	0.80
CR_{bound}	CR adaptation boundaries	[0.00,1.00]
τ_1	F adaptation parameter	0.10
τ_2	CR adaptation parameter	0.10
nr	Number of vectors used for mutation	4
slp	Gradient descent slope	0.20
df	Stop condition - relative improvement of the obj f.	0.001
NP	Population size	50
n_{gen}	Maximum number of generations	300
K_{rms}	Objective function weight for d_{rms}	100
K_{haus}	Objective function weight for d_{haus}	15
K_{dratio}	Objective function weight for d_{ratio}	10
$K_{ebratio}$	Objective function weight for $e_{bend-ratio}$	10
$K_{u_{RMS}}$	Objective function weight for u_{RMS}	50
$steptime$	Simulink fixed step time	0.01 [s]

Table 4.20. LQR tuning - jDE configuration parameters

Parameter	Search Bounds	Optimised Value	Manual Tuning Value
Q_{11}	[2.00 , 5.00]	4.06	1.80
Q_{22}	[0.00 , 0.05]	0.02	0.30
Q_{33}	[5.00 , 15.00]	10.48	3.00
Q_{44}	[0.00 , 0.10]	0.00	0.80
R_{11}	[0.30 , 0.70]	0.44	0.50
R_{22}	[0.40 , 0.90]	0.76	0.60
K_α	-	1.000	0.90
K_y	-	0.900	1.00
K_{I_ρ}	-	0.01	0.01

Table 4.21. LQR tuning- Parameters obtained with DE plus the manually tuned parameters

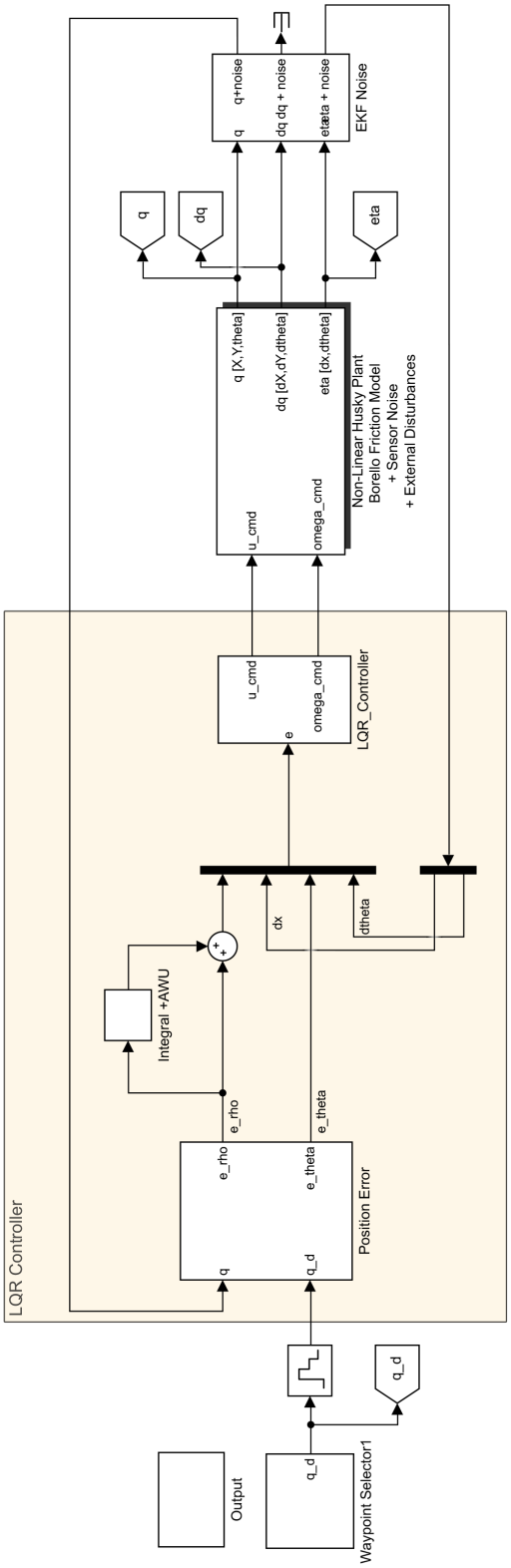

Figure 4.26. LQR controller architecture - Simulation set-up

4.4.4 Simulink Results

Linear Trajectory - Case 1

Linear trajectory generated imposing an average linear velocity of 0.25 m/s.

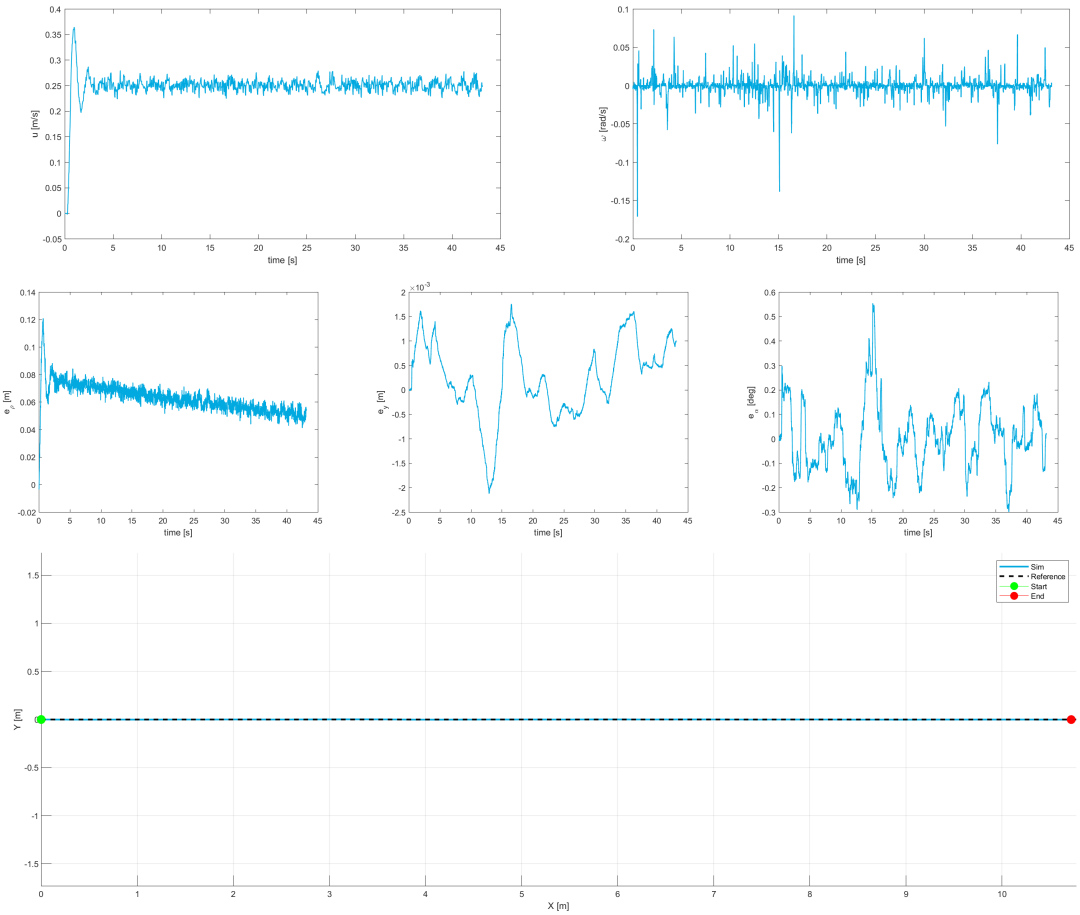

Figure 4.27. LQR simulation results - Linear trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0516	0.0073	0.9952

Table 4.22. LQR trajectory evaluation metrics - Linear trajectory at 0.25 m/s - Reference vs. Simulated

Linear Trajectory - Case 2

Linear trajectory generated imposing an average linear velocity of 0.50 m/s.

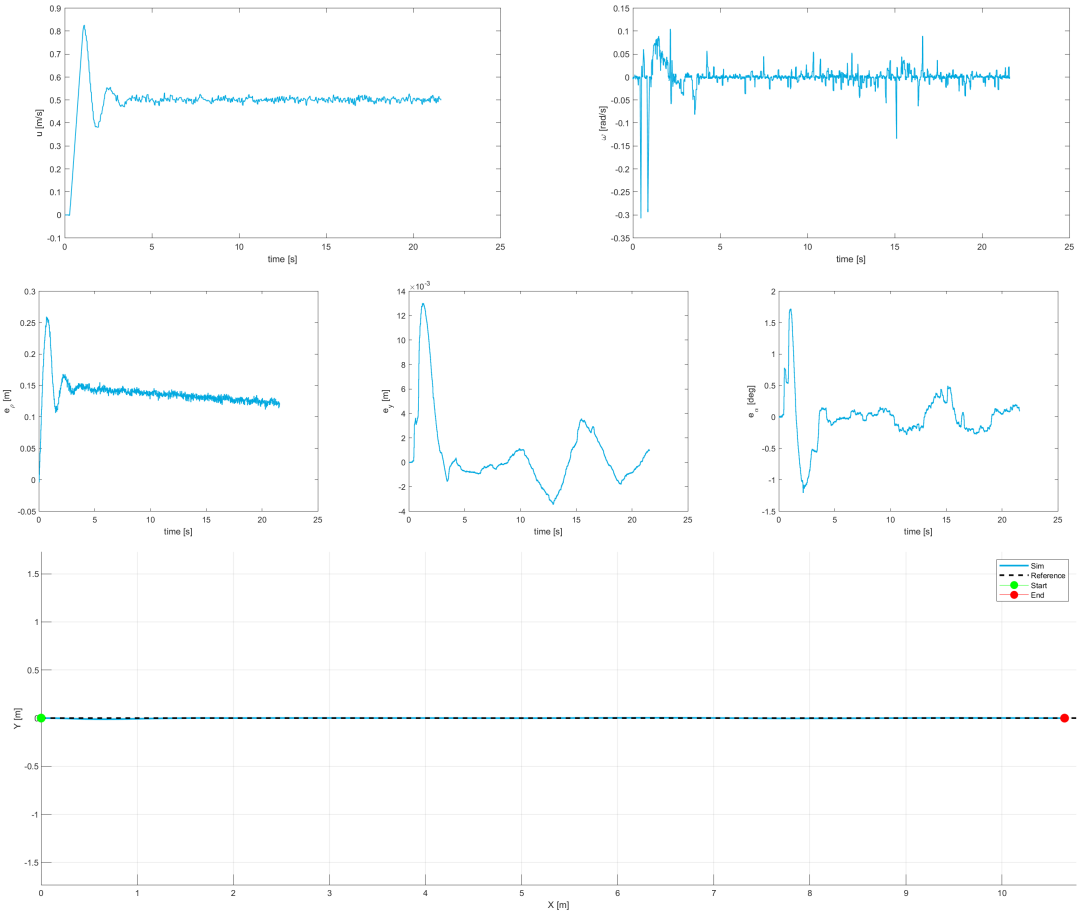

Figure 4.28. LQR simulation results - Linear trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.1209	0.0109	0.9888

Table 4.23. LQR trajectory evaluation metrics - Linear trajectory at 0.25 m/s - Reference vs. Simulated

Circular Trajectory - Case 1

Circular trajectory generated imposing an average linear velocity of 0.25 m/s and a radius of 1.5 m.

