Analysis of Inverse Simulation Algorithms with an Application to Planetary Rover Guidance and Control

Thaleia Flessa

Submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Aerospace Sciences Research Division

James Watt School of Engineering

College of Science and Engineering

University of Glasgow

December 2021

Author's Declaration

I declare that, except where explicit reference is made to the contribution of others, this dissertation is the result of my own work and has not been submitted for any other degree at the University of Glasgow or any other institution.

Thaleia Flessa

Acknowledgement

I would like to thank my research supervisors, Dr Euan McGookin and Dr Douglas Thomson for giving me the opportunity to conduct research for this work.

My thanks also to the Engineering and Physical Sciences Research Council (EPSRC) for providing the funding (EPSRC/1369575) which allowed me to carry out the work presented in this thesis; without it, this work would not be possible. Also, to the School of Engineering and the Aerospace Sciences Division for providing me with all necessary resources.

A very special thanks to Dimitrios for his love, encouragement, patience, and support and for being with me on every step of this journey; the highs, the lows, and the plateaus.

And a final thanks to you, reader for picking this up and giving it a go.

This thesis is dedicated to my father, Professor George P. Flessas, who passed away so very unexpectedly.

Abstract

Rover exploration is a contributing factor to driving the relevant research forward on guidance, navigation, and control (GNC). Yet, there is a need for incorporating the dynamic model into the controller for increased accuracy. Methods that use the model are limited by issues such as linearity, systems affine in the control, number of inputs and outputs. Inverse Simulation is a more general approach that uses a mathematical model and a numerical scheme to calculate the control inputs necessary to produce a desired response defined using the output variables.

This thesis develops the Inverse Simulation algorithm for a general state space model and utilises a numerical Newton-Raphson scheme to converge to the inputs using two approaches: The **Differentiation method** converges based on the state and output equations. The **Integration method** converges based on whether the output matches the desired and is suitable for grey or black-box models. The thesis offers extensive insights into the requirements and application of Inverse Simulation and the performance parameters. Attention is given to how the inputs and outputs affect the Jacobian formulation and ensure an efficient solution. The linear case and the relationship with feedback linearisation are examined. Examples are given using simple mechanical systems and an example is also given as to how Inverse Simulation can be used for determining system input disturbances.

Inverse Simulation is applied for the first time for guidance and control of a four-wheeled, differentially driven rover. The desired output is the time history of the desired trajectory and is used to produce the required control inputs. The control inputs are nominal and are applied to the rover without additional correction. Using insights from the system's physics and actuation, the Differentiation and Integration schemes are developed based on the general method presented in this thesis. The novel Differentiation scheme employs a non-square Jacobian. The method provides very accurate position and orientation control of the rover while considering the limitations of the model used. Finally, the application of Inverse Simulation to the rover is supported by a review of current designs that resulted in a rover taxonomy.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Motivation

Current research on guidance, navigation, and control (GNC) for wheeled, mobile robots is a vast, constantly evolving field. A search at IEEEXplore using the keywords “motion”, “control” and “mobile” returns 16,508 results published in conferences, journals, magazines, and books indexed by IEEE between 1980 and 2020, of which 14,637 were published between 2000 – 2020. A similar search at ScienceDirect using the keywords “motion”, “control” and “mobile” returns 62,904 results between 2010 and 2020; adding the term “ground” (e.g., to exclude mobile manipulators) returns 17,785 results. A review article published in 2018 outlining the ten great challenges in science robotics (Yang *et al.*, 2018), identified the exploration capabilities of mobile robots as one of these great challenges that also have “a wider impact on all application areas of robotics”, adding that “The associated challenges are therefore much greater than those encountered today”.

The term rover usually applies to a system that uses wheels for locomotion; however, they may alternatively use tracks, legs or a combination of these (Yoshida *et al.*, 2008). Throughout this work, the term rover is used to refer to an unmanned, mobile, wheeled robot that can explore a surface with varying degrees of autonomy. Thus, rover as a term can be considered equivalent to a wheeled mobile robot (WMR) in this work. The term robot is used more generally to indicate any mobile system, regardless of its means of locomotion.

Rovers in terms of exploration capabilities have significant advantages: they can traverse different terrain types, slopes and overcome obstacles and can explore a large area. The recent achievements of the NASA Mars Exploration Rovers (MER) Spirit and Opportunity (2004), Curiosity (2012), Perseverance (2020) and the forthcoming ESA ExoMars (2022) mission have captured the public’s imagination. They have also been a significant contributing factor in driving the relevant research forward on control, control architectures, guidance, navigation, autonomy, mobility, computer processing, vision and communications (Quadrelli *et al.*, 2015; Mateo Sanguino, 2017).

To manoeuvre a rover (and any robot in general) three tasks are required: guidance, navigation, and control (GNC). An additional requirement is motion planning: the determination of the desired path or trajectory that can provide the reference condition needed to determine a guidance command (Siegwart *et al.*, 2011). A path is a route through space that has only spatial constraints, whereas a trajectory also has temporal constraints (Siegwart *et al.*, 2011). For example, in Cartesian space, a trajectory can be defined as a series of ($x(t)$, $y(t)$, $z(t)$) or their derivatives. The following definitions for GNC are used, based on (Fossen, 2011).

Navigation is the determination of the system's current state: position, attitude, distance travelled and, in some cases, velocity and acceleration are determined as well.

Guidance is the action or the system that continuously computes the reference (desired) position, velocity, and acceleration to be used by the motion control system. The guidance system determines the reference trajectories to be fed to the control system from the vehicle's current location (from navigation) to a designated target, as well as desired changes in velocity, rotation, and acceleration for following that reference trajectory.

Control, or more specifically motion control, is the action of determining the necessary control forces and moments to be provided to satisfy a certain control objective. The desired control objective is usually seen in conjunction with the guidance system. Constructing the control algorithm involves the design of feedback and feedforward control laws. The outputs from the navigation system, (position, velocity, and acceleration) are used for feedback control while feedforward control is implemented using signals available in the guidance system and when available, other external sensors.

For mobile robots, the overall objective is (a) follow a desired path, (b) follow a desired trajectory or (c) achieve a certain pose (Morin *et al.*, 2008). Case (a) and (b) are referred to as output tracking. Ideally, the tracking error converges to zero. Sometimes this is not possible, and the concept of practical stabilisation is defined when the tracking error is small but not zero (Morin *et al.*, 2008). Point-to-point motion (also known as point or posture stabilisation) is defined as starting from an initial configuration, the robot must reach a desired goal configuration

(Morin *et al.*, 2008). This contrasts with output tracking, where we care not only about the robot's end configuration but also about how it is achieved.

The key to extracting the maximum scientific value is to ensure that the rovers can operate effectively on the surface and be capable of efficiently and safely traversing as much of the terrain as possible. Therefore, the capability to perform output tracking for a desired trajectory is of interest in this work.

There is a plethora of methodologies to control a rover. These approaches range from the simplest to the most complex; (Aström *et al.*, 2014) provide a thorough historical review of the general problem of control and (Garcia *et al.*, 2007) an evolution of robotics research across industrial, service and field applications. Specifically, for mobile robot control developed over the last decades, (Tzafestas, 2018) provides a thorough overview of control methodologies for mobile robots, a selection of relevant books and a summary of survey papers.

So, why would anyone think that they can add something new to this exciting field? To answer this, this thesis takes a somewhat different approach. The developed control methods take the view of obtaining a specific system model (dynamic, kinematic or both), simplifying it when necessary, constructing a control law based on an objective and then attempting to prove that the control law is stable, i.e., it will not lead the system astray but rather to its goal.

In this work, the focus shifts somewhat: a general algorithm that depends only on the model itself is developed and then is applied to the rover model. Within this general framework, a novel guidance and control method is proposed based on Inverse Simulation.

Inverse Simulation is a method that incorporates a mathematical model which is representative of the system and calculates the control inputs necessary to produce a desired, predefined response. The desired response is defined using the system's output variables. Inverse simulation works by taking a model of the system and solving it in a conventional form over a discrete time step. It is a model-based, numerical, iterative process where step changes in the various controls are applied until the system's response matches the desired, predefined response within a certain tolerance (Murray-Smith, 2000; Thomson *et al.*, 2006).

Applications for Inverse Simulation so far have ranged from the aerospace domain and rotorcraft flight control such as (Murray-Smith, 2000; Thomson *et al.*, 2006), to applications to unmanned underwater vehicles such as (Murray-Smith *et al.*, 2008; Murray-Smith, 2014), unmanned aerial vehicles (Murray-Smith *et al.*, 2015), hypersonic vehicles (Forbes-Spyratos *et al.*, 2014) or investigating the handling qualities of a manually controlled rendezvous and docking system (Zhou *et al.*, 2017). An in-depth review is in Chapter 3. Inverse Simulation has been used to produce the required control signals for specific manoeuvres, design and test the feasibility of a desired output, and for model validation and pilot training.

It is, therefore, necessary to present Inverse Simulation not just in the context of a particular application, with all its specific complications, but in a broader, abstracted way.

A key factor is the availability of a suitable mathematical model that is a good approximation of the system. An important aspect of the Inverse Simulation algorithm is that the actuator and rover dynamics are incorporated into the system's mathematical model and are used during the calculation of the control signals. This results in control signals that consider the limitations of the rover and the actuators. The model can also be updated to include degradation to the actuators, power supply, mechanical structure or compensate for any damage that may occur. The control signals generated by Inverse Simulation will compensate for this degradation.

The desired output for Inverse Simulation is a trajectory to a goal destination. First, a trajectory to the destination is determined as a series of waypoints, e.g., from terrain maps or sensor information. This information will provide the desired trajectory for the Inverse Simulation, which in turn will generate the required control inputs to follow the trajectory. The method can be applied in situ given a defined trajectory, the rover can calculate the necessary control inputs; or offline: define the trajectory, the control inputs are calculated, uploaded to the rover, which then executes the trajectory.

Essentially, in Inverse Simulation the system’s model is used to generate a series of control inputs that will drive the system to a desired output, instead of applying an additional external controller. The model incorporates the dynamics, states, inputs, outputs, and hardware limitations of the system and this allows for the consideration of these parameters when calculating a control input to achieve a specified output – a trajectory (path) in the case of rovers. Moreover, because the method depends explicitly on the system model, the model parameters, hardware constraints, actuators and the trajectories, their influence on the control inputs can be investigated to identify problem areas in advance.

The approach in this work differs in that an attempt is made to describe a general algorithm and then apply it to the specific system to obtain a control input. This control input is entirely and exclusively based on the model used, without using any additional external controllers or imposing any simplifications on the existing model – apart from those needed to construct a model in the first place, but that is unavoidable. The method provides the best available inputs for a given system and a desired output, and these inputs are nominal. We are not interested in imposing an external control law on the system; rather we are looking to see if the system as is, can achieve the goal defined in terms of its outputs.

Therefore, Inverse Simulation can be used for guidance by providing the changes in velocity, rotation, and acceleration for following a desired trajectory and for control by using these inputs to execute the desired trajectory. Hence, Inverse Simulation is a method for guidance and control. The Inverse Simulation control inputs are nominal, and, in this work, they are applied to a forward simulation without additional corrections. Thus, their validity depends on the model used and the assumptions made. In the last chapter of this work, two different schemes for using these nominal signals are included in cases where there may be significant changes and unmodelled disturbances that may require additional correction.

1.2 Thesis Aims

The main aims of this thesis are summarised as the following:

- Develop the Inverse Simulation algorithm for the general case, develop its main requirements and establish a theoretical background using two different approaches: Differentiation and Integration.

- Establish and examine the parameters that affect the performance of Inverse Simulation, and the type of solution Inverse Simulation finds, given a desired output and system model.

- Apply Inverse Simulation for output tracking to a four-wheeled rover model.

- Examine the parameters that affect the performance of Inverse Simulation when applied to the rover, within the general framework established in the thesis.

To support these main aims and to place the method within the broader context of GNC, the secondary aims are:

- Establish the current state of rover design and control.

- Establish the current state of Inverse Simulation.

- Provide Inverse Simulation application examples using common mechanical systems.

1.3 Thesis Outline

First, in Chapter 2 a review of the state of the art of rover system design and control methodologies is presented. The system review aims to propose a taxonomy for rover design from which a baseline design emerges. The review provides an overview of the current methods and where Inverse Simulation fits within that paradigm.

Chapter 3 is the review of the existing applications of Inverse Simulation, the two main implementations (Differentiation and Integration), and the application considerations from previous experience.

Following Chapter 3, Chapter 4 builds on this review and examines in depth the two main implementations of Inverse Simulation for the standard state space model, itself an abstraction for a variety of systems. The algorithms are for the general, non-square case of an unequal number of inputs and outputs. There is an in-depth discussion on the parameters that affect the numerical stability of the algorithms and the issue of an unequal number of inputs and outputs. Next, the relation of Inverse Simulation to feedback linearisation is investigated. Both Inverse Simulation and feedback linearisation require the expression of the input in terms of the output and thus an exploration of their differing qualities is important. Then, the linear case of Inverse Simulation is presented, as it showcases certain aspects of the method as to what can be considered an appropriate desired output based on the system model and how the control input can be expressed using the output. Finally, application examples are given to show how the method can be applied. Chapter 4 concludes with a set of tunning recommendations for the general case of Inverse Simulation. While some parameters are by necessity application specific, certain general considerations are highlighted.

Chapter 5 presents the dynamics and kinematics of the rover and the trajectories that will be used as desired outputs for Inverse Simulation.

Having established what Inverse Simulation is and how it works, Chapter 6 applies the Differentiation and the Integration algorithm to the specific case of a four-wheeled rover, using the model and output trajectories from Chapter 5. The goal is to find the inputs to achieve a desired trajectory and several trajectories of varying length and duration are tested. Then, having found these inputs, they are applied in a standard forward simulation to see if they indeed achieve the desired trajectory, thus fulfilling the goal of trajectory tracking. The point is to check that using these inputs the rover can track the trajectory with a very small error, based only on the system model used. The implementation specifics for the Differentiation and Integration are discussed and their comparative performance is investigated.

In Chapter 7, the rover model is augmented with the motor dynamics. The Inverse Simulation algorithms already developed are applied for this new model and it is shown how can this be done, without fundamentally altering the algorithm.

An important element to consider is what type of solution the Inverse Simulation algorithms find and even if there is one to begin with, as well as what parameters affect its numerical performance. In Chapter 3 a review of the stability of Inverse Simulation is presented based on previous applications and in Chapter 4 the general recommendations were discussed. In Chapter 8 a closer look is taken at the specific algorithms applied to the rover and the choices made, based on the tunning recommendations for the general case of Inverse Simulation in Chapter 4.

Finally, in Chapter 9, the main findings are summarised and recommendations and opportunities to further develop Inverse Simulation are presented, including two different schemes for using the Inverse Simulation nominal signals in combination with another control method.

1.4 Contributions

During the research for this thesis, the following publications were made. They are detailed below, together with the relevant chapter that includes their results.

Flessa, T., McGookin, E. W., and Thomson, D. G. (2014) Taxonomy, Systems Review and Performance Metrics of Planetary Exploration Rovers. In: 13th International Conference on Control, Automation, Robotics and Vision (ICARCV'14), Marina Bay Sands, Singapore, 10-12 Dec 2014, pp. 1554-1559. The results are included in Chapter 2.

Worrall, K., Thomson, D., McGookin, E. and Flessa, T. (2015) Autonomous Planetary Rover Control Using Inverse Simulation. In: 13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2015), ESA/ESTEC, Noordwijk, 11-13 May 2015. The results are included in 6, 7, and 8.

Flessa, T., McGookin, E. and Thomson, D. (2016) Numerical Stability of Inverse Simulation Algorithms Applied to Planetary Rover Navigation. In: 24th Mediterranean Conference on Control and Automation (MED 2016), Athens, Greece, 21-24 June 2016, pp. 901-906, The results are included in Chapters 6, 7, and 8.

Flessa, T., McGookin, E., Thomson, D. and Worrall, K. (2016) Numerical Efficiency of Inverse Simulation Methods Applied to a Wheeled Rover. In: 9th EUROSIM Congress on Modelling and Simulation, Oulu, Finland, 12-16 Sep 2016. The results are included in Chapters 6, 7, and 8.

Ireland, M. L., Flessa, T., Thomson, D. and McGookin, E. (2017) Comparison of nonlinear dynamic inversion and inverse simulation. Journal of Guidance, Control, and Dynamics, 40(12), pp. 3304-3309. The results are included in Chapter 4.

Furthermore, the Integration algorithm developed in Chapter 4 was used as a method for fault detection in the following two publications. These results are not included in the thesis, as the fault detection results were primarily developed by the co-authors, but they do showcase the validity of the general algorithm and its more general application for finding inputs given a model and an output.

Ireland, M. L., Worrall, K. J., Mackenzie, R., Flessa, T., McGookin, E. and Thomson, D. (2017) A Comparison of Inverse Simulation-Based Fault Detection in a Simple Robotic Rover with a Traditional Model-Based Method. In: 19th International Conference on Autonomous Robots and Agents (ICARA 2017), Madrid, Spain, 26-27 Mar 2017.

Ireland, M. L., Mackenzie, R., Flessa, T., Worrall, K. J., Thomson, D. G. and McGookin, E. W. (2017) Inverse Simulation as a Tool for Fault Detection and Isolation in Planetary Rovers. In: 10th International ESA Conference on Guidance, Navigation and Control Systems, Salzburg, Austria, 29 May - 02 Jun 2017.

Chapter 2 Review of Rover Systems and Control Methodologies

The literature review focuses on the following two aspects: (a) the state of the art of planetary exploration systems, with an emphasis on rovers and related experimental systems and (b) the state of the art of control methodologies for wheeled systems, whether used on Earth or for planetary exploration.

The review of the state of the art of planetary exploration systems includes those used in missions and selected experimental designs. A general taxonomy of rovers is proposed based on this review. The taxonomy can be used for comparing rover characteristics from a set of possible configurations, thus facilitating a more systematic design process. From this review, the baseline design is proposed, which is then used as an appropriate system for Inverse Simulation in this work.

The review of the GNC methodologies is intended as an overview and focuses on the methods applied to differentially driven wheeled robots, as these are widely used in the research community and achieve high manoeuvrability with decreased complexity (Siegwart *et al.*, 2011). Furthermore, the baseline design proposed in this chapter is driven differentially as well as the rover model used for Inverse Simulation.

2.1 Rover Taxonomy

The autonomous robotic exploration of Mars, the Moon, asteroids, and other celestial bodies is a necessary step for space exploration and the expansion of human presence in space. These robots can take the form of rovers, stationary landers, hoppers, and probes. The most mature locomotion method is wheeled, whereas legged and tracked locomotion are still in an experimental phase for space applications (Yoshida *et al.*, 2008; Siegwart *et al.*, 2011; Mateo Sanguino, 2017). The focus is on wheeled rovers, as these have demonstrated their ability to perform with varying degrees of autonomy robustly and reliably over more than fifty years of active research (Yoshida *et al.*, 2008; Mateo Sanguino, 2017); to date all successful planetary rover missions employ wheels.

Rovers have significant advantages compared with landers and probes: they can traverse different terrain types, slopes and overcome obstacles and so they can explore a large area. This is the concept of the “robotic field geologist”, where a rover was used instead of an astronaut team (Lindemann *et al.*, 2006). Landers, such as NASA’s InSight lander (2018) (*InSight Mission Overview*, 2018), are stationary and their exploration capabilities are limited to the landing site. Hoppers and probes have been used in asteroid landing and sample collection, for example in ESA’S Rosetta mission and Philae lander (2004) (Ulamec *et al.*, 2006, 2016). The complexity of rovers for planetary exploration means that to date there have only been seven successful missions, detailed in Table 2.2.

An exploration rover consists of the following subsystems: (a) instrumentation, (b) communications, (c) on board data handling (OBDH), (d) guidance, navigation and control (GNC), (e) power, (f) thermal, (g) chassis & structures (e.g. camera mast, arm), (h) locomotion including the suspension (Sellers, 2005). A question arises as to how we can categorize the different configurations and what a meaningful baseline design looks like. To this end, a taxonomy is presented in Table 2.1 with regards to locomotion method (mobility type), suspension type, steering system configuration and chassis articulation.

Table 2.1: Rover Taxonomy

Criterion	Type	Example
Mobility	Continuous	Wheeled Tracked Crawling Tumbling
	Discrete	Legged
	Hybrid	Wheels on legs Tracks and wheels Circulating Wheels
Steering Configuration	Wheeled Locomotion	Skid Coordinated (e.g., Ackerman steering) Independent (incl. crab steering) Differential
Suspension	Active/Semi Active	Independent Dynamic
	Passive	Rocker – Bogie Multiple Rockers Multiple Bogies Kinematic (other) Mass Spring Damper
Chassis	Articulation	Articulated (actively or passively controlled) Fixed

2.1.1 Mobility Type

Wheels are very well suited for operating on flat terrain and do not have the control complexity, energy efficiency and power distribution issues associated with legged and tracked vehicles (Siegwart *et al.*, 2011; Nie *et al.*, 2013; Reina *et al.*, 2013). Legs perform better overall at the expense of increased complexity and power requirements (Bartlett *et al.*, 2008; Siegwart *et al.*, 2011; Nie *et al.*, 2013; Reina *et al.*, 2013). Tracks perform better on soft terrain than wheels or legs but have reliability issues and a high power-to-weight ratio (Wong *et al.*, 2006; Bartlett *et al.*, 2008; Siegwart *et al.*, 2011; Nie *et al.*, 2013; Reina *et al.*, 2013). Hybrid systems are still in the experimental stage and include wheels on legs, tracks and wheels and circulating wheels. The most popular hybrid system is the combination of wheels and legs, as it combines the efficiency of wheels with the adaptability on different terrains of legs (Siegwart *et al.*, 2011; Nie *et al.*, 2013). The number of wheels is also an important consideration. Three wheels are the minimum for static stability (Siegwart *et al.*, 2011). The number of wheels is usually four or six; eight or more wheels are cumbersome and difficult to control. Four wheels have reduced motion resistance, power requirements, and design complexity, and can be actuated with as little as two motors. Six wheels are generally better for traversing obstacles, reducing the pressure at each wheel and maintaining a smooth chassis pitch adjustment (Siegwart *et al.*, 2011).

2.1.2 Steering Configuration

An important design aspect is the steering configuration. For planetary exploration rovers, all wheels are usually driven (actuated by a motor), and at least some of them are steered (an additional motor is added to change where the wheel is pointing). In independent (or explicit) steering each wheel is driven and steered with a dedicated motor assembly; this increases the ability of the rover to manoeuvre but also increases the overall complexity (Shamah, 1999). When each wheel is driven and steered, crab steering is achieved: all wheels point in the same direction by the same angle and the rover can move sideways. In skid steering, each set of wheels on the left and right sides of the rover is independently powered and a zero-turn radius is possible (Siegwart *et al.*, 2011). Skidding increases the traction of the robot but requires more power and imposes considerable stress on the chassis and the wheels (Siegwart *et al.*, 2011).

Ackerman steering is the most well-known example of coordinated steering, used in commercial automobiles and the hobby robotics market but requires a turning diameter that is larger than the vehicle, thus making it unsuitable for mobile robot exploration (Siegwart *et al.*, 2011).

A wheeled mobile robot (WMR) with differential steering has the advantage of being able to turn on the spot (Campion *et al.*, 2008; Cook, 2011; Siegwart *et al.*, 2011). The wheels on one side of the robot are controlled and are always actuated with the same speed, in contrast with the skid steering method (Siegwart *et al.*, 2011). By coordinating the two different speeds (left and right wheel speed), the robot can turn on the spot, move in a straight line or move in a circular path (Campion *et al.*, 2008; Cook, 2011; Siegwart *et al.*, 2011). Mobile exploration robots most frequently use this type of steering in practice or experimental designs (Siegwart *et al.*, 2011; Mateo Sanguino, 2017).

The mobility type and steering configuration also influence whether the rover is holonomic or not and this is something that impacts how the rover is controlled, a point that will be discussed in 2.3.1. A system is holonomic if the controllable degrees of freedom are equal to the total degrees of freedom (Siegwart *et al.*, 2011). A differentially driven WMR is always non-holonomic because its controllable degrees of freedom are two (left and right wheel speed), but the total degrees of freedom are three: x, y, and orientation θ using a Cartesian reference frame. In practice, a non-holonomic WMR can achieve any feasible pose x, y, θ but this may require more time and energy than a holonomic WMR because the robot may need to first orient itself and then move in that particular direction (Siegwart *et al.*, 2011). Essentially, the non-holonomic constraints arise from the fact that the robot cannot move sideways (Siegwart *et al.*, 2011). A more general definition is that a non-holonomic system is a non-integrable one (Siegwart *et al.*, 2011): there are kinematic constraints that involve the derivatives of the position variables, and these constraints cannot be integrated to provide a relationship using only the position variables.

2.1.3 Suspension

The suspension system is how the means of locomotion are connected to the rest of the rover and influence how the rover interacts with the terrain. For robots with more than three wheels, a suspension system is normally required to maintain wheel contact with the ground. A passive suspension uses springs and dampers with a predefined damping ratio to absorb the dynamical loads whereas a semi-active suspension has a controllable damper but does not introduce any additional energy into the system (Dixon, 2007; Savaresi *et al.*, 2010). An active suspension uses a powered actuator to actively control the damping ratio; energy is now introduced into the system (Dixon, 2007; Savaresi *et al.*, 2010). In terms of performance, response time and reduction of impulse forces, active suspensions are superior; however, they are costly, complex and require a dedicated power supply (Dixon, 2007; Savaresi *et al.*, 2010).

A further distinction is made between kinematic and dynamic suspensions (Wettergreen *et al.*, 2010; Reina *et al.*, 2013). Dynamic suspensions use springs, torsion tubes, dampers, and high-speed actuators to adjust the damping ratio. These are used when a fast response to comply with the terrain is needed. Kinematic suspensions use freely pivoting joints with unsprung and undamped passive linkages, and they are well suited to slow-moving vehicles.

The speed of a planetary rover is less than[1] $5\ \mathrm{cm/s}$ (Biesiadecki *et al.*, 2007; Mateo Sanguino, 2017) and in planetary mobile robotics, if a suspension is chosen, it will probably be a kinematic one (Reina *et al.*, 2013). In fact, below a speed of about $8\ \mathrm{m/s}$, sprung suspensions are an impediment to mobility since they change the force each wheel exerts on the ground, as obstacles are negotiated (Reina *et al.*, 2013). The simplest approach to suspension suitable for low speeds and not significantly uneven terrain is to utilise the flexibility of the wheel by using a deformable tyre made of soft rubber for the wheel (Siegwart *et al.*, 2011)

The suspension most often used in the successful missions to date is the rocker-bogie (Lindemann *et al.*, 2005; Reina *et al.*, 2013), Figure 2.1. The rocker-bogie is a passive, kinematic suspension that keeps all wheels in contact with the surface at all times and no wheel sinks more than the rest (Lindemann *et al.*, 2005). There are three main components: the rocker, the bogie, and the differential, Figure 2.1. The differential is a passive, motion-reversal joint that constrains the two sides to equal and opposite motion and keeps the rover level. The rocker-bogie

[1] For comparison, the average speed of a common garden snail is 1.3 cm/s.

suspension can be adapted for four wheels by using a rocker and a pivoting joint for each side, connected with a differential. Another variation is the three-bogie system used in the ExoMars Rosalind Franklin rover; each wheel pair is suspended on a pivoted bogie (Patel *et al.*, 2010; Silva *et al.*, 2013).

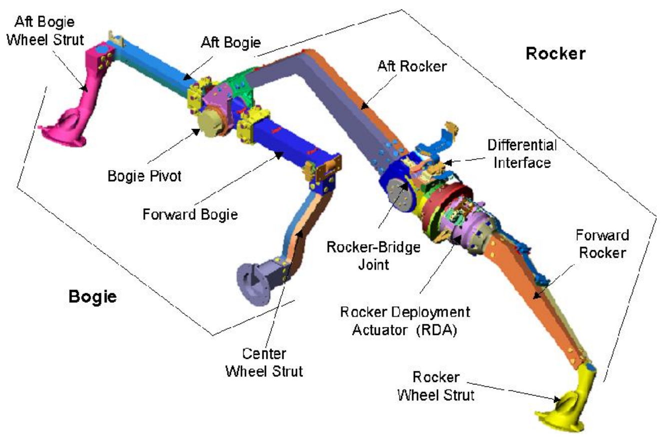


Figure 2.1: Rocker – Bogie Suspension (Lindemann *et al.*, 2005)

2.1.4 Chassis

The ability of the rocker-bogie suspension to maintain the average pitch angle between the two sides is called body averaging. Body averaging is the case where the two chassis sides are connected via a joint or a linkage (active or passive) to maintain the average pitch between them. More generally, active articulation of the chassis transforms the chassis size for different configurations, such as stowing and driving (Rollins *et al.*, 1998; Wagner *et al.*, 2005) or actively lowering the chassis and extending its wheel base when drilling (Bartlett *et al.*, 2008; Wettergreen *et al.*, 2010). Adding actively controlled articulation joints at the chassis increases the complexity and power requirements and therefore this type of design is still limited to experimental systems.

2.2 Analysis of Rover Design

A review of the planetary exploration rovers successfully used (Table 2.2) and selected experimental designs (Table 2.3), focusing on wheeled, legged or hybrid systems, is presented to provide an overview of ongoing research and to highlight the different configurations based on Table 2.1. From Table 2.2 and Table 2.3, a baseline design is then proposed in Section 2.2.3.

2.2.1 Flown Exploration Rovers

Table 2.2 presents all the successful rover exploration missions to date. The values for speed, obstacle height and tilt are the maximum. The first planetary vehicles were the Apollo Lunar Roving Vehicles (1971, 1972) and the first teleoperated rovers were the Lunokhod rovers (1971, 1973) (Young, 2007). Since then, research efforts have mostly focused on developing rovers for Mars exploration: NASA's Sojourner rover (1996) (Muirhead, 2004), Mars Exploration Rovers (MER) Spirit and Opportunity (2003) (Lindemann *et al.*, 2005), Curiosity (2011) (Heverly *et al.*, 2013) and Perseverance (2020), which is based on the design of Curiosity rover with upgraded hardware and new scientific instruments (Chu *et al.*, 2017). ESA's ExoMars Rosalind Franklin rover (Silva *et al.*, 2013) is scheduled for launch in 2022. The Rosalind Franklin rover has a wheel walking ability; the rover can lift each wheel to adjust its attitude and ground clearance and to create a type of walking ability so that the rover can slowly walk out of adverse terrain, e.g. soft soil (Michaud *et al.*, 2008; Silva *et al.*, 2013). In July 2020 China launched the Tianwen-1 mission for Mars, which consists of an orbiter, a lander, and a six-wheeled rover but no further details are available at this point (Mallapaty, 2020) and so it is not included. China has also developed a robotic lunar exploration programme. In December 2013, the rover Yutu ("Jade Rabbit") landed on the Moon (Sun et al., 2013) but the rover was unable to move after the end of the second lunar night. The follow up mission had the same rover design and landed on the far side of the moon in January 2019 (Jia et al., 2018).

A full-scale size comparison of three generations of NASA Mars exploration rovers is in Figure 2.2 and the Rosalind Franklin rover is in Figure 2.3.

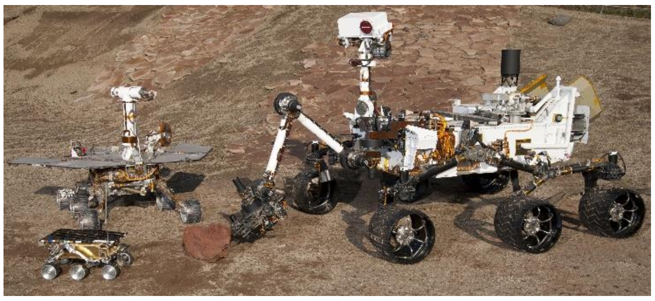


Figure 2.2: Sojourner (F), MER (L), Curiosity (R) (courtesy of NASA)

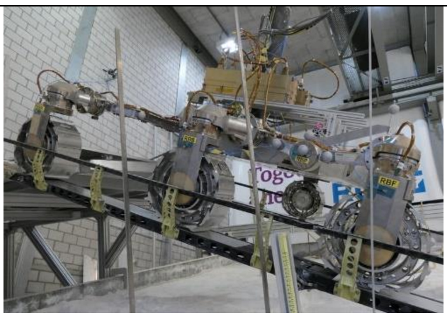


Figure 2.3: Rosalind Franklin Prototype (courtesy of ESA)

In Table 2.2, the maximum speed, obstacle, and tilt may be exceeded in some cases (e.g., level hard ground with high traction). The maximum object height is equal to the wheel diameter when using the rocker-bogie suspension. All rovers in Table 2.2 use wheels and a passive suspension. The maximum speed, permissible obstacle height, maximum tilt, and weight increase over time. In terms of operation and autonomy capabilities, the Apollo Lunar Vehicles were operated in-situ by astronauts (Young, 2007) and the Lunokhod rovers were teleoperated (Yoshida et al., 2008). All other rovers have a degree of autonomy to explore the nearby area and that capability is increased with each mission, especially as the computational efficiency increases (Bajracharya et al., 2008; Correal et al., 2016).

Table 2.2: Planetary Exploration Rovers

Name (Launch)	Institution	kg	Size (m)	Locomotion Steering	Suspension	Speed (cm/s)	Obstacle Height (cm) Tilt (deg)
Apollo LRV (1971, 1972)	NASA	210	1.14(h) x 1.83(w) x 3.1(l)	4 wheels Ø51cm Ackerman Steering	Passive: suspension arms and torsion bars	360	30
Lunokhod (1971, 1973)	NPO Lavochkin	840	1.35(h) x 1.6(w) x 1.7(l)	8 wheels Ø51cm Skid steering	Passive: independent at each wheel	55.5	n/a
Sojourner (1996)	NASA	11	0.3(h) x 0.48(w) x0.65(l)	6 wheels Ø13cm 6 drive 4 steer	Rocker-Bogie	1	13 15
Spirit & Opportunity (2003)	NASA	176	1.5(h) x 1.2(w) x 1.4(l)	6 wheels Ø25cm 6 drive 4 steer	Rocker-Bogie	4.6	26 16
Curiosity (2011)	NASA	900	2.2(h) x 2.8(w) x 2.8(l)	6 wheels Ø50cm 6 drive 4 steer	Rocker-Bogie	4.6	50 28
Yutu (2013, 2019)	CNSA	140	1.5(h) x 1(w) x 1.1(l)	6 wheels 6 drive 4 steer	Rocker-Bogie	5.5	n/a
Perseverance (2020)	NASA	102 5	2.2(h) x 2.7(w) x 3(l)	6 wheels Ø52.5cm 6 drive 4 steer	Rocker-Bogie	4.2	52.5 45
Rosalind Franklin (2022)	ESA	310	2(h) x 1.1(w) x 1.2(l)	6 wheels Ø25cm 6 drive 6 steer	Three Bogies	3.6	25 40

2.2.2 Selected Experimental Designs

Table 2.3 presents selected experimental designs to provide an overview of current research and of the different configurations. The table presents selected experimental designs for planetary exploration rovers to show the design variety, especially in comparison with the flown systems in Table 2.2. Historical reviews such as (Yoshida *et al.*, 2008; Mateo Sanguino, 2017) provide a broad overview of the development efforts for planetary exploration and include results pre-2008.

Table 2.3: Selected Experimental Designs

Name (Year)	Institution	kg	Size (m)	Locomotion Steering	Suspension	Speed (cm/s)
Scarab (2008)	Carnegie Melon University	28	1.2 wheelbase	4 wheels Ø71 cm 4-wheel drive skid steering	Passive: Two Rockers Active body roll control	6
Nanokhod (2008)	ESA	3	0.65(h)x0.16(w)x0.24(l)	Two Tracks skid steering	n/a	0.14
AMALIA (2010) (Della Torre *et al.*, 2010)	Team Italia for Google LunarXPrize	30	n/a	4 wheels Ø21 cm	Passive	n/a
SpaceClimber (2010)	DFKI	23	0.17(h)x0.2(w)x0.85(l)	6 legs 4 DOF each	n/a	17.5
CESAR (2012)	University of Bremen	13.3	0.5(h)x0.82(w)x0.98(l)	2 hybrid legs / wheels	n/a	n/a
ATHLETE (2012)	NASA	300	4(h)x2.75(w)x2.75(l)	6 Hybrid legs / wheels Ø71 cm 6 DOF legs	n/a	83cm/s
Axel (2012)	NASA	40	1.5 m × 0.9 m	2 wheels Ø30 cm	n/a	n/a
Cataglyphis (2015) (Gu *et al.*, 2017)	West Virginia University	65	1.5(h) total volume < 1.5m^3	6 wheels	Rocker Bogie	2
Artemis Jr. (2015) (Reid *et al.*, 2015)	CSA	260kg	1.53(h)x1.62(w)x1.47(l)	4 wheels Ø30 cm 4-wheel drive skid steering	Passive	n/a

SCARAB (Bartlett et al., 2008) is an example of a four-wheeled rover with an actively transforming chassis. The SCARAB design (Bartlett *et al.*, 2008; Wettergreen *et al.*, 2010) combines a passive rocker suspension for pitch adjustment with an active part for chassis transformation, Figure 2.4.

The Nanokhod rover (Schiele *et al.*, 2008) is a small, rugged explorer developed for exploring more environmentally extreme bodies such as Mercury. It is one of the few examples of tracked locomotion, Figure 2.5. To reduce Nanokhod's size, while maintaining its scientific payload, it is tethered to a lander that provides power, communications, control and navigation commands (Schiele *et al.*, 2008).

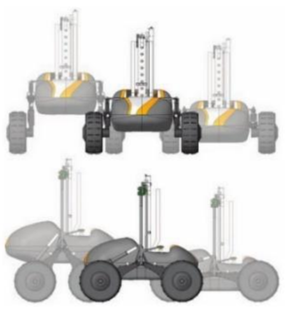


Figure 2.4: SCARAB (Wettergreen *et al.*, 2010)

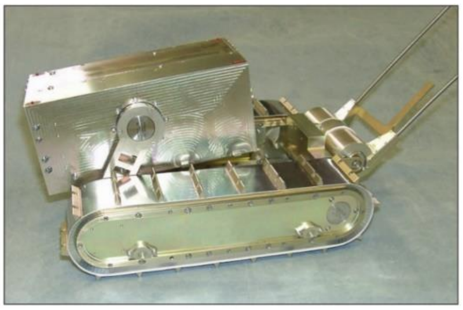


Figure 2.5: Nanokhod model (courtesy of ESA)

Space Climber, Figure 2.6, was designed as a small scout for steep slopes up to 40 deg. It has six legs, each with four DOF, for locomotion and a tether is used for power and commands (Bartsch *et al.*, 2012). CESAR was developed for an ESA lunar crater robotic exploration challenge (Belo *et al.*, 2012), Figure 2.7. It utilises a wheel/leg hybrid and an additional module, a repeater, is used for teleoperation.

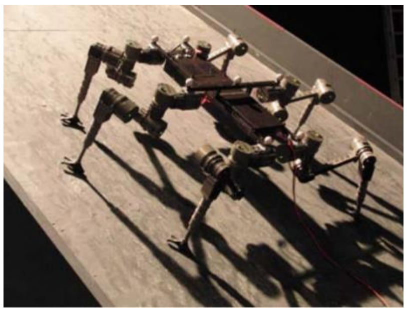


Figure 2.6: SpaceClimber, 25 deg slope (Bartsch *et al.*, 2010)

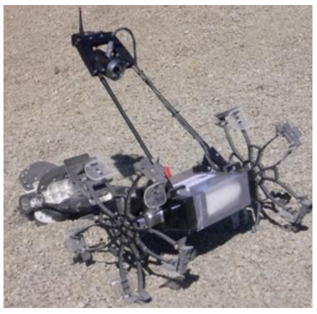


Figure 2.7: CESAR prototype (courtesy of ESA)

ATHLETE in Figure 2.9 is designed for carrying cargo, hence its large size and speed; it is a hexagonal platform on six legs and each leg is individually actuated (SunSpiral *et al.*, 2012; Wilcox, 2012). Each leg has a wheel and the system uses the locked wheels as "feet" to "walk" when in challenging terrain (SunSpiral *et al.*, 2012; Wilcox, 2012). The wheels and their actuators are sized for nominal terrain, which requires less torque and smaller wheel diameter, which results in a mass saving of up to 25% (Wilcox, 2012).

The Axel rover, Figure 2.8, is a two-wheeled tethered robot capable of rappelling down steep slopes and traversing rocky terrain. (Nesnas *et al.*, 2012). Only three actuators are used to control its wheels, caster arm, and tether (Nesnas *et al.*, 2012). Two or more Axel rovers can be combined to form a larger, untethered system (Nesnas *et al.*, 2012).

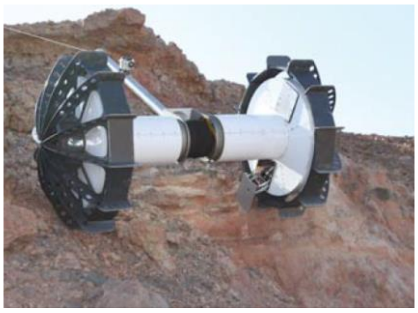


Figure 2.8: Axel Rover (courtesy of NASA/JPL-Caltech)

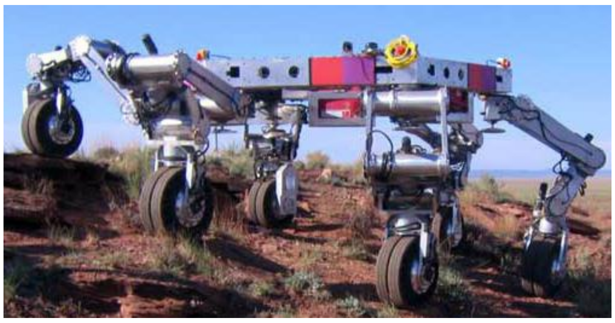


Figure 2.9: Athlete on a Hill (courtesy of Nasa/JPL-Caltech)

The Nanokhod rover, the Axel rover and the Space Climber rover are all small systems that use a tether to connect to the main module. This is a mother-daughter systems configuration (Nesnas *et al.*, 2012). The mother ship handles tasks such as long-range communications and onboard scientific analysis and so the daughter ship is lighter and simpler and is suitable for exploring areas where a larger rover could not go (Nesnas *et al.*, 2012).

The AMALIA, Cataglyphis and Artemis Jr designs all share certain design elements: wheels and a passive suspension. The AMALIA rover is a small, four-wheeled rover, each wheel is directly driven and is connected with an independent suspension to the main body that has a torsional spring but no damper (Della Torre *et al.*, 2010). The Cataglyphis rover (Gu *et al.*, 2017) was equipped with a manipulator for sample collecting and used six wheels and rocker-bogie passive suspension (Gu *et al.*, 2017). The Artemis Jr. rover has six wheels actuated by only two motors for locomotion to reduce complexity, mass and cost; it is skid steered and has a passive suspension (Reid *et al.*, 2015).

Most of the experimental designs in Table 2.3 use skid-steering, in addition to all-wheel drive. When using skid steering, the traction increases, and the rover can better negotiate difficult terrain (e.g., loose soil) and achieve a zero-turn radius. Skidding also requires more power and imposes considerable stress on the chassis and the wheels. In experimental designs, however, there is more freedom to investigate different means of locomotion and steering. For most cases, the suspension of choice (when present), is a passive, kinematic one. A trend is apparent in using hybrid wheel/legged locomotion combining the maturity of wheels with the versatility of legs, as seen in ATHLETE, CESAR, and Rosalind Franklin from Table 2.2 and Table 2.3. Of the eight robots participating in the ESA Lunar challenge (Belo *et al.*, 2012), CESAR was the only one that completed the mission and used a wheel/leg hybrid. The other robots included wheeled, tracked and track/wheel hybrids.

2.2.3 Review Summary and Baseline Design

There is a proliferation of experimental designs that shows the intense research effort over a long period of time, the need to optimise designs, the trial-error methodology adopted and the unique challenges faced (Nie *et al.*, 2013; Mateo Sanguino, 2017). There are several designs for planetary exploration rovers; however, all rovers successfully used in a mission since Sojourner share a similar design: wheels, passive kinematic suspension (using the rocker-bogie configuration), independent all-wheel driving and selected wheel steering. The main difference is the weight, size, and power requirements. With each successive mission and as technology and launch capabilities advance, the scale and demands of the objectives are increased as well as the system’s capabilities. Most experimental designs in Table 2.3 use four wheels, only two use six wheels and just one has two wheels. Furthermore, a survey of over 100 rovers over the past 50 years (Mateo Sanguino, 2017) showed that 31% had six wheels and 29% had four wheels and in terms of steering, 23% had four wheels with differential steering versus 25% that also had four wheels and skid steering,

Therefore, from Table 2.2, Table 2.3 and the subsequent analysis, a rover baseline design emerges: (a) wheeled locomotion using four to six wheels (b) all-wheel drive and selected wheel steering, (c) passive suspension, usually the rocker-bogie system with a differential for steering. Using a passive suspension reduces complexity and increases reliability: fewer actuators and fewer moving parts reduce the chance of mechanical failure and the control requirements. A passive kinematic suspension has the benefits of simplicity, stiffness and a more equal distribution of weight (Lindemann *et al.*, 2005; Bartlett *et al.*, 2008; Wettergreen *et al.*, 2010).

A simpler design, suitable for simulation and experiments would incorporate the following characteristics: (a) 4 wheels, (b) differential drive, and (c) passive suspension. The simplest approach to suspension suitable for not significantly uneven terrain is to design flexibility into the wheel itself, e.g. by using a deformable tyre made of soft rubber for the wheel, and not including any other suspension type (Siegwart *et al.*, 2011). It is this baseline design, without suspension, that will be used further on in this work for applying the Inverse Simulation algorithms, as it represents a good analogy of the actual systems currently used.

2.3 Overview of Guidance, Navigation and Control for wheeled vehicles

2.3.1 Control for Mobile Robots with a Differential Drive

A wheeled mobile robot is described by a dynamic model and a kinematic model (Morin *et al.*, 2008), where: $\mathbf{q}$ is the system's configuration vector in the body-fixed frame, $\boldsymbol{\tau}$ is a vector of independent motor torques, $\mathbf{H}$ is the invertible mass matrix, $\boldsymbol{\Gamma}$ is the combined effect of the Coriolis forces ($\mathbf{C}$), gravitational forces ($\mathbf{G}$), friction and other damping forces ($\mathbf{D}$) and $\mathbf{J}(\boldsymbol{\eta})$ relates $\mathbf{q}$ from the body-fixed frame to $\dot{\boldsymbol{\eta}}$ in the Earth-fixed frame:

$$\boldsymbol{\tau} = \mathbf{H}\dot{\mathbf{q}} + \underbrace{\mathbf{C}(\mathbf{q}) + \mathbf{D}(\mathbf{q})\mathbf{q} + \mathbf{G}(\boldsymbol{\eta})}_{\boldsymbol{\Gamma}} \tag{2.1}$$

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\mathbf{q} \tag{2.2}$$

The system described by Eq. (2.1) and Eq.(2.2) forms a control system; Eq.(2.1) is the dynamic model and Eq.(2.2) is the kinematic model (Morin *et al.*, 2008). The kinematic model on its own is also a control system with $\boldsymbol{\eta}$ being the state vector and $\mathbf{q}$ the control input vector (Morin *et al.*, 2008). The kinematic model is very often used on its own to develop control algorithms for wheeled mobile robots, as it is simpler than the dynamic model, has lower computational requirements, is easier to implement and the resulting controllers exhibit good performance at moderate speeds. An additional reason is that depending on the system, the architecture may not allow the usage of torque or acceleration inputs (Oriolo *et* *al.*, 2002). From a kinematic viewpoint, the control of a differential drive robot is equivalent to that of the unicycle (i.e. a vehicle with only one wheel) (De Luca *et al.*, 2001) and so a lot of the controllers proposed are for unicycles.

There are numerous control methodologies to design the control system, from the simplest to the most complex; (Aström *et al.*, 2014) provide a thorough historical review of the general problem of control and (Garcia *et al.*, 2007) an evolution of robotics research across industrial, service and field applications. Specifically, for mobile robots, this work (Tzafestas, 2018) provides an overview of control methodologies, a selection of relevant books and a summary of survey papers.

For vehicles with non-holonomic constraints, which is the case of a differential driven, wheeled robot, the trajectory following case is relatively simpler to control compared with the point-to-point motion and position stabilisation (i.e., remain at a set point), due to the nature of the non-holonomic kinematics. It was first established by (Brockett, 1983) that point stabilisation is not achievable with continuous, constant feedback control that uses only the states. It can however be achieved by time-varying control laws, discontinuous feedback, or both and the authors of (De Luca *et al.*, 2001; Oriolo *et al.*, 2002; Morin *et al.*, 2008) provide intuitive examples as to how this can be circumvented by using time-varying control laws, discontinuous feedback or both control laws.

At its most fundamental, the control on a plane of a mobile robot is achieved by defining its heading and forward velocity (Cook, 2011). For a differentially driven robot, this is equivalent to controlling the velocity of the left (v_{left}) and of the right side (v_{right}) (Cook, 2011); these two velocities are the system's two control inputs. To achieve a particular heading and position the rover can (a) turn while travelling (i.e. spread the rotation uniformly while moving at the maximum speed) or (b) turn then travel (i.e. turn in place using the maximum rotational speed and then move forward using the maximum speed) (Cook, 2011). The time required to achieve the desired heading and position is shorter for the turn-then-travel strategy and the greater the change in heading is, the faster it is compared to the turn-while-travelling strategy (Cook, 2011). The computed control method (Cook, 2011) assumes that the heading and velocity cannot change instantaneously and the steering system to change them behaves as a second-order system with a specified natural frequency and damping ratio (Cook, 2011).

Two well-known and widely used methodologies to achieve mobile robot control is PID (Proportional Integral Derivative) control and pole placement control. These methods are extensively described in the literature, such as (Skogestad *et al.*, 2005; Ogata, 2008). A common element of all these methods is that they are usually applied to the kinematic model of the robot, without considering the dynamics, and are based on linear control theory. The performance of a PID controller, a pole placement controller and a sliding mode controller when applied to a four-wheeled differential drive robot were compared in (Worrall *et al.*, 2006; Worrall, 2010). The performance of the pole placement controller was superior in terms of the tracking time and error for a given trajectory, but it did require a linearised model of the system (Worrall, 2010).

Overall, the standard controllers used are: proportional plus integral controller and its variations, Lyapunov function-based controller and computed control torque (Tzafestas, 2018). More advanced controllers include feedback linearisation-based controllers, adaptive and robust control (Tzafestas, 2018). Adaptive control is suitable for systems that involve slowly varying parameters or uncertainties/disturbances (Tzafestas, 2018). Robust control, such as sliding mode control, is applied in cases where there are strong parameter variations or uncertainties and can face fast disturbances, variations and unmodelled characteristics (Tzafestas, 2018).

Another overview and comparison of commonly used controllers for path and trajectory following that employ the kinematic model can be found in (De Luca *et al.*, 2001; Morin *et al.*, 2008; Paden *et al.*, 2016). These focus on feedback linearisation control and Lyapunov based control design. One of the first results for trajectory tracking that proposes a control rule for determining the linear and the rotation velocity and proves its stability using a Lyapunov function is (Kanayama *et al.*, 1990).

Feedback linearisation is applied to single-input and output, control affine[2] systems in Eq.(2.3), where $\mathbf{x}$ is the state (m states), u the control input and y the output:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \xi(\mathbf{x})u,\ y = h(\mathbf{x}) \tag{2.3}$$

Feedback linearisation transforms the system to an equivalent linear one, thus allowing the usage of linear control methods. There are two types of feedback linearisation: input-output linearization and input-state linearisation (Slotine *et al.*, 1991; Khalil, 2003).

In input-output linearisation, there is a direct, linear relationship between the output and the input and the state equation is partially linearised (Slotine *et al.*, 1991; Khalil, 2003). In input-state linearisation, the system is fully transformed into an equivalent linear system, i.e. both the state and output equation (Slotine *et al.*, 1991; Khalil, 2003). Non-holonomic systems, however, such as a differentially driven robot, cannot be input-state linearised but can be input-output linearised (Yun *et al.*, 1992), another reason for using the kinematic model.

For single-input and output systems, there is a ready form of input-output linearisation (Slotine *et al.*, 1991; Khalil, 2003). For multi-input and multi-output (MIMO) affine systems, with an equal number of inputs and outputs (square systems) (Slotine *et al.*, 1991; Isidori, 1995, 1999) provide a generalisation, but the process is far more involved. The results for single-input systems have been adapted for differential drive robots and then the trajectory tracking control was solved for the resulting input-output linearised systems, using conventional linear state-feedback control (Tzafestas, 2018).

The method of feedback linearisation (input-output) for trajectory tracking has received particular interest in the robotics community since it transforms the input-output relation to a linear one and the developed controllers exhibit good results (i.e. small errors when following the trajectory and asymptotic stability

[2] A system is control affine when it is linear in the control input, i.e. linear in the action but nonlinear with respect to the state (LaValle, 2006).

proved via Lyapunov theory), such as in (De Luca *et al.*, 2001; Oriolo *et al.*, 2002; Morin *et al.*, 2008; Blažič, 2011; Rodríguez-Seda *et al.*, 2014; Paden *et al.*, 2016).

Lyapunov based controller design is based on the concept of selecting a feedback control $\mathbf{u}$ and then selecting a Lyapunov function[3] V to prove that the closed-loop dynamics are stable, as defined by Lyapunov theory (Slotine *et al.*, 1991; Khalil, 2003). This can be done in two ways for the nonlinear system, where the state is $\mathbf{x} \in \Re^m$ and the input is $\mathbf{u} \in \Re^k$ (Slotine *et al.*, 1991; Khalil, 2003).

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \mathbf{f}(\mathbf{0},\mathbf{0}) &= \mathbf{0} \end{aligned} \tag{2.4}$$

The first technique is to select a control $\mathbf{u}(\mathbf{x})$ and then substitute it so that the closed-loop dynamics are:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}(\mathbf{x})) \tag{2.5}$$

Then, a candidate Lyapunov function $V(\mathbf{x}):\Re^m \to \Re$ is selected and the next step is to prove that the derivative of $V(\mathbf{x})$ along the trajectories of Eq.(2.4) is at least negative semi-definite, thus validating the original choice of $\mathbf{u}(\mathbf{x})$.

$$\dot{V} = \frac{\partial V}{\partial \mathbf{x}}\dot{\mathbf{x}} = \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x},\mathbf{u}) \le 0 \tag{2.6}$$

The second technique selects first a Lyapunov candidate function $V(\mathbf{x})$ and then attempts to find a control law $\mathbf{u}(\mathbf{x})$ so that $V(\mathbf{x})$ is indeed a Lyapunov function.

This process can be applied to essentially all systems in the form of Eq.(2.5) but the real issue is finding the Lyapunov function. There is no general method for doing so, though there are guidelines for constructing the Lyapunov function, such as the variable gradient method (Slotine *et al.*, 1991; Khalil, 2003). None of these

[3] A Lyapunov function $V(\mathbf{x})$ is defined as a function that is continuously differentiable, $V(0)=0$, $V(\mathbf{x})>0\ \forall \mathbf{x} \ne \mathbf{0}$ and $\dot{V}(\mathbf{x}) \le 0$ (Slotine *et al.*, 1991; Khalil, 2003).

methods however guarantee that a suitable $V(\mathbf{x})$ will be found (Slotine *et al.*, 1991; Khalil, 2003). It is a trial-and-error method guided by intuition, experience and physical insights into the system (Slotine *et al.*, 1991; Khalil, 2003). Sometimes it is possible to know in advance that a Lyapunov function exists for that particular system, so at least the search is not hopeless (Khalil, 2003).

An adaptive controller is one whose parameters are variable and there is a mechanism for adjusting these parameters based on signals from the system (Slotine *et al.*, 1991). There are two main approaches for adaptive control: the model reference adaptive control (MRAC) and the self-tuning method (ST) (Slotine *et al.*, 1991). In MRAC control, the parameters are updated so that the tracking errors between the plant output and the reference model output are minimised (Slotine *et al.*, 1991). In ST systems, the parameters are updated to minimise the data fitting error in input-output measurements (Slotine *et al.*, 1991). ST controllers are more flexible compared with MRAC controllers but their stability and convergence are more difficult to guarantee (Slotine *et al.*, 1991). Usually, however, the adaptive control techniques require some linearisation of the dynamics around an operational set-point (Slotine *et al.*, 1991; Tzafestas, 2018).

Sliding mode control (Slotine *et al.*, 1991) was originally applied to a single-input single-output nonlinear system, though it can be extended to multi-input and output systems. Sliding mode control has the benefit of reducing the system to a series of first-order ordinary differential equations and can handle the uncertainty at a trade-off against performance. Examples of sliding mode control methods that use in some form the kinematic and dynamic model for trajectory tracking of two-wheeled mobile robots include (Solea *et al.*, 2009; Hwang *et al.*, 2013; Tian *et al.*, 2014).

Model predictive control (MPC) based methods are used when a high fidelity model is available (kinematic or dynamic) and emergency or aggressive manoeuvres may need to be performed (Martins *et al.*, 2008; Paden *et al.*, 2016; Sharma *et al.*, 2017; Škrjanc *et al.*, 2017). The approach is often to solve the control problem over a short time horizon, apply the control inputs in an open loop and while executing, solve the problem for the next time interval (Paden *et al.*, 2016). MPC as a control technique requires the solution of a (convex) quadratic program at each step. For this reason, MPC methods are computationally expensive to perform online and model linearisation is often used in an attempt to simplify the process (Paden *et al.*, 2016; Sharma *et al.*, 2017; Škrjanc *et al.*, 2017). As the computational capabilities of the onboard CPUs improve, the use of MPC methods is also expected to increase over time.

2.3.2 GNC for Planetary Rovers

While a planetary rover is essentially a mobile robot and thus the control methods are those presented in the previous section, there are several very specific challenges for a rover operating on a distant planet: time delays, uncertainty over the terrain type and varied terrain (e.g., loose sand, hard soil, rocks, varying slopes), unknown dynamic environment, limited communication bandwidth and high latency, minimum or zero capability of ground control intervention (Quadrelli *et al.*, 2015; Correal *et al.*, 2016).

The Apollo LRVs were operated by astronauts and the Lunokhods were teleoperated; Earth operators sent driving commands in real-time. All other rovers have autonomous navigation capabilities: Earth operators upload instructions for the rover to follow and the rover can also plot its path and place its instruments on a selected target using the onboard navigation software (Bajracharya *et al.*, 2008; Correal *et al.*, 2016). Early systems devoted most of their power to computing rather than to mobility (Yoshida *et al.*, 2008). Later systems have a more balanced approach and future systems will likely devote most of their power to mobility (Yoshida *et al.*, 2008).

The control of the successful NASA planetary rovers (Table 2.2) is achieved using a combination of non-, semi-, and fully autonomous operating modes. For the non-autonomous mode (directed driving), operators upload instructions for the rover to follow and then evaluate the results (Bajracharya *et al.*, 2008; Correal *et al.*, 2016). The distance travelled can be over $100\,\mathrm{m}$, with limited correction for errors and uncertainties while driving (Wright *et al.*, 2006; Biesiadecki *et al.*, 2007; Bajracharya *et al.*, 2008).

In the case of the semi- and fully autonomous modes, the rover can be given a target and a path made up of a series of waypoints. The onboard navigation software is then used to follow the path and overcome any unforeseen obstacles (Bajracharya *et al.*, 2008; Correal *et al.*, 2016). The rover can also analyse terrain images, detect hazards, and plot its path for a limited number of motions using the onboard navigation software. Semi-autonomous is when the rover is commanded to travel to a particular destination and the on-board systems evaluate the path and control inputs to navigate to the destination (M. W. Maimone *et al.*, 2006; Biesiadecki *et al.*, 2007; Maimone *et al.*, 2007; Bajracharya *et al.*, 2008; Woods *et al.*, 2014; Arvidson *et al.*, 2017).

When in fully autonomous mode, the rover receives very limited commands and selects areas of interest on its own, based on existing parameters. In this case, the rover must select a target destination, choose suitable waypoints by analysing stereo images, plot a safe path and finally execute the movement. This is a very slow process, necessary to ensure the rover's safety (Maimone *et al.*, 2006; Biesiadecki *et al.*, 2007; Bajracharya *et al.*, 2008; Quadrelli *et al.*, 2015; Correal *et al.*, 2016).

An overview of the mobility trends for the Curiosity rover during its first seven years of operation is in (Rankin *et al.*, 2020). The average drive distance is $28.9\ \mathrm{m}$ and the total distance travelled is $21{,}318.5\ \mathrm{m}$ (Rankin *et al.*, 2020). Curiosity attempted 738 drives, of which 622 drives were completed successfully and 116 drives were stopped by the onboard fault-protection software (Rankin *et al.*, 2020). The maximum rotation rate of each wheel is $0.168\ \mathrm{rad/s}$, which is equivalent to a maximum speed of $4.2\ \mathrm{m/s}$, though the wheel speed can vary depending on the terrain (Rankin *et al.*, 2020). The Mars 2020 Perseverance rover also utilises similar modes of operation (Barfoot *et al.*, 2011; Correal *et al.*, 2016).

The designs developed for the ESA ExoMars Rosalind Franklin rover have addressed the issue of control and the degree of its autonomy by including an element of autonomous control within the guidance paradigm (Silva *et al.*, 2013; Woods *et al.*, 2014; Correal *et al.*, 2016). The Rosalind Franklin rover has two main navigation functions: one for short range and one for long range (Silva *et al.*, 2013). For the short range, the rover drives from one waypoint to the next without updating its information (Silva *et al.*, 2013). For the long range, the rover has moved several waypoints but still utilises past acquired information to refine its movement (Silva *et al.*, 2013). There is also a differentiation between normal traverse and drill placement, due to the specific requirements of drilling (Silva *et al.*, 2013).

Allowing the rover to determine its path to a user-specified destination is more efficient as the motion of the vehicle is controlled locally and does not rely on continuous commands being received remotely (Maimone *et al.*, 2006; Biesiadecki *et al.*, 2007; Maimone *et al.*, 2007; Bajracharya *et al.*, 2008). Considering the unknown environment and the time delays, it takes a significant amount of time to implement manoeuvres in advance and drive the rover to points of interest. However, the resulting systems are more complex and need additional onboard power and computational capabilities (Madison *et al.*, 2007; Barfoot *et al.*, 2011; Howard *et al.*, 2012; Correal *et al.*, 2016). They also depend on the accuracy of the method used to select and execute the rover's trajectory (Maimone *et al.*, 2006; Biesiadecki *et al.*, 2007; Bajracharya *et al.*, 2008; Quadrelli *et al.*, 2015), which may accumulate significant error over time.

2.3.3 Trajectory Generation

A necessary element of GNC is motion planning: the generation of the desired trajectory that will enable the robot to reach its goal destination efficiently and safely (Siegwart *et al.*, 2011). Motion planning provides the reference condition that will be fed into the guidance and then into the control system (Howard *et al.*, 2007; Siegwart *et al.*, 2011; Quadrelli *et al.*, 2015; Paden *et al.*, 2016; Wolek *et al.*, 2017). The authors of (Howard *et al.*, 2007) define trajectory generation for mobile robots as the problem of finding a feasible motion that will permit to robot to move from an initial state to a final state, given some model and a number of constraints. This results in a nonlinear differential equation describing an optimal control problem that is often solved using numerical and optimisation methods. The methods for solving the trajectory problem can be broadly categorised as (a) algorithms that produce a motion assuming a flat terrain and (b) algorithms that produce a discrete group of motions over a rough terrain, without any or few connecting in between motions (Howard *et al.*, 2007). This is very much an open research field, and an overview is provided here for context.

In general, trajectory (path) generation starts from a selection of waypoints and each waypoint is defined using Cartesian coordinates x-y-z and they represent the safe points through which the robot must pass (Fossen, 2011). These points form a group of waypoints that consists of:

$$wpt = \{(x_0, y_0, z_0), (x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)\} \tag{2.7}$$

From this group, a trajectory (path) must be generated that connects one waypoint to the next. To do so a curve must be fitted, and the simplest case is that of connecting straight lines and circular arcs, Figure 2.10. In this case, a straight line connects two waypoints (x_{i-1}, y_{i-1}), (x_i, y_i) and for turning, a circle of radius R_i is inscribed between two successive straight lines to form a curved path (Fossen, 2011). This choice is motivated by the result (Dubins, 1957) that "The shortest path with minimum time between two configurations of a particle moving with a constant forward speed is a path formed by straight lines and circular arc segments". A path that consists of straight lines and circular arcs is called a Dubins path (Fossen, 2011).

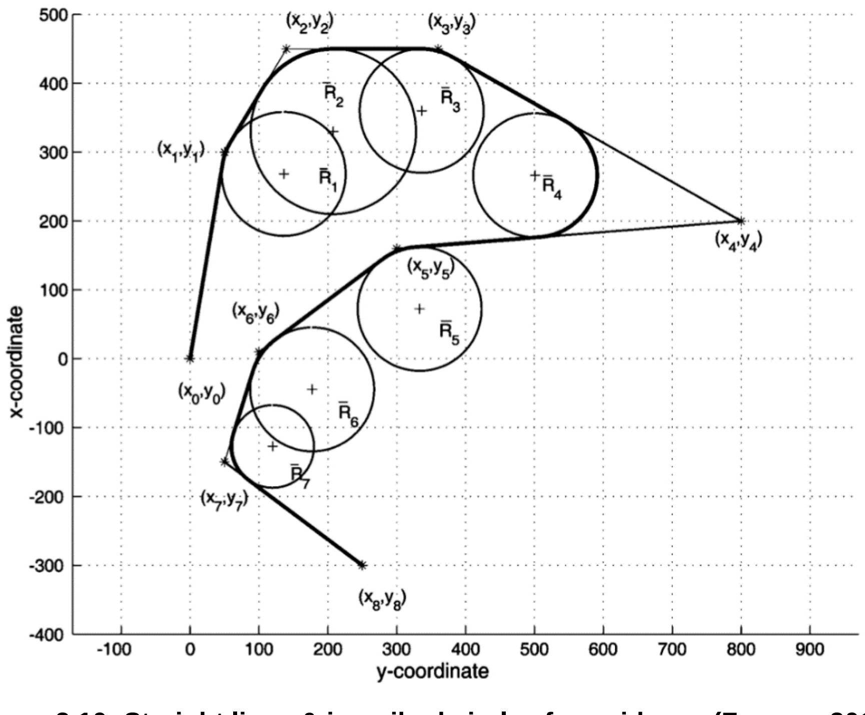


Figure 2.10: Straight lines & inscribed circles for guidance (Fossen, 2011)

There are two main drawbacks to this method (Howard *et al.*, 2007; Fossen, 2011). First, the desired yaw rate is zero along the straight line and constant (non-zero) on the circle arc; thus, the yaw derivative is not continuous during the transition from the straight line to the circular arc. Second, the arc connecting two successive waypoints may not necessarily pass through the two waypoints but in their vicinity; this may or may not be acceptable depending on the application.