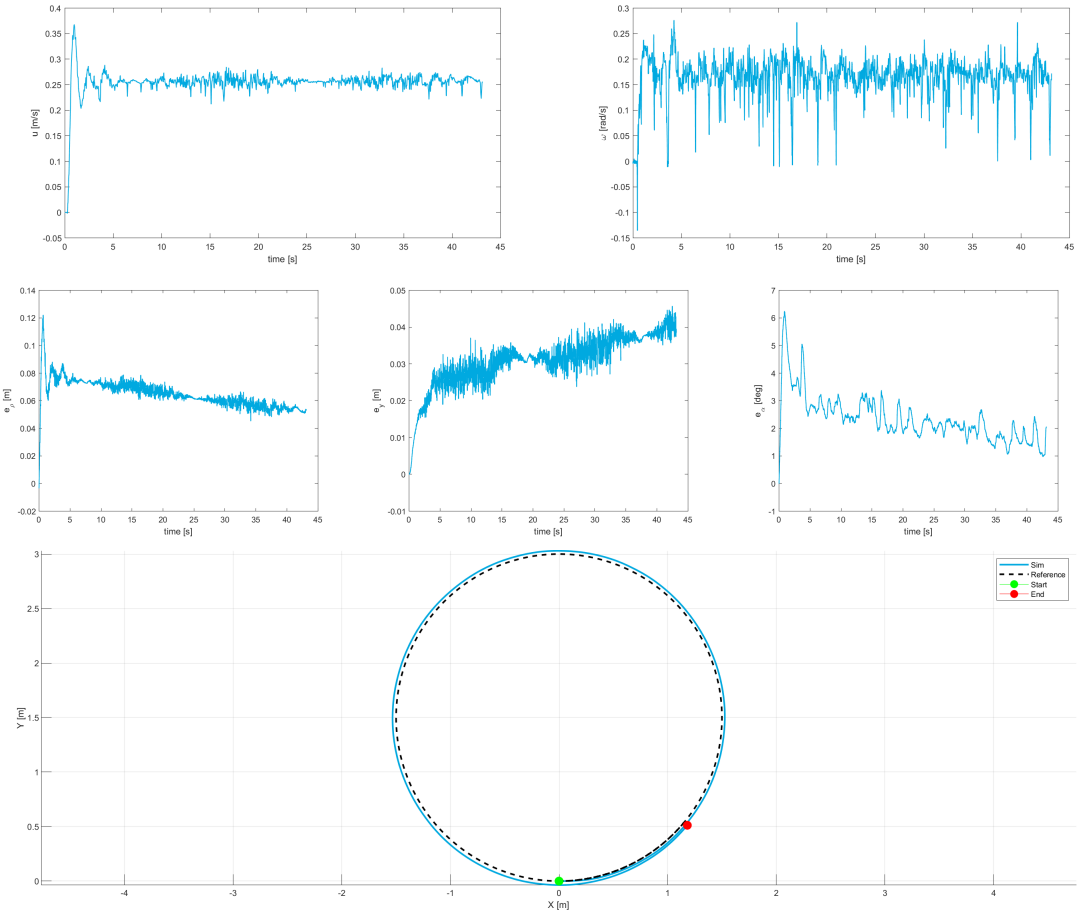

Figure 4.29. LQR simulation results - Circular trajectory at 0.25 m/s and radius 1.5 m - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0372	0.0286	1.0165

Table 4.24. LQR trajectory evaluation metrics - Circular trajectory at 0.25 m/s and radius 1.5 m - Reference vs. Simulated

Circular Trajectory - Case 2

Circular trajectory generated imposing an average linear velocity of 0.50 m/s and a radius of 1.5 m.

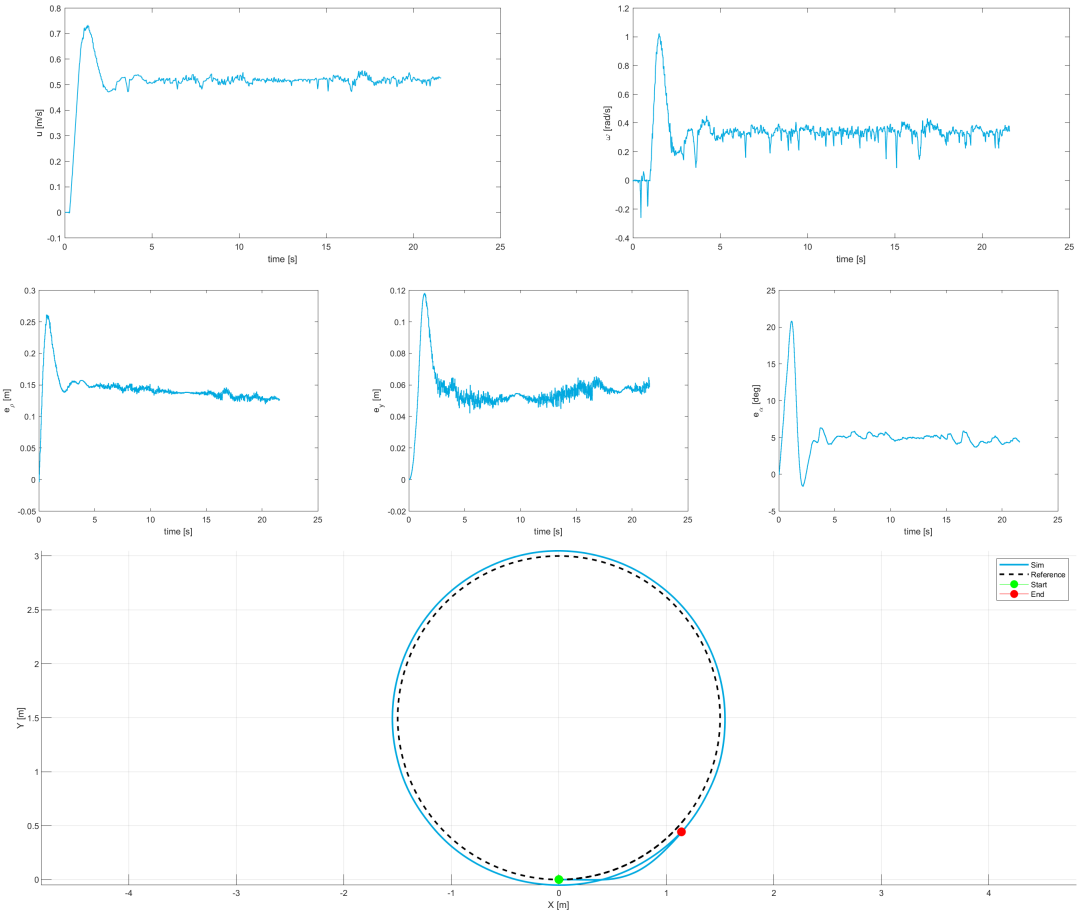

Figure 4.30. LQR simulation results - Circular trajectory at 0.50 m/s and radius 1.5 m - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0583	0.0483	1.0237

Table 4.25. LQR trajectory evaluation metrics - Circular trajectory at 0.50 m/s and radius 1.5 m - Reference vs. Simulated

Sinusoidal Trajectory - Case 1

Sinusoidal trajectory generated imposing an average linear velocity of 0.25 m/s.