If a continuous curve is desired, then a suitable method must be selected that ensures this, such as using a polynomial, splines (e.g. cubic splines, Hermite polynomials) or Bezier curves (Moler, 2004; Howard *et al.*, 2007; Fossen, 2011; Lekkas *et al.*, 2014). A Bezier curve and its generalisation (B-splines) that is fitted through n waypoints is only guaranteed to pass through the first ($i=0$) and last waypoint ($i=n$) (Moler, 2004; Lekkas *et al.*, 2014).

There is a unique polynomial in x of a degree less than n, that passes through all n points defined by Eq.(2.7) (Moler, 2004). The number of data points is also the number of coefficients, although some of the leading coefficients might be zero, so the degree might be less than n (Moler, 2004). There are many different formulas for the polynomial, but they all define the same function. This polynomial is called the interpolating polynomial because it exactly reproduces the given data (Moler, 2004). The more data points that are used in the interpolation, the higher the degree of the resulting polynomial. Such a polynomial tends to show excessive variations between waypoints and overshoots them while guaranteeing that the graph will pass through each waypoint used (Moler, 2004). There is a trade-off between a good fit and a smooth, well-behaved fitting function. For this reason, full degree polynomial interpolation for a large number of waypoints (usually more than five) is hardly used in practice (Moler, 2004). Common methods of fitting to n points a unique polynomial of degree n-1 are Newton's Interpolating Polynomials and Lagrange Interpolating Polynomials (Chapra *et al.*, 2001; Moler, 2004).

A way to solve this problem of a high degree polynomial through multiple waypoints is to fit the polynomial through every few waypoints and use appropriate boundary continuity conditions. This process is known as piecewise interpolation can be done using linear interpolation, splines and Bezier curves as well as other suitable, higher degree polynomials (Moler, 2004).

For a planar moving robot, the faster way to travel through several waypoints is a path of connecting straight lines and circular arcs (Cook, 2011; Wolek *et al.*, 2017). In the ideal case where the forward speed is fixed and there are few disturbances, the fastest path is the Dubins path already described (Wolek *et al.*, 2017). In practice, the robot cannot move forward or turn always at a constant speed and there are speed limits on the surge and yaw velocity, therefore the Dubins path with a variable speed is constructed (Wolek *et al.*, 2017). In that case, a forward speed profile is assigned to the straight segment and a yaw speed profile to the turning segment. Additionally, a robot that is differentially driven can turn on the spot. There are two contrasting strategies for moving between waypoints. The first is the turn while travelling method (Cook, 2011): the rotation is spread uniformly over the entire trajectory while travelling at a constant forward speed. The second is the turn then travel or stop and turn method (Cook, 2011): the robot turns on the spot to achieve the desired heading and then travels forwards to the next waypoint, i.e. a Dubins path, with a circular arc of zero radius. The time required to achieve the desired heading and position is smaller for the turn-then-travel strategy and the greater the change in heading is, the faster it is compared to the turn-while-travelling strategy (Cook, 2011).

Beyond considerations such as the shortest path and minimum time to travel, other issues come into play when designing a trajectory (Fossen, 2011; Siegwart *et al.*, 2011): obstacle avoidance, energy use minimisation, onboard computational power constraints, environmental disturbances, speed limits, curvature limits. Planetary rovers face additional challenges: time delays, uncertainty over the terrain type and varied terrain (e.g., loose sand, hard soil, rocks, varying slopes), unknown dynamic environment, limited communication bandwidth and high latency, minimum or zero capability of ground control intervention through teleoperation (Quadrelli *et al.*, 2015; Correal *et al.*, 2016). A historical review of trajectory (or path) generation methods and challenges for mobile robots can be found in (Howard *et al.*, 2007; Paden *et al.*, 2016) and specifically for planetary mobile robots in (Quadrelli *et al.*, 2015; Correal *et al.*, 2016). The question of how to design a suitable trajectory from a set of waypoints is very specific to the task at hand and the required application, as the next examples from the literature show.

The authors of (Yongguo Mei *et al.*, 2006), develop a power model for a mobile robot and then examine the best way to manage the speed of a group of robots to maximise the travelling distance, under time and energy constraints. The authors of (Connors *et al.*, 2007) use splines to generate a trajectory that avoids obstacles in a cluttered environment. In (Liu *et al.*, 2011, 2014), the authors propose an algorithm that generates a Bezier curve between every two successive waypoints, with appropriate continuity conditions, that is optimised for energy consumption, based on the arrival time and velocity at each waypoint. Using monotone cubic Hermite splines (Lekkas *et al.*, 2013, 2014), a path between successive waypoints is interpolated, allowing for better shape control and avoiding excessive variations between waypoints. A real-time smooth trajectory generation based on piecewise Bezier curves for obstacle avoidance is proposed in (Renny Simba *et al.*, 2016), albeit for a teleoperated robot in an indoor environment. (Kolter *et al.*, 2009) develop an algorithm for designing a smooth trajectory using cubic splines optimisation. The method optimises the initial waypoint positions (within a user-specified limit), while also obeying additional constraints on velocity, acceleration, kinematics and obstacles and was used for designing the trajectory of a four-legged robot (Kolter *et al.*, 2009). All these methods were developed under the assumption of planar movements.

All of the current NASA Mars Rovers (Spirit & Opportunity, Curiosity, Perseverance) use the same algorithm for computing a path, the GESTALT local planner (Grid-based Estimation of Surface Traversability Applied to Local Terrain), designed by NASA/JPL (Biesiadecki *et al.*, 2006; M. W. Maimone *et al.*, 2006; Carsten *et al.*, 2009; Correal *et al.*, 2016). The planning strategy consists of a set of routines that decide the next best direction for a rover to move, given environment & terrain sensor data and the final location or waypoint (Wright *et al.*, 2006; Carsten *et al.*, 2009; Correal *et al.*, 2016; Rankin *et al.*, 2020). A set of candidate arcs (short paths from the current rover location) is then produced and considered. These paths can be straight or arc trajectories, depending on mechanical characteristics and mission constraints (Wright *et al.*, 2006; Carsten *et al.*, 2009; Correal *et al.*, 2016; Rankin *et al.*, 2020). Nominally, the arc set consists of forward and backward arcs of varying curvature, as well as point turns to a variety of headings (Wright *et al.*, 2006; Carsten *et al.*, 2009; Correal *et al.*, 2016). Each arc is evaluated based on three criteria: avoiding hazards, minimizing steering time, and reaching the goal (Wright *et al.*, 2006; Carsten *et al.*, 2009; Correal *et al.*, 2016; Rankin *et al.*, 2020). The algorithm then chooses from among the safe arcs one that will best ensure the rover reaches the goal (Wright *et al.*, 2006; Carsten *et al.*, 2009; Correal *et al.*, 2016; Rankin *et al.*, 2020).

2.3.4 Review Summary

From the control methods presented in this chapter, there is a need for incorporating the dynamic model into the controller for increased accuracy. Methods that use the model are limited by issues such as linearity, affine in the control, number of inputs and outputs and non-holonomic constraints.

Overall, rovers drive from one waypoint to the next and for shorter traverses of a few meters, the rover can move autonomously, in the context already described. The two main strategies can be summed up as the following (Correal *et al.*, 2016). The rover can go to a given location by executing a pre-defined path without any corrections (Correal *et al.*, 2016), similarly to open-loop control (Silva *et al.*, 2013). The rover can also move autonomously to a given location by sensing the environment and making its own decision (Correal *et al.*, 2016). Finally, there are hybrids of these two strategies, such as having the first part of the trajectory already planned and the remaining distance travelled autonomously (Correal *et al.*, 2016). For the Curiosity rover, the average drive distance is $28.9\ \mathrm{m}$, the shortest drive is $2.6\ \mathrm{cm}$ and the longest is $142.5\ \mathrm{m}$ (Rankin *et al.*, 2020).

Chapter 3 Review of Inverse Simulation

In this chapter, a review of the existing applications of Inverse Simulation, the two main implementations, and the application considerations are presented.

Inverse Simulation is a method that uses a representative model of the system and calculates the control inputs necessary to produce the desired response. This desired response is defined in terms of the system's output variables, represented as a time history. It is a model-based, numerical, iterative method, where step changes in the various controls are applied until the response matches the desired outcome within the desired tolerance (Murray-Smith, 2000; Thomson *et al.*, 2006). The inverse simulation techniques can be potentially used in almost all areas where the direct simulation[4] technique is applicable. Then, inverse simulation can be defined as a process where computer simulation methods are used to find a set of unknown input variables that realize a set of known and required model output responses (Du, 2013).

Inverse Simulation has two main requirements for its operation: the desired output and a suitable model of the system. A general nonlinear system can be used with m state equations, p output equations and k control inputs in the usual state-space form. The control inputs calculated from Inverse Simulation are nominal for the given model and output. Accordingly, the output from Inverse Simulation and the corresponding states are also the nominal ones for the given model and desired output. Essentially, the system dynamics – in the form of a representative model – and desired outputs are used to drive the system to the desired output, instead of applying an additional external controller.

Inverse Simulation considers how the system responds over the course of the complete output manoeuvre and there is an emphasis on how the desired response is achieved. This is especially pertinent to the case of nonlinear systems, where there might be singularities along the course of the manoeuvre and the nonlinear mathematical model can only be solved numerically.

[4] This is the forward process where the behaviour (simulation output) of a system under design is found from a given model structure and a set of model input variables

Compared with other methods for nonlinear system inversion, such as feedback linearisation, Inverse Simulation numerically inverts the system model through a series of discretised time points (e.g., using the Newton-Raphson method) whereas conventional methods invert the model in advance and have the advantage of being suitable for a wide variety of systems with model discontinuities, discontinuities in the manoeuvres, control and saturation constraints, non-minimum phase systems and systems not affine in the control (Murray-Smith, 2000; Thomson *et al.*, 2006; Lu *et al.*, 2008; Ireland *et al.*, 2017).

3.1 Review of Inverse Simulation Applications

Applications of Inverse Simulation are so far predominantly within the aerospace domain. The application to rotorcraft flight control, compound helicopters and more generally aircraft control has been a major area of research (Hess *et al.*, 1991, 1993; Rutherford *et al.*, 1996; Murray-Smith, 2000; Avanzini *et al.*, 2001, 2013; Bottasso *et al.*, 2001; Su *et al.*, 2002; Thomson *et al.*, 2006; Lu *et al.*, 2007; Karelahti *et al.*, 2008; Blajer *et al.*, 2009; Bagiev *et al.*, 2012; Ferguson *et al.*, 2016; Kim *et al.*, 2020). Inverse Simulation is used in these cases to (a) produce the required control inputs for specific flight manoeuvres, (b) investigate the handling qualities of the system, (c) investigate whether these manoeuvres are achievable, and (d) investigate the control commands the pilot must and for pilot training. Inverse Simulation has also been used as a model validation method (Bradley *et al.*, 1990; Thomson *et al.*, 1990; Gray, 1992; Rutherford *et al.*, 1996; Murray-Smith, 2000; Avanzini *et al.*, 2010, 2017).

The method has also been applied to autonomous underwater vehicles (Murray-Smith *et al.*, 2008; Murray-Smith, 2014); unmanned aerial vehicles as the design basis of flight control systems (Murray-Smith *et al.*, 2015); for the design and trajectory evaluation of hypersonic vehicles (Forbes-Spyratos *et al.*, 2014); for developing a method to estimate the fatigue of aircraft (Öström, 2007); for evaluating the proper control actions to achieve an orbit change of a plane (de Divitiis, 1999) and for investigating the handling qualities of a manually controlled rendezvous and docking system (Zhou *et al.*, 2017). Finally, Inverse Simulation has also been used for fault detection using the nominal control inputs calculated from Inverse Simulation (Ireland *et al.*, 2017a; Ireland *et al* 2017b).

From this review, Inverse Simulation applications can be categorized into two main categories. First, is the a priori evaluation of the required control inputs for a given, desired output. This process finds the time histories of input variables (i.e., the control inputs) that, for a given model, match the desired output response. When using a dynamic and kinematic model, such as the general described in Eq.(2.1) and Eq.(2.2), the control inputs found are the forces or torques needed to produce desired motions. This approach can also be used to design and test the feasibility of the desired output and to investigate the model parameters.

Second, as a method of validation and fault detection. In this case, the actual system output is passed on to the Inverse Simulation scheme, which in turn produces the corresponding control input. Then, the inputs from Inverse Simulation can then be used to reconstruct the actual system input signals.

To deal with unforeseen disturbances (Bagiev *et al.*, 2012) use a receding horizon predictive approach for applications involving aggressive helicopter manoeuvres. Another approach is the work by (Avanzini *et al.*, 2013), which introduces an element of adaptability to the Inverse Simulation algorithm applied to rotorcraft control based on a model predictive control scheme. In (Du, 2013) an optimisation approach is developed that given a set of output variables, along with the distributions of a set of uncertain input variables, the distributions of the unknown input variables are obtained. This method is then applied to an impact problem involving two rigid bodies to perform traffic accident reconstruction (Du, 2013).

There are two main implementations for Inverse Simulation: Differentiation (Thomson, 1987; Bradley *et al.*, 1990; Thomson *et al.*, 1990, 2006; Rutherford *et al.*, 1996; Murray-Smith, 2000; Lu *et al.*, 2008; Murray-Smith *et al.*, 2015) and Integration (Hess *et al.*, 1991, 1993; Rutherford *et al.*, 1996; Murray-Smith, 2000; Su *et al.*, 2002; Thomson *et al.*, 2006; Karelahti *et al.*, 2008; Lu *et al.*, 2008; Blajer *et al.*, 2009; Du, 2013; Forbes-Spyratos *et al.*, 2014; Kim *et al.*, 2020). As can been for these examples, the Integration method is the more popular of the two.

The underlying algorithm (Figure 3.1) is similar for both methods but the method of convergence to the control input is different.

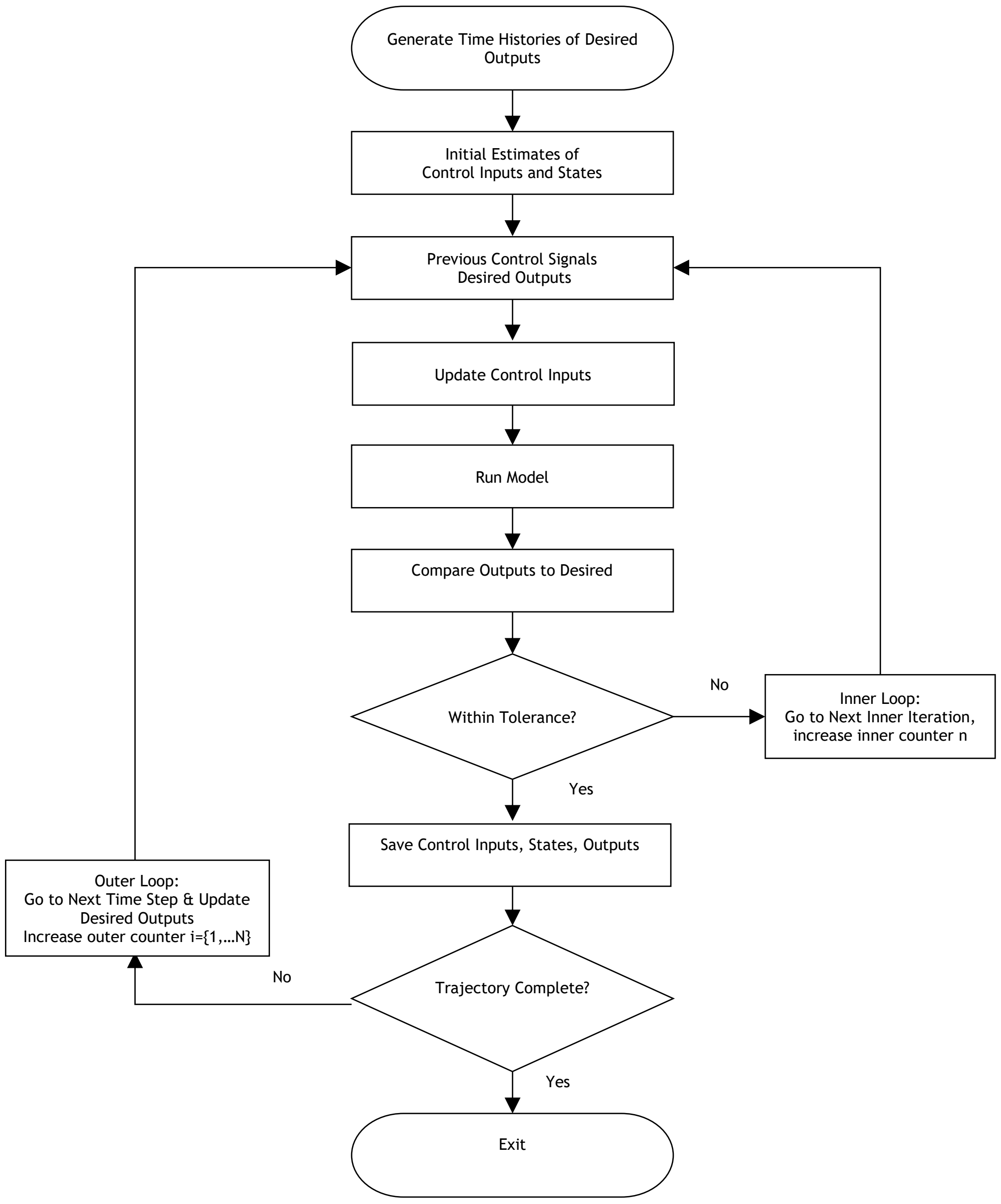

Generate Time Histories of Desired Outputs
Initial Estimates of Control Inputs and States
Previous Control Signals Desired Outputs
Update Control Inputs
Run Model
Compare Outputs to Desired
Within Tolerance?
No
Inner Loop: Go to Next Inner Iteration, increase inner counter n
Yes
Save Control Inputs, States, Outputs
Outer Loop: Go to Next Time Step & Update Desired Outputs Increase outer counter i={1,...N}
No
Trajectory Complete?
Yes
Exit

Figure 3.1: Inverse Simulation Flowchart

The basic Inverse Simulation algorithm in Figure 3.1 employs a Newton-Raphson scheme and a Jacobian that can be approximate (e.g., based on finite differences) or more rarely analytical for solving the nonlinear model equations and computing the outputs, which are then compared with the desired. It runs through a time interval that is discretised N times using the time step dt; this is the outer loop. The inner loop runs until convergence is achieved based on pre-defined tolerance or until the maximum number of iterations has been exceeded (see also Appendix B, and C for the algorithm steps).

For the Differentiation method, a numerical differentiation scheme is used, and the convergence is based on the system's state and output equations. The acceleration terms in the equations of motion are estimated by backward differencing (or another appropriate numerical differentiation formula) and the differential equations effectively become algebraic. The Newton-Raphson algorithm then uses these equations directly in the error function rather than the difference between desired and actual response used in Integration methods. For the Integration method, a numerical integration scheme is used, and the convergence is based on whether the system's output matches the desired.

For both methods, existing research has been mostly restricted to the case of equal numbers of inputs and outputs because the Jacobian is a square matrix and this simplifies the calculations (Hess *et al.*, 1991, 1993; Murray-Smith, 2000; Thomson *et al.*, 2006; Avanzini *et al.*, 2017). When there are more outputs than inputs, this is an under-actuated system and a factorisation can be used for the Jacobian (Hess *et al.*, 1991, 1993; Thomson *et al.*, 2006).

Since the Differentiation method needs the states and outputs, the model must be in a suitable mathematical description where both are readily available. For the Integration method, the model can be a mathematical model or a grey or black box model if the outputs remain the same. This is because there are three main model types. A white-box model is a model based on physical laws with corresponding physical parameters (Keesman, 2011). If some of these parameters are estimated from the data, this is then a grey box model (Keesman, 2011). A black-box model is viewed only in terms of its inputs and outputs (Keesman, 2011). For example, in control applications, a black-box model may be a linear model, which does not necessarily refer to the underlying physical laws and relationships of the system (Keesman, 2011). For both methods, convergence to a point is dependent on the tolerance value set within the Inverse Simulation algorithm. A control input is accepted if the output is within the defined convergence tolerance. This tolerance is dependent on the actual capabilities of the system and is typically set to ensure convergence. If convergence is not reached after a set number of iterations, then this is an indication that the system has either been set up with inadequate parameters or the system is unable to achieve the current set trajectory.

A variation on the Integration method that uses the derivative-free Nelder-Mead search-based optimisation algorithm applied to models used in ship steering control can be found in (Lu *et al.*, 2008) and an approach based on sensitivity analysis applied to a helicopter model is in (Lu *et al.*, 2007). Furthermore, (Celi, 2000) presents an Inverse Simulation approach based on numerical optimisation, which operates on a family of desired trajectories for a helicopter model.

Overall, the Integration method based on (Hess *et al.*, 1991, 1993) is the most widely used for Inverse Simulation. It requires only the inputs and outputs of the system and thus is suitable for complex models and it can also handle more easily problems where the number of inputs exceeds the number of outputs (Rutherford *et al.*, 1996; Murray-Smith, 2000; Thomson *et al.*, 2006; Lu *et al.*, 2008; Avanzini *et al.*, 2017). Nonetheless, the Differentiation method may be more complex to set up but it also converges faster to the required control inputs (Rutherford *et al.*, 1996; Murray-Smith, 2000; Thomson *et al.*, 2006; Lu *et al.*, 2008).

3.2 Review of Application Considerations of Inverse Simulation Algorithms

This section presents an overview of the key results for the application and stability of Inverse Simulation based on the literature review in 3.1

Integration can use any representative model of the system if the outputs and inputs remain the same. Differentiation requires both the states and the outputs, so any change in the model needs a reformulation of the algorithm. This results in Differentiation being more model specific and more time consuming to set up and maintain when changing the model. For the Integration method, the model can be more easily modified, if the outputs remain the same, which is a significant advantage and enables the use of a grey or black-box model.

The issue of the stability of Inverse Simulation is examined in (Lin *et al.*, 1995; Rutherford *et al.*, 1996; Murray-Smith, 2000; Thomson *et al.*, 2006; Lu *et al.*, 2007, 2008; Lu *et al.*, 2007), with these references also providing a discussion on parameter selection, desired output selection and application examples primarily in selected systems from the flight dynamics domain and ship steering control in the case of (Lu *et al.*, 2007). Each algorithm has advantages and disadvantages, detailed next based on (Lin *et al.*, 1995; Rutherford *et al.*, 1996; Murray-Smith, 2000; Thomson *et al.*, 2006; Lu *et al.*, 2007, 2008).

The Integration method converges using only the outputs and so has a convergence rate that is an order of magnitude slower than that of the Differentiation method (Hess *et al.*, 1993; Murray-Smith, 2000; Thomson *et al.*, 2006; Lu *et al.*, 2008). Integration however is generally more stable than Differentiation; what is gained in flexibility and stability, is lost in computing time. Moreover, the Integration method produces smoother control signals (Hess *et al.*, 1993; Murray-Smith, 2000; Thomson *et al.*, 2006; Lu *et al.*, 2008).

The numerical properties of both Differentiation and Integration have been examined when applied mostly to flight dynamics (Hess *et al.*, 1993; Lin *et al.*, 1995; Rutherford *et al.*, 1996; Thomson *et al.*, 2006; Lu *et al.*, 2008). The authors of (Thomson *et al.*, 2006) examine the stability properties of the method in this context. When using Differentiation, it has been observed that there are oscillations in the response of the uncontrolled states (so-called "constraint oscillations") (Thomson *et al.*, 2006). However, these oscillations depend more on the dynamical properties of the system and its uncontrollable states and zero dynamics rather than the method used and its numerical properties (Lu *et al.*, 2008). It has also been observed that there are low amplitude, high frequency oscillations superimposed on the calculated control input. These oscillations are due to several reasons (Hess *et al.*, 1993; Thomson *et al.*, 2006; Lu *et al.*, 2008): redundancy issues, non-square Jacobian and multiple solutions, several local minima of the error function Eq.(4.6) or Eq.(4.10). The authors of (Lu *et al.*, 2008) propose for the Integration case a derivative-free method using constrained Nelder–Mead search-based optimisation to overcome issues related to the Jacobian. Another approach for the Integration method is to calculate the Jacobian by solving a sensitivity equation, thus increasing the calculation accuracy (Lu *et al.*, 2007). The low amplitude, high frequency oscillations are increased when the discretisation step dt is too small, as it could excite the uncontrollable states (Lin *et al.*, 1995; Lu *et al.*, 2008). Nonetheless, a relatively small dt can have a positive effect because it captures the changes in the system dynamics and this may reduce or even remove them, as well as increase accuracy (Lin *et al.*, 1995; Rutherford *et al.*, 1996; Lu *et al.*, 2008).

Since the two main algorithms of Inverse Simulation presented here are numerical implementations, the accuracy and execution time of Inverse Simulation varies with the number of iterations required for convergence, the tolerance limit set for convergence and the discretisation step dt used in Figure 3.1. Selecting a dt is a case of compromising between adequately following the system as it evolves over time, accuracy and possibly exciting the uncontrollable states. Furthermore, for systems where there are fast responding dynamics and slower responding dynamics, as is the case for helicopter applications, a “two timescale” Integration method using a reduced-order system model that eliminates the high-frequency oscillations has been developed by (Avanzini *et al.*, 2001).

The authors of (Lin *et al.*, 1995) perform a global error analysis of Inverse Simulation for Differentiation and Integration. The definition for the global error of Inverse Simulation, which will be used is the following. The input from Inverse Simulation is applied to the forward system and the output is obtained and compared with the desired. This is the global error that measures the fidelity of Inverse Simulation (Lin *et al.*, 1995). This definition mirrors the convergence criterion for the Inverse Simulation algorithms: a control input is accepted if the state & output or output is within the defined convergence tolerance.

The global error of the Differentiation method relative to dt is of the same order as the method used to evaluate the derivatives (Lin *et al.*, 1995). For example, a first-order difference formula has a local truncation error[5] of order $O(dt)$ and so if dt decreases by an order of magnitude so does the global error. For Integration, the states are obtained via integration, which is less sensitive to numerical errors due to time step size, and the smaller the time step dt, the smaller the local error in the integration formula (Lin *et al.*, 1995). The convergence is based on the outputs and control signals and the convergence tolerance can be set as desired; thus, the global error can be reduced accordingly and there is no direct correlation with the dt as in Differentiation. That said, when there are uncontrolled states and a very small dt, numerical instability may arise for the Integration method (Lin *et al.*, 1995). This is expected since as described the method converges using the inputs and outputs only.

[5] The local truncation error of a difference formula results from the use of the Taylor series to approximate the derivative. A first order formula results from using the first term of the Taylor series and thus has first order error and so on.

The choice of the desired output is also important since the validity of the calculated inputs depends on it. The proposed output should represent a realistic expectation of what the system can achieve (Thomson et al., 2006). The C^i continuity order is a parameter that needs to be carefully selected (Rutherford *et al.*, 1996; Thomson *et al.*, 2006; Ireland *et al.*, 2017). In theory, for Inverse Simulation the desired output can be of any order and does not depend on the need to have a specific relative degree. In practice, this output needs to be sufficiently smooth; the derivative information is needed since the output equation may need to be differentiated. The stability of Inverse Simulation is improved when the desired outputs used are of higher order, e.g. acceleration or velocity compared to position and orientation (Rutherford *et al.*, 1996; Thomson *et al.*, 2006). A high degree polynomial for describing the desired trajectory is an efficient and flexible method for defining desired outputs and usually, a polynomial of at least C^2 order is preferred (Rutherford *et al.*, 1996; Thomson *et al.*, 2006). Furthermore, the elements of the desired output vector should be equal to or less than the number of elements of the control vector, to ensure adequate actuation (Rutherford *et al.*, 1996; Murray-Smith, 2000; Thomson *et al.*, 2006).

3.3 Summary

From this review, it is evident that Inverse Simulation has been applied to a large variety of systems for the a priori evaluation of the required control inputs to produce a given, desired output, given a model of the system that provides at least the inputs and outputs. Two main algorithms were identified: Differentiation and Integration. Most of the applications use the Integration Method, as it is simpler to set up since it requires the model's inputs and outputs only, can handle more easily problems where the number of inputs exceeds the number of outputs and produces smoother inputs. All these however come at the expense of increased execution time, compared to Differentiation which also uses the system's state equations.

Inverse Simulation is a model-based method, and its successful application depends on several parameters. First, the choice of the model and the desired output is important as the validity of the calculated inputs depends on these. Second, the number of inputs and outputs is important as they impact the

Jacobian used for the Newton-Raphson scheme and more outputs than inputs result in an under-actuated system. Third, since this is a numerical method, it depends on numerical parameters: the tolerance limit set for convergence, the discretisation step dt used, and the number of iterations required for convergence. Of these, the tolerance limit set for convergence and the discretisation step dt are the most important.

Chapter 4 Analysis of Inverse Simulation with Application Examples

In Chapter 3, a review of the existing Inverse Simulation applications was presented, together with the application considerations. Having established this overview of the existing work on Inverse Simulation, in this Chapter the two main implementations are discussed in more detail not just in the context of a particular application, with all its specific complications, but in a broader, abstract way. Additional material is presented on how to overcome the issue of unequal numbers of inputs and outputs to facilitate the usage of Inverse Simulation in a wider variety of systems, so long as they are in the general state-space form. Finally, the background of Inverse Simulation from feedback linearisation and linear systems is examined.

4.1 Analysis of Inverse Simulation

4.1.1 General Algorithm for Differentiation and Integration

Inverse Simulation has two main requirements for its operation: the desired output (e.g., a trajectory) represented as a time history with an appropriate time step and a model of the system. The model's inputs and outputs must be representative of the inputs and outputs of the actual system. A wide variety of systems are suitable if these requirements hold. A general nonlinear system is used with m state equations, p output equations and k control inputs in the usual state-space form.

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x},\mathbf{u}\right),\ \mathbf{x} \in \Re^{m},\ \mathbf{u} \in \Re^{k} \tag{4.1}$$

$$\mathbf{y} = \mathbf{g}\left(\mathbf{x},\mathbf{u}\right),\ \mathbf{y} \in \Re^{p},\ \mathbf{u} \in \Re^{k} \tag{4.2}$$

The desired output $\mathbf{g}_d\left(t\right)$ is defined over the time interval T.

$$\mathbf{y}_d\left(t\right) = \mathbf{g}_d\left(t\right),\ t \in \mathrm{T} \tag{4.3}$$

$$\mathrm{T} = \left\{t_0,\ldots,t_i,\ldots,t_N\right\},\ t_i = idt,\ i \in \left\{0,1,\ldots,N\right\} \tag{4.4}$$

The time interval T is discretised N times using the time step dt in Eq.(4.4). The time step (or discretisation step) is selected after consideration of the rate of change of the system, the response time of its actuators, the motor time constant and the required response time.

4.1.1.1 Differentiation

For the Differentiation method, the convergence to an appropriate control input for the current time step is based on the system's state and output equations.

The state and output equations are discretised N times over the time interval T with a step of dt:

$$\begin{aligned} \frac{\mathbf{x}(t_i)-\mathbf{x}(t_{i-1})}{\underbrace{t_i - t_{i-1}}_{dt}} &= \mathbf{f}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right) \\ \mathbf{y}(t_i) &= \mathbf{g}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right) \end{aligned} \tag{4.5}$$

By using the desired output $\mathbf{g}_d(t_i)$ at every discretisation point i, a time series of suitable control inputs $\mathbf{u}$ and the corresponding states $\mathbf{x}$ is found. The functions $\mathbf{F_1}$ and $\mathbf{F_2}$ are defined to find the values of input $\mathbf{u}$ and the states $\mathbf{x}$:

$$\begin{aligned} \mathbf{F}_1(t_i) &= \mathbf{f}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right) - \frac{\mathbf{x}(t_i)-\mathbf{x}(t_{i-1})}{t_i - t_{i-1}} \\ \mathbf{F}_2(t_i) &= \mathbf{g}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right) - \mathbf{g}_d(t_i) \end{aligned} \tag{4.6}$$

In this work, the system described by Eq.(4.6) is solved using the Newton-Raphson method until the calculated values of the control input $\mathbf{u}$ and the states $\mathbf{x}$ are such that $\mathbf{F_1}$ and $\mathbf{F_2}$ are both close to zero within a certain tolerance.

At each inner iteration n, $\mathbf{u}$ and $\mathbf{x}$ are updated, and $\mathbf{J}$ is the Jacobian of $\mathbf{F_1}$ and $\mathbf{F_2}$.

$$\begin{bmatrix} \mathbf{x}_n(t_i) \\ \mathbf{u}_n(t_i) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{n-1}(t_i) \\ \mathbf{u}_{n-1}(t_i) \end{bmatrix} - \underbrace{\begin{bmatrix} \frac{\delta \mathbf{F}_1}{\delta \mathbf{x}}(t_i) & \frac{\delta \mathbf{F}_1}{\delta \mathbf{u}}(t_i) \\ \frac{\delta \mathbf{F}_2}{\delta \mathbf{x}}(t_i) & \frac{\delta \mathbf{F}_2}{\delta \mathbf{u}}(t_i) \end{bmatrix}_{n-1}^{-1}}_{\mathbf{J}} \begin{bmatrix} \mathbf{F}_1\left(\mathbf{x}_{n-1}(t_i),\mathbf{u}_{n-1}(t_i)\right) \\ \mathbf{F}_2\left(\mathbf{x}_{n-1}(t_i),\mathbf{u}_{n-1}(t_i)\right) \end{bmatrix} \tag{4.7}$$

The algorithm for the Differentiation method of Inverse Simulation given a system defined by Eq.(4.1), Eq.(4.2) and the desired trajectory $\mathbf{g}_d(t)$ over a discretised time interval $\mathbf{T}$ is in Appendix B. This is for the general case, so the steps in Appendix B can be used for any system in the form of Eq.(4.1), Eq.(4.2).

The algorithm has two loops: the inner loop (steps 2–9 in Appendix B) that converges after n iterations and an outer loop (steps 1–10 in Appendix B) that runs through each desired output at each discretisation point i (see also Figure 3.1). The convergence is estimated based on how close the values of $\mathbf{F}_1$ and $\mathbf{F}_2$, which correspond to the state and output equations, are to zero within a tolerance. The Differentiation method requires the differentiation of the system's state and output equation at each t_i, which means that any changes in these equations need the algorithm to be changed to accommodate them.

Several parameters are critical for the convergence of the algorithm and the quality of the calculated inputs: the tolerance limit set for convergence, the discretisation step dt used, the number of maximum iterations required for convergence and how the Jacobian for the Newton-Raphson scheme is approximated and inverted in Eq.(4.7). The tolerance limit, the discretisation step dt used are the main numerical parameters.

For the dt, it should take into consideration how fast the system dynamics are evolving and what the actual capabilities of the system are (e.g., the response time of the actuators, motor time step). The choice of dt also impacts the global error of the Differentiation method as discussed in (Lin *et al.*, 1995) and in Section 3.2. This point will be further elaborated on in Section 4.1.1.3.

The selection of the convergence tolerance is related to how the condition of convergence is estimated given the estimated values of $\mathbf{F}_1$ and $\mathbf{F}_2$ from Eq.(4.6) and the condition that both should be close to zero (within the convergence tolerance, see also Appendix B) This can be done in two ways. First, by using the relative error which represents the qualitative side: how accurate is the estimate relative to the value of the desired. Second, by using the absolute error, which measures the total error and represents the quantitative side. The absolute error depends on the magnitude of the measured quantities and frames the result within the interval or tolerance tol. Appendix H provides a detailed background of the relative and the absolute error.

Eq.(4.7) is a system of $(m+k)$ algebraic equations and is treated as a linear system to solve for $\mathbf{x}_n(t_i)$, $\mathbf{u}_n(t_i)$. The inversion of the Jacobian $\mathbf{J}$ is necessary and so its dimensions are of interest. From Eq.(4.1), (4.2) and (4.7) there are m states, k control input variables and p outputs and the dimension of $\mathbf{J}$ is $(m+p)\times(m+k)$.

If there is an equal number of inputs and outputs $p=k$ the Jacobian is a square matrix and existing research has been mostly restricted to this case (Hess *et al.*, 1991, 1993; Murray-Smith, 2000; Thomson *et al.*, 2006; Avanzini *et al.*, 2017).

If the number of inputs and outputs is not equal then a few factorisation methodologies are used to solve Eq.(4.7) and achieve the least square solution, which is the best available solution (Higham, 2002; Strang, 2009). There are the following two cases.

If there are more inputs than outputs $k>p$, this is an over-actuated system. From a linear algebra viewpoint, this is an underdetermined system without a unique solution. If the system in Eq.(4.7) is consistent then an input can be found for each desired output; the remaining $k-p$ inputs are free parameters, which can be allocated using a suitable factorisation method.

If there are more outputs than inputs $k<p$, this is an under-actuated system. From a linear algebra viewpoint, this is an overdetermined system and so an input cannot be found for each desired output. If the system in Eq.(4.7) is consistent all k inputs can be found and the remaining $p-k$ outputs cannot be directly associated with an input. Again, a suitable factorisation method can be used to ensure a least-squares solution.

Several available factorisation methods can provide the least square solution, such as LU ($\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix), QR ($\mathbf{Q}$ is a matrix with orthonormal columns and $\mathbf{R}$ an upper triangular matrix), Cholesky or the pseudoinverse. (Strang, 2009). The Jacobian in Eq.(4.7) may not necessarily have only one unique factorisation but Eq.(4.7) does have a unique least square solution (Strang, 2009), which means the best available solution for the control input $\mathbf{u}_n(t_i)$ and the state $\mathbf{x}_n(t_i)$ will be found.

The best factorisation method for any matrix to provide a computationally efficient least square solution depends on its dimension and rank (Strang, 2009). Choosing this factorisation among those available is as much an art as is a science and is highly specific to the problem at hand (Davis, 2013). The use of the commonly available formulas in linear algebra textbooks for square and non-square systems, such as those in (Strang, 2009), is strongly discouraged for numerical computations (Higham, 2002; Davis, 2013; Foster *et al.*, 2013). Furthermore, estimating the rank of a non-square matrix in numerical computations to select the best factorisation at every iteration of Eq.(4.7) is computationally expensive and not always straightforward (Strang, 2009; Davis, 2013). For all these reasons, it is recommended to carefully select the factorisation implementation to be used. Appendix G provides a detailed background on the solution of a non-square linear system and the available factorisations to ensure a least-squares result.

4.1.1.2 Integration

For the Integration method, the convergence to an appropriate control input for the current time step is based on whether the system's output matches the desired output.

To start, the state and output equations are again discretised N times over the time interval $\mathbf{T}$ with a step of dt. The desired output is $\mathbf{g}_d(t)$ and $t \in [0, t_N]$. At t_i the state Eq.(4.1) is numerically integrated to obtain the state $\mathbf{x}$ and then the output $\mathbf{y}$ is calculated.

$$\begin{aligned} \mathbf{x}(t_i) &= \int_{t_{i-1}}^{t_i} \dot{\mathbf{x}}(\tau)\, d\tau + \mathbf{x}(t_{i-1}) \\ \mathbf{y}(t_i) &= \mathbf{g}\left(\mathbf{x}(t_i), \mathbf{u}(t_{i-1})\right) \end{aligned} \tag{4.8}$$

If using the Euler rule for the numerical integration of $\mathbf{x}$:

$$\mathbf{x}(t_i) = \dot{\mathbf{x}}(t_i)\,dt + \mathbf{x}(t_{i-1}) \tag{4.9}$$

The error function between the current output and the desired is:

$$\mathbf{f}_e = \mathbf{g}\left(\mathbf{x}(t_i), \mathbf{u}(t_{i-1})\right) - \mathbf{g}_d(t_i) = \mathbf{y}(t_i) - \mathbf{g}_d(t_i) \tag{4.10}$$

In this work, Eq.(4.10) is solved using the Newton-Raphson method until convergence, that is until a suitable input $\mathbf{u}$ is found so that the error function is zero within an acceptable tolerance. At each iteration the value of input $\mathbf{u}$ is updated using Eq.(4.11), where n is the current Newton-Raphson inner iteration and $\mathbf{J}_e$ is the Jacobian of the error function $\mathbf{f}_e$ or equivalently the Jacobian of the system outputs $\mathbf{y}$ when perturbing the control inputs $\mathbf{u}$.

$$\begin{gathered}\mathbf{u}_n(t_{i-1}) = \mathbf{u}_{n-1}(t_{i-1}) - \mathbf{J}_e^{-1}\left(\mathbf{x}_{n-1}(t_i), \mathbf{u}_{n-1}(t_{i-1})\right) \cdot \mathbf{f}_e\left(\mathbf{x}_{n-1}(t_i), \mathbf{u}_{n-1}(t_{i-1})\right) \\ \mathbf{J}_e = \frac{\delta \mathbf{y}}{\delta \mathbf{u}}\end{gathered} \tag{4.11}$$

The algorithm for the Integration method of Inverse Simulation given a system defined by Eq.(4.1), Eq.(4.2) and a desired trajectory output $\mathbf{g}_d(t)$ over the discretised time interval $\mathbf{T}$ is in Appendix C.

The algorithm has two loops: the inner loop (steps 2 – 9 in Appendix C) that converges after n iterations and an outer loop (steps 1 – 10 in Appendix C) that runs through each desired output at each discretisation point. The convergence is estimated based on minimising the error function in Eq.(4.10), i.e. how close the current output is to the desired within a tolerance.

Several parameters are critical for the convergence of the algorithm and the quality of the calculated inputs: the tolerance limit set for convergence, the discretisation step dt used, the number of maximum iterations required for convergence and how the Jacobian for the Newton-Raphson scheme is approximated and inverted in Eq.(4.11). For the tolerance limit and the discretisation step the same considerations with the Differentiation method apply here.

As was the case for the Differentiation algorithm, Eq.(4.11) is a system of k algebraic equations and so can be treated as a linear system to solve for $\mathbf{u}_n\left(t_{i-1}\right)$. The dimension of $\mathbf{J_e}$ is $p \times k$ and if the number of inputs and outputs is the same then the Jacobian is a square matrix. If, however, the number of inputs and outputs are not equal, then a least-square solution is available using a suitable factorisation such as LU, QR, Cholesky decomposition or the Moore-Penrose pseudoinverse (Strang, 2009; Davis, 2013), see also Appendix G. If there are more inputs k than outputs p, then this is an over-actuated system and from a linear algebra viewpoint, an underdetermined system, so there never is a unique solution. When there are more outputs p than inputs k, this is an under-actuated system and so an overdetermined system; then a factorisation can be used. In this case, the calculated outputs are a least-square fit to the desired outputs (Hess *et al.*, 1991, 1993; Thomson *et al.*, 2006). Viewed from the point of linear algebra, this is an overdetermined system and usually has no unique solution; an input cannot be found for each desired output. Generally, the problems are restricted to $p \leq k$ where there is sufficient actuation to directly influence the dynamics of the system, but a least-square solution is available even for underactuated problems.

A key point here is that the Integration method does not require the differentiation of the system's state and output equation. Instead, the Jacobian of the output vector when perturbing the input is used. This means that any changes in the model's state equations do not require the algorithm to be changed to accommodate them. This can also be seen by comparing Eq.(4.6), Eq.(4.7) for Differentiation, which use both the states and the outputs equations, with Eq.(4.10) and Eq.(4.11) for Integration, which use only the output equations. The system's state equation can be called from an external function when needed (step 5 in Appendix C) and so is isolated from the main algorithm, thus the method's suitability for grey or black-box models.

4.1.1.3 Numerical Differentiation and Integration for Inverse Simulation

Consider again the Differentiation method in Section 4.1.1.1 that is (as the name suggests) based on evaluating the time derivatives of the states and then attempting to find the control inputs to achieve the outputs. The process is repeated until convergence within a tolerance at each point in time (t_i), see Eq.(4.7). Therefore, the nature of numerical differentiation is important and has a direct impact on the method.

From numerical analysis (Kreyszig, 2014), divided-difference formulas for differentiation can be generated by the Taylor series expansion. For example, for $dt = t_{i+1} - t_i$ a first-order approximation is:

$$f'(t_i) = \frac{f(t_{i+1}) - f(t_i)}{dt} + O(dt) \tag{4.12}$$

The local truncation error (i.e. the local error) results from the use of the Taylor series (Kreyszig, 2014) and is of order $O(dt)$, which in this case is the discretization time step for Inverse Simulation. When the truncation error is of the order $O(dt)$, this is a first-order method.

An idea would be to reduce the local truncation error, by reducing dt. However, when dt is reduced too much, there is a point where the truncation error is reduced but the round off error starts to dominate (Higham, 2002; Kreyszig, 2014). Rounding errors occur due to the way computers represent numerical values and thus cannot be influenced (Higham, 2002) (see also Appendix H). The difficulty with differentiation is tied in with the definition of the derivative, which is the limit of the difference quotient, and, in that quotient, there is the division of a large quantity by a small quantity or the difference between two large and nearly equal terms; both cases can cause numerical instability (Kreyszig, 2014). Similar difficulties occur with all differentiation formulas, and overall, for these reasons numerical formulas for differentiation are considered less (numerically) stable (Kreyszig, 2014).

The Integration method starts by integrating the states $\dot{\mathbf{q}}$ and then calculating the output, Eq.(4.8). Then, the error $\mathbf{f}_e$ between the actual and desired output, Eq.(4.10), is calculated based on the control input estimation. The process is repeated, and the control input estimation is refined until the outputs converge to the desired ones, i.e., the error $\mathbf{f}_e$ between the actual and the desired output converges within a tolerance for a given control input at each point in time (t_i). Therefore, the nature of numerical integration is also important and has a direct impact on the method's results.

From numerical analysis (Kreyszig, 2014) the simplest and most often used method in engineering problems for integrating differential equations as in Eq.(4.13) is the forward Euler method.

$$\frac{dy}{dt} = f(y,t) \tag{4.13}$$

The derivative can be approximated by:

$$f(y,t) \simeq \frac{y_{n+1} - y_n}{dt}, dt = t_{n+1} - t_n \tag{4.14}$$

Then the forward Euler method is:

$$y_{n+1} = y_n + f(y_n, t_n) \cdot dt \tag{4.15}$$

Eq.(4.15) involves two types of error: the round off error and the truncation error. The truncation error arises from using the approximation of the derivative in Eq.(4.14). Same with differentiation, the local truncation error is of interest because it is the one that can be influenced (see also Appendix H).

The truncation error results from the application of the Euler method over a single interval (t_n, t_{n+1}) in Eq.(4.15) (Kreyszig, 2014). The truncation error occurs because the true solution is approximated using what is essentially a Taylor series at Eq.(4.15) and is proportional to the square of the step size $O(dt^2)$ (Kreyszig, 2014). The global error results from the approximations done in all previous steps to evaluate Eq.(4.13) using Eq.(4.15) over a time interval and is $O(dt)$, proportional to the step size (Kreyszig, 2014). The useful conclusion from this analysis is that if the dt is made sufficiently small, then the accuracy is increased. In practice, this may not be computationally efficient (Chapra *et al.*, 2001; Kreyszig, 2014) and is constrained by application-specific issues (for example exciting the uncontrollable states). This implies that the method is stable as $\mathrm{d}t$ approaches zero, i.e., as $\mathrm{d}t$ approaches zero the error also does. This property of integration is in contrast with differentiation, where when $\mathrm{d}t$ is reduced too much, the truncation error is reduced but the round off error starts to dominate (Higham, 2002; Kreyszig, 2014). A point however to consider is that for a very small $\mathrm{d}t$, the estimation of the derivative $f\left(y_n, t_n\right)$ in Eq.(4.15) may not be accurate, for the reasons that have to do with the definition of the derivative.

Finally, note that the Euler method in Eq.(4.15) uses straight line segments to approximate the solution and is so a first-order method (Chapra *et al.*, 2001). In practice, most functions are not linear and therefore the method will yield good results from a $\mathrm{d}t$ small enough that a local linear approximation is valid. How small that $\mathrm{d}t$ is, depends on the quantity being approximated.

Overall, numerical integration is a smoothing process and is not very sensitive to small inaccuracies in function values and reducing the step size has a directly beneficial effect (Kreyszig, 2014). Numerical differentiation generally provides values that, for very small time steps, are dominated by the rounding error (Kreyszig, 2014).

These fundamental differences are the reason that the Integration method of Inverse Simulation is considered more numerically stable than the Differentiation method and why the global error of the Differentiation method in terms of the time step $\mathrm{d}t$ is of the same order as the method used to evaluate the derivatives as discussed in Section 3.2.

4.1.2 Inverse Simulation and Feedback Linearisation

In Inverse Simulation, the goal is to find a suitable input $\mathbf{u}$ given a desired output $\mathbf{g}_d\left(t\right)$ for the general nonlinear system in Eq. (4.1) and Eq.(4.2). However, Eq. (4.2) usually cannot be solved directly for $\mathbf{u}$. For this reason, it is differentiated again (Thomson *et al.*, 2006):

$$\left.\begin{aligned}\dot{\mathbf{y}} &= \frac{\partial \mathbf{g}}{\partial \mathbf{x}}\left(\mathbf{x}\right)\dot{\mathbf{x}} \\ \dot{\mathbf{x}} &= \mathbf{f}\left(\mathbf{x},\mathbf{u}\right)\end{aligned}\right\} \dot{\mathbf{y}} = \mathbf{g}_{\mathbf{x}}\left(\mathbf{x}\right)\mathbf{f}\left(\mathbf{x},\mathbf{u}\right) \tag{4.16}$$

If Eq.(4.16) can be solved for $\mathbf{u}$, we do so by setting:

$$\mathbf{y} = \mathbf{y_d} = \mathbf{g_d}(\mathbf{x}) \rightarrow \dot{\mathbf{y}} = \dot{\mathbf{g}}_\mathbf{d}(\mathbf{x}) \tag{4.17}$$

Then a suitable $\mathbf{u}$ can be found from:

$$\dot{\mathbf{g}}_\mathbf{d} = \mathbf{g_x}(\mathbf{x})\mathbf{f}(\mathbf{x},\mathbf{u}) \tag{4.18}$$

If it cannot be solved, a second differentiation can be performed:

$$\ddot{\mathbf{y}} = \mathbf{g}_{\mathbf{x}x}(\mathbf{x})\mathbf{f}^2(\mathbf{x},\mathbf{u}) + \mathbf{g_x}(\mathbf{x})\left[\mathbf{f}_x(\mathbf{x},\mathbf{u})\mathbf{f}(\mathbf{x},\mathbf{u}) + \mathbf{f}_u(\mathbf{x},\mathbf{u})\dot{\mathbf{u}}\right] \tag{4.19}$$

Eq.(4.19) can then be solved for $\mathbf{u}$. If that is not possible, then further differentiations can be performed (Thomson *et al.*, 2006), provided that the system in Eq. (4.1), (4.2) and $\mathbf{g}_d(t)$ is sufficiently smooth.

The idea of differentiating Eq.(4.16) until the output can be written in terms of the input, recalls the definition of the relative degree for a nonlinear, single input, single output, affine in the control system defined by Eq.(4.20), (4.21) (Khalil, 2003), i.e. the number of differentiations needed for the input to appear at the output equation Eq.(4.21).

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \xi(\mathbf{x})u, \ \mathbf{x} \in \Re^m, u \in \Re \tag{4.20}$$

$$y = h(\mathbf{x}), \ y \in \Re, \mathbf{x} \in \Re^m \tag{4.21}$$

$$\dot{y} = \frac{\partial h}{\partial \mathbf{x}}\left[\mathbf{f}(\mathbf{x}) + \xi(\mathbf{x})u\right] \triangleq L_f h(\mathbf{x}) + L_\xi h(\mathbf{x})u \tag{4.22}$$

The Lie derivative of $h(\mathbf{x})$ with respect to $\mathbf{f}(\mathbf{x})$ and to $\xi(\mathbf{x})$ is defined as:

$$\begin{aligned} L_f h(\mathbf{x}) &= \frac{\partial h}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}) \\ L_\xi h(\mathbf{x}) &= \frac{\partial h}{\partial \mathbf{x}}h(\mathbf{x}) \end{aligned} \tag{4.23}$$

If $L_{\xi}h(\mathbf{x})=0$ then Eq.(4.22) cannot be solved for u. Further differentiation until $L_{\xi}^{r-1}h(\mathbf{x})\neq 0$ yields Eq.(4.24).

$$y^{(r)}=L_{f}^{(r)}h(\mathbf{x})+L_{\xi}L_{f}^{(r-1)}h(\mathbf{x})u \tag{4.24}$$

Then, the control input u:

$$u=\frac{1}{L_{\xi}L_{f}^{(r-1)}h(\mathbf{x})}\left[-L_{f}^{(r)}h(\mathbf{x})+v\right] \tag{4.25}$$

reduces the output equation Eq.(4.24) to:

$$y^{(r)}=v \tag{4.26}$$

The relative degree of the system is defined as r, where $r\leq m$ (Slotine *et al.*, 1991; Khalil, 2003). The output is now a chain of r integrators and this process is called input-output linearisation (Slotine *et al.*, 1991; Khalil, 2003). In input-output linearisation, there is a direct, linear relationship between the output and the input and the state equation is partially linearised (Slotine *et al.*, 1991; Khalil, 2003). If the relative degree r is equal to the number of states $(r=m)$ then it can be fully feedback linearised, i.e. full-state linearisation: it can be transformed into an equivalent linear system (Slotine *et al.*, 1991; Khalil, 2003). The definitions of the relative degree, input-output linearisation and feedback linearisation are similarly expanded for nonlinear, control affine, multi-input and multi-output (MIMO) systems, with an equal number of inputs and outputs (square systems), though the notation can become quite complex (Slotine *et al.*, 1991; Isidori, 1995, 1999).

Overall, systems described by Eq. (4.20) and (4.21) can be analytically inverted to provide an expression for the input in terms of the state and output. The control design based on input-output linearization consists of three stages: differentiate the output until the input appears, Eq.(4.20) to Eq.(4.26), choose a control input to cancel the non-linearities and finally study the behaviour of the non-linearised states (Slotine *et al.*, 1991). If the system can be fully feedback linearised, then there are no non-linearised states, which simplifies things (Slotine *et al.*, 1991) -

though that is never the case for non-holonomic systems such as the four-wheeled differential rover (Yun *et al.*, 1993).

This approach to input-output and full-state linearisation is also known as nonlinear dynamic inversion (NDI). NDI has the advantage of providing an analytical solution that depends on the system model and its relative degree. Systems with a high relative degree can experience drift due to minute numerical errors and as the system's complexity increases, its inversion through NDI becomes more difficult. In (Ireland *et al.*, 2017) a comparison of tracking a desired trajectory between a system that is input-output linearised and inverted using Inverse Simulation Integration is given. Inverse Simulation is found to be more accurate in terms of tracking the desired output, at the expense of greater computational effort but this is offset by the more flexible nature of Inverse Simulation (Ireland *et al.*, 2017).

Compared with NDI, Inverse Simulation is a more general method that can be used for MIMO systems that are not control-affine and are not square. In practice, most systems are not control-affine and are quite complex and so this ability is a major advantage. Also, from Section 2.3.1, a non-holonomic system, such as a differentially driven robot, cannot be input-state linearised but can be input-output linearised (Yun *et al.*, 1993). Inverse Simulation depends on the system model, but there is no analytical inversion. Practical implementations usually require model changes and Inverse Simulation can handle these better than NDI. This is especially useful in the case of Integration (see Section 4.1.1.2) where the method is essentially decoupled from the system itself, so long as the inputs and outputs remain the same. As a system grows in complexity, its analytical inversion becomes less trivial. Conversely, any model change requires that NDI's inverse model be redefined, and practical implementation is almost guaranteed to require such model changes. Furthermore, the relative degree of the system is fixed, whereas, for Inverse Simulation, the desired output can be of any order, though in practice the order of the desired output is a parameter that needs to be carefully selected (Rutherford *et al.*, 1996; Thomson *et al.*, 2006), as was also discussed in Section 3.2.

4.1.3 Inverse Simulation for a Linear Time Invariant System

Inverse Simulation is simplified in the case of a linear, time-invariant (LTI) system. A general LTI system is used with: m state equations, p output equations and k control inputs in the usual state-space form, Eq.(4.27) and Eq.(4.28). The desired output $\mathbf{g}_d\left(t_i\right)$ is defined over the time interval $\mathbf{T}$, which is discretised with a time step dt, same as in Eq.(4.3).

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu},\ \mathbf{x} \in \Re^m,\ \mathbf{u} \in \Re^k,\ \mathbf{A} \in \Re^{m\times m},\ \mathbf{B} \in \Re^{m\times k} \tag{4.27}$$

$$\mathbf{y} = \mathbf{Cx},\ \mathbf{y} \in \Re^p,\ \mathbf{C} \in \Re^{p\times m} \tag{4.28}$$

The Inverse Simulation problem is stated as: Given a desired output $\mathbf{g}_d\left(t\right)$ over the time interval $\mathbf{T}$, find a corresponding control input $\mathbf{u}$ over the time interval $\mathbf{T}$.

Differentiating Eq.(4.28) gives:

$$\mathbf{y} = \mathbf{C}\dot{\mathbf{x}} \tag{4.29}$$

Substituting Eq.(4.27) to Eq.(4.29) and solving for $\mathbf{u}$:

$$\left(\mathbf{CB}\right)\mathbf{u} = \dot{\mathbf{y}} - \left(\mathbf{CA}\right)\mathbf{x},\ \left(\mathbf{CB}\right) \in \Re^{p\times k} \tag{4.30}$$

If the number of inputs is not equal to the number of outputs, matrix $\mathbf{CB}$ is not square and cannot be inverted. If there are more inputs than outputs, $k > p$, this is an over-actuated system that results in an underdetermined system, so there is never a unique solution. If there are more outputs than inputs, $k < p$, this is an under-actuated system that results in an overdetermined system. An input cannot be found for each desired output. If the system in Eq.(4.30) is consistent, all k inputs can be found and the remaining $p-k$ outputs cannot be directly associated with an input. A suitable factorisation method can be used to ensure a least-squares solution (Appendix G).

A unique control $\mathbf{u}$ can be found if the number of outputs p is equal to the number of inputs k and the determinant of matrix $\mathbf{CB}$ is not zero, i.e., $\mathbf{CB}$ is square and full rank:

$$\begin{gathered} \mathbf{u} = (\mathbf{CB})^{-1} \cdot \left[\dot{\mathbf{y}} - (\mathbf{CA})\mathbf{x}\right] \\ rank(\mathbf{CB}) = p = k \leftrightarrow \det(\mathbf{CB}) \neq 0 \end{gathered} \tag{4.31}$$

The LTI system in Eq.(4.27), Eq.(4.28) is defined as output controllable if it is possible to find an unconstrained control input $\mathbf{u}(t)$ that will transfer any given initial output $\mathbf{y}_0$ to any desired, final output $\mathbf{y}$ in a finite amount of time (Ogata, 2008). To check if a system is output controllable, the rank of the following matrix (this is known as the controllability matrix and its size is $\mathrm{p}\times(\mathrm{m}+\mathrm{k})$) is calculated (Ogata, 2008):

$$rank\left[\begin{matrix} \mathbf{CB} & \mathbf{CAB} & \mathbf{CA}^2\mathbf{B} & \dots & \mathbf{CA}^{m-1}\mathbf{B} \end{matrix}\right] = p \tag{4.32}$$

The system is output controllable if and only if the rank is p, where p is the number of outputs. The condition for output controllability is of practical importance because the goal of Inverse Simulation is to control the system's output to match it to the desired.

By comparing Eq.(4.31) and Eq.(4.32), the condition for finding a unique control input $\mathbf{u}$ for an output $\mathbf{y}$ for the LTI system in Eq.(4.27), Eq.(4.28) can be written as (Murray-Smith, 2000; Thomson *et al.*, 2006):

$$p = k = rank\left[\begin{matrix} \mathbf{CB} & \mathbf{CAB} & \mathbf{CA}^2\mathbf{B} & \dots & \mathbf{CA}^{m-1}\mathbf{B} \end{matrix}\right] \tag{4.33}$$

The condition $p = k$ recalls the case of the numerical Differentiation and Integration algorithms where if there is an equal number of inputs and outputs, then the Jacobian is a square matrix.

If instead of Eq.(4.28), the output is written as:

$$\mathbf{y} = \mathbf{Cx} + \mathbf{Du},\ \mathbf{y} \in \Re^m,\ \mathbf{u} \in \Re^q,\ \mathbf{C} \in \Re^{p\times m},\ \mathbf{D} \in \Re^{p\times q} \tag{4.34}$$

Then, the controllability condition is (Ogata, 2008):

$$rank\begin{bmatrix} \mathbf{CB} & \mathbf{CAB} & \mathbf{CA}^2\mathbf{B} & \dots & \mathbf{CA}^{m-1}\mathbf{B} & \mathbf{D} \end{bmatrix} = p \qquad (4.35)$$

The presence of the $\mathbf{Du}$ term in Eq.(4.34) always helps to establish output controllability (Ogata, 2008), since they add one more entry to the controllability matrix. From an Inverse Simulation point of view, this is expected since in Eq.(4.34) there is already a relationship between the output $\mathbf{y}$ and the input $\mathbf{u}$ and there is no need to differentiate $\mathbf{y}$ for $\mathbf{u}$ to appear. Whether of course Eq.(4.34) can be solved for $\mathbf{u}$ depends on the matrices $\mathbf{C}$ and $\mathbf{D}$.

Substituting the control input from Eq.(4.31) to Eq.(4.27) yields a new LTI system, defined by the matrices $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$:

$$\dot{\mathbf{x}} = \underbrace{\left\{\mathbf{A} - \left[\mathbf{B}\times(\mathbf{CB})^{-1}\times\mathbf{CA}\right]\right\}}_{\tilde{\mathbf{A}}}\mathbf{x} + \underbrace{\mathbf{B}\times(\mathbf{CB})^{-1}}_{\tilde{\mathbf{B}}}\dot{\mathbf{y}} \qquad (4.36)$$

The stability of this system is now determined by the matrix $\tilde{\mathbf{A}}$ which represents the new system dynamics. The matrices $\mathbf{A}$ and $\mathbf{B}$ are given from Eq.(4.36), but the selection of matrix $\mathbf{C}$ can vary. Therefore, the selection of desired outputs to control can affect the stability of the system.

In the case where the determinant of matrix $\mathbf{CB}$ is zero, a second differentiation of Eq.(4.28) can be attempted. This process is repeated until it is possible to express the output $\mathbf{y}^{(\mathbf{n})}$ in terms of the input $\mathbf{u}$, where $\mathbf{y}^{(\mathbf{n})}$ is a linear combination of derivatives of $\mathbf{y}$, $\det(\tilde{\mathbf{D}}) \neq 0$, $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{D}}$ are a linear combination of the rows of $\mathbf{CA}$, $\mathbf{CA}^2$, and $\mathbf{CB}$, $\mathbf{CAB}$ respectively and so on (Thomson *et al.*, 2006).

$$\begin{aligned} &\ddot{\mathbf{y}} = \mathbf{CA}^2\mathbf{x} + \mathbf{CABu} + \mathbf{CB}\dot{\mathbf{u}} \\ &\vdots \\ &\mathbf{y}^{(\mathbf{n})} = \tilde{\mathbf{C}}\mathbf{x} + \tilde{\mathbf{D}}\mathbf{u} \end{aligned} \qquad (4.37)$$

Then, the input $\mathbf{u}$ is:

$$\mathbf{u} = \tilde{\mathbf{D}}^{-1}\left(\mathbf{y}^{(n)} - \tilde{\mathbf{C}}\mathbf{x}\right) \tag{4.38}$$

and the new system dynamics are:

$$\dot{\mathbf{x}} = \underbrace{\left(\mathbf{A} - \mathbf{B}\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{C}}\right)}_{\tilde{\mathbf{A}}}\mathbf{x} + \underbrace{\left(\mathbf{B}\tilde{\mathbf{D}}^{-1}\right)}_{\tilde{\mathbf{B}}}\mathbf{y} \tag{4.39}$$

The stability of this system is now determined by the matrix $\tilde{\mathbf{A}}$, which represents the new system dynamics. The matrices $\mathbf{A}, \mathbf{B}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}}$ are given from Eq.(4.27) and Eq.(4.37) and the selection of desired outputs to control affects the stability of the system. In Sections 4.2.2, and 4.3.2 examples will be given as to how the selection of output affects the system stability and so the Inverse Simulation results.

For the LTI case, Inverse Simulation using Eq.(4.36) or Eq.(4.39) is the simplest case of the Differentiation algorithm. The Differentiation algorithm requires the appearance of the output in terms of the states and input. This contrasts with the general case for Integration which deals only with the input and output; the states and thus the system dynamics are not affected. This is an important difference in terms of the stability of Inverse Simulation Differentiation and Integration and the selection of appropriate desired outputs.

4.2 Application example: Mass Spring Damper System

To demonstrate the application of Inverse Simulation two well-known mechanical systems are used. The first system examined is the mass spring damper (MSD).

For comparison, the linear case of Inverse Simulation from Section 4.1.3 is examined alongside the Integration algorithm from Section 4.1.1.2. This provides the opportunity to demonstrate why selecting an appropriate output is important in Section 4.2.2. Also consider that in practice, a system may be locally linearised and so using the linear case may be a reasonable choice.