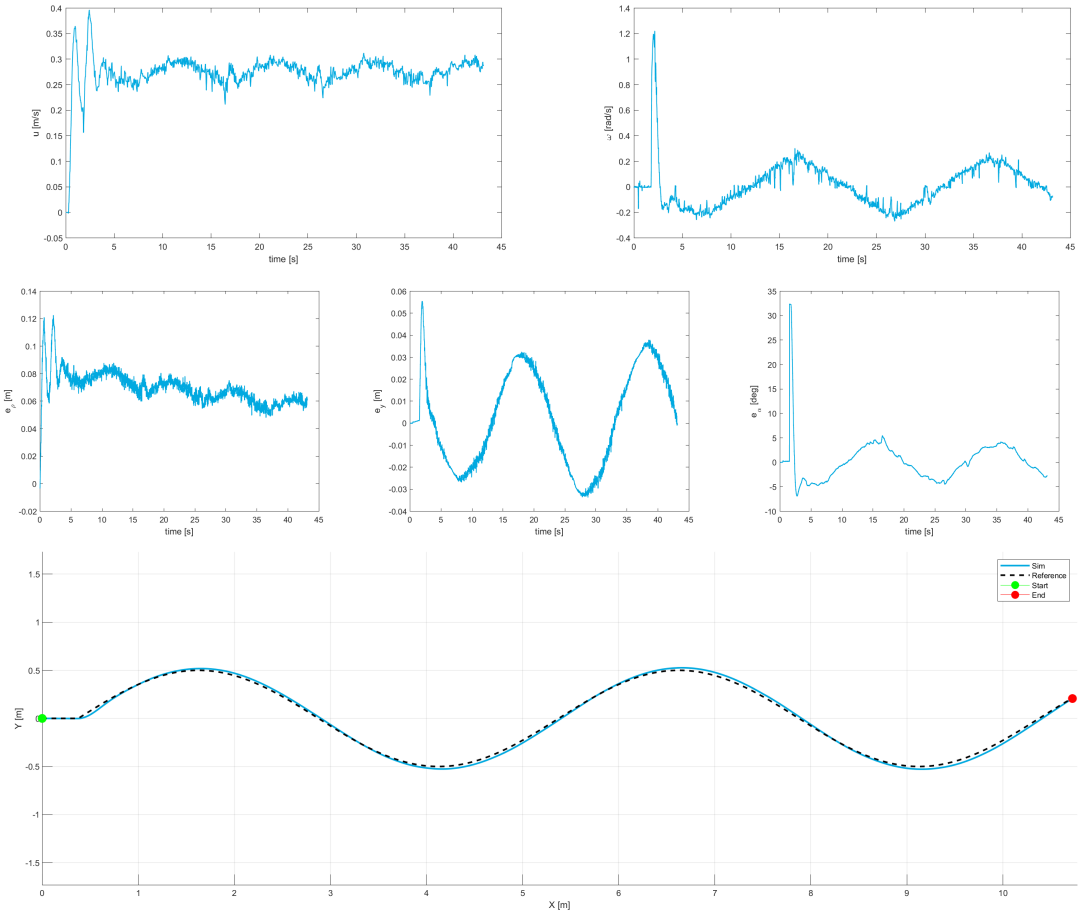

Figure 4.31. LQR simulation results - Sinusoidal trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.0590	0.0218	1.0033

Table 4.26. LQR trajectory evaluation metrics - Sinusoidal trajectory at 0.25 m/s - Reference vs. Simulated

Sinusoidal Trajectory - Case 2

Sinusoidal trajectory generated imposing an average linear velocity of 0.50 m/s.

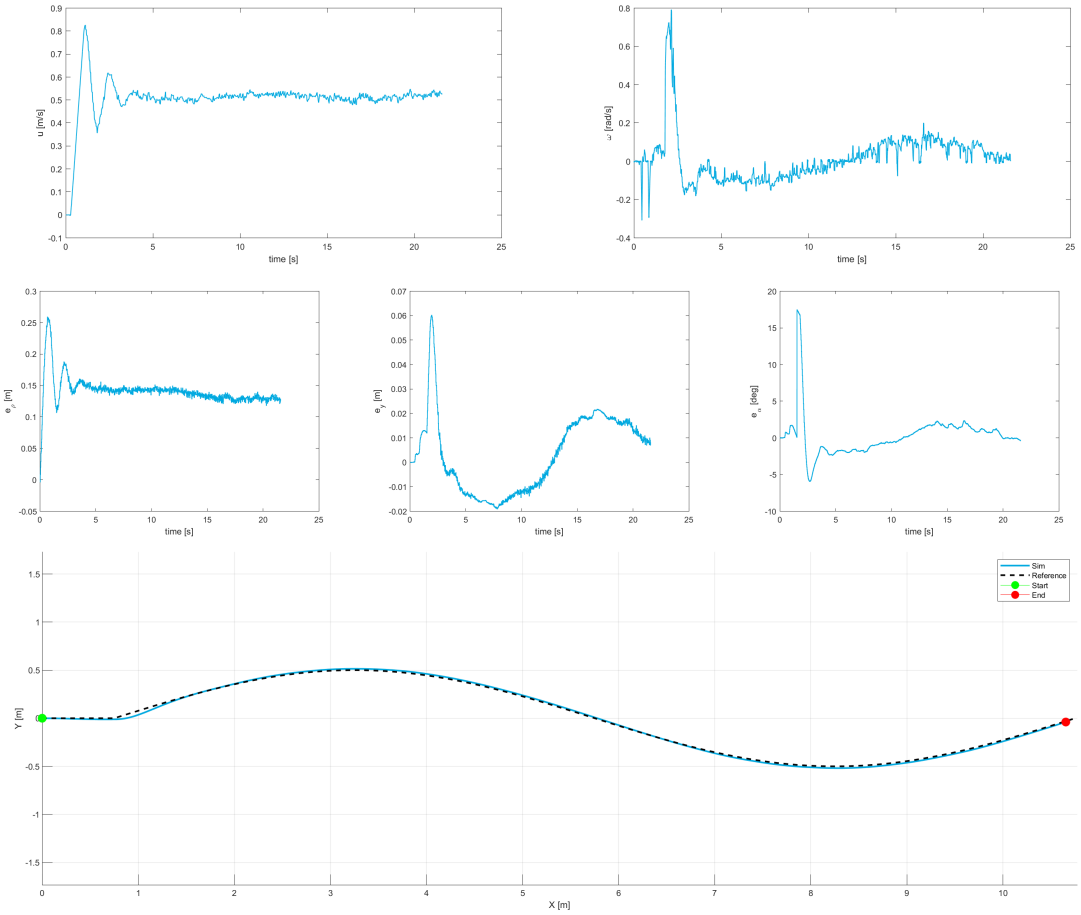

Figure 4.32. LQR simulation results - Sinusoidal trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.1272	0.0178	0.9906

Table 4.27. LQR trajectory evaluation metrics - Sinusoidal trajectory at 0.50 m/s - Reference vs. Simulated

Generic Trajectory - Case 1

Generic trajectory generated imposing an average linear velocity of 0.25 m/s.

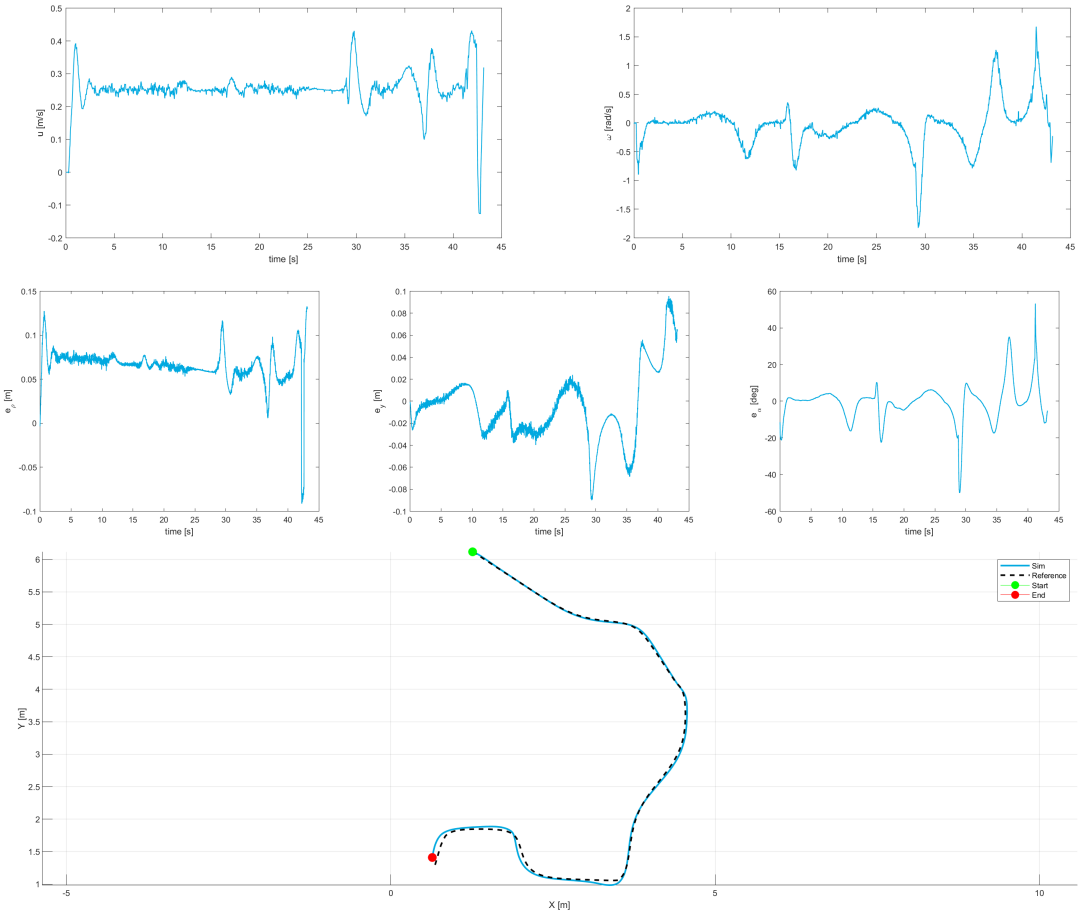

Figure 4.33. LQR simulation results - Generic trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.1297	0.0314	1.0170

Table 4.28. LQR trajectory evaluation metrics - Generic trajectory at 0.25 m/s - Reference vs. Simulated

Generic Trajectory - Case 2

Generic trajectory generated imposing an average linear velocity of 0.50 m/s.

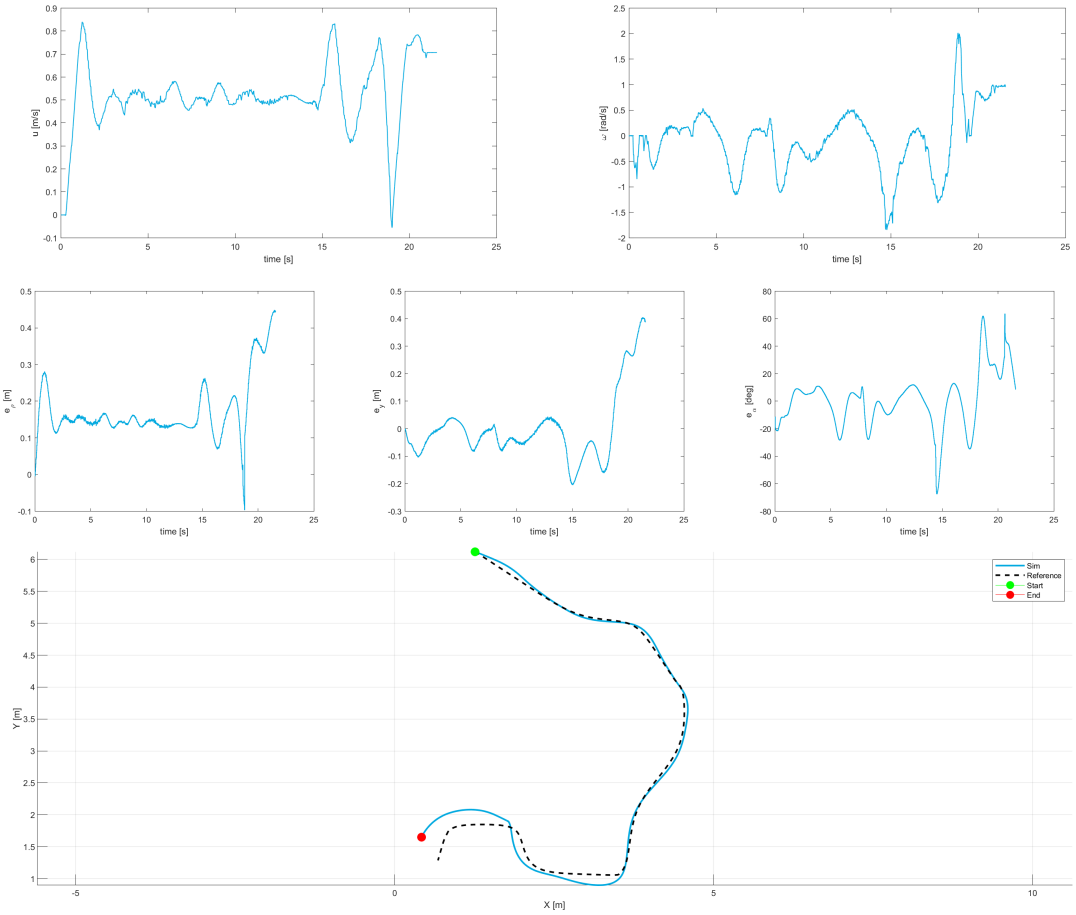

Figure 4.34. LQR simulation results - Generic trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated

d_{haus}	d_{rms}	d_{ratio}
0.4454	0.1123	1.0417

Table 4.29. LQR trajectory evaluation metrics - Generic trajectory at 0.50 m/s - Reference vs. Simulated

4.4.5 Experimental Results

Linear Trajectory - Case 1

Linear trajectory generated imposing an average linear velocity of 0.25 m/s.

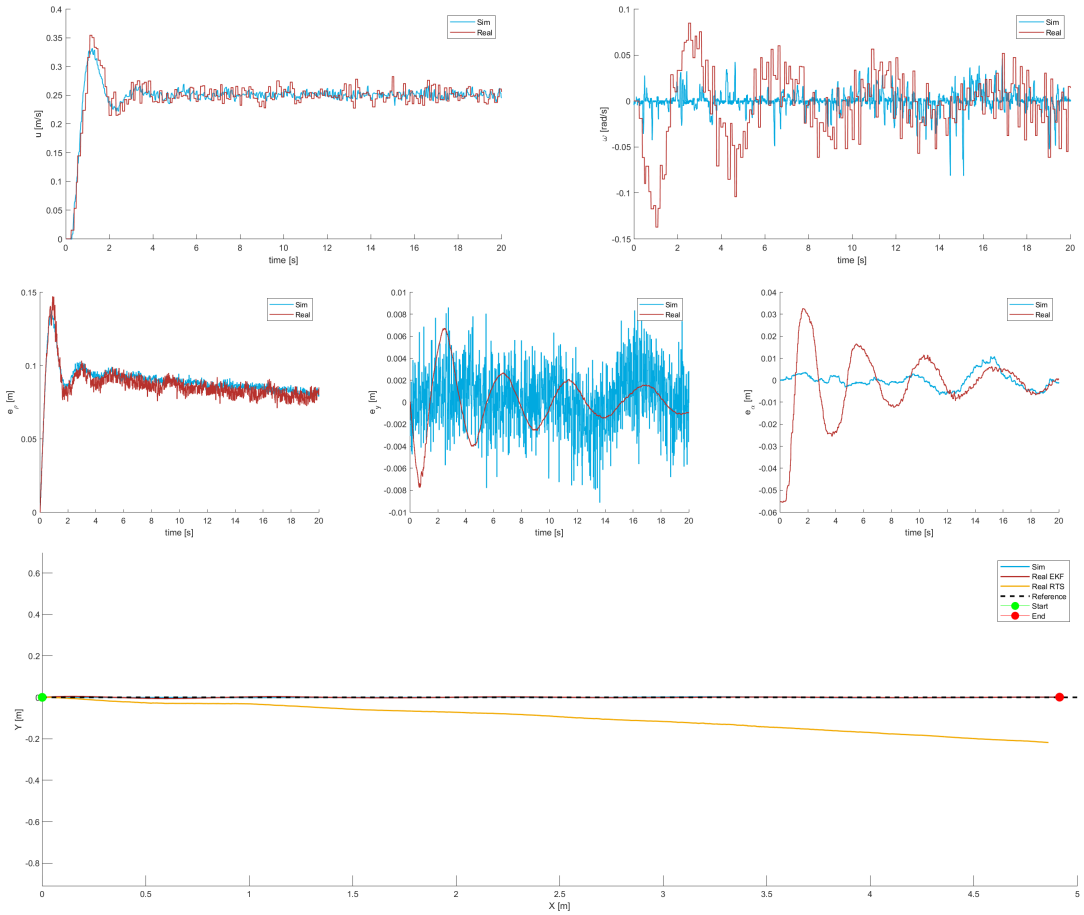

Figure 4.35. LQR experimental results - Linear trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0047	0.0020	1.0002

Table 4.30. LQR trajectory evaluation metrics - Linear trajectory at 0.25 m/s - Simulated vs. Real

Linear Trajectory - Case 2

Linear trajectory generated imposing an average linear velocity of 0.50 m/s.

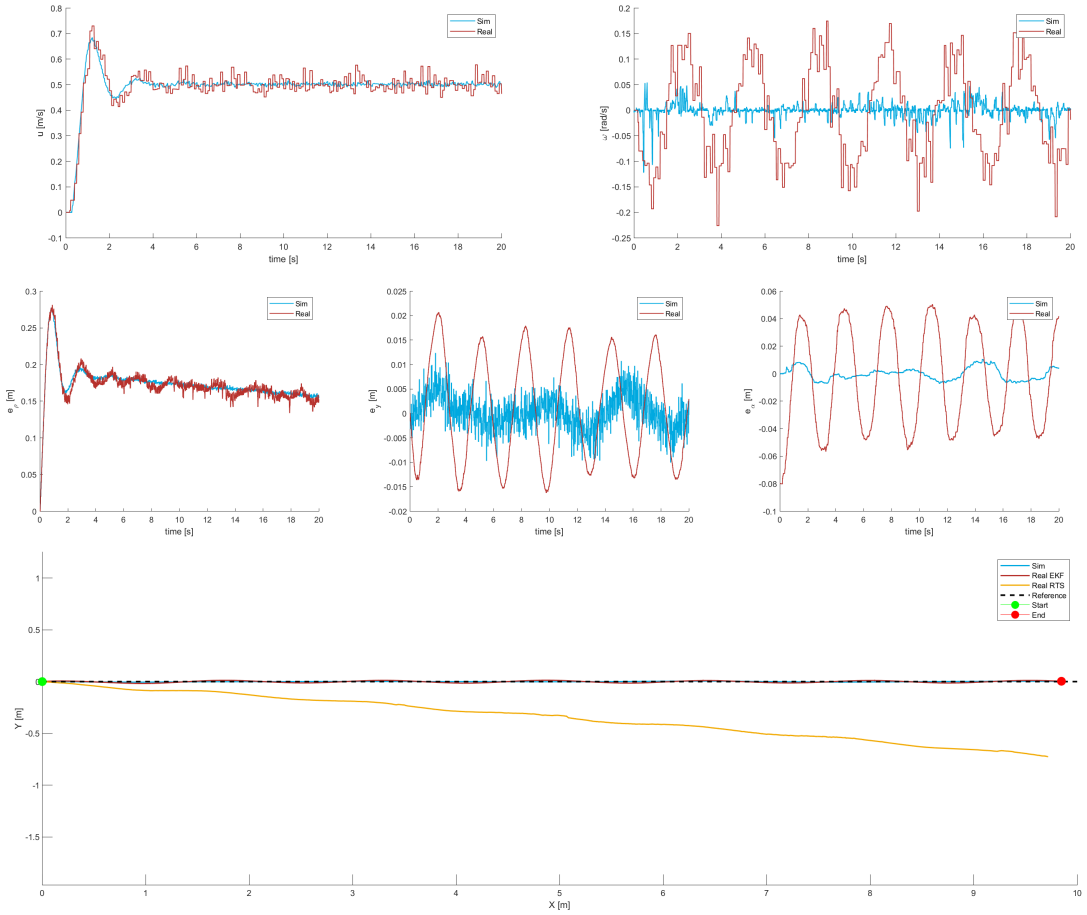

Figure 4.36. LQR experimental results - Linear trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0158	0.0088	0.9979

Table 4.31. LQR trajectory evaluation metrics - Linear trajectory at 0.50 m/s - Simulated vs. Real

Circular Trajectory - Case 1

Circular trajectory generated imposing an average linear velocity of 0.25 m/s and a radius of 1.5 m.