The state equation for the MSD system is:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\end{bmatrix}=\underbrace{\begin{bmatrix}0 & 1\\ \frac{\text{-k}}{\text{m}} & \frac{\text{-b}}{\text{m}}\end{bmatrix}}_{\mathbf{A}}\begin{bmatrix}x\\ \dot{x}\end{bmatrix}+\underbrace{\begin{bmatrix}0\\ \frac{1}{\text{m}}\end{bmatrix}}_{\mathbf{B}}u \tag{4.40}$$

This is a linear system with one input u and two states: the velocity $\dot{x}$ of the mass and its acceleration $\ddot{x}$. Matrix $\mathbf{A}$ has two conjugate complex eigenvalues, both with a negative real part and so the system is asymptotically stable (Appendix D). The desired output, i.e., the desired system response, can be the position x, its velocity $\dot{x}$, its acceleration $\ddot{x}$ or a combination. Their profiles are in Figure 4.1.

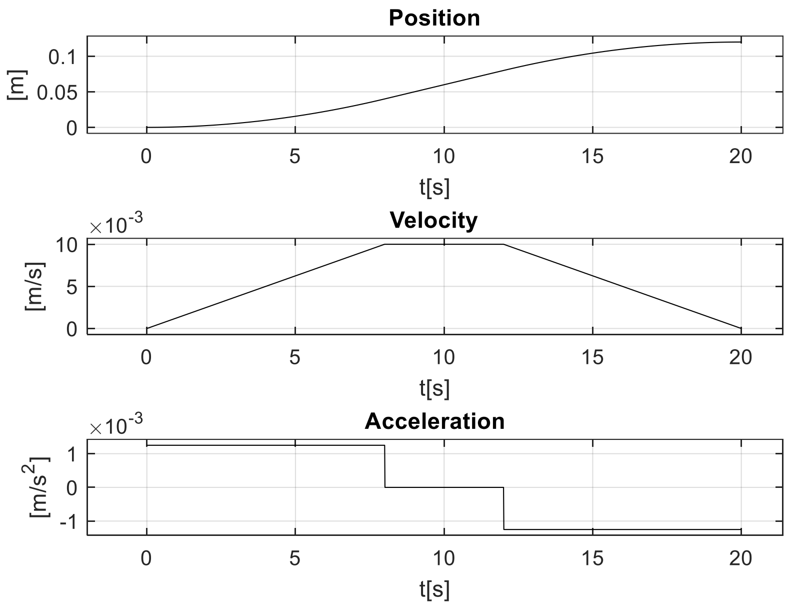


Figure 4.1: Position, velocity, acceleration

In Sections 4.2.2 and 4.2.3, the following methodology is used.

First, the desired output is defined, Figure 4.1. Then, the input from Inverse Simulation is found and applied to the forward system. The resulting output is then compared with the desired.

To facilitate a further comparison, a PID controller is used to achieve the same desired output, Figure 4.1, with Inverse Simulation. The result from the PID controller (the position x) is compared with the result from Inverse Simulation (the position x) to show how the system response compares. Additionally, the PID control input is compared with that from Inverse Simulation.

In all cases, the total trajectory time is discretised with a time step of $\mathrm{dt} = 0.01\mathrm{s}$, and the controller (PID or Inverse Simulation) is applied at each discretised point in time. All relevant parameters are in Appendix D.

4.2.1 PID Controller Response

A PID controller (see Appendix D) is used to achieve position x in Figure 4.1. The PID controller is applied to the system with a time step of $\mathrm{dt} = 0.01\ \mathrm{s}$.

The error of position x is in Figure 4.2 and its average value is $1.36\ 10^{-4}\ \mathrm{m}$. Figure 4.3 is the PID control input. Note that for $\mathrm{dt} = 0.1\ \mathrm{s}$ the PID fails because the time step is too large.

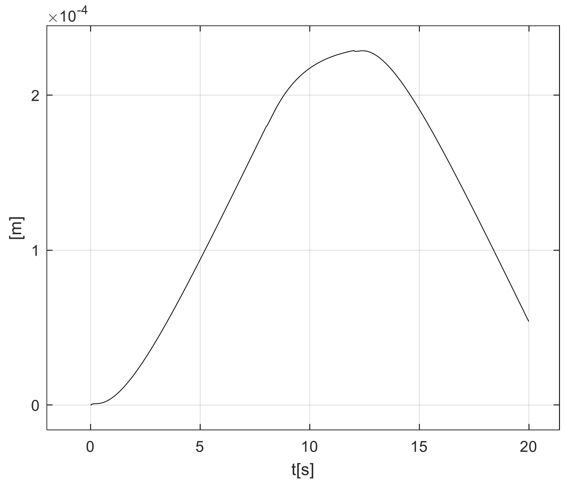


Figure 4.2: MSD PID Error between desired and actual position

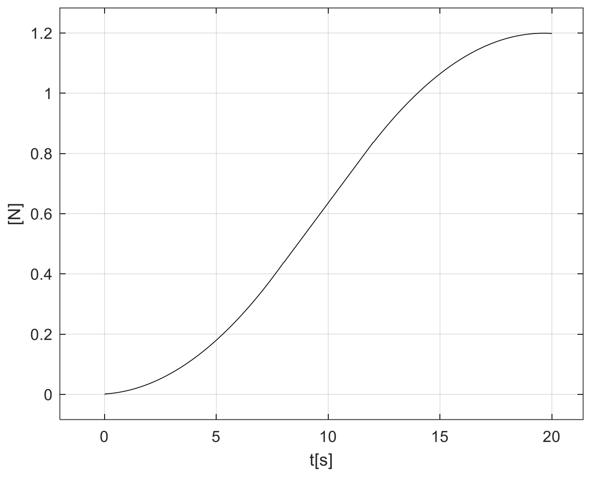


Figure 4.3: MSD PID Control Input

4.2.2 MSD Linear Inverse Simulation

The output equation is:

$$\mathbf{y} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{4.41}$$

For the linear case, to solve directly for the input u, matrix $\mathbf{CB}$ must be factorised. From Eq.(4.40), Eq.(4.41):

$$\mathbf{CB}\cdot u = \left(\dot{\mathbf{y}} - \mathbf{CAx}\right), \mathbf{CB} = \begin{bmatrix} 0 \\ \frac{1}{\mathrm{m}} \end{bmatrix} \qquad (4.42)$$
$$u = \left(\mathbf{CB}\right)^{-1}\left(\dot{\mathbf{y}} - \mathbf{CAx}\right)$$

It would be reasonable to select only x as the desired output, using the output matrix $\tilde{\mathbf{C}}$. In that case, however, u does not appear as in Eq.(4.42) and a second differentiation would be needed.

$$\tilde{\mathbf{C}} = \begin{bmatrix} 1 & 0 \end{bmatrix} \rightarrow \tilde{\mathbf{C}}\mathbf{B} = \mathbf{0} \qquad (4.43)$$

If the velocity $\dot{x}$ is chosen as desired, then:

$$\underset{\sim}{\mathbf{C}} = \begin{bmatrix} 0 & 1 \end{bmatrix} \rightarrow \underset{\sim}{\mathbf{C}}\mathbf{B} = \frac{1}{\mathrm{m}} \qquad (4.44)$$

This would simplify things, however, the new system dynamics matrix $\underset{\sim}{\mathbf{A}}$, calculated using Eq.(4.36) and Eq.(4.44), now has two zero eigenvalues:

$$\underset{\sim}{\mathbf{A}} = \left\{\mathbf{A} - \left[\mathbf{B}\left(\underset{\sim}{\mathbf{C}}\mathbf{B}\right)^{-1}\underset{\sim}{\mathbf{C}}\mathbf{A}\right]\right\} \qquad (4.45)$$

This example demonstrates how the selection of desired outputs to control affects the stability of the system, even in this simple case.

The time step is the same as the one used for the PID: dt = 0.01 s. The average position error is $1.15\ 10^{-7}$ m for the linear Inverse Simulation, Figure 4.4, which is less compared with that of the PID ($1.36\ 10^{-4}$ m.). This difference is because the linear case for Inverse Simulation uses the backslash (\) operator in MATLAB (see Appendix G for the available factorisation methods) that guarantees the best solution for Eq.(4.42) and both the position and the velocity (which are coupled) are used for converging to the control input

Figure 4.5 shows the control input calculated by Eq.(4.42) compared with the one required by the PID. The control input required for both cases is similar, the mean difference between them is $2.56\ 10^{-3}$ N.

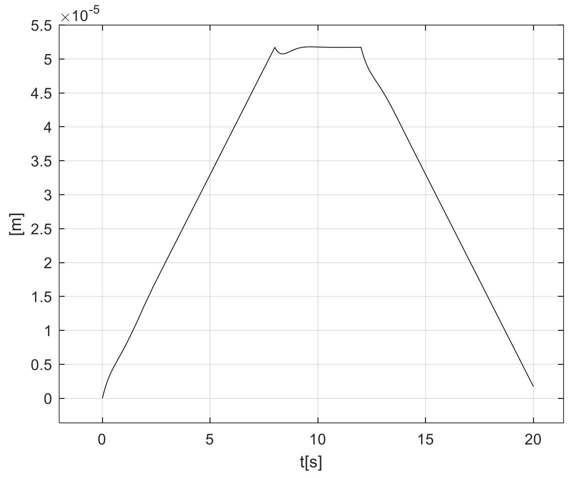


Figure 4.4: MSD IS Linear Error between desired and actual positions

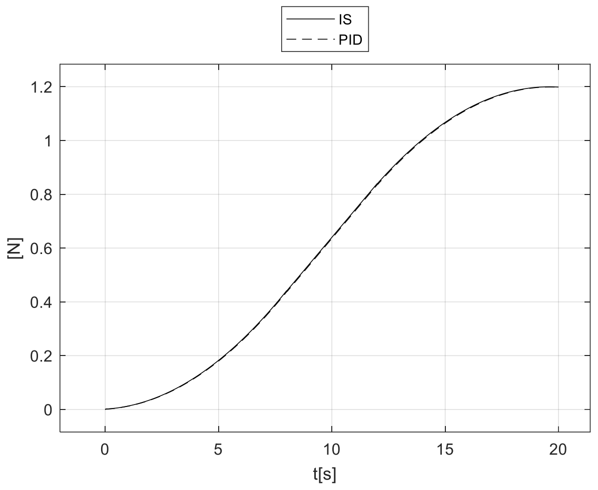


Figure 4.5: MSD IS Linear Control Input vs PID Input

For $\mathrm{dt} = 0.1\ \mathrm{s}$ the PID controller fails but the linear Inverse Simulation provides good results, the average position error is $1.73\ 10^{-6}\ \mathrm{m}$.

4.2.3 MSD Integration Inverse Simulation

For the Integration algorithm, position x is the output.

$$\begin{aligned} y &= \tilde{\mathbf{C}} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \rightarrow y = x \\ \tilde{\mathbf{C}} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \end{aligned} \tag{4.46}$$

There is one input and one output and so the Jacobian is replaced by the single partial derivative of the output when the input is perturbed. Using central differences and y from Eq.(4.46), the Jacobian is:

$$\begin{aligned} J_e &= \frac{\partial y}{\partial u} = \frac{y(u+\delta u) - y(u-\delta u)}{2\cdot\delta u} = \frac{\delta y}{2\cdot\delta u} \\ J_e^{-1} &= \frac{2\cdot\delta u}{\delta y} \\ \delta u &= |u|\cdot dt \end{aligned} \tag{4.47}$$

The perturbation of the control input δu is based on the previous estimate of the input multiplied by dt. There is a minimum acceptable δu and δy, to avoid division by zero. The method gives good results for $\mathrm{dt} = 0.01\ \mathrm{s}$ and $0.1\ \mathrm{s}$.

For dt = 0.01 s the average position error is $3.16\ 10^{-5}$ m, Figure 4.6. Compared with the PID position error which is $1.36\ 10^{-4}$ m, Integration provides better results. In comparison, however, with the linear case, where the average position error is $1.15\ 10^{-7}$ m, Integration provides worse results. This difference is because the linear case for Inverse Simulation uses the backslash (\) operator in MATLAB (see Appendix G for the available factorisation methods) that guarantees the best solution for Eq.(4.42) but only the position is considered for converging to the control input in Integration. In practice, these position errors are all small and provide a good system response in terms of position accuracy.

Figure 4.7 shows the control input calculated by Eq.(4.47) compared with the one required by the PID. The control input required for both cases is similar, the mean difference between them is $2.81\ 10^{-3}$ N.

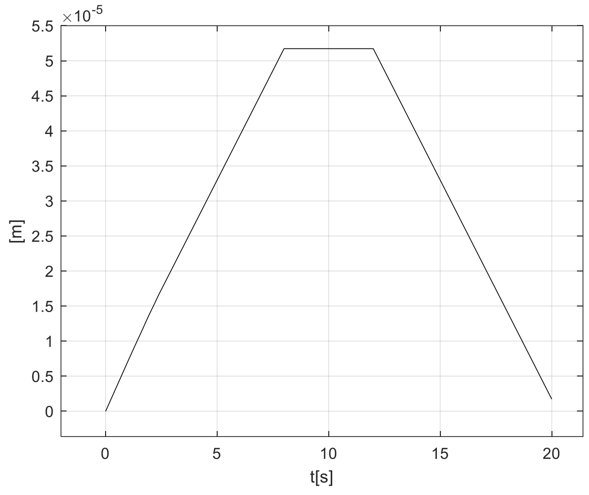


Figure 4.6: MSD IS Integration Position Error

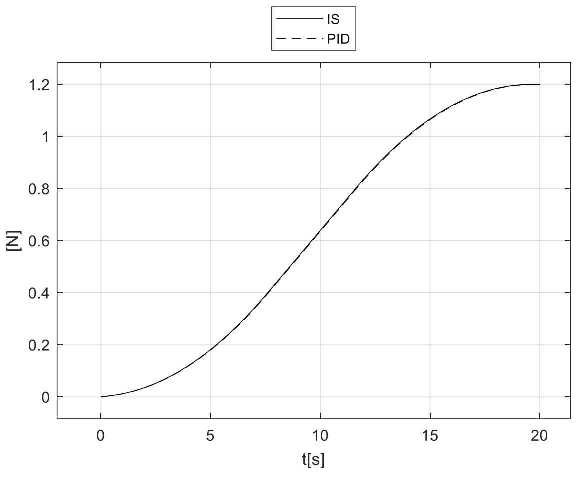


Figure 4.7: MSD IS Integration Control Input vs PID Input

For dt = 0.1 s the PID controller fails but the Integration Inverse Simulation provides good results, the average position error is $2.98\ 10^{-4}$ m.

Overall, a dt = 0.01 s is sufficient for Inverse Simulation to achieve good results in terms of the position error and the control effort required for following the desired trajectory profile. Furthermore, if there is a need for a larger dt, Inverse Simulation provides results, whereas the PID controller fails.

4.3 Application Example: Active Quarter Car Model

The quarter car (QC) model, Figure 4.8, allows the behaviour of the system in relation to its suspension type to be examined. The sprung mass $\mathrm{m_{sq}}$ represents one-quarter of the vehicle mass and the unsprung mass $\mathrm{m_{us}}$ represents one-quarter of the tyre mass. The spring force $\mathrm{k_s}$ is proportional to the displacement x_1 and the viscous damping force is proportional to velocity $\dot{x}_1$. In the case of the active suspension, a controllable actuator force F_s is used. For the passive suspension, this force is set to zero. The system parameters are in Appendix E.

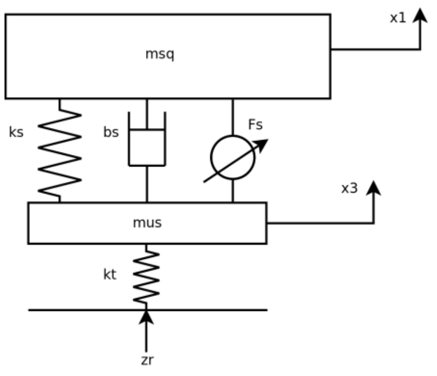


Figure 4.8: Quarter Car Model

The state equation for the QC is presented next, and the system is asymptotically stable (see Appendix E).

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{bmatrix} = \underbrace{\begin{bmatrix}0 & 1 & 0 & 0\\ \dfrac{\mathrm{-k_s}}{\mathrm{m_{sq}}} & \dfrac{\mathrm{-b_s}}{\mathrm{m_{sq}}} & \dfrac{\mathrm{k_s}}{\mathrm{m_{sq}}} & \dfrac{\mathrm{b_s}}{\mathrm{m_{sq}}}\\ 0 & 0 & 0 & 1\\ \dfrac{\mathrm{k_s}}{\mathrm{m_{us}}} & \dfrac{\mathrm{b_s}}{\mathrm{m_{us}}} & \dfrac{\mathrm{-(k_s+k_t)}}{\mathrm{m_{us}}} & \dfrac{\mathrm{-b_s}}{\mathrm{m_{us}}}\end{bmatrix}}_{\mathbf{A}} \cdot \begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix} + \underbrace{\begin{bmatrix}0\\ \dfrac{1}{\mathrm{m_{sq}}}\\ 0\\ \dfrac{-1}{\mathrm{m_{us}}}\end{bmatrix}}_{\mathbf{B}} \cdot F_s + \underbrace{\begin{bmatrix}0\\ 0\\ 0\\ \dfrac{\mathrm{k_t}}{\mathrm{m_{us}}}\end{bmatrix}}_{\mathbf{R}} \cdot z_r \tag{4.48}$$

In Eq.(4.48), the velocity of the sprung mass x_2 is a measure of the ride comfort of the vehicle, while x_4 is the velocity of the unsprung mass and is a measure of the vehicle's terrain handling ability (Dixon, 2007; Savaresi *et al.*, 2010). The unsprung mass velocity x_4 is chosen here as the output of interest, as this is relevant to the vehicle's ability to navigate rough terrain.

The road disturbance is modelled as a sine wave with a duration of $2\ \mathrm{s}$ and the total simulation time is $10\ \mathrm{s}$. There are more complex models of road disturbance, (Dixon, 2007), but for the purposes here, a sine wave is sufficient.

$$\begin{gathered} z_r = h\sin\left(2\pi ft\right) \\ 0 \le t \le 2s,\ f = 7Hz, h = 0.1m \end{gathered} \tag{4.49}$$

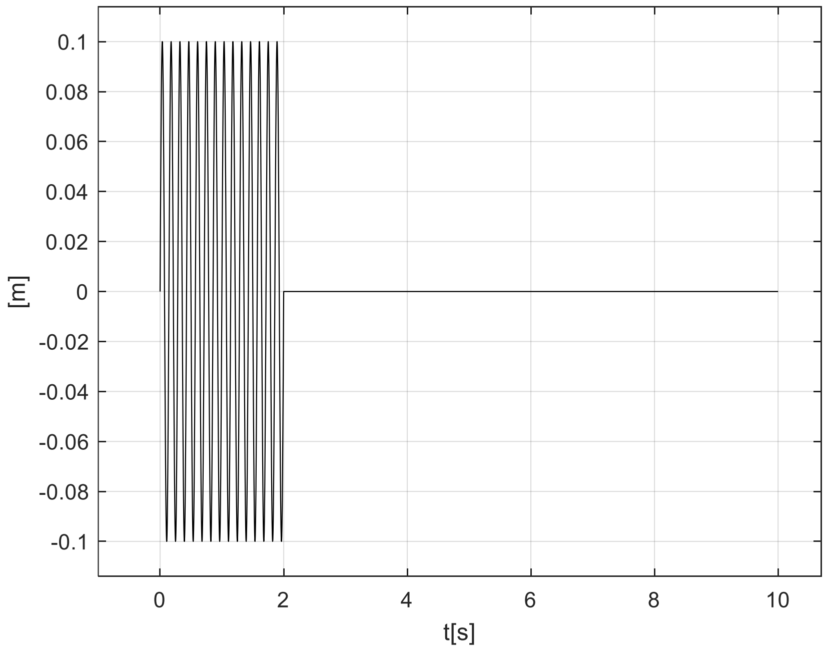


Figure 4.9: Sine Road Disturbance

In Sections 4.3.1 and 4.3.2, the following methodology is used.

First, a PID controller is applied to the system in Eq.(4.48) to reduce the oscillations from the road disturbance, Eq.(4.49), for the unsprung mass velocity x_4. Then, the system response (in terms of x_4) from the PID controller is chosen as the desired output for Inverse Simulation. In this way, Inverse Simulation has a realistic desired output and the control effort from both PID and Inverse Simulation is compared. Note this choice of desired output is more realistic than setting the desired output directly to zero; in such a system the goal is to minimise the oscillations and drive them to zero in a reasonable amount of time.

The numerical Integration method of Inverse Simulation is used, and the linear case of Inverse Simulation is also examined, as this is a linear system, though in practice that is rarely the case. In all cases, the total trajectory time is discretised with a time step of dt of 0.001 s, and the controller (PID or Inverse Simulation) is applied at each discretised point in time.

4.3.1 Desired Output

A PID controller (the gains are in Appendix E) is applied and then the system response (x_4) in Figure 4.10 is used as the desired output for Inverse Simulation. The PID control input is shown in Figure 4.11.

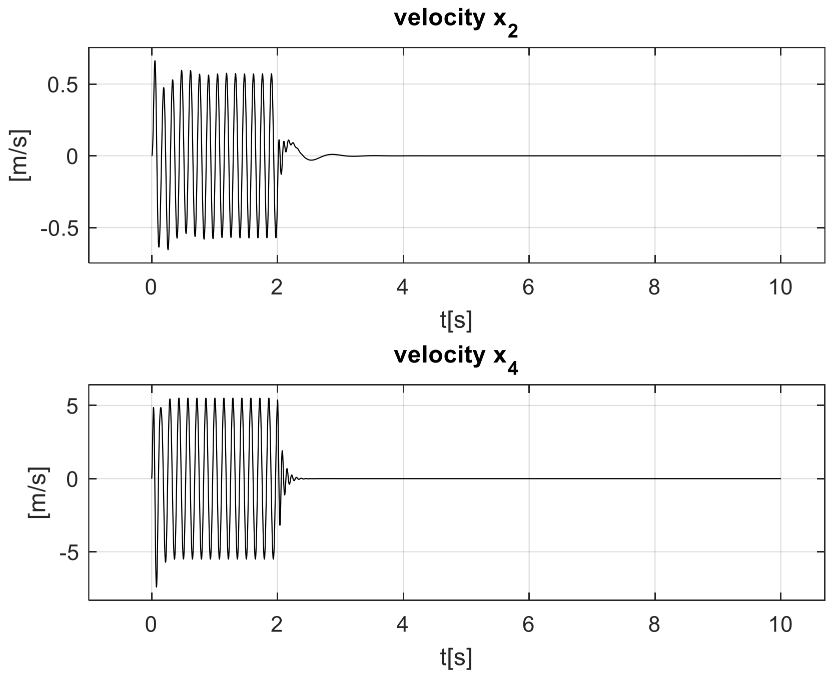


Figure 4.10: QCA PID Response

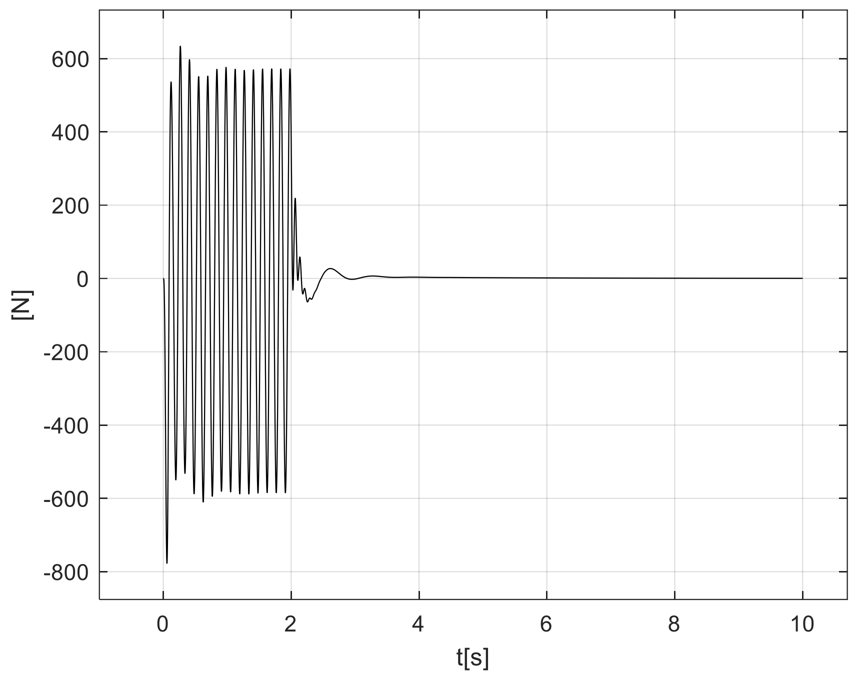


Figure 4.11: QCA PID Control Input

4.3.2 QCA Linear Inverse Simulation

From Eq.(4.48), and Eq. (4.53) the control input is:

$$F_s = (\mathbf{CB})^{-1}\left[\dot{y} - \mathbf{CA}x - \mathbf{CR}z_r\right] \tag{4.50}$$

The unsprung velocity x_4 from Figure 4.10 is selected as the desired output for Inverse Simulation. The output matrix is:

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix},\ \mathbf{CB} = -\frac{1}{\mathrm{m_{us}}} \tag{4.51}$$

The new system dynamics matrix is calculated by substituting the control input from Eq.(4.50) to Eq.(4.48) and is now:

$$\dot{\mathbf{x}} = \left\{ \underbrace{\mathbf{A} - \left[\mathbf{B} \times (\mathbf{CB})^{-1} \times \mathbf{CA}\right]}_{\tilde{\mathbf{A}}} \right\} \mathbf{x} + \underbrace{\mathbf{B} \times (\mathbf{CB})^{-1}}_{\tilde{\mathbf{B}}} \dot{\mathbf{y}} + \underbrace{\left[\mathbf{R} - \mathbf{B} \times (\mathbf{CB})^{-1} \times \mathbf{CR}\right]}_{\tilde{\mathbf{R}}} z_r \tag{4.52}$$

When x_4 is selected as the desired output, matrix $\tilde{\mathbf{A}}$ again from Eq.(4.52) has two imaginary eigenvalues, in a conjugate pair, and a double zero eigenvalue. The system is marginally stable.

If x_2 is selected, then $\mathbf{C}=\begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}$, $\mathbf{CB}=-\frac{1}{\mathrm{m_{sq}}}$ and $\tilde{\mathbf{A}}$ from Eq.(4.52) has two imaginary eigenvalues, in a conjugate pair, and a double zero eigenvalue. The system is marginally stable. Since both choices result in a similar condition, the unsprung mass velocity x_4 is preferred as it best represents the handling abilities of the system.

Note that if the output matrix $\mathbf{C}=\begin{bmatrix}0 & 1 & 0 & 1\end{bmatrix}$ (x_2, x_4 as outputs) was used, then the new system dynamics matrix, would have two zero eigenvalues, one negative real and one positive real and so would be unstable. Additionally, selecting x_1 or x_3 as the desired output is not possible, because in both cases it is $\mathbf{CB}=\mathbf{0}$. Therefore, x_4 is the desired output and the required control input is calculated and then applied to the forward system. The simulation parameters are in Appendix E.

Figure 4.12 shows the error between the actual output from linear Inverse Simulation and the desired. The output x_4 matches closely the desired; the maximum error is 0.042 m/s at the start of the simulation and the average error is 0.035 m/s. The spike at 2 s occurs at the point where the road disturbance ends.

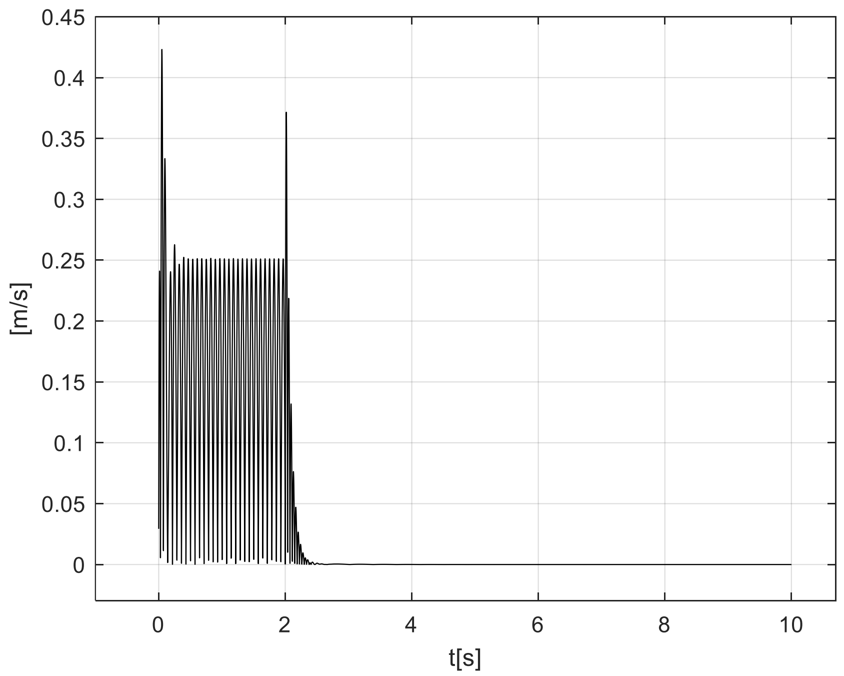


Figure 4.12: QCA Linear IS unsprung velocity error

Figure 4.13 shows the linear Inverse Simulation control input versus the PID control input. The control input calculated from the Inverse Simulation is larger than that of the PID controller, which signifies that a larger control effort is required, although both control inputs are comparable. Between the PID and the linear case, the PID is more accurate, though the linear case is significantly easier to set up. Using the output matrix from Eq.(4.51) results in a new system dynamics matrix, that has two zero eigenvalues and two imaginary. In this more complex system, this effect can be seen indirectly in the fact that the results for the linear case of Inverse Simulation are worse than the PID and a larger effort is required to achieve the desired output.

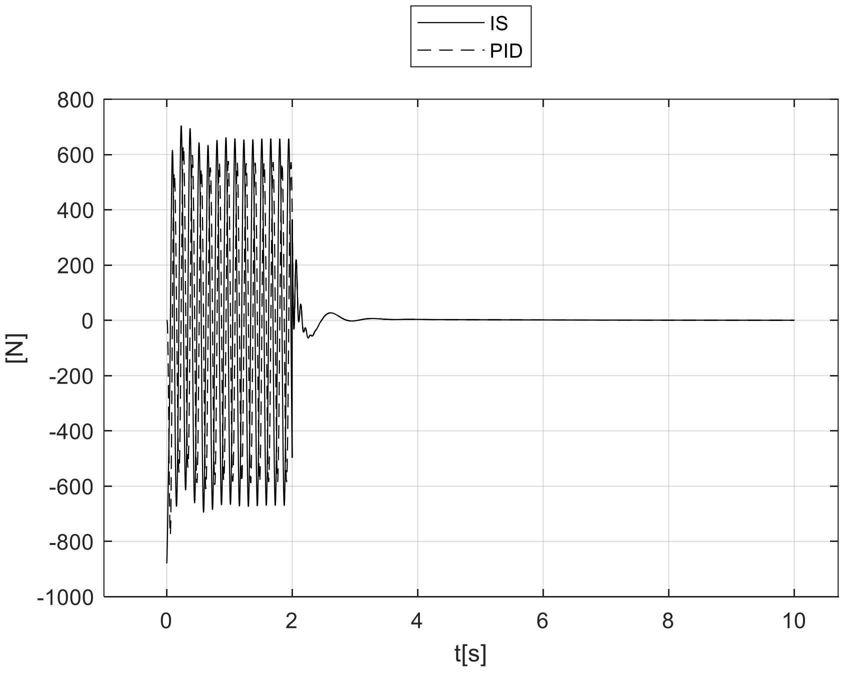


Figure 4.13: QCA Linear IS Control vs PID Control

4.3.3 QCA Integration Inverse Simulation

The simulation parameters for the numerical Integration method of Inverse Simulation are in Appendix E.

There is one input (F_s), and one output (x_4) in Eq. (4.53). The goal is to eventually drive to zero the velocity x_4, using as the desired the PID output from Figure 4.10.

$$y = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}}_{C} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{4.53}$$

The Jacobian is a number, the partial derivative of the single output when the single input is perturbed. Using central differences and y from Eq.(4.53), the Jacobian is similarly derived as in Eq.(4.47).

Figure 4.14 shows the error in x_4 when using the PID output as the desired. The output follows very closely the desired; the maximum error is 10^{-5} m/s, around the time when the road disturbance stops.

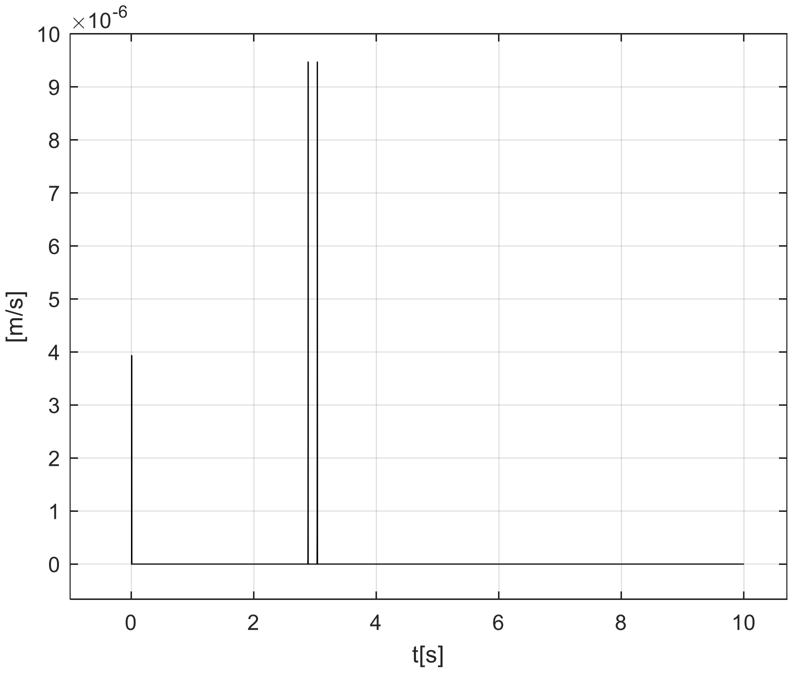


Figure 4.14: QCA IS Integration unsprung velocity error

As was the case for the Linear Inverse Simulation, the choice of the desired output is important. From Eq.(4.48), all the states are coupled and so any one of them can be chosen as the desired output. When the sprung mass velocity x_2 is chosen as the output to control, the Inverse Simulation algorithm exhibits significant numerical errors. However, when the unsprung mass velocity x_4 is chosen, the algorithm converges without significant errors. Intuitively, this is because x_2 is quite small compared to x_4 and changes rapidly, Figure 4.10, thus the algorithm cannot follow.

Figure 4.15 shows the Inverse Simulation control input versus the PID control input. In contrast to the linear case of Inverse Simulation, the control inputs, the two control inputs are almost identical. The Integration method does not impact the system dynamics and follows the desired output very closely.

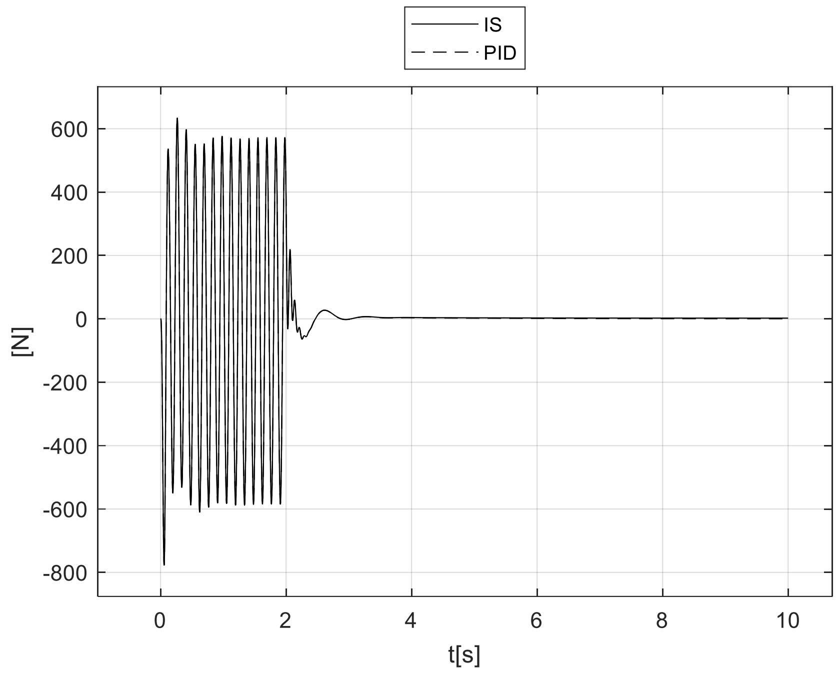


Figure 4.15: QCA IS Integration Control vs PID Control

4.4 Application Example: Road Disturbance Identification

The passive QC model (QCP) is derived from Eq.(4.48) by setting the actuator force to zero. In this case, the road disturbance z_r is considered as the input to the system.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-k_s}{m_{sq}} & \frac{-b_s}{m_{sq}} & \frac{k_s}{m_{sq}} & \frac{b_s}{m_{sq}} \\ 0 & 0 & 0 & 1 \\ \frac{k_s}{m_{us}} & \frac{b_s}{m_{us}} & \frac{-(k_s+k_t)}{m_{us}} & \frac{-b_s}{m_{us}} \end{bmatrix}}_{\mathbf{A}} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{k_t}{m_{us}} \end{bmatrix}}_{\mathbf{R}} \cdot z_r \qquad (4.54)$$

Motivated by the previous examples, Inverse Simulation is used to find the road disturbance (considered an input), given the output. In this way, the road disturbance is identified.

To do this, a disturbance is applied to the system and, as in the case of the active QC model, the unsprung mass velocity x_4 is chosen as the output of interest since it is a measure of the vehicle's terrain handling ability (Dixon, 2007; Savaresi *et al.*, 2010) and was used with good results for the QC active case. Then, the resulting control input from Inverse Simulation is compared with the road disturbance imposed on the forward system. The output equation is:

$$y = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathrm{C}} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{4.55}$$

This process is repeated for two types of disturbances z_r: a sine function as in Eq.(4.49) (plot repeated here for clarity, Figure 4.16) and a disturbance with a trapezoidal profile that has a maximum height of $0.1\ \mathrm{m}$ and a duration of $2.5\ \mathrm{s}$, Figure 4.17.

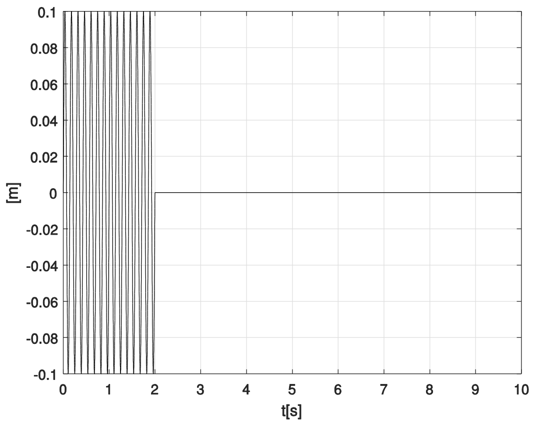


Figure 4.16: Road disturbance, sine

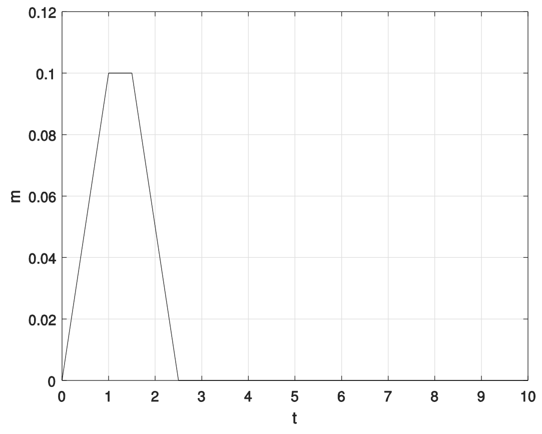


Figure 4.17: Road disturbance, trapezoid

There is one input and one output, and the Integration Inverse Simulation method is used. The Jacobian is calculated using central differences, as in Eq.(4.47). The perturbation is based on the previous estimate multiplied by dt and there is no minimum perturbation size. The simulation parameters are shown in Appendix E.

Figure 4.18 shows the results when the road disturbance is a sine function, the dashed line indicates the actual disturbance. The control input identified by the Inverse Simulation matches very well the initial disturbance, the average error is $1\ \mathrm{mm}$, and the maximum is $5\ \mathrm{mm}$.

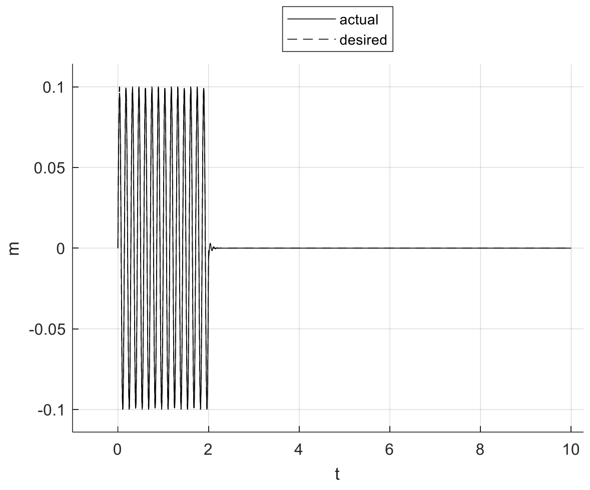


Figure 4.18: Sine Road Disturbance identified by IS

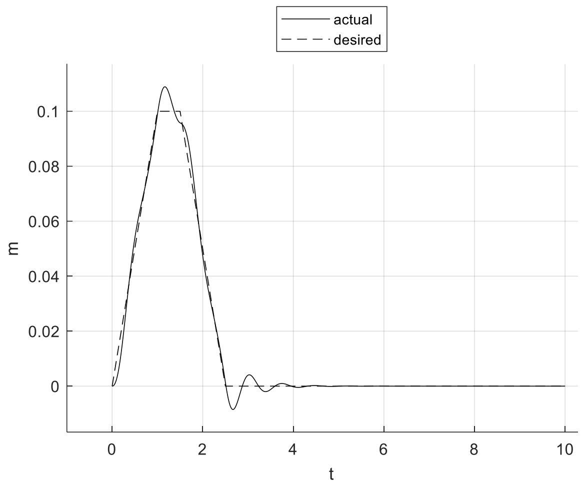


Figure 4.19: Trapezoid Road Disturbance identified by IS

Figure 4.19 shows the results when the road disturbance is a trapezoid, the dashed line indicates the actual disturbance. The control input identified is very close to the initial disturbance; the average error is 2 mm, and the maximum error is 1 cm. The identified road disturbance in Figure 4.19 is less smooth, especially when the input changes from ascending to a straight line and again when it starts to descend. However, in reality, the disturbance will not have well-defined edges as the trapezoid function used here.

The identification of the road disturbance affecting a vehicle is an important issue that influences the navigation capability of any ground vehicle and certainly so in the case of a rover operating in a distant, hostile environment. Currently, the issue of identifying road disturbances and road defects, either on paved roads or on off-road terrain, is discussed in several papers and different methods are used. A few examples include the use of line scan sensors, LiDAR sensors, stereography using digital cameras (Botha *et al.*, 2015), the reconstruction of road defects using neural networks (Ngwangwa *et al.*, 2014), the use of nonlinear observers (Rath *et al.*, 2015) and the use of Kalman filters (Dawkins, 2014). Reference (Chhaniyara *et al.*, 2012) is a survey of various terrain characterisation techniques (remote, in situ sensing, direct sensors) for planetary exploration rovers and notes that “in a number of areas, suitable light-weight, compact, power-efficient technology has yet to be fully realised”. The ability to identify these disturbances from the outputs of the system and without any need of additional equipment is an important advantage of Inverse Simulation.

4.5 Inverse Simulation Tuning Recommendations

Inverse Simulation is affected by numerical and stability issues that depend on certain parameters. These are identified as: the tolerance limit set for convergence, the discretisation step dt used, the number of maximum iterations required for convergence and how the Jacobian for the Newton-Raphson scheme is approximated and inverted. Of these, the tolerance limit, and the discretisation step dt used are the numerical parameters of the problem.

The time step dt is important for the stability of the Inverse Simulation, especially for Differentiation, and must correspond to the physical limitations and response times of the system, such as the onboard actuators. For the Differentiation method, increasing dt corresponds to an analogous increase in the error between the actual and the desired and reducing dt reduces the error only up to a point. For the Integration case, a small dt results in a smaller error, that in theory can be reduced to zero as the dt decreases. In practice, no numerical computation will achieve this. Selecting a dt is a case of compromising between adequately following the system as it evolves over time, accuracy and possibly exciting the uncontrollable states, as was also discussed in the review in Section 3.2.

The convergence tolerance can be set according to the acceptable error between the actual and the desired, considering that this is a numerical method and thus no computer will produce a perfect zero error. Additionally, the convergence tolerance should be relevant to the scale of the desired output. In Appendix H additional mathematical background is provided.

The size of the Jacobian and thus the solution for the Inverse Simulation depends on the number of the system's k control input variables and p outputs. An equal number of inputs and outputs is the preference, as this is neither an overactuated nor an underactuated control problem. In practice though, that may not be the case. Additionally, when solving numerically and since all measurements are never perfect, the choice should always be to select the best factorisation method to provide a computationally efficient least-square solution. It is important to have a clear idea of the size of the system to solve for Differentiation and Integration and examining the system's physical properties should provide the necessary information.

Selecting the desired output is necessary for formulating the algorithms and the physical properties of the system should be considered. The proposed output should represent a realistic expectation of what the system can achieve. In cases where there is ambiguity as to what outputs to select, it is worth remembering that for Differentiation the desired outputs can affect the overall stability, as shown for the linear case in Section 4.1.3. The linear case itself is the simplest version of the Differentiation algorithm, which requires the appearance of the output in terms of the states and input. This contrasts with the general case for Integration which deals only with the input and output, the states and thus the system dynamics are not affected. Furthermore, this output needs to be sufficiently smooth, especially if the derivative information may be needed to differentiate the output equation.

A final observation is that the Integration method does not require the differentiation of the system’s state and output equation. Instead, the Jacobian of the output vector when perturbing the input is used. The state and output equations can be called from an external function and so are isolated from the main algorithm, thus the method’s suitability for grey or black-box models, Differentiation converges based on the system’s state and output equations and their Jacobian when the input is perturbed. Therefore, any changes in the system dynamics (e.g., number of states) require a change in the algorithm.

4.6 Summary of Application Results

Overall, for the MSD, the active and the passive QC models, Integration provided consistently good results in terms of accurately following the desired output. Even for a seemingly simple LTI system such as the MSD, care should be taken when selecting the desired output or outputs and using the linear case of Inverse Simulation is not always the best choice. Furthermore, the selection of outputs for the linear case of Inverse Simulation and how they influence the system dynamics can provide insight into which is the best output to select if there are several choices. This was shown in the MSD example in Section 4.2 and the QCA example in Section 4.3. Finally, the Inverse Simulation algorithm can be used to determine in general an input given an output, without that input being necessarily a control input. This extends the usage of Inverse Simulation.

Chapter 5 Rover Mathematical Model and Trajectory Generation

The purpose of this chapter is to present the mathematical model and the test trajectories that will be used for Inverse Simulation for the rover in Chapter 6 and Chapter 7. The rover model used conforms to the baseline design from Section 2.2.3: (a) 4 wheels, (b) differential drive. Instead of using a passive suspension, the simpler approach is adopted of using the flexibility of the wheel itself for terrain that is not significantly uneven (Siegwart *et al.*, 2011).

5.1 Rover Model Overview

A realistic, accurate model that describes the behaviour of the robot under a defined range of operating conditions is required to create a simulation capable of testing control algorithms. More so in the case of Inverse Simulation when the model is numerically inverted to find the input for a desired output.

A wheeled mobile robot is described by a kinematic model and a dynamic model. The kinematic model is the most basic study of how the system behaves (Siegwart *et al.*, 2011) and provides a description of the pose of the robot given a frame of reference, the system's configuration vector, and the vector of independent velocity variables associated with the system's degrees of freedom (Campion *et al.*, 2008; Siegwart *et al.*, 2011). The kinematic model provides a description of the robot that can be used on its own, as it is simpler than the dynamic model, has lower computational requirements and is easier to implement (Morin *et al.*, 2008; Cheein *et al.*, 2014; Paden *et al.*, 2016). The dynamic model provides a complete description of all the forces and torques that act on the robot, including the forces provided by the actuators (Campion *et al.*, 2008). Together, these two models completely describe the behaviour of the robot.

The model used in this work is that of a four-wheel, differentially driven rover with no suspension. The rover's left and right sides are symmetrical to each other, and each side is actuated independently, with the wheels on each side always being actuated with the same signal. This model is a close analogue to the baseline model in Section 2.2.3 and has the same three basic characteristics: (a) wheeled locomotion using four wheels (b) wheel drive, and (c) differentially driven. There is no suspension. The simplest approach that is suitable for terrain that is not significantly uneven, is to design flexibility into the wheel itself, e.g. by using a deformable tyre made of rubber for the wheel (Siegwart *et al.*, 2011). In this model, the wheel conformity to the ground and the associated terramechanics are not considered, the assumption is that the terrain is not significantly uneven or soft and the wheels can traverse the terrain with no issues. For this work, the robot moves on a planar, smooth, and rigid terrain.

Note that there is no general definition in the literature about what is exactly soft terrain or uneven terrain (Nie *et al.*, 2013; Ghotbi *et al.*, 2016). For wheeled robots moving on rigid, flat ground, when it can be assumed that the robot wheels roll without (significant) slipping and no sinkage occurs, then this is the type of terrain that is not significantly uneven or soft (Ghotbi *et al.*, 2016). These assumptions may be violated when the vehicle moves on soft terrain (Thueer *et al.*, 2010; Ghotbi *et al.*, 2016), but this is not the case for this model in this work. Reference (Nie *et al.*, 2013) provides a planar to rough terrain classification using the obstacle size relative to the robot's size. The obstacles (Nie *et al.*, 2013) are subdivided into four categories: single, continuous, slopes, and gaps.

The rover's chassis is a Lynxmotion 4WD3 model that employs four identical DC motors and has rubber, treaded wheels, Figure 5.1. The chassis width is $0.2488\ \mathrm{m}$ (from the centre of the left wheel to the centre of the right wheel), the length is $0.35\ \mathrm{m}$, and the wheel height is $0.127\ \mathrm{m}$. The complete rover specifications are in Appendix A. The model used has been previously extensively presented in (Worrall *et al.*, 2006; Worrall, 2010) based on the method in (Fossen, 2002). The model has also been experimentally validated (Worrall, 2010).

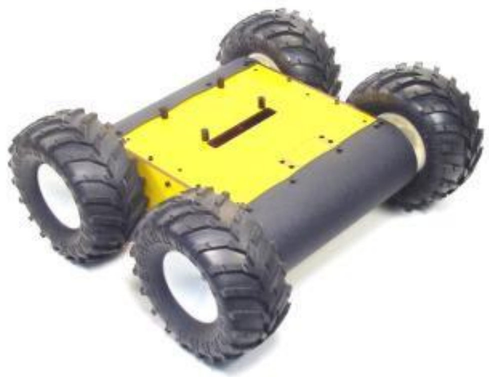


Figure 5.1 Lynxmotion 4WD3 Chassis (*Lynxmotion*, 2018)

Using a validated model ensures that the simulation results will be like those of the actual robot. The validation process is fully described in (Worrall, 2010) and was done using two comparison methods: Analogue Matching (also known as visual inspection) and Integral Least Squares. When using Analogue Matching, the simulation model output is compared graphically with available experimental data by superimposing the two plots (Gray, 1992). The simulation data that best fit the experimental data correspond to the model that best represents the physical system (Gray, 1992). The Integral Least Squares method is a quantitative method that calculates the least-squares error between the experimental data and the simulation data; the model that has the smallest error is the best representation of the physical system (Worrall, 2010). During the validation process, seven different experiments of increasing complexity were carried out (Worrall, 2010): (1) drive the robot forward in a straight line, (2) drive the robot forward in a straight line and then execute a left turn, (3) drive the robot in a square, (4) drive the robot forward on a small up-down ramp with a maximum of $15\,\mathrm{deg}$ incline: the robot moves forward, one set of wheels drives up the ramp, then down the ramp then continues to move forward, this is to evaluate the coupling between roll and pitch (5) drive the robot forward on a small up-down ramp with a maximum of $15\,\mathrm{deg}$ incline, this time the whole robot is on the ramp, (6) drive the robot forward on a flat surface to evaluate the roll and pitch coupling, (7) drive the robot in a zig-zag pattern. In each case, the linear and angular accelerations and velocities were recorded and compared with those from simulation (Worrall, 2010). In Appendix A, the validation results from the second experiment are shown.

5.1.1 Model Variables and Frame of Reference

The inertial Earth-fixed frame e and the rover body-fixed frame b are shown in Figure 5.2.

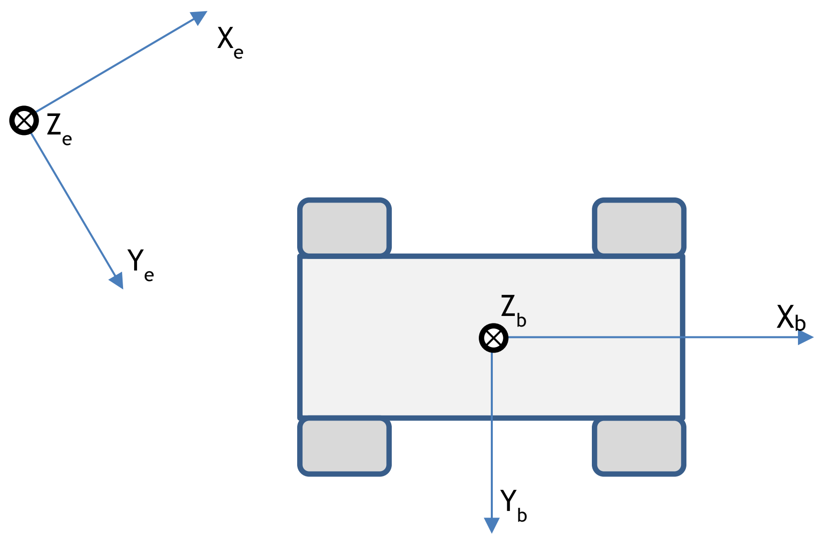


Figure 5.2: Rover Frames of Reference

The origin of the body-fixed frame is at the centre of mass of the rover, which is also the centre of the robot, Figure 5.2. Table 5.1 presents the model variables. The rover's specifications (such as mass, wheel radius, moments of inertia etc) are in Appendix A.

Table 5.1: Model Variables

DOF	Axis	Name (Type)	Velocity	Force or Moment	Position or Orientation
1	X_e / X_b	Surge (Translation)	u (m/s)	X (N)	x (m)
2	Y_e / Y_b	Sway (Translation)	v (m/s)	Y (N)	y (m)
3	Z_e / Z_b	Heave (Translation)	w (m/s)	Z (N)	z (m)
4	X_e / X_b	Roll (Rotation)	p (rad/s)	K (Nm)	φ (rad)
5	Y_e / Y_b	Pitch (Rotation)	q (rad/s)	P (Nm)	θ (rad)
6	Z_e / Z_b	Yaw (Rotation)	r (rad/s)	M (Nm)	ψ (rad)

The following notation is used. Vector **q** describes the linear and angular velocities in the body-fixed frame, these are the independent velocity variables. It is further decomposed in $\mathbf{q_1}$ and $\mathbf{q_2}$ to represent the body-fixed translation and orientation velocities respectively.

$$\mathbf{q} = \begin{bmatrix} \underbrace{u \quad v \quad w}_{\text{translation}} & \underbrace{p \quad q \quad r}_{\text{rotation}} \end{bmatrix}^T \tag{5.1}$$

$$\mathbf{q}_1 = \begin{bmatrix} u & v & w \end{bmatrix}^T \tag{5.2}$$

$$\mathbf{q}_2 = \begin{bmatrix} p & q & r \end{bmatrix}^T \tag{5.3}$$

Vector $\boldsymbol{\tau}$ combines the independent forces $\mathbf{F}$ and moments $\mathbf{M}$ that act on the system.

$$\boldsymbol{\tau} = \left[\underbrace{X \quad Y \quad Z}_{\mathbf{F}} \; \underbrace{M_x \quad M_y \quad M_z}_{\mathbf{M}} \right]^T \tag{5.4}$$

Vector $\boldsymbol{\eta}$ is the configuration vector and describes the pose in the Earth-fixed frame. It is further decomposed into $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$ to represent the Earth-fixed position and orientation respectively.

$$\boldsymbol{\eta} = \left[\underbrace{x \quad y \quad z}_{\text{position}} \; \underbrace{\varphi \quad \theta \quad \psi}_{\text{orientation}} \right]^T \tag{5.5}$$

$$\boldsymbol{\eta}_1 = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{5.6}$$

$$\boldsymbol{\eta}_2 = \begin{bmatrix} \varphi & \theta & \psi \end{bmatrix}^T \tag{5.7}$$

5.1.2 Dynamics

The complete dynamic model is presented here:

$$\boldsymbol{\tau} = \mathbf{H}\dot{\mathbf{q}} + \mathbf{C}(\mathbf{q})\mathbf{q} + \mathbf{D}(\mathbf{q})\mathbf{q} + \mathbf{G}(\boldsymbol{\eta}) \tag{5.8}$$

In Eq.(5.8) $\mathbf{H}$ is the mass matrix, $\mathbf{C(q)}$ are the Coriolis forces, $\mathbf{D(q)}$ are damping forces and $\mathbf{G}(\boldsymbol{\eta})$ are the gravitational forces. In the next sections, the individual elements of the dynamics described by Eq.(2.1) are presented. These are the forces & moments in vector $\boldsymbol{\tau}$ in Eq.(5.4) that cause the rover's movement, the rigid body dynamics described by matrices $\mathbf{H}$, $\mathbf{C(q)}$, the damping forces & moments $\mathbf{D(q)}$ and the gravitational forces $\mathbf{G}(\boldsymbol{\eta})$. Note that for the cosines and sines, c is used instead of cos and s instead of sin.

5.1.2.1 Forces & Moments

The forces are the forces that drive the rover and are generated by each wheel when actuated. The force F_i is the result of the torque T_i applied to the wheel, where i refers to each wheel (fl, fr, bl, br) as in Figure 5.3, Figure 5.4, and r_w is the wheel radius.

$$F_i = \frac{T_i}{r_w} \tag{5.9}$$

The rover has a differential drive configuration that uses four motors wired in parallel so that the motors on each side always receive the same input. Thus, two motors drive the left-hand side wheels and two drive the right-hand side wheels. The wheels at each side are always actuated with the same input by the motors. The rover motion is controlled by specifying the torque to be sent to each side. Later, with the inclusion of the motor dynamics in Section 5.1.4, that input will be a voltage, but the basic functionality remains the same.

To achieve forward (surge) motion each wheel is actuated with identical torques (Figure 5.3). To rotate the rover, the wheels on opposite sides are driven in opposite directions. For example, to rotate clockwise, the wheels on the outside of the turn rotate forward and the wheels on the inside of the turn rotate backwards (Figure 5.4). To rotate on the spot, the signals to the left side and right side are equal and in opposite directions.

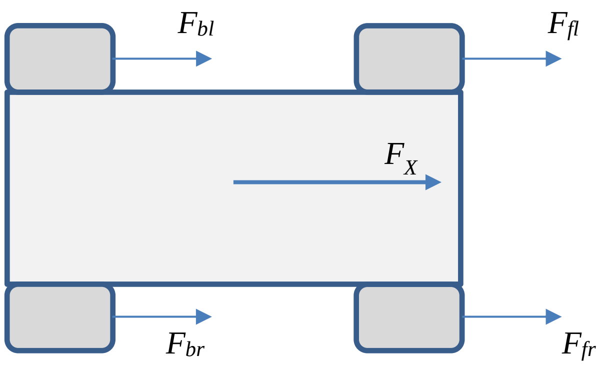


Figure 5.3: Forward Surge Motion

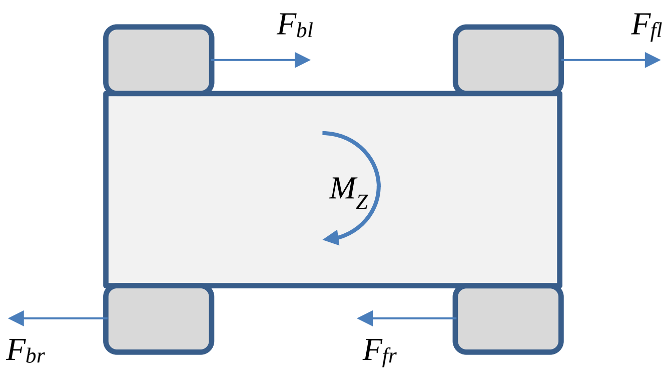


Figure 5.4: Clockwise Turn

Using this wheel drive configuration, the movement along the surge direction and the yaw direction is directly controlled, X and Mz respectively in Eq.(5.4).

Before moving on to the calculation of the propulsion forces, it is worth going over the issue of lateral slip. This is different to roll (or longitudinal) slip, where in an ideal rolling wheel, the velocity of the contact point between the wheel and the ground is zero (Schramm, Hiller and Bardini, 2014). In the case of lateral slip, a wheel that is acted upon by a lateral force gets a velocity component that is lateral to the rolling direction (Pacejka, 2012; Schramm *et al.*, 2014). The forward (surge) speed is the longitudinal component of the total velocity vector at the wheel centre and the sway is the lateral component (Pacejka, 2012; Schramm *et al.*, 2014). This is present, for example, when the rover is turning (Worrall *et al.*, 2006) in one direction and the wheels are pointing in the previous direction. This is because the wheels are driven but not steered, the wheel cannot turn around a vertical axis passing through the centre of the wheel and the ground contact point to change its direction (Siegwart *et al.*, 2011). The lateral (or skew) wheel slip is then defined as the ratio of the lateral and the forward velocity of the wheel, which corresponds to the tangent of the slip angle β (Pacejka, 2012; Schramm *et al.*, 2014). The slip angle β is (Pacejka, 2012; Schramm *et al.*, 2014):

$$\beta = \arcsin\left(\frac{v}{\sqrt{\left(u^2+v^2\right)}}\right) \tag{5.10}$$

Ideally, this slip angle should be as close to zero as possible and there would be no sway force acting on the wheels, so the sway velocity should be zero. For a slow-moving system on flat terrain, this is not an unreasonable assumption (Tian *et al.*, 2014; Paden *et al.*, 2016). Recall also that the rover is differentially driven and thus it can control its forward speed u and its orientation, but not the sway speed v. The slip angle is set to zero when the rover is not moving. Eq.(5.10) provides a relation between the surge and the yaw velocity and this coupling will further be used when developing the Inverse Simulation algorithm for the rover.

Force X, where F_{fl} is the force at the front left wheel and so on and β is the slip angle is (Worrall *et al.*, 2006; Worrall, 2010):

$$X = \left(F_{fl} + F_{fr} + F_{bl} + F_{br}\right) \mathrm{c}\,\beta \tag{5.11}$$

The yaw moment M_z, where F_{fl} is the force at the front left wheel and so on and $\mathrm{r_m}$ is the moment arm is (Worrall *et al.*, 2006; Worrall, 2010):

$$M_z = \left[\left(F_{fl} + F_{bl}\right) - \left(F_{rl} + F_{br}\right)\right] r_m \tag{5.12}$$

So far, the surge force X and the yaw moment M_z have been defined in Eq.(5.4). The remaining forces and torques at Eq.(5.4) cannot be directly controlled and are therefore the system's unmatched dynamics: sway (Y), heave (Z), roll (M_x) and pitch (M_y). These forces & moments are the result of the interaction between surge, yaw, and the environment. The sway force Y, in particular, is the result of the robot slipping on the ground (angle β) when turning, Eq.(5.13) (Worrall *et al.*, 2006; Worrall, 2010):

$$Y = \left(F_{fl} + F_{fr} + F_{bl} + F_{br}\right) \mathrm{s}\,\beta \tag{5.13}$$

Comparing Eq.(5.13) with Eq.(5.11) for the sway force X and Eq.(5.12) for the yaw moment M_z, it can be seen that while Y is not directly controlled, it does depend on the wheel forces and thus on the actuation torque. Also, when the slip angle is very small due to the small angle approximation[6], the effect on the surge force X is very small and the sway force Y is almost zero.

[6] Small angle approximation (Fossen, 2002): $\sin\beta \approx \beta$ or even $\sin\beta \approx 0$, $\cos\beta \approx 1$ for $\beta \leq 0.17\ \mathrm{rad}\ (10\,\mathrm{deg})$, which results in less than 1% error.

5.1.2.2 Rigid Body Dynamics

The rigid body dynamics are a simpler form of the complete dynamics in Eq.(5.8) and are obtained by omitting the effect of the damping and gravitational forces, matrices $\mathbf{D}$ and $\mathbf{G}$. This is the system's response when subjected only to a force $\mathbf{F}$ and a moment $\mathbf{M}$, Eq. (5.4). Because the centre of the body-fixed frame (Figure 5.2) coincides with the system's centre of gravity and with the principal axes of inertia, the mass matrix $\mathbf{H}$ is diagonal (Popp *et al.*, 2010).

The rigid body dynamics are:

$$\boldsymbol{\tau} = \mathbf{H}\dot{\mathbf{q}} + \mathbf{C}(\mathbf{q})\mathbf{q} \tag{5.14}$$

In Eq.(5.14), $\mathbf{H}$ is the mass matrix and $\mathbf{C}$ is the Coriolis matrix:

$$\mathbf{H} = diag\begin{pmatrix} \mathrm{m} & \mathrm{m} & \mathrm{m} & \mathrm{I_{xc}} & \mathrm{I_{yc}} & \mathrm{I_{zc}} \end{pmatrix} \tag{5.15}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & \mathrm{m}w & -\mathrm{m}v \\ 0 & 0 & 0 & -\mathrm{m}w & 0 & \mathrm{m}u \\ 0 & 0 & 0 & \mathrm{m}v & -\mathrm{m}u & 0 \\ 0 & \mathrm{m}w & -\mathrm{m}v & 0 & \mathrm{I_{zc}}r & -\mathrm{I_{yc}}q \\ -\mathrm{m}w & 0 & \mathrm{m}u & -\mathrm{I_{zc}}r & 0 & \mathrm{I_{xc}}p \\ \mathrm{m}v & -\mathrm{m}u & 0 & \mathrm{I_{yc}}q & -\mathrm{I_{xc}}r & 0 \end{bmatrix} \tag{5.16}$$

In Eq.(5.15) and Eq.(5.16) m is the robot's mass, $\mathrm{I_{xc}}$ is the inertia around the $\mathrm{X_b}$ axis, $\mathrm{I_{yc}}$ around the $\mathrm{Y_b}$ axis and $\mathrm{I_{zc}}$ around the $\mathrm{Z_b}$ axis (Figure 5.2 and Appendix A).