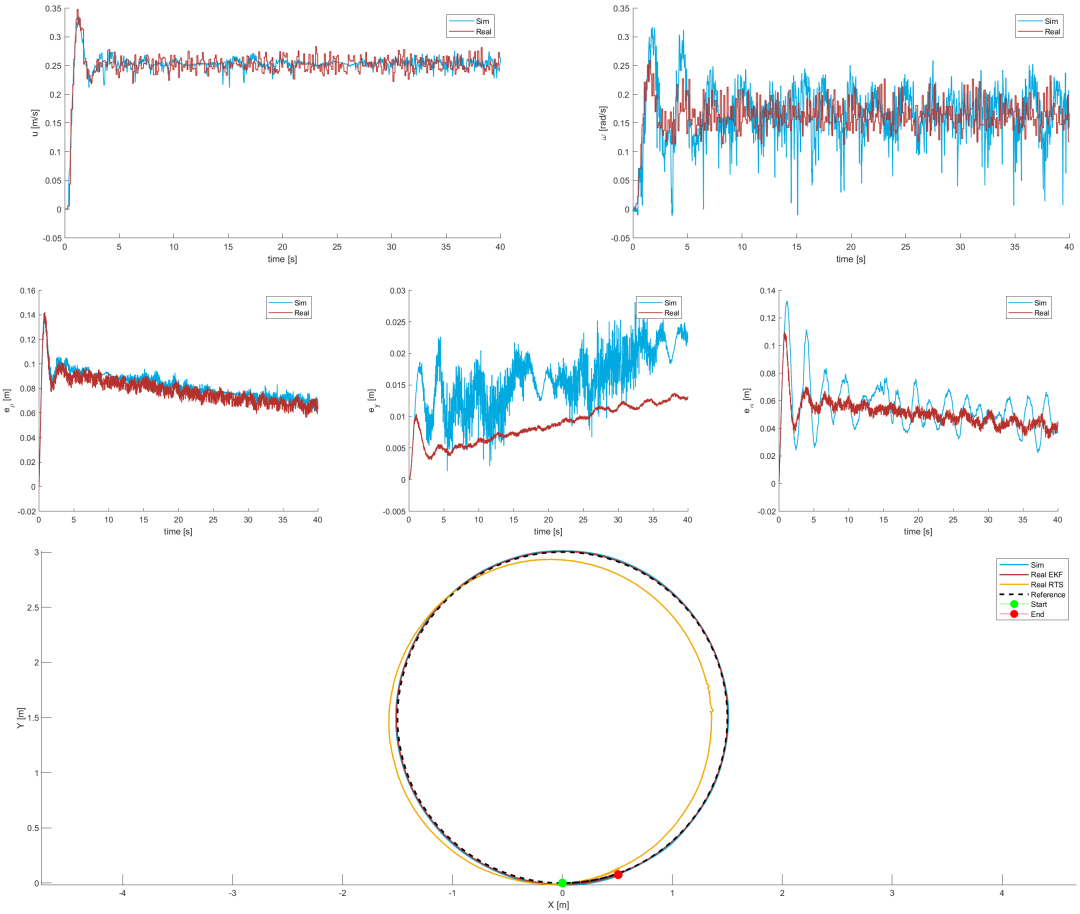

Figure 4.37. LQR experimental results - Circular trajectory at 0.25 m/s and radius 1.5 m- From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0117	0.0069	1.0035

Table 4.32. LQR trajectory evaluation metrics - Circular trajectory at 0.25 m/s and radius 1.5 m - Simulated vs. Real

Circular Trajectory - Case 2

Circular trajectory generated imposing an average linear velocity of 0.50 m/s and a radius of 1.5 m.

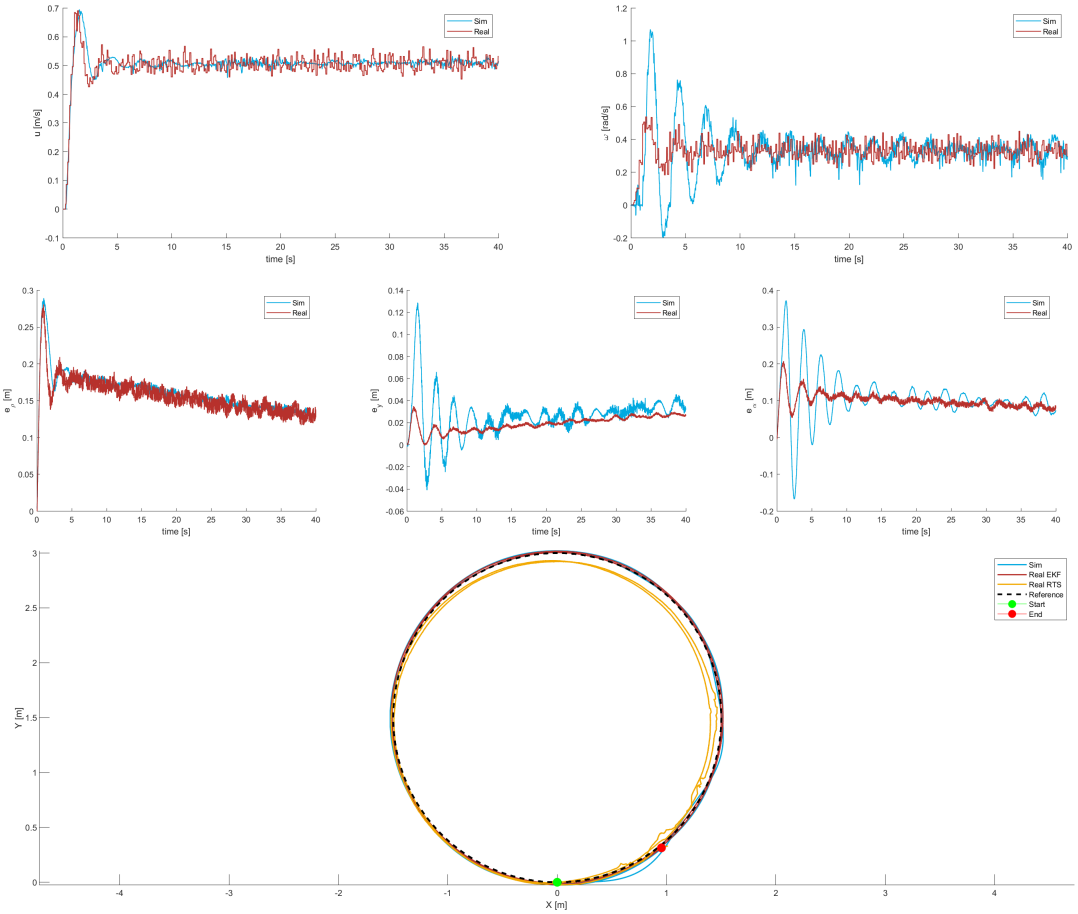

Figure 4.38. LQR experimental results - Circular trajectory at 0.50 m/s and radius 1.5 m- From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0221	0.0070	1.0057

Table 4.33. LQR trajectory evaluation metrics - Circular trajectory at 0.50 m/s and radius 1.5 m - Simulated vs. Real

Sinusoidal Trajectory - Case 1

Sinusoidal trajectory generated imposing an average linear velocity of 0.25 m/s.

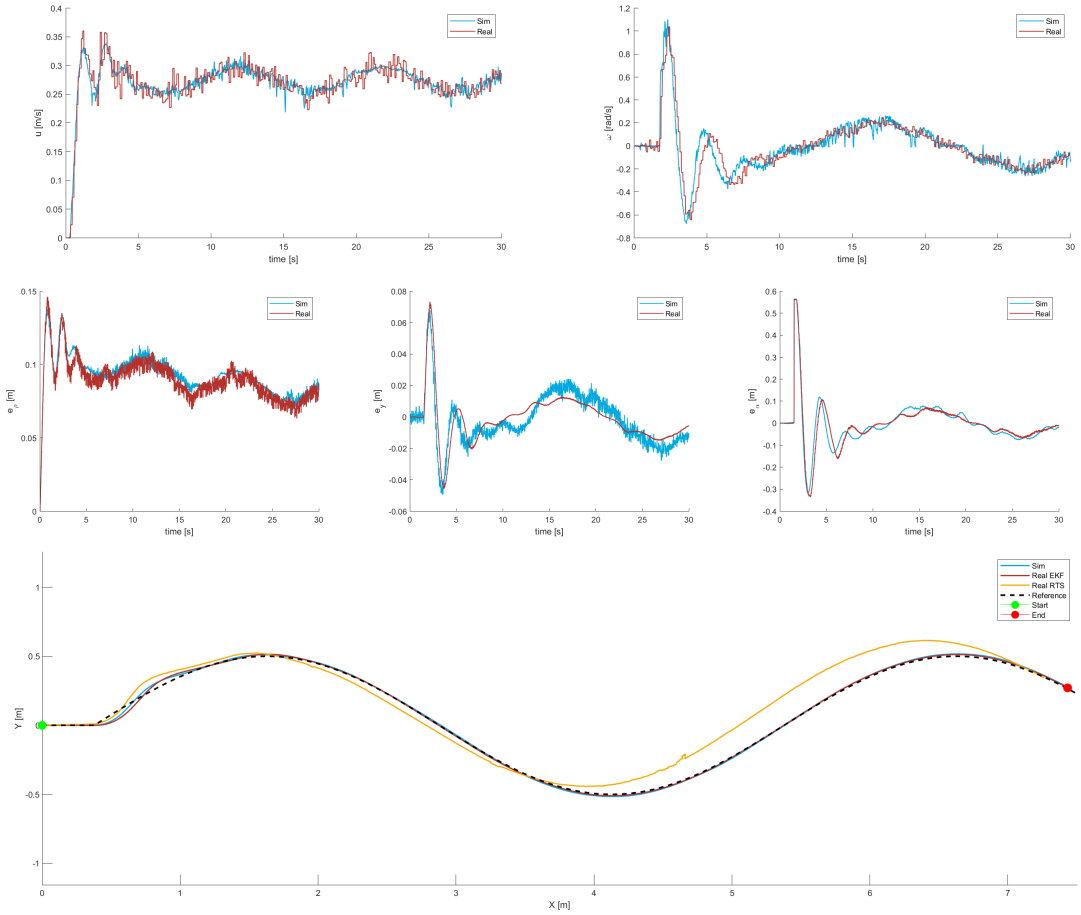

Figure 4.39. LQR experimental results - Sinusoidal trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0230	0.0063	0.9990

Table 4.34. LQR trajectory evaluation metrics - Sinusoidal trajectory at 0.25 m/s - Simulated vs. Real

Generic Trajectory - Case 1

Generic trajectory generated imposing an average linear velocity of 0.25 m/s.

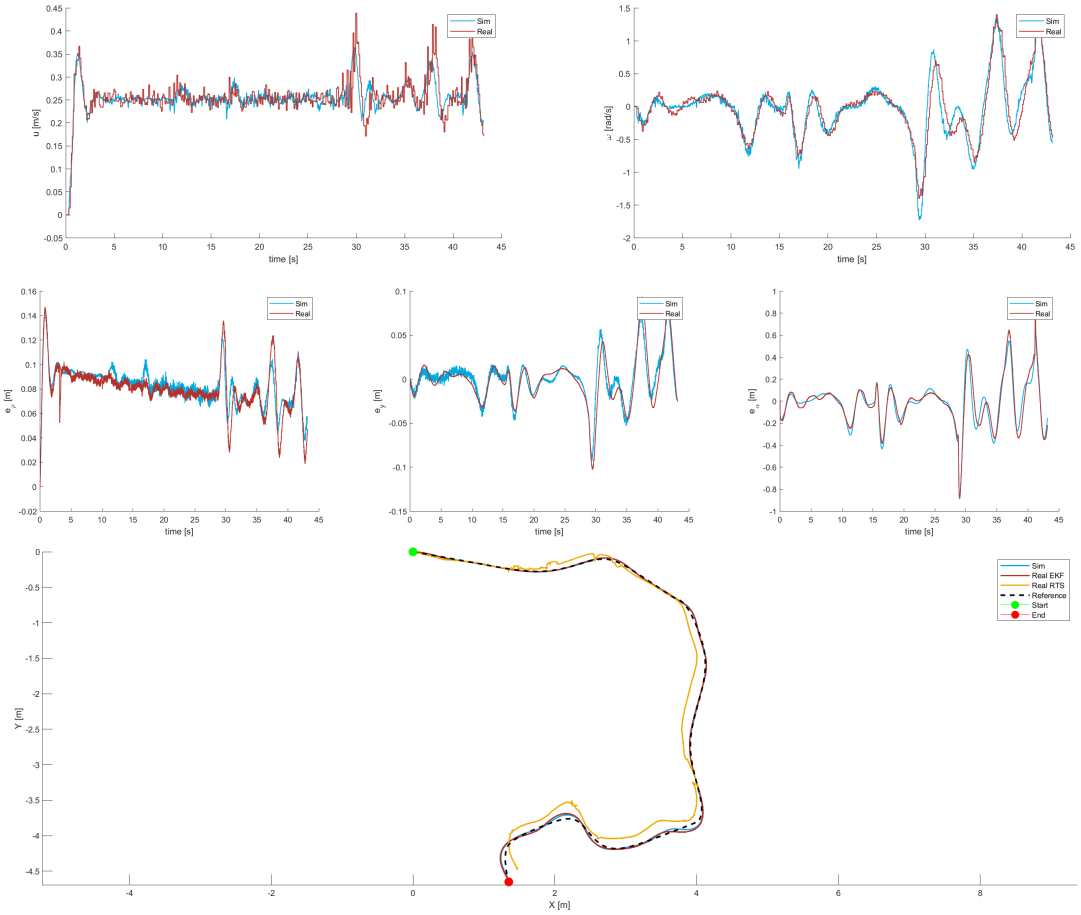

Figure 4.40. LQR experimental results - Generic trajectory at 0.25 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.0413	0.0095	0.9896

Table 4.35. LQR trajectory evaluation metrics - Generic trajectory at 0.25 m/s - Simulated vs. Real

Generic Trajectory - Case 2

Generic trajectory generated imposing an average linear velocity of 0.50 m/s.

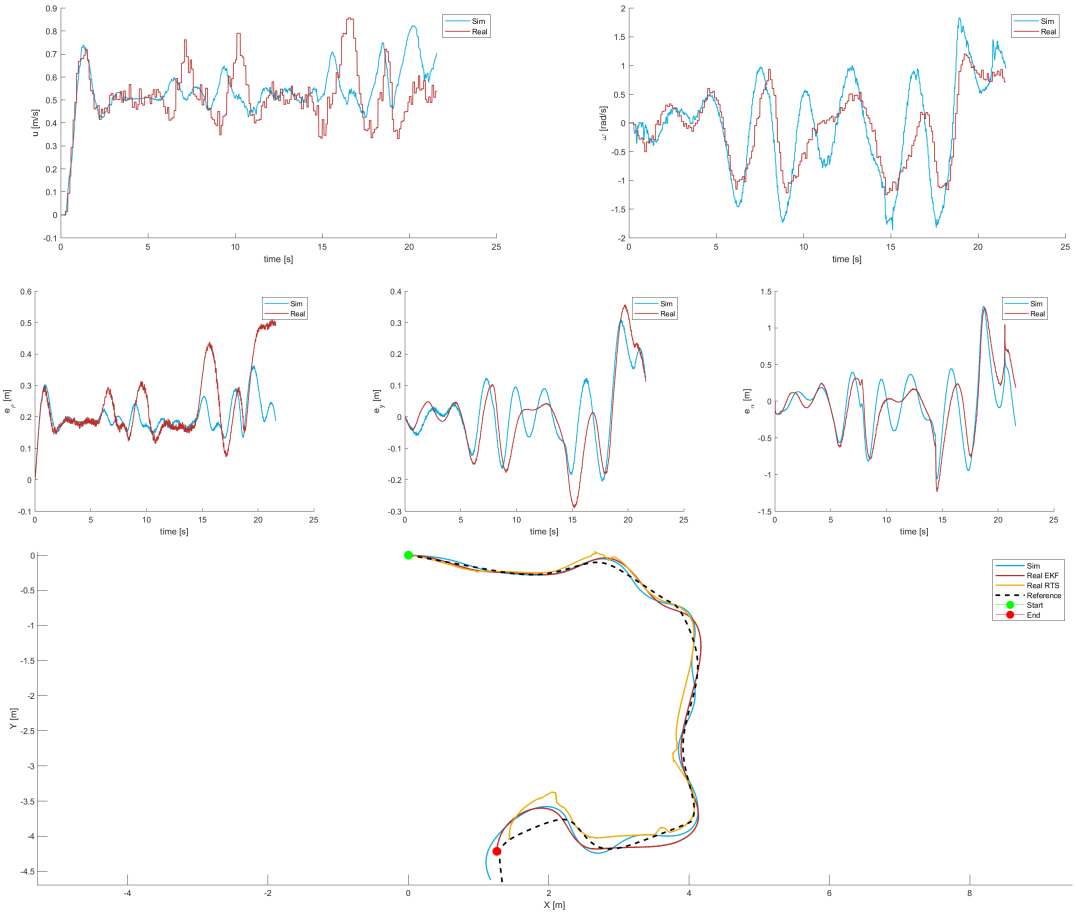

Figure 4.41. LQR experimental results - Generic trajectory at 0.50 m/s - From top left to bottom right: linear velocity u, angular velocity ω, error e_ρ, error e_y, error e_α, trajectory comparison reference vs. simulated vs. real

d_{haus}	d_{rms}	d_{ratio}
0.1735	0.0497	1.0484

Table 4.36. LQR trajectory evaluation metrics - Generic trajectory at 0.50 m/s - Simulated vs. Real

4.5 Controllers Analysis and Comparison

Looking at both the simulations and the experimental results, it is possible to extrapolate some trends. This brief analysis regards both the vehicle modelling and then the performance of the control systems. Regarding the SSMR model, it is possible to state that:

- The mathematical model shows a high level of fidelity to what was measured experimentally. The higher the speed of the manoeuvres, to less faithful the model becomes. Such behaviour can be clearly seen in the trajectory traversed at 0.50 m/s with tight corners. The phenomenon can be traced back to the nonlinearity of the model: not all the nonlinearities were included in the first place in the mathematical formulation, while others were removed by assumptions. It is possible to consider it an acceptable result, as the speed at which the model significantly diverges from the real scenario is very close to the maximum operative speed of the rover.
- The model did not take into account sensors biases. When comparing the RTS measurements - which ought to be the most reliable - it is evident that, even in straight trajectories, what is measured by the onboard EKF diverges from the measurement of the tracking system. It is supposed to be caused by mechanical damage of the wheel encoders. As the control system only relies on the feedback of the EKF, such divergence can not be measured and corrected.