5.1.2.3 Damping & Gravitational Forces

A wheeled rover encounters two damping forces: friction $\mathbf{F}_f$, air resistance $\mathbf{F}_{ar}$.

$$\mathbf{D}(\mathbf{v}) = \mathbf{F}_f(\mathbf{v}) + \mathbf{F}_{ar}(\mathbf{v}) \tag{5.17}$$

The friction reaction $\mathbf{F}_f$ depends on the rover's weight and a frictional coefficient. The friction reaction $\mathbf{F}_f$ is decomposed into the friction forces $\mathbf{F}_{fric}(\mathbf{v})$ and moments $\mathbf{M}_{fric}$:

$$\mathbf{F}_{fric}(\mathbf{v}) = \mathrm{mg} \cdot \begin{bmatrix} \sigma_x & 0 & 0 \\ 0 & \sigma_y & 0 \\ 0 & 0 & \sigma_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{5.18}$$

$$\mathbf{M}_f = \mathrm{mg} \cdot \begin{bmatrix} r_p\sigma_p & 0 & 0 \\ 0 & r_q\sigma_q & 0 \\ 0 & 0 & r_r\sigma_r \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{5.19}$$

To increase the accuracy, it was found that an additional variable is required, the numerical velocity component along or about each axis (Worrall *et al.*, 2006), seen in Eq.(5.19). This velocity term has the effect of scaling the frictional term to suit the current wheel velocity (Worrall *et al.*, 2006). In Eq.(5.18), Eq.(5.19) the parameters g (gravity acceleration), σ_i (the friction coefficient) and r_j (moment arm) are in Appendix A.

The air resistance force F_{ar} is caused by the movement of the robot through the air:

$$\mathbf{F}_{ar}(u) = \begin{bmatrix} \mathrm{C_d A} \dfrac{\rho \cdot u^2}{2} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{5.20}$$

In Eq. (5.20) (Worrall, 2010; Nakayama, 2018), C_d is the drag coefficient, A is the surface area presented to the direction of travel, ρ is the air density and u is the velocity in the direction of travel. Axis X_b is the main axis of motion and so the other velocities and their effect on the air resistance force is negligible. The drag coefficient C_d depends on the shape of the robot and the values of parameters C_d, ρ and A are in Appendix A. Note that at low speeds or low density, the air resistance force is negligible. For example, using the values from Appendix A, the rover's air resistance is less than 0.001 N for a speed of 0.1 m/s.

Vector $\mathbf{G}(\boldsymbol{\eta})$ (Worrall, 2010), represents the effect of any gravitational forces and moments that act on the robot, where θ is the pitch angle and φ is the roll angle.

$$\mathbf{G}(\boldsymbol{\eta}) = \begin{bmatrix} -\mathrm{mg\,s}\theta \\ -\mathrm{mg\,s}\varphi\,\mathrm{c}\theta \\ -\mathrm{mg\,c}\varphi\,\mathrm{c}\theta \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.21}$$

Eq.(5.21) is based on the way gravitational forces act on the robot when moving on an incline, with the additional multiplication term $\mathrm{c}\theta$ along the $\mathrm{Y_e}$ and $\mathrm{Z_e}$ axis. This term is added to represent the coupling when both a roll and a pitch exist (Worrall, 2010).

5.1.3 Kinematics

The system's kinematics are the geometric transformations that map the body-fixed velocities to the Earth-fixed reference frame (Fossen, 2002):

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\mathbf{q} \tag{5.22}$$

Matrix $\mathbf{J}(\boldsymbol{\eta})$ relates the body-fixed linear and angular velocities $\mathbf{q}$ from Eq.(5.1) to the Earth-fixed position and orientation configuration vector $\boldsymbol{\eta}$ from Eq.(5.5).

The kinematics are further decomposed to the linear and angular velocities using $\mathbf{q}_1$ from Eq.(5.2) and $\mathbf{q}_2$ from Eq.(5.3) for the body-fixed velocities and $\boldsymbol{\eta}_1$ from Eq.(5.6) and $\boldsymbol{\eta}_2$ from Eq.(5.7) for the Earth-fixed position and orientation respectively.

Using the Euler angles φ, θ, ψ and the zyx convention (Fossen, 2002), the rotation matrix from the body-frame b to the Earth-fixed frame e is:

$$\begin{aligned} \mathbf{R}_b^e &= \mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\varphi} \\ \mathbf{R}_b^e &= \begin{bmatrix} c\psi c\theta & -s\psi c\varphi - c\psi s\theta s\varphi & s\psi s\varphi - c\psi c\varphi s\theta \\ s\psi c\theta & c\psi c\varphi - s\varphi s\theta s\psi & -c\psi s\varphi - s\theta s\psi c\varphi \\ s\theta & c\theta s\varphi & c\theta c\varphi \end{bmatrix} \end{aligned} \tag{5.23}$$

From the Earth-fixed frame to the local frame the rotation matrix is:

$$\mathbf{R}_e^b = \left(\mathbf{R}_b^e\right)^{-1} = \mathbf{R}^T{}_{x,\varphi}\mathbf{R}^T{}_{y,\theta}\mathbf{R}^T{}_{z,\psi} \tag{5.24}$$

The relationship between the body-fixed translation velocities $\mathbf{q_1}$ and those in the Earth-fixed frame is:

$$\dot{\boldsymbol{\eta}}_1 = \mathbf{R}_b^e\mathbf{q}_1 \tag{5.25}$$

Expanding Eq.(5.25) gives:

$$\begin{aligned} \dot{x} &= uc\psi c\theta + v\left(-c\psi s\theta s\varphi - s\psi c\varphi\right) + w\left(s\psi s\varphi - c\psi c\varphi s\theta\right) \\ \dot{y} &= us\psi c\theta + v\left(c\psi c\varphi - s\varphi s\theta s\psi\right) \ + w\left(-s\theta s\psi c\varphi - c\psi s\varphi\right) \\ \dot{z} &= us\theta \quad + vc\theta s\varphi \qquad\qquad + wc\theta c\varphi \end{aligned} \tag{5.26}$$

Using Euler integration, where dt is the time step and $i+1$ is the current instance, the numerical computation of the global position based on Eq.(5.25) is:

$$\boldsymbol{\eta}_1\left(i+1\right) = \boldsymbol{\eta}_1\left(i\right) + dt\left[\mathbf{R}_b^e\left(i\right)\mathbf{q}_1\left(i\right)\right] \tag{5.27}$$

If the translation velocities in the local frame are needed, then:

$$\mathbf{q}_1 = \left(\mathbf{R}_b^e\right)^{-1}\boldsymbol{\eta}_1 \tag{5.28}$$

Similarly, the relationship between the body-fixed angular velocity vector $\mathbf{q_2}$ and the Earth-fixed angular velocity is:

$$\dot{\boldsymbol{\eta}}_2 = \mathbf{T}_b^e\mathbf{q}_2 \tag{5.29}$$

The rotation matrix is:

$$\mathbf{T}_b^e = \begin{bmatrix} 1 & -s\varphi t\theta & c\varphi t\theta \\ 0 & c\varphi & s\varphi \\ 0 & {}^{-s\varphi}/_{c\theta} & {}^{c\varphi}/_{c\theta} \end{bmatrix}, \ \theta \neq \pm {}^{\pi}/_{2} \tag{5.30}$$

Expanding Eq.(5.29):

$$\begin{aligned}\dot{\varphi} &= p - qs\varphi t\theta + rc\varphi t\theta \\ \dot{\theta} &= \quad qc\varphi \quad + rs\varphi \\ \dot{\psi} &= \; -q\, {}^{s\varphi}/_{c\theta} + r\, {}^{c\varphi}/_{c\theta}\end{aligned} \tag{5.31}$$

The angular velocity vector $\mathbf{q}_2$ in the body frame cannot be integrated to obtain the angular coordinates[7] (Fossen, 2002). Instead, vector $\boldsymbol{\eta}_2$ from Eq.(5.29), Eq.(5.31) can be integrated to obtain the angles φ, θ, ψ.

For small angles $\delta\varphi, \delta\theta$ of less than 0.17 rad (10 deg) to achieve less than 1% error, the rotation matrix in Eq.(5.30) is simplified (Fossen, 2002):

$$\mathbf{T}_b^e \approx \begin{bmatrix} 1 & 0 & \delta\theta \\ 0 & 1 & \delta\varphi \\ 0 & -\delta\varphi & 1 \end{bmatrix} \tag{5.32}$$

Overall, the kinematics described by Eq.(5.22) and matrix $\mathbf{J}(\boldsymbol{\eta})$ are:

$$\begin{aligned}\dot{\boldsymbol{\eta}} &= \mathbf{J}(\boldsymbol{\eta})\mathbf{q} \\ \mathbf{J}(\boldsymbol{\eta}) &= \begin{bmatrix} \mathbf{R}_b^e & [\mathbf{0}]_{3x3} \\ [\mathbf{0}]_{3x3} & \mathbf{T}_b^e \end{bmatrix}_{6x6}\end{aligned} \tag{5.33}$$

5.1.4 Motor Dynamics

The model of the rover is augmented by the inclusion of the motor dynamics, the parameters are in Appendix A. The rover employs four identical DC motors and both wheels on each side receive the same input. This reduces the voltage inputs to two: a voltage V_1 to the left side and a voltage V_2 to the right side:

$$\mathbf{V} = [V_1 \quad V_2]^T \tag{5.34}$$

Since each side receives the same voltage input, the equations are written for the left ($i=1$) and right side ($i=2$).

[7] This is because (and in contrast to translation) in general rotations around different axis, do not commute; the order in which rotations are applied is important. Therefore, integrating the angular velocity to find angular position does not work in the general case.

The motor dynamics consist of an electrical and a mechanical component. The electrical component describes the behaviour of the current (I_i) in response to the voltage input (V_i), where $\mathrm{L_a}$ is the inductance of the circuit, R is the resistance and $\mathrm{K_e}$ is the EMF constant.

$$\frac{dI_i}{dt} = \frac{-\mathrm{R_a} I_i - \mathrm{K_e}\omega_i + V_i}{\mathrm{L_a}} \tag{5.35}$$

The dynamics of the mechanical component describe the interaction of the motor speed (ω_i) with the current (I_i); $\mathrm{J_m}$ is the motor moment of inertia, $\mathrm{K_t}$ is the torque constant, b is the viscous torque constant and ξ is the base friction coefficient.

$$\frac{d\omega_i}{dt} = \frac{\mathrm{K_t} I_i - \mathrm{b}\omega_i - \xi\omega_i}{\mathrm{J_m}},\ i = [1,2] \tag{5.36}$$

Each motor generates the torque τ_i and η_i is the motor efficiency.

$$\tau_i = \mathrm{K_t} I_i \eta_i,\ i = [1,2] \tag{5.37}$$

The force per wheel is the propulsion force that drives the rover and is generated by each wheel when actuated, where $\mathrm{r_w}$ is the wheel radius. Because each side receives the same voltage input, the front left and back left wheels produce the same force ($F_{fl} = F_{bl}$) and so do the front right and back right wheel ($F_{fr} = F_{br}$). This is the point where the motor dynamics are connected to the system dynamics via the wheel force from Eq.(5.9).

5.2 Trajectory generation for the four-wheeled robot

The trajectory of the rover is represented as a series of waypoints on a plane, each defined by a ($\mathrm{X_e}$, $\mathrm{Y_e}$) coordinate with a common origin, such as the example in Figure 5.5. The waypoints represent safe points and have been pre-selected by an appropriate methodology. The algorithm creates the trajectory between each successive waypoint with the robot stopping at each waypoint to turn on the spot to achieve the desired orientation, then move again; this is the turn then travel strategy described in Section 2.3.3 and when the rover moves there is either a surge motion or a yaw rotation. The benefit is that the time required to achieve the desired heading and position is smaller for the stop and turn strategy and the greater the change in heading is, the faster it is compared to the turn-while-travelling strategy (Cook, 2011).

Through this process, the trajectory is described by a series of forward surge movements along the X_b axis (defined on the local frame), each followed by a turn along the Z_b axis. The surge and turn movements are in a format that can then be used by the Inverse Simulation algorithm.

To define the trajectory, the desired constant speed is needed for both the forward and rotational velocity as well as the required time for acceleration and deceleration. The system's operating limits for the maximum surge and yaw are also considered. This information is used along with the distances and angles calculated from one waypoint to the next to evaluate the acceleration along the X_b axis (when moving forward) or the acceleration around the Z_b axis (when rotating) using a fixed time step, which is the same as the time step dt used later for the Inverse Simulation algorithm. The motion consists of an acceleration stage, followed by a stage of constant velocity and then a deceleration stage, such as the example in Figure 5.6.

The profile for the surge and yaw velocity is generated using a 6th order polynomial between two successive waypoints and the method presented here is based on (Thomson *et al.*, 2006; Worrall *et al.*, 2015). A 6th order polynomial ensures smooth trajectory profiles due to the continuity of the higher-order derivatives. This is the process of piecewise interpolation, where the polynomial is fitted through two successive waypoints and uses appropriate boundary continuity conditions to connect them.

The distance to travel between waypoint (x_{i-1}, y_{i-1}) to waypoint (x_i, y_i) is calculated for a maximum speed of 0.1 m/s for the surge velocity and 10 deg/s for the angular velocity next. These limits are in line with the actual capabilities of the robot, ensure that the air resistance is almost negligible and act as an indirect constraint on the minimum traverse time from one waypoint to the next. At each waypoint, it is determined if the rover is at the correct angle for the next traversal forward. If not, then the rover is commanded to turn on the spot until the desired angle of travel is achieved. The acceleration and deceleration $\dot{u}(\dot{r})$ are calculated using the following polynomial:

$$\dot{u} = \left[-7\left(\frac{20\mathrm{u}_{\max}}{e^7}\right)t_i\right]^6 + \left[6\left(\frac{70\mathrm{u}_{\max}}{e^6}\right)t_i\right]^5 - \left[5\left(\frac{84\mathrm{u}_{\max}}{e^5}\right)t_i\right]^4 + \left[4\left(\frac{35\mathrm{u}_{\max}}{e^4}\right)t_i\right]^3 \quad (5.38)$$

In Eq.(5.38), $\mathrm{u}_{\max}$ $(\mathrm{r}_{\max})$ is the maximum value of the surge (yaw) velocity and t_i is the current time. From Eq.(5.38), a time history for the acceleration is obtained; it is then numerically integrated to provide the velocities which are then further integrated to provide the displacements.

For example, Figure 5.5 shows a desired trajectory in the $\mathrm{X_e}$ - $\mathrm{Y_e}$ plane and the waypoints are marked with a cross (+). Figure 5.6 shows the corresponding surge and yaw velocities generated by this method.

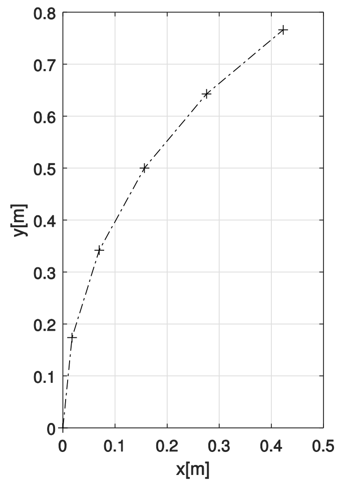


Figure 5.5: Arc Trajectory

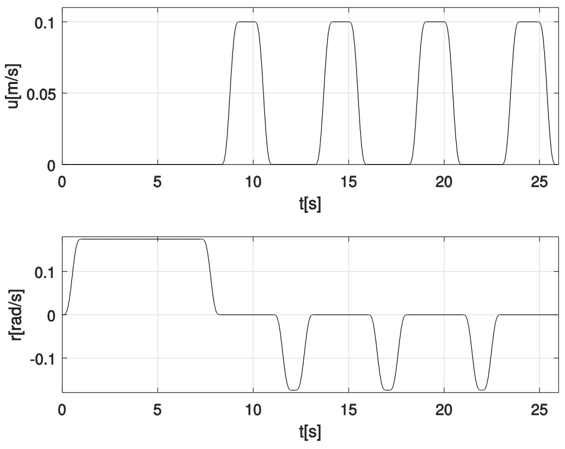


Figure 5.6: Arc Desired Surge Velocity (top), Desired Yaw Velocity (bottom)

The trajectory is an arc that is defined by six waypoints in the $\mathrm{X_e}$ - $\mathrm{Y_e}$ plane, the total duration is 25.9 s, and the total drive distance is 0.73 m. In Figure 5.6 the motion consists of an acceleration stage, followed by a stage of constant velocity and then a deceleration stage (a trapezoidal profile), with 0.1 m/s set for maximum surge velocity and 10 deg/s (0.17 rad/s) for maximum yaw velocity. When the rover moves there is either a surge motion or a yaw rotation.

These limits are reasonable for a rover traverse scenario, such as the one in (Correal *et al.*, 2016) and the actual drive data from Curiosity (Rankin *et al.*, 2020), where the maximum rotation rate is 0.168 rad/s.

The choice of outputs and the polynomial is further motivated by the fact that the trajectories will be used further along for demonstrating the Inverse Simulation algorithms and the following points were considered for the trajectory, based on those discussed in Section 3.2.

It is good practice to specify as desired outputs variables that are related to those that can be more strongly controlled and are a realistic representation of what the rover can achieve. The rover is differentially driven, which means that the rover's left and right sides are symmetrical to each other, and each side is actuated independently, with the wheels on each side always being actuated with the same signal. Therefore, the two directly controlled variables are the surge velocity u and the yaw angular velocity r and these two variables are sufficient for completely describing the pose of the robot moving on a plane.

The preference for using a high order polynomial for Inverse Simulation is balanced against the need to avoid the oscillations present in polynomial interpolation, which further motivates the choice of using an interpolating polynomial between two waypoints for the turn-then-travel method. In this way, a trajectory is designed from a set of waypoints with considerations specific to the task at hand and the application of Inverse Simulation.

Chapter 6 Application of Inverse Simulation to the Rover I: Non-linear Model

In this chapter, the general algorithm is applied to the specific problem of using Inverse Simulation for guidance by providing the changes in velocity, rotation, and acceleration for following the desired trajectory and for control by using these inputs to execute the desired trajectory, thus performing output tracking. The non-linear dynamic and kinematic rover model from Section 5.1 is used. The desired output is a series of trajectories that were produced using waypoints and the stop and turn method detailed in Section 5.2. The benefit is that the time required to achieve the desired heading and position is smaller for the stop and turn strategy (Cook, 2011). Finally, the assumption is made that the terrain is not significantly uneven or soft and that the robot moves on a planar, smooth, and rigid terrain.

The method of evaluating the Inverse Simulation signals is by applying them in a standard forward simulation and checking if the response matches the desired. In this way, the rover moves autonomously for traverses up to a few meters. This is similar to the strategy in Section 2.3.2, where the rover can go to a given location by executing a pre-defined path without any corrections (Correal *et al.*, 2016), similarly to open-loop control (Silva *et al.*, 2013).

The rover is differentially driven and the motion along the surge axis and the yaw rotation is directly controlled (the left and right wheel speed); these are sufficient so that the rover can turn on the spot, move in a straight line or move in a circular path. Therefore, the rover control inputs $\hat{\mathbf{u}}$ are reduced to torque $\hat{\tau}_1$ to the left side and torque $\hat{\tau}_2$ to the right side.

$$\hat{\mathbf{u}} = \hat{\boldsymbol{\tau}} = \begin{bmatrix} \hat{\tau}_1 & \hat{\tau}_2 \end{bmatrix}^T \tag{6.1}$$

The state-space model describing the rigid body dynamics of the four-wheeled rover, with m=6 states and k=2 control inputs, is:

$$\dot{\mathbf{q}} = \mathbf{f}\left(\mathbf{q}(t), \hat{\mathbf{u}}(t)\right) = \mathbf{H}^{-1}\left\{\boldsymbol{\tau} - \mathbf{C}(\mathbf{q})\mathbf{q} - \mathbf{D}(\mathbf{q})\mathbf{q} - \mathbf{G}(\boldsymbol{\eta})\right\} \tag{6.2}$$

Vector $\boldsymbol{\tau}$ are the forces and moments acting on the rover, vector $\mathbf{q}$ represents the state velocity vector in the body-fixed frame, and $\hat{\mathbf{u}}$ is the control input.

The output equation $\mathbf{y}$ is:

$$\mathbf{y} = \begin{bmatrix} u \\ r \end{bmatrix} = \mathbf{g}\left(\mathbf{q}(t), \hat{\mathbf{u}}(t)\right) \tag{6.3}$$

The desired output $\mathbf{g}_d$ is defined over the time interval $\mathbf{T}$ and is discretised with a time step dt.

$$\mathbf{g}_d = \begin{bmatrix} u_d \\ r_d \end{bmatrix} \tag{6.4}$$

$$\begin{aligned} &\mathbf{y}_d(t_i) = \mathbf{g}_d(t_i), \forall t \in \mathbf{T} \\ &\mathbf{T} = \begin{bmatrix} t_0 & \dots & t_i & \dots & t_N \end{bmatrix} \\ &t_i = idt, i \in [0, N] \end{aligned} \tag{6.5}$$

The Inverse Simulation problem is stated as:

Given a desired output $\mathbf{g}_d(t)$ defined as a vector of surge velocities $(\mathbf{u}_d)$ and a vector of yaw velocities $(\mathbf{r}_d)$ over a time interval $\mathbf{T}$, find the vector of suitable control inputs $\hat{\mathbf{u}}$ so that the vector of outputs $\mathbf{g}$ of the system is equal to the desired $\mathbf{g}_d(t)$ within the specified tolerance. The control inputs are the left and right side torques, therefore $\hat{\mathbf{u}} = \hat{\boldsymbol{\tau}} = \begin{bmatrix} \hat{\tau}_1 & \hat{\tau}_1 \end{bmatrix}^T$.

6.1 Differentiation

There are six states, two desired outputs $(u_d,\ r_d)$, two system outputs to control $(u,\ r)$ and two control inputs $(\hat{\tau}_1,\ \hat{\tau}_2)$ to identify at every t_i over the time interval $\mathbf{T}$. The state and output equations Eq.(6.2), Eq.(6.3) are discretised N times over the time interval $\mathbf{T}$ with a step of dt. At t_i given the desired output $\mathbf{g}_d(t_i)$, the functions $\mathbf{F}_1$ and $\mathbf{F}_2$ are defined to find the value of the input $\hat{\boldsymbol{\tau}}$ and the states $\mathbf{q}$ respectively, as described in Section 4.1.1.1.

The size of $\mathbf{F}_1$ is 6×1 and of $\mathbf{F}_2$ is 2×1.

$$\mathbf{F}_1 = \mathbf{f}\left(\mathbf{q}(t_i), \hat{\boldsymbol{\tau}}(t_i)\right) - \left[\frac{\mathbf{q}(t_i) - \mathbf{q}(t_{i-1})}{t_i - t_{i-1}}\right] = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 & F_5 & F_6 \end{bmatrix}^T \tag{6.6}$$

$$\mathbf{F}_2 = \mathbf{g}(t_i) - \mathbf{g_d}(t_i) \begin{bmatrix} u(t_i) - u_d(t_i) \\ r(t_i) - r_d(t_i) \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \tag{6.7}$$

Then, by using the desired output at every i discretisation point, a time series of suitable control inputs $\hat{\boldsymbol{\tau}}(t_i)$ and the corresponding states $\mathbf{q}(t_i)$ is found. This is done by solving the system described by Eq.(6.6), (6.7) using the Newton-Raphson method.

At each inner iteration n, $\mathbf{q}$ and $\hat{\boldsymbol{\tau}}$ are updated using Eq.(6.8), where $\mathbf{J}(t_i)$ is the Jacobian of $\mathbf{F}_1$ and $\mathbf{F}_2$ at t_i. The method converges when the calculated values of the control input and the states are such that $\mathbf{F}_1$ and $\mathbf{F}_2$ are both equal to zero, within a certain tolerance.

$$\begin{bmatrix} \mathbf{q}_n(t_i) \\ \hat{\boldsymbol{\tau}}_n(t_i) \end{bmatrix} = \begin{bmatrix} \mathbf{q}_{n-1}(t_i) \\ \hat{\boldsymbol{\tau}}_{n-1}(t_i) \end{bmatrix} - \underbrace{\begin{bmatrix} \dfrac{\delta \mathbf{F}_1}{\delta \mathbf{q}}(t_i) & \dfrac{\delta \mathbf{F}_1}{\delta \hat{\boldsymbol{\tau}}}(t_i) \\ \dfrac{\delta \mathbf{F}_2}{\delta \mathbf{q}}(t_i) & \dfrac{\delta \mathbf{F}_2}{\delta \hat{\boldsymbol{\tau}}}(t_i) \end{bmatrix}^{-1}}_{\mathbf{J}} \cdot \begin{bmatrix} \mathbf{F}_1\left(\mathbf{q}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_i)\right) \\ \mathbf{F}_2\left(\mathbf{q}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_i)\right) \end{bmatrix} \tag{6.8}$$

If all six states are perturbed in the Jacobian, its size will be 8×8. However, only the states u and r are directly controllable, whereas the remaining four states (v, w, p, q) are not. Therefore, it makes physical sense to only perturb u and r, in which case the Jacobian becomes:

$$\mathbf{J} = \begin{bmatrix} \frac{\delta F_1}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta F_1}{\delta r} & \frac{\delta F_1}{\delta \tau_1} & \frac{\delta F_1}{\delta \tau_2} \\ \frac{\delta F_2}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta F_2}{\delta r} & \frac{\delta F_2}{\delta \tau_1} & \frac{\delta F_2}{\delta \tau_2} \\ \frac{\delta F_3}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta F_3}{\delta r} & \frac{\delta F_3}{\delta \tau_1} & \frac{\delta F_3}{\delta \tau_2} \\ \frac{\delta F_4}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta F_4}{\delta r} & \frac{\delta F_4}{\delta \tau_1} & \frac{\delta F_4}{\delta \tau_2} \\ \frac{\delta F_5}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta F_5}{\delta r} & \frac{\delta F_5}{\delta \tau_1} & \frac{\delta F_5}{\delta \tau_2} \\ \frac{\delta F_6}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta F_6}{\delta r} & \frac{\delta F_6}{\delta \tau_1} & \frac{\delta F_6}{\delta \tau_2} \\ \frac{\delta g_1}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta g_1}{\delta r} & \frac{\delta g_1}{\delta \tau_1} & \frac{g_1}{\delta \tau_2} \\ \frac{\delta g_2}{\delta u} & 0 & 0 & 0 & 0 & \frac{\delta g_2}{\delta r} & \frac{\delta g_2}{\delta \tau_1} & \frac{\delta g_2}{\delta \tau_2} \end{bmatrix} \tag{6.9}$$

The rank of $\mathbf{J}$ in Eq.(6.9) is 4. This is expected since the system has two directly controllable states (u, r) and two control inputs ($\hat{\tau}_1$, $\hat{\tau}_2$).

If only the directly controllable states are considered for the Jacobian, then it is reduced to a 4×4:

$$\mathbf{J} = \begin{bmatrix} \frac{\delta F_1}{\delta u} & \frac{\delta F_1}{\delta r} & \frac{\delta F_1}{\delta \hat{\tau}_1} & \frac{\delta F_1}{\delta \hat{\tau}_2} \\ \frac{\delta F_2}{\delta u} & \frac{\delta F_2}{\delta r} & \frac{\delta F_2}{\delta \hat{\tau}_1} & \frac{\delta F_2}{\delta \hat{\tau}_2} \\ \frac{\delta g_1}{\delta u} & \frac{\delta g_1}{\delta r} & \frac{\delta g_1}{\delta \hat{\tau}_1} & \frac{\delta g_1}{\delta \hat{\tau}_2} \\ \frac{\delta g_2}{\delta u} & \frac{\delta g_2}{\delta r} & \frac{\delta g_2}{\delta \hat{\tau}_1} & \frac{\delta g_2}{\delta \hat{\tau}_2} \end{bmatrix} \tag{6.10}$$

The sway velocity v is not matched dynamically to the system's actuators and therefore is not directly controllable. As was discussed in Section 5.1.2.1, however, the sway velocity is strongly coupled to the surge velocity via the slip angle, Eq.(5.10) and Section 5.1.2.1. This interaction provides an indirect control of the sway and acts as an additional constraint when solving for the control input. Consider also that a differential driven system by its nature does not move sideways (i.e., slip) and for low speeds, this is a reasonable assumption. Hence, $\mathbf{v}_d = 0$ is used.

Furthermore, it was observed during the initial simulations that including the perturbation of the sway velocity v and selecting as an additional output to control the sway velocity, the overall results were significantly improved. This is examined further in Section 8.2.3. Therefore, there are three vectors of desired outputs over the time interval: $\mathbf{u}_d$, $\mathbf{r}_d$ and the sway velocity $\mathbf{v}_d$ which is set to zero.

The outputs to control are now $(u,\ v,\ r)$ the inputs $(\hat{\tau}_1,\ \hat{\tau}_2)$ to identify are as before. The number of outputs is now higher than the number of the control inputs, making the system overdetermined. All outputs, however, are directly related to those that can be strongly controlled. The output equation $\mathbf{F}_2$ is 3×1 and the Jacobian is 6×5:

$$\mathbf{F}_2 = \begin{bmatrix} u(t_i) - u_d(t_i) \\ v(t_i) - v_d(t_i) \\ r(t_i) - r_d(t_i) \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} \tag{6.11}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\delta F_1}{\delta u} & \frac{\delta F_1}{\delta v} & \frac{\delta F_1}{\delta r} & \frac{\delta F_1}{\delta \hat{\tau}_1} & \frac{\delta F_1}{\delta \hat{\tau}_2} \\ \frac{\delta F_2}{\delta u} & \frac{\delta F_2}{\delta v} & \frac{\delta F_2}{\delta r} & \frac{\delta F_2}{\delta \hat{\tau}_1} & \frac{\delta F_2}{\delta \hat{\tau}_2} \\ \frac{\delta F_6}{\delta u} & \frac{\delta F_6}{\delta v} & \frac{\delta F_6}{\delta r} & \frac{\delta F_6}{\delta \hat{\tau}_1} & \frac{\delta F_6}{\delta \hat{\tau}_2} \\ \frac{\delta g_1}{\delta u} & \frac{\delta g_1}{\delta v} & \frac{\delta g_1}{\delta r} & \frac{\delta g_1}{\delta \hat{\tau}_1} & \frac{\delta g_1}{\delta \hat{\tau}_2} \\ \frac{\delta g_2}{\delta u} & \frac{\delta g_2}{\delta v} & \frac{\delta g_2}{\delta r} & \frac{\delta g_2}{\delta \hat{\tau}_1} & \frac{\delta g_2}{\delta \hat{\tau}_2} \\ \frac{\delta g_3}{\delta u} & \frac{\delta g_3}{\delta v} & \frac{\delta g_3}{\delta r} & \frac{\delta g_3}{\delta \hat{\tau}_1} & \frac{\delta g_3}{\delta \hat{\tau}_2} \end{bmatrix} \tag{6.12}$$

The Jacobian is calculated using finite differences and a first-order, forward difference formula is used. A suitable factorisation method has to be used to find the pseudoinverse of $\mathbf{J}$ required for Eq.(6.8). During the simulations, the MATLAB package presented by (Davis, 2013) was used that is suitable for square, orthogonal, rank deficient and over/under determined systems and provides reliable results, even for near-singular systems. The algorithm extends the functionality of the backslash operator (\) by improving how rank-deficient systems are handled and selects the best factorisation method for a matrix $\mathbf{A}$ m×n that is then used to solve the system to provide the least square solution. The selection is between LU ($\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix), QR ($\mathbf{Q}$ is a matrix with orthonormal columns and $\mathbf{R}$ an upper triangular matrix), SVD (Singular Value Decomposition), Cholesky and COD (complete orthogonal decomposition) (Davis, 2013). This method ensures that the solution is the best and most efficient in terms of errors and execution time (Davis, 2013); a warning is given if no suitable method is found. For the Jacobian $\mathbf{J}$ defined in Eq.(6.12), COD is the selected method, which was expected since the Jacobian matrix is orthogonal and rank. This choice is further examined in Section 8.1 and Appendix G contains more details on the factorisation methods.

Because only the states $u,\ v,\ r$ are used for the Jacobian, only $u,\ v,\ r\ ,\ \hat{\tau}_1,\ \hat{\tau}_2$ are updated in Eq.(6.8), which results in an updated reduced vector for the states and a fully updated control vector, Eq.(6.14).

$$\underbrace{\begin{bmatrix} u_n \\ v_n \\ r_n \\ \hat{\tau}_{1,n} \\ \hat{\tau}_{2,n} \end{bmatrix}}_{\mathbf{Q}_n \quad t_i} = \underbrace{\begin{bmatrix} u_{n-1} \\ v_{n-1} \\ r_{n-1} \\ \hat{\tau}_{1,n-1} \\ \hat{\tau}_{2,n-1} \end{bmatrix}}_{\mathbf{Q}_{n-1} \quad t_i} - \underbrace{\begin{bmatrix} \dfrac{\delta \mathbf{F}_1}{\delta \mathbf{q}}(t_i) & \dfrac{\delta \mathbf{F}_1}{\delta \hat{\boldsymbol{\tau}}}(t_i) \\ \dfrac{\delta \mathbf{F}_2}{\delta \mathbf{q}}(t_i) & \dfrac{\delta \mathbf{F}_2}{\delta \hat{\boldsymbol{\tau}}}(t_i) \end{bmatrix}}_{\mathbf{J}}^{-1} \cdot \begin{bmatrix} \mathbf{F}_1\left(\mathbf{q}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_i)\right) \\ \mathbf{F}_2\left(\mathbf{q}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_i)\right) \end{bmatrix} \tag{6.13}$$

$$\mathbf{Q}_n\big|_{t_i} = \begin{bmatrix} \mathbf{q}_r \\ \hat{\boldsymbol{\tau}} \end{bmatrix}\Bigg|_{t_i}$$
$$\mathbf{q}_r = \begin{bmatrix} u_n \\ v_n \\ r_n \end{bmatrix},\ \hat{\boldsymbol{\tau}} = \begin{bmatrix} \hat{\tau}_{1,n} \\ \hat{\tau}_{2,n} \end{bmatrix} \tag{6.14}$$

Therefore, the remaining states $(w,\ p,\ q)$ must be estimated. These correspond to moving along the body-fixed $\mathrm{Z_b}$ axis (heave), rotating around the body-fixed $\mathrm{X_b}$ axis (roll), and rotating around the body-fixed $\mathrm{Y_b}$ axis (pitch). When moving on a flat plane (or a surface that can be considered flat with a good degree of accuracy), the states $(\mathrm{w},\ \mathrm{p},\ \mathrm{q})$ must all be zero due to the physics of the problem. This also confirms the choice to not perturb w, p, q in Eq.(6.9). In total, the states $\left(w_{n-1}, p_{n-1}, q_{n-1}\right)\Big|\left(t_i\right)$ from the previous iteration n-1 are always set to zero and the full state vector with the updated only $u,\ v,\ r\ ,\ \hat{\tau}_1,\ \hat{\tau}_2$ is now:

$$\mathbf{q}_n\left(t_i\right)=\left[u_n \quad v_n \quad w_{n-1} \quad p_{n-1} \quad q_{n-1} \quad r_n\right]\left(t_i\right) \tag{6.15}$$

Then, the new values for iteration n for $\mathbf{F}_1$, Eq.(6.6), and $\mathbf{F}_2$, Eq.(6.11), are calculated using $\mathbf{q}_n\left(t_i\right)$ from Eq.(6.15). If both are equal to zero, within a certain tolerance, the algorithm converges.

$$\begin{aligned} \left\| \mathbf{F_1}\big|_n - \mathbf{F_1}\big|_{n-1} \right\| \le tol \\ \left\| \mathbf{F_2}\big|_n - \mathbf{F_2}\big|_{n-1} \right\| \le tol \end{aligned} \tag{6.16}$$

The dynamics for the next step $\dot{\mathbf{q}}\left(t_{i+1}\right)$ are calculated and integrated from Eq.(6.2) using the updated control vector and states from Eq.(6.14) and Eq.(6.15). If convergence is not achieved, the vector state for the next iteration n+1 is Eq.(6.15) and the input is from Eq.(6.14).

6.2 Integration

The algorithm follows the general process discussed in Section 4.1.1.2. From the state equation Eq.(6.2) and the output equation Eq.(6.3) there are two desired outputs $\left(u_d,\ r_d\right)$, two system outputs to control $\left(u,\ r\right)$ and two control inputs $\left(\hat{\tau}_1,\ \hat{\tau}_2\right)$ to identify at every t_i over the time interval T. The error function $\mathbf{f}_e$ between the actual output $\mathbf{y}$ from Eq.(6.3) and the desired output $\mathbf{g_d}\left(t\right)$ from Eq. (6.4) is:

$$\mathbf{f_e}=\mathbf{y}\left(t_i\right)-\mathbf{g_d}\left(t_i\right)=\mathbf{g}\left(\mathbf{q}\left(t_i\right),\hat{\boldsymbol{\tau}}\left(t_{i-1}\right)\right)=\begin{bmatrix} u\left(t_i\right)-u_d\left(t_i\right) \\ r\left(t_i\right)-r_d\left(t_i\right) \end{bmatrix} \tag{6.17}$$

The Jacobian $\mathbf{J}_e$ of the error function is square and has a size of 2×2 since there are two inputs and two outputs.

$$\mathbf{J_e} = \frac{\delta \mathbf{y}}{\delta \hat{\boldsymbol{\tau}}} = \begin{bmatrix} \dfrac{\delta y_1}{\delta \hat{\tau}_1} & \dfrac{\delta y_1}{\delta \hat{\tau}_2} \\ \dfrac{\delta y_2}{\delta \hat{\tau}_1} & \dfrac{\delta y_2}{\delta \hat{\tau}_2} \end{bmatrix} \tag{6.18}$$

The Jacobian in Eq.(6.18) is calculated using central finite differences. To do so, the perturbations of the control inputs need to be considered and then the corresponding change in the outputs. The perturbation matrix for the control inputs is:

$$\begin{gathered} p(\hat{\tau}_1, \hat{\tau}_2) = \begin{bmatrix} \hat{\tau}_1 + \delta\hat{\tau}_1 & \hat{\tau}_2 - \delta\hat{\tau}_2 \\ \hat{\tau}_1 - \delta\hat{\tau}_1 & \hat{\tau}_2 + \delta\hat{\tau}_2 \\ \hat{\tau}_1 + \delta\hat{\tau}_1 & \hat{\tau}_2 + \delta\hat{\tau}_2 \\ \hat{\tau}_1 - \delta\hat{\tau}_1 & \hat{\tau}_2 - \delta\hat{\tau}_2 \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{\tau}}_-^+ \\ \hat{\boldsymbol{\tau}}_+^- \\ \hat{\boldsymbol{\tau}}_+^+ \\ \hat{\boldsymbol{\tau}}_-^- \end{bmatrix} \\ \hat{\boldsymbol{\tau}}_-^+ = \begin{bmatrix} \hat{\tau}_1 + \delta\hat{\tau}_1 & \hat{\tau}_2 - \delta\hat{\tau}_2 \end{bmatrix} \\ \ldots \\ \delta\hat{\tau} = \delta\hat{\tau}_1 = \delta\hat{\tau}_2 \end{gathered} \tag{6.19}$$

The perturbation matrix represents four different cases that correspond to all the possible movement combinations of the rover. The first line, $\hat{\boldsymbol{\tau}}_-^+$, corresponds to the case where the left side torque is increased and the right side is decreased (i.e., a right turn), and so on for the rest (left turn, forward movement, backward movement). For each of the four cases, the control inputs from Eq.(6.19) are applied to the system equation Eq.(6.2), the new outputs are produced for each case from Eq.(6.3) and the error for each case from Eq.(6.17) is:

$$\begin{gathered} \mathbf{f}_{\mathbf{e}-}^{+}\left(\hat{\boldsymbol{\tau}}_-^+\right) = \begin{bmatrix} u_-^+ - u_d \\ r_-^+ - r_d \end{bmatrix} = \begin{bmatrix} \delta u_{+-} \\ \delta r_{+-} \end{bmatrix} \\ \mathbf{f}_{\mathbf{e}+}^{-}\left(\hat{\boldsymbol{\tau}}_+^-\right) = \begin{bmatrix} u_+^- - u_d \\ r_+^- - r_d \end{bmatrix} = \begin{bmatrix} \delta u_{-+} \\ \delta r_{-+} \end{bmatrix} \\ \mathbf{f}_{\mathbf{e}+}^{+}\left(\hat{\boldsymbol{\tau}}_+^+\right) = \begin{bmatrix} u_+^+ - u_d \\ r_+^+ - r_d \end{bmatrix} = \begin{bmatrix} \delta u_{++} \\ \delta r_{++} \end{bmatrix} \\ \mathbf{f}_{\mathbf{e}-}^{-}\left(\hat{\boldsymbol{\tau}}_-^-\right) = \begin{bmatrix} u_-^- - u_d \\ r_-^- - r_d \end{bmatrix} = \begin{bmatrix} \delta u_{--} \\ \delta r_{--} \end{bmatrix} \end{gathered} \tag{6.20}$$

Using matrix notation, the error matrix is $\mathbf{F}_{error}$ and its size is 2x4:

$$\mathbf{F}_{error} = \begin{bmatrix} \mathbf{f}_{e-}^{+} & \mathbf{f}_{e+}^{-} & \mathbf{f}_{e+}^{+} & \mathbf{f}_{e-}^{-} \end{bmatrix} = \begin{bmatrix} \delta u_{+-} & \delta u_{-+} & \delta u_{++} & \delta u_{--} \\ \delta r_{+-} & \delta r_{-+} & \delta r_{++} & \delta r_{--} \end{bmatrix} \tag{6.21}$$

The Jacobian $\mathbf{J}_e$ can now be calculated using central differences. Then, From Eq.(4.11), the new control inputs are calculated, where n is the current Newton-Raphson inner iteration. Using these new input estimates the states and outputs are updated using the system dynamics Eq.(5.8). Then, the error function $\mathbf{f}_{e,n}$ is calculated using the updated outputs, Eq.(6.23). If the error $\mathbf{f}_{e,n}$ is within the acceptable range, then the calculated control input vector $\hat{\boldsymbol{\tau}}_n$ is the required for this time instance, $\hat{\boldsymbol{\tau}}_n(t_{i-1}) = \hat{\boldsymbol{\tau}}_n(t_{i-1})$.

$$\hat{\boldsymbol{\tau}}_n(t_{i-1}) = \hat{\boldsymbol{\tau}}_{n-1}(t_{i-1}) - \mathbf{J}_e^{-1}\left(\mathbf{x}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_{i-1})\right) \cdot \mathbf{f}_e\left(\mathbf{x}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_{i-1})\right) \tag{6.22}$$

$$\mathbf{f}_{e,n} = \left. \begin{bmatrix} u(\hat{\boldsymbol{\tau}}_n) - u_d \\ r(\hat{\boldsymbol{\tau}}_n) - r_d \end{bmatrix} \right|_{t_{i-1}} \tag{6.23}$$

The MATLAB backslash operator (\) was selected as the best method for solving a square system like that in Eq.(4.11). This choice is further examined in Section 8.1 and Appendix G provides more details.

6.3 Test Trajectories

The trajectories used are calculated using the stop and turn method and the polynomial from Section 5.2. All the trajectories are a combination of linear travel along the surge axis and a yaw rotation and are depicted in the global X_e, Y_e system, as in Figure 5.2. The following two basic trajectories are first demonstrated.

Forward: Move forward 1 m, Figure 6.1. This test shows that the algorithms can generate the required control inputs for driving linearly without heading changes.

Left Right Turn: Forward for 1 m, turn 90 deg counter clockwise, forward 1 m, turn 90 deg clockwise and move 1m forward, Figure 6.2. This shows that the required control inputs to enable forward movement followed by yaw rotation, followed by a consecutive forward, and turn movement, can be generated.

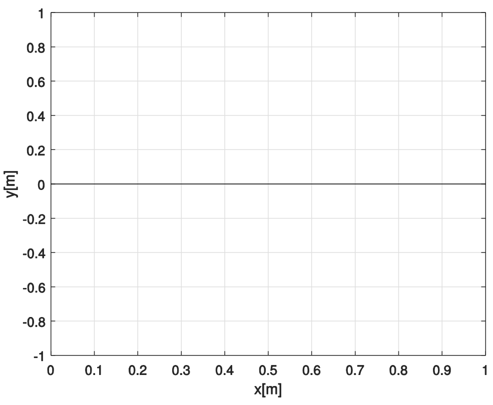


Figure 6.1: Forward

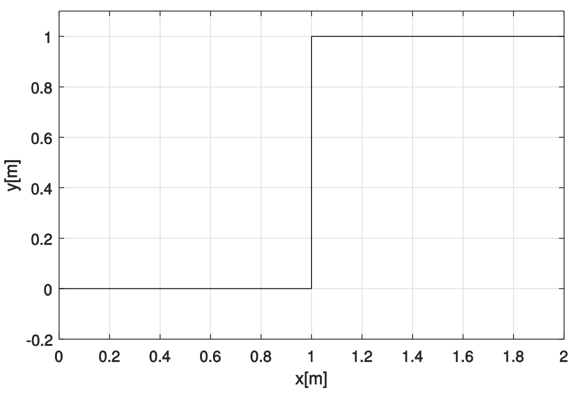


Figure 6.2: Left – Right Turn

Having demonstrated these two basic trajectories, four more complex trajectories are examined, to ensure that control inputs for the desired trajectories can be generated.

Arc: The rover will have to move between closely spaced waypoints. Two consecutive waypoints relate to a forward line and a turn, creating overall an approximation of an arc, Figure 6.3.

Rhombus: This trajectory demonstrates a closed path with multiple turns, Figure 6.4.

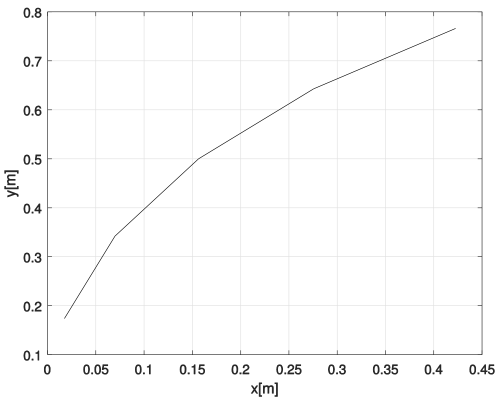


Figure 6.3: Arc

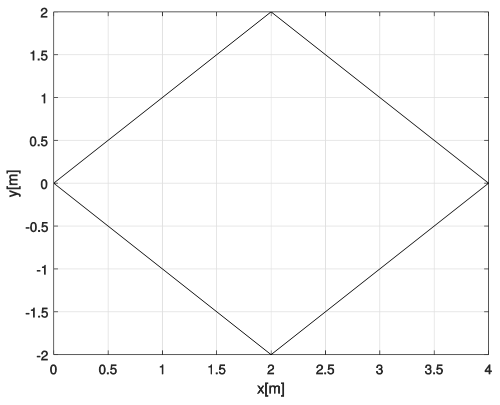


Figure 6.4: Rhombus

Valley: One of the most difficult scenarios would be a valley run, where obstacles are present on both sides and in proximity. This run consists of a series of forward motions with rotations within tight spaces, Figure 6.5.

Long Arc: The long arc run involves traversing tens of meters with several pose alterations, Figure 6.6. This test will examine the error built over time over a demanding path.

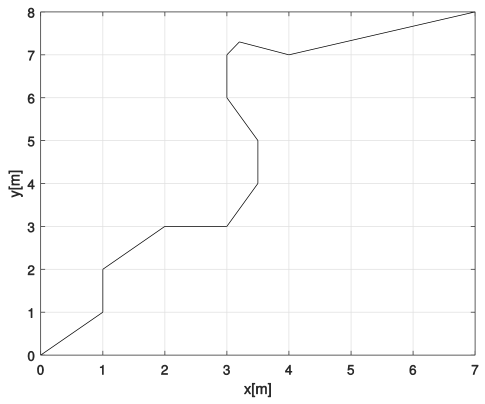


Figure 6.5: Valley

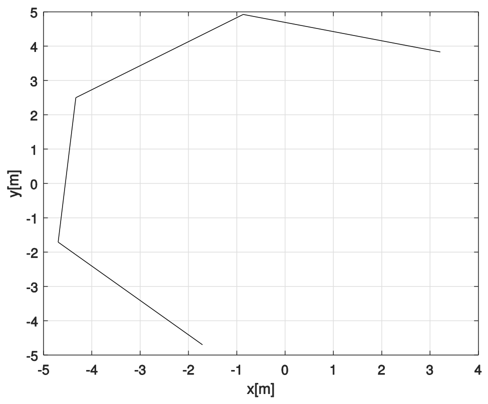


Figure 6.6: Long Arc

The waypoints used for each test scenario are shown in Table 6.1.

Table 6.1: Test Trajectories Overview

Test	Total Drive Distance [m]	Duration [s]	Waypoints
Forward	1	11.01	[0 0; 1 0]
Left Right Turn	3	53	[0 0; 1 0; 1 1; 2 1]
Arc	0.73	25.9	[0.0175 0.174; 0.0698 0.342; 0.156 0.500; 0.276 0.643; 0.423 0.766]
Rhombus	11.31	152.63	[0 0; 2 2; 4 0; 2 -2; 0 0]
Valley	13.44	203.70	[0 0; 1 1; 1 2;2 3;3 3;3.5 4; 3.5 5; 3 6;3 7;3.2 7.3; 4 7; 7 8];
Long Arc	16.91	208.55	[3.2139 3.8302; -0.862 4.9240; -4.3301 2.500; -4.6985 -1.7101; -1.7101 -4.6985]

6.4 Results for Stop and Turn Trajectories

This section presents the results of Inverse Simulation when applied to trajectory tracking. For each trajectory, the objective is to calculate suitable control inputs that when applied to the rover the desired trajectory is achieved. A series of waypoints are first defined and then a trajectory between them is generated. Inverse Simulation calculates the control inputs for each trajectory. Then, these inputs are applied to the forward system, and it is checked whether the resulting trajectory matches the desired. The control inputs from Inverse Simulation are nominal and the resulting outputs and states are also nominal. The desired trajectory is represented in the local frame by a vector of desired surge velocities

$\mathbf{u_d}$ and a vector of desired yaw velocities $\mathbf{r_d}$ that have been evaluated in advance. Throughout this analysis, the rover operates on a horizontal plane.

The simulation parameters used for both the Differentiation and Integration Inverse Simulation algorithm are the discretisation time step dt, the convergence tolerance for the Newton-Raphson method, the maximum number of iterations for convergence and the initial control estimate. The rover starts from rest and the initial motor torque is zero, with a very small non-zero value used as an initial estimate. The number of iterations is set to 30 per time step and in practice, it is usually between 1 to 4 iterations per time step. The values for these parameters were selected to ensure accuracy and decreased execution time and are shown in Table 6.2. Additionally, when selecting the time step dt, the time constant of the motors was considered.

Table 6.2: Inverse Simulation Parameters (Baseline)

Parameter	Value
dt [s]	0.01
convergence tolerance (tol) [Nm]	$5\ 10^{-7}$
initial control estimate [Nm]	$2.5\ 10^{-7}$
maximum iterations	30

6.4.1 Trajectory Results Figures

The results for each trajectory when applying the Inverse Simulation control inputs are presented in the following sections. For each case, the actual path (solid line) versus the desired (dashed line) are depicted in the global X_e, Y_e coordinate system and the related calculated control inputs are shown. The results for the Forward, Rhombus and Valley trajectories are in Appendix F.

6.4.1.1 Left Right

The trajectory results are presented in Figure 6.7, Figure 6.8 and there is no discernible deviation from the desired trajectory. The control inputs from Inverse Simulation are in Figure 6.9, Figure 6.10.

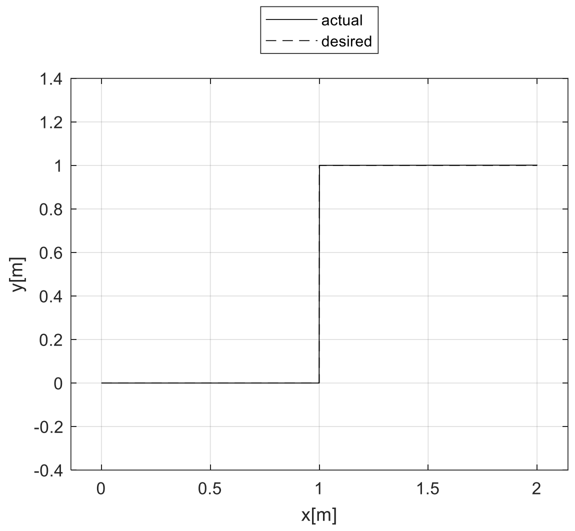


Figure 6.7: Left – Right, Differentiation

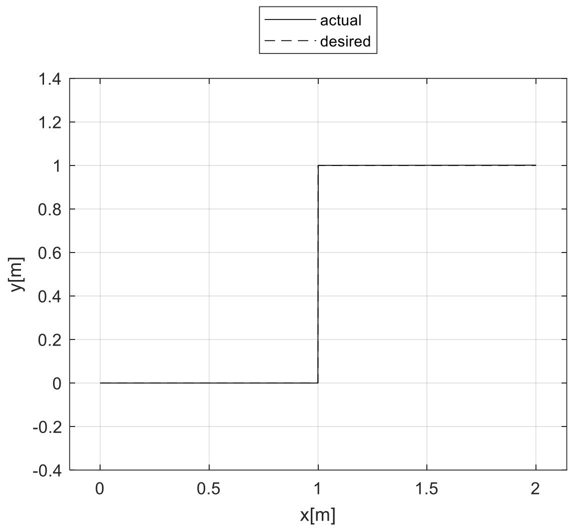


Figure 6.8: Left – Right, Integration

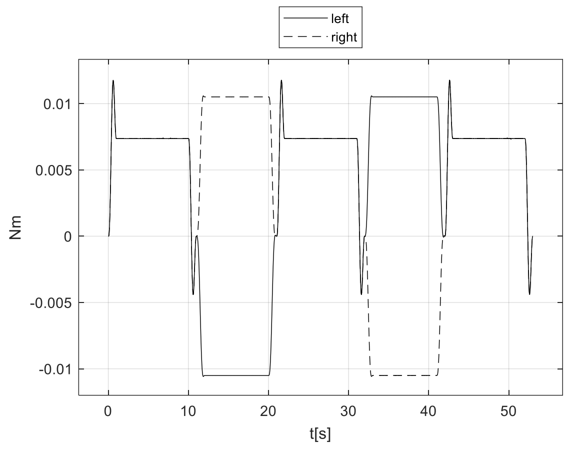


Figure 6.9: Left – Right Control Input, Differentiation

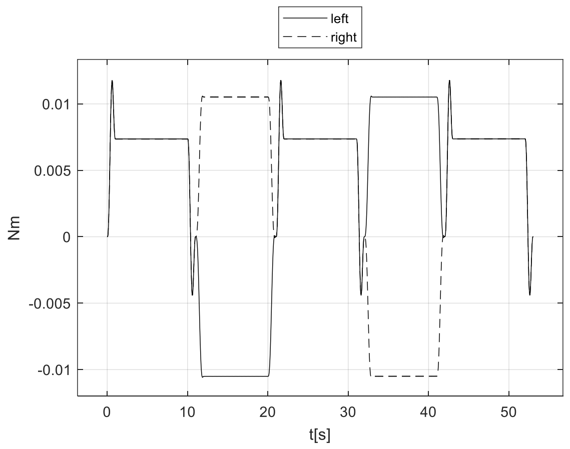


Figure 6.10: Left – Right Control Input, Integration

For Differentiation, in Figure 6.7 the final error is $1.61\ 10^{-3}\ \mathrm{m}$ on the global X_e axis and $1.31\ 10^{-3}\ \mathrm{m}$ on the Y_e axis. For Integration, in Figure 6.8 the final error is $1.55\ 10^{-3}\ \mathrm{m}$ on the X_e axis and $1.22\ 10^{-3}\ \mathrm{m}$ on the Y_e axis. The Left-Right trajectory tests the ability of the algorithm to produce a series of forward movements and sharp turns and the method responds well.

6.4.1.2 Arc

The trajectory results are presented in Figure 6.11, Figure 6.12 and there is no significant deviation from the desired trajectory. The control inputs from Inverse Simulation are in Figure 6.13, Figure 6.14. Both methods produce good results when required to produce a series of control inputs for navigating between closely spaced waypoints.

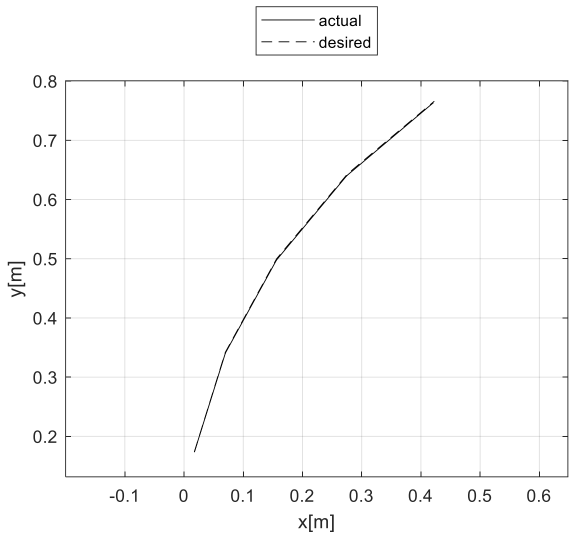


Figure 6.11: Arc, Differentiation

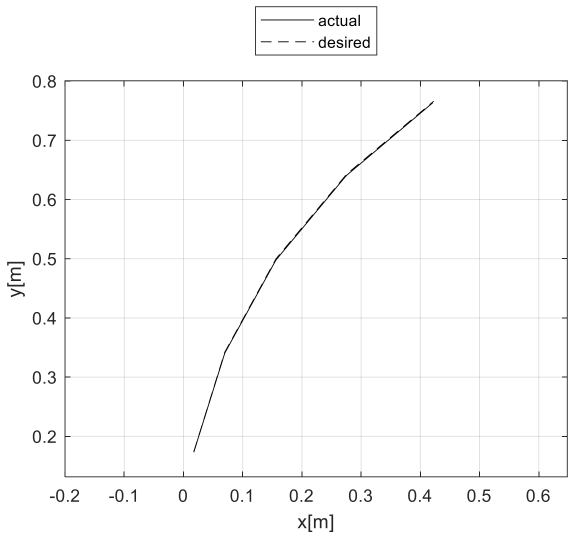


Figure 6.12: Arc, Integration

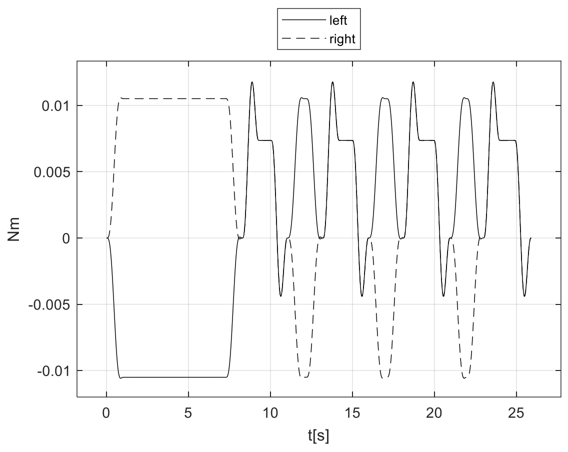


Figure 6.13: Arc Control Input, Differentiation

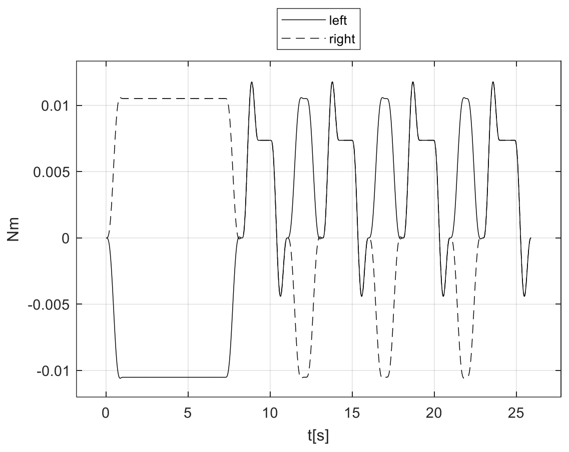


Figure 6.14: Arc Control Input, Integration

For Differentiation, in Figure 6.11 the final error is $1.80\ 10^{-3}\ \mathrm{m}$ on the global X_e axis and $2.71\ 10^{-3}\ \mathrm{m}$ on the Y_e axis. For Integration, in Figure 6.12 the final error is $1.80\ 10^{-3}\ \mathrm{m}$ in the X_e axis and $2.78\ 10^{-3}\ \mathrm{m}$ in the Y_e axis.

6.4.1.3 Long Arc

The trajectory results are presented in Figure 6.15, Figure 6.16 and there is no serious deviation from the desired trajectory. For Differentiation, in Figure 6.15 the final error is $0.87\ 10^{-2}\ \mathrm{m}$ on the global X_e axis and $2.54\ 10^{-2}\ \mathrm{m}$ on the Y_e axis. For Integration, in Figure 6.16 the final error is $1.10\ 10^{-2}\ \mathrm{m}$ on the X_e axis and $2.42\ 10^{-2}\ \mathrm{m}$ on the Y_e axis. Both algorithms perform well when required to produce inputs over a longer, complex drive. The control inputs from Inverse Simulation are in Figure 6.17, Figure 6.18. In Figure 6.17 high frequency and low amplitude oscillations in the control signal from Differentiation can be seen, for example around 25 s, 75 s, 125 s, 150 s and 180 s. That is not the case for the Integration control signal in Figure 6.18.

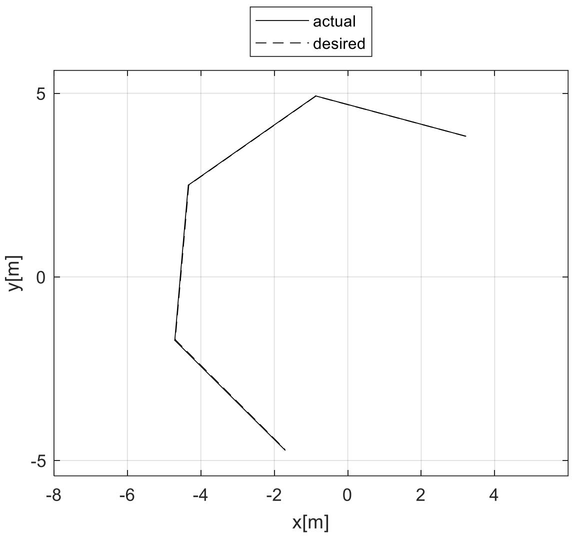


Figure 6.15: Long Arc, Differentiation

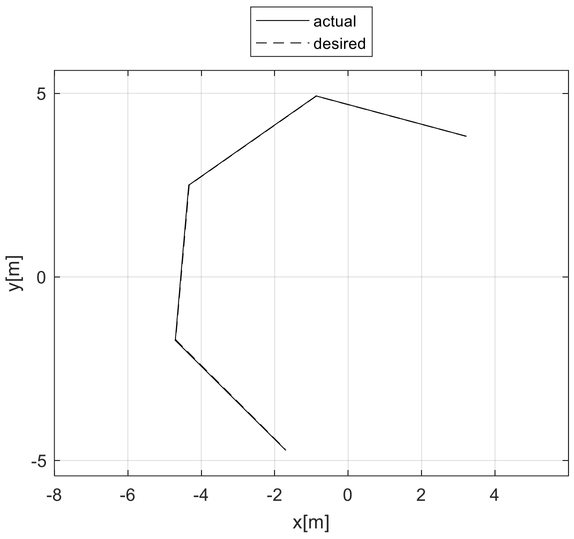


Figure 6.16: Long Arc, Integration

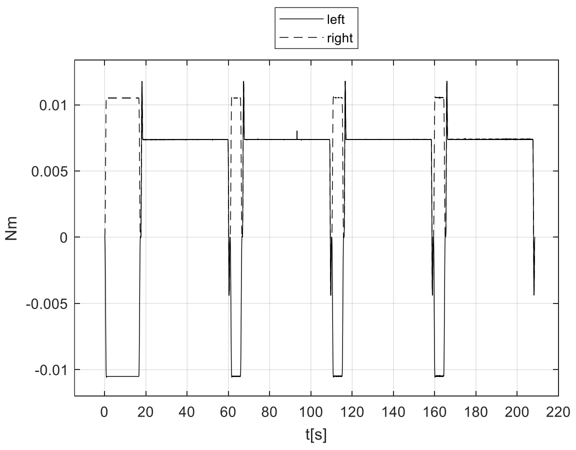


Figure 6.17: Long Arc Control Input, Integration

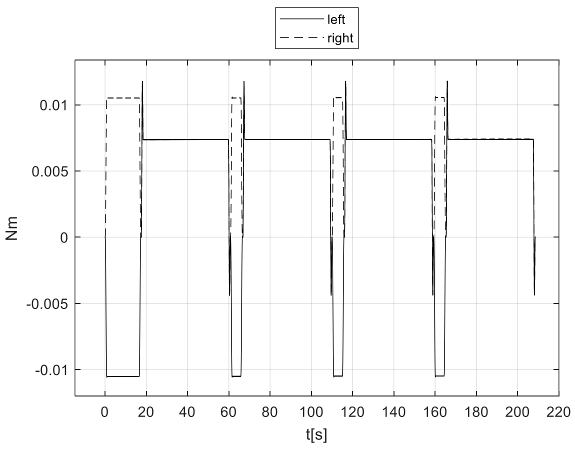


Figure 6.18: Long Arc Control Input, Differentiation

It can be seen, e.g. in Figure 6.17, Figure 6.18, that when the robot must turn, the control signals are symmetrical (i.e., equal