Moving on to the control systems, it is possible to conclude that:

- The PID control system is able to easily execute all the trajectories it was tested with. The least satisfactory results were registered with the generic trajectory, especially at 0.50 m/s. It was concluded that the vehicle is physically unable to move along the provided trajectory at speeds higher than 0.25 m/s: at 0.50 m/s the wheel actuators were ofter operating at their maximum torque. In other words, the system reached a saturation point. More precise tracking can be achieved by imposing a so-called *lookahead distance*: instead of aiming at reducing the e_ρ error to zero, the control system is set to settle at a value greater than zero. Looking at the trajectory run at 0.25 m/s, the average e_ρ is approximately 0.05 m; it was measured that by imposing a lookahead distance of 0.1 m the vehicle is able to traverse the entire trajectory with a significantly higher degree of geometrical precision. Of course, this type of solutions impacts of what is the main goal of trajectory tracking: reaching the desired waypoints at an exact instant in time.
- The LQR control system shows similar behaviour to the PID and all the considerations proposed above are valid for it as well. The overall precision and stability of the LQR is worse than what was achieved with the PID but this is

due to the abrupt stop of the research: further computer simulations proved that the optimised control system had a significant margin of improvement when compared to the one tested with the manually tuned parameters. The improved results, however, couldn't be tested experimentally. The greatest difference between the two control system lies in their degree of complexity: while the PID requires two cascade control loop for a total of twelve parameters to be tuned, the LQR only needs one close loop for a total of six unknown parameters, much more easily identifiable.

The last comment pertains to the IR tracking system. Despite being able of very high precision, it is believed that either the disposition of the emitters or the disposition of the markers was not optimal, as it is possible to notice some unpredicted oscillations in the measurements collected via the IR system. Such oscillations are not classifiable as noise and are physically impossible: they do not represent the real movement of the robot.

	PID			**LQR**		
	d_{haus}	d_{rms}	d_{ratio}	d_{haus}	d_{rms}	d_{ratio}
Linear 0,25 m/s	0,0033	0,0017	0,9995	0,0047	0,0020	1,0002
Linear 0,50 m/s	0,0108	0,0040	0,9994	0,0158	0,0088	0,9979
Circular 0,25 m/s	0,0031	0,0016	0,9994	0,0117	0,0069	1,0035
Circular 0,50 m/s	0,0231	0,0056	1,0006	0,0221	0,0070	1,0057
Sinusoidal 0,25 m/s	0,0192	0,0048	0,9988	0,0230	0,0063	0,9990
Generic 0,25 m/s	0,0392	0,0118	0,9832	0,0413	0,0095	0,9896
Generic 0,50 m/s	0,4569	0,1014	0,9918	0,1735	0,0497	1,0484

Table 4.37. PID - LQR performance comparison - Values measured experimentally

	PID			**LQR**		
	d_{haus}	d_{rms}	d_{ratio}	d_{haus}	d_{rms}	d_{ratio}
Linear 0,25 m/s	-	-	-	+42,42 %	+17,65 %	+0,07 %
Linear 0,50 m/s	-	-	-	+46,29 %	+120,00 %	-0,15 %
Circular 0,25 m/s	-	-	-	+277, 42 %	+331,25 %	+0,41 %
Circular 0,50 m/s	-	-	-	-4,33 %	+25,00 %	+ 0,51 %
Sinusoidal 0,25 m/s	-	-	-	+34,86 %	+34,88 %	+0,02 %
Generic 0,25 m/s	-	-	-	+5,36 %	-19,49 %	+ 0,65 %
Generic 0,50 m/s	-	-	-	-62,03 %	-50,98 %	+5,70 %

Table 4.38. PID - LQR performance comparison, percentage variation taking the PID as reference - Values measured experimentally

Chapter 5

Conclusions

The present MSc thesis illustrates the trajectory tracking problem for a four-wheel SSMR, starting from the eduction of its mathematical model, to the design, simulation and testing of two different controllers, conceived for a real robotic mobile platform: the Clearpath Husky A200. Two mathematical models are compared, and the best one is linearised about an equilibrium point. The unknown physical parameters are determined using a type of AI: Differential Evolution. The theoretical foundation of the two controllers, namely a PID and an LQR, is introduced before their architecture definition. Furthermore, they are tuned employing a DE algorithm, and their numerical simulations are compared to experimental results. Despite being promising, the limited time available for testing left room for improvement. More mature and reliable results, therefore, appear to be possible.

The results obtained demonstrate that the mathematical models described in Chapter 2, whose unknown parameters are determined in Chapter 3, are reliable when used to simulate conditions of limited skidding. The PID controller presented in Chapter 4 proved to guarantee satisfying tracking performances, and the tuning process via DE resulted in the effortless determination of ten gains which granted the achievement of a global optimum with respect to the selected objective function. However, the tracking performance of the PID was obtained at the expense of system robustness to parametric uncertainties, sensor noise and external disturbances. A trade-off between these two characteristics must be evaluated. On the other hand, the DE tuning results of the LQR controller introduced in Chapter 4 could not be tested experimentally due to the interruption of the project. The tests conducted on its manually tuned parameters were too oscillatory but demonstrated how much easier it is to tune an optimal control.

Future developments of the proposed research could aim at concluding the tuning and testing phases of the controllers. Satisfactory results could allow for the coding of the control system on an MCU and more refined tests thereof. Further improvements could be made to the DE tuning algorithm, whose objective function could include new parameters related to time instead of geometry and therefore

obtain better controller performances. While in the present research some manual tuning was still required - for instance, the feedforward gains of the PID - more in-depth analyses could lead to a fully automated tuning process, whose output can be used straight away. The mathematical model can still be developed to discard the bidimensional assumption and account for dynamics which involve the vertical axis. Such improvement would lead to the computation of the exact weight distribution on the four wheels and, consequently, of better wheel-ground interactions estimation. The LQR could be improved by iteratively linearising the system about the instantaneous operating condition and by continuously computing the optimal gain matrix. Lastly, an in-depth study of the sensory component of the vehicle could lead to better dead reckoning estimations on different types of surface; more sensors could also be introduced in the control loop to enhance the robot capabilities, like obstacle avoidance.

Bibliography

[1] McKinsey&Company, "Industrial robotics: Insights into the sector's future growth dynamics," July 2019.

[2] H. R. Everett, *Unmanned systems of World Wars I and II*. Intelligent robotics and autonomous agents, Cambridge, Massachusetts: The MIT Press, 2015.

[3] A. Finn and S. Scheding, *Developments and Challenges for Autonomous Unmanned Vehicles*, vol. 3 of *Intelligent Systems Reference Library*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[4] W. G. Walter, "An Imitation of Life," *Scientific American 182*, pp. 42–45, 1950.

[5] X. Chen, Y. Chen, and J. Chase, "Mobiles Robots - Past Present and Future," tech. rep., University of Canterbury, New Zealand, Department of Mechanical Engineering, 2009.

[6] N. J. Nilsson, "Shakey the Robot," tech. rep., SRI International Menlo Park CA, Apr. 1984.

[7] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, Dec. 1971.

[8] H. P. Moravec, "The Stanford Cart and the CMU Rover," in *Autonomous Robot Vehicles* (I. J. Cox and G. T. Wilfong, eds.), pp. 407–419, New York, NY: Springer New York, 1990.

[9] M. F. Silva and J. Tenreiro Machado, "A Historical Perspective of Legged Robots," *Journal of Vibration and Control*, vol. 13, pp. 1447–1486, Sept. 2007.

[10] C. Angle, *Genghis, a six legged autonomous walking robot*. Doctoral Dissertation, Massachusetts Institute of Technology, 1989.

[11] I. Kato, S. Ohteru, K. Shirai, T. Matsushima, S. Narita, S. Sugano, T. Kobayashi, and E. Fujisawa, "The robot musician 'wabot-2' (waseda robot-2)," *Robotics*, vol. 3, pp. 143–155, June 1987.

[12] G. Giralt, R. Sobek, and R. Chatila, "A multi-level planning and navigation system for a mobile robot: a first approach to HILARE," in *Proceedings of the 6th international joint conference on Artificial intelligence - Volume 1*, IJCAI'79, (Tokyo, Japan), pp. 335–337, Morgan Kaufmann Publishers Inc., Aug. 1979.

[13] S. Sekhavat, F. Lamiraux, J. Laumond, G. Bauzil, and A. Ferrand, "Motion

planning and control for Hilare pulling a trailer: experimental issues," in *Proceedings of International Conference on Robotics and Automation*, vol. 4, (Albuquerque, NM, USA), pp. 3306–3311, IEEE, 1997.

[14] N. R. C. (U.S.) and N. R. C. (U.S.), eds., *Technology development for Army unmanned ground vehicles*. Washington, D.C: National Academies Press, 2002. OCLC: ocm51746797.

[15] R. Siegwart and I. R. Nourbakhsh, *Introduction to autonomous mobile robots*. Intelligent robots and autonomous agents, Cambridge, Mass: MIT Press, 2004.

[16] C. Ilas, "Electronic sensing technologies for autonomous ground vehicles: A review," in *2013 8TH INTERNATIONAL SYMPOSIUM ON ADVANCED TOPICS IN ELECTRICAL ENGINEERING (ATEE)*, (Bucharest, Romania), pp. 1–6, IEEE, May 2013.

[17] F. Gustafsson, *Statistical sensor fusion*. Studentlitteratur, 2010.

[18] J. Pentzer, S. Brennan, and K. Reichard, "Model-based Prediction of Skid-steer Robot Kinematics Using Online Estimation of Track Instantaneous Centers of Rotation: Model-based Prediction of Skid-steer Robot Kinematics," *Journal of Field Robotics*, vol. 31, pp. 455–476, May 2014.

[19] G. Anousaki and K. Kyriakopoulos, "A dead-reckoning scheme for skid-steered vehicles in outdoor environments," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, (New Orleans, LA, USA), pp. 580–585 Vol.1, IEEE, 2004.

[20] D. M. Helmick, S. I. Roumeliotis, Y. Cheng, D. S. Clouse, M. Bajracharya, and L. H. Matthies, "Slip-compensated path following for planetary exploration rovers," *Advanced Robotics*, vol. 20, pp. 1257–1280, Jan. 2006.

[21] A. Angelova, L. Matthies, D. Helmick, and P. Perona, "Slip Prediction Using Visual Information," in *Robotics: Science and Systems II*, Robotics: Science and Systems Foundation, Aug. 2006.

[22] L. Caracciolo, A. de Luca, and S. Iannitti, "Trajectory tracking control of a four-wheel differentially driven mobile robot," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 4, (Detroit, MI, USA), pp. 2632–2638, IEEE, 1999.

[23] K. Koz\lowski and D. Pazderski, "Modeling and control of a 4-wheel skid-steering mobile robot," *International journal of applied mathematics and computer science*, vol. 14, pp. 477–496, 2004.

[24] W. E. Dixon, D. M. Dawson, E. Zergeroglu, and A. Behal, *Nonlinear Control of Wheeled Mobile Robots*, vol. 262 of *Lecture Notes in Control and Information Sciences*. London: Springer London, 2001.

[25] Y. Wu, T. Wang, J. Liang, J. Chen, Q. Zhao, X. Yang, and C. Han, "Experimental kinematics modeling estimation for wheeled skid-steering mobile robots," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, (Shenzhen, China), pp. 268–273, IEEE, Dec. 2013.

[26] V. Rajagopalan, C. Mericli, and A. Kelly, "Slip-aware Model Predictive optimal

control for Path following," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, (Stockholm), pp. 4585–4590, IEEE, May 2016.

[27] J. Yi, D. Song, J. Zhang, and Z. Goodwin, "Adaptive Trajectory Tracking Control of Skid-Steered Mobile Robots," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, (Rome, Italy), pp. 2605–2610, IEEE, Apr. 2007.

[28] J. Cerkala and A. Jadlovská, "NONHOLONOMIC MOBILE ROBOT WITH DIFFERENTIAL CHASSIS MATHEMATICAL MODELLING AND IMPLEMENTATION IN SIMULINK WITH FRICTION IN DYNAMICS," *Acta Electrotechnica et Informatica*, vol. 15, pp. 3–8, Sept. 2015.

[29] S. Jeon and W. Jeong, "The Stable Trajectory Tracking Control of a Skid-Steered Mobile Platform with Dynamic Uncertainties," *International Journal of Advanced Robotic Systems*, vol. 12, p. 122, Sept. 2015.

[30] J. Majumdar, S. C Gupta, and B. Prassanna Prasath, "Linear and Non-Linear Control Design of Skid Steer Mobile Robot on an Embedded," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 7, p. 185, Sept. 2018.

[31] L. Borello and M. Dalla Vedova, "Dry Friction Discontinuous Computational Algorithms," in *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 3, Feb. 2014.

[32] R. C. Dorf and R. H. Bishop, *Modern control systems.* Hoboken: Pearson, thirteenth edition ed., 2016.

[33] G. Cohen, "Theoretical consideration of retarded control," *Trans. Asme*, vol. 75, pp. 827–834, 1953.

[34] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *trans. ASME*, vol. 64, no. 11, 1942.

[35] W. K. Ho, C. C. Hang, and L. S. Cao, "Tuning of PID controllers based on gain and phase margin specifications," *Automatica*, vol. 31, no. 3, pp. 497–502, 1995. Publisher: Elsevier.

[36] K. J. \AAström and T. Hägglund, "Automatic tuning of simple regulators with specifications on phase and amplitude margins," *Automatica*, vol. 20, no. 5, pp. 645–651, 1984. Publisher: Elsevier.

[37] C. C. Hang, K. J. \AAström, and W. K. Ho, "Refinements of the Ziegler–Nichols tuning formula," in *IEE Proceedings D (Control Theory and Applications)*, vol. 138, pp. 111–118, IET, 1991. Issue: 2.

[38] R. S. Ali, A. A. Aldair, and A. K. Almousawi, "Design an optimal PID controller using artificial bee colony and genetic algorithm for autonomous mobile robot," *International Journal of Computer Applications*, vol. 100, no. 16, pp. 8–16, 2014.

[39] P. B. de Moura Oliveira, "Modern heuristics review for PID control optimization: A teaching experiment," in *2005 international conference on control and automation*, vol. 2, pp. 828–833, IEEE, 2005.

[40] P. Šuster and A. Jadlovská, "Tracking trajectory of the mobile robot Khepera II

using approaches of artificial intelligence," *Acta Electrotechnica et Informatica*, vol. 11, no. 1, pp. 38–43, 2011. Publisher: De Gruyter Open Sp. z oo.

[41] A. Alouache and Q. Wu, "Genetic Algorithms for Trajectory Tracking of Mobile Robot Based on PID Controller," in *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 237–241, IEEE, 2018.

[42] C. Kumar and T. Alwarsamy, "Solution of economic dispatch problem using differential evolution algorithm," *International Journal of Soft Computing and Engineering*, vol. 1, no. 6, pp. 236–241, 2012.

[43] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[44] V. Pano and P. R. Ouyang, "Comparative study of ga, pso, and de for tuning position domain pid controller," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pp. 1254–1259, IEEE, 2014.

[45] W. Wang, X. Yuan, and J. Zhu, "Automatic PID tuning via differential evolution for quadrotor UAVs trajectory tracking," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2016.

[46] L. Sun and J. Gan, "Researching of two-wheeled self-balancing robot base on LQR combined with PID," in *2010 2nd International Workshop on Intelligent Systems and Applications*, pp. 1–5, IEEE, 2010.

[47] A. N. K. Nasir, M. A. Ahmad, and R. R. Ismail, "The control of a highly nonlinear two-wheels balancing robot: A comparative assessment between LQR and PID-PID control schemes," *World Academy of Science, Engineering and Technology*, vol. 70, pp. 227–232, 2010.

[48] H.-j. Zhang, J.-w. Gong, Y. Jiang, G.-m. Xiong, and H.-y. Chen, "An iterative linear quadratic regulator based trajectory tracking controller for wheeled mobile robot," *Journal of Zhejiang University SCIENCE C*, vol. 13, no. 8, pp. 593–600, 2012. Publisher: Springer.

[49] A. Abbasi and A. J. Moshayedi, "Trajectory Tracking of Two-Wheeled Mobile Robots, Using LQR Optimal Control Method, Based On Computational Model of KHEPERA IV," *Journal of Simulation and Analysis of Novel Technologies in Mechanical Engineering*, vol. 10, pp. 41–50, Jan. 2018. Publisher: Islamic Azad University, Khomeinishahr Branch.

[50] F. Lin, Z. Lin, and X. Qiu, "LQR controller for car-like robot," in *2016 35th Chinese Control Conference (CCC)*, pp. 2515–2518, July 2016. ISSN: 1934-1768.

[51] X. Heng, D. Cabecinhas, R. Cunha, C. Silvestre, and X. Qingsong, "A trajectory tracking LQR controller for a quadrotor: Design and experimental evaluation," in *TENCON 2015-2015 IEEE Region 10 Conference*, pp. 1–7, IEEE, 2015.

[52] I. D. Cowling, J. F. Whidborne, and A. K. Cooke, "Optimal trajectory planning and LQR control for a quadrotor UAV," in *International Conference on Control*, 2006.

[53] Vishal and J. Ohri, "GA tuned LQR and PID controller for aircraft pitch control," in *2014 IEEE 6th India International Conference on Power Electronics (IICPE)*, pp. 1–6, Dec. 2014. ISSN: 2160-3170.

[54] M. Assahubulkahfi, Y. Md. Sam, A. Maseleno, and M. Huda, "LQR Tuning by Particle Swarm Optimization of Full Car Suspension System," *International Journal of Engineering & Technology*, vol. 7, p. 328, Apr. 2018.

[55] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, vol. 2, pp. 1980–1987, IEEE, 2004.

[56] S. Das and P. N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," *IEEE Transactions on Evolutionary Computation*, vol. 15, pp. 4–31, Feb. 2011.

[57] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2008. Publisher: IEEE.

[58] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE transactions on evolutionary computation*, vol. 10, no. 6, pp. 646–657, 2006. Publisher: IEEE.

[59] G. Ermacora, D. Sartori, L. Pei, and W. Yu, "An Evaluation Framework for the Deployment of Mobile Robots Performing Real-World Indoor Autonomous Navigation,"

[60] E. Saggini, E. Zereik, M. Bibuli, G. Bruzzone, M. Caccia, and E. Riccomagno, "Performance Indices for Evaluation and Comparison of Unmanned Marine Vehicles' Guidance Systems," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 12182–12187, 2014.

[61] N. D. Munoz Ceballos, J. Alejandro, and N. Londono, "Quantitative Performance Metrics for Mobile Robots Navigation," in *Mobile Robots Navigation* (A. Barrera, ed.), InTech, Mar. 2010.

[62] G. R. Jensen and S. G. Krantz, eds., *150 years of mathematics at Washington University in St. Louis: sesquicentennial of mathematics at Washington University, October 3-5, 2003, Washington University, St. Louis, Missouri.* No. 395 in Contemporary mathematics, Providence, R.I: American Mathematical Society, 2006. OCLC: ocm62408845.

[63] H. B. Pacejka and I. Besselink, *Tire and vehicle dynamics.* Engineering Automotive engineering, Amsterdam: Elsevier/Butterworth-Heinemann, 3. ed ed., 2012. OCLC: 796260687.

[64] H. Olsson, K. Åström, C. Canudas de Wit, M. Gäfvert, and P. Lischinsky, "Friction Models and Friction Compensation," *European Journal of Control*, vol. 4, pp. 176–195, Jan. 1998.

[65] D. Rowell, "State-Space Representation of LTI Systems," *MIT - 2.14 Analysis and Design of Feedback Control Systems*, p. 18, 2004.

[66] F. Neri and V. Tirronen, "Recent advances in differential evolution: a survey and experimental analysis," *Artificial Intelligence Review*, vol. 33, no. 1-2, pp. 61–106, 2010. Publisher: Springer.

[67] R. Mallipeddi, P. N. Suganthan, Q.-K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Applied soft computing*, vol. 11, no. 2, pp. 1679–1696, 2011. Publisher: Elsevier.

[68] N. Minorsky, "Directional stability of automatically steered bodies," *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, pp. 280–309, 1922. Publisher: Wiley Online Library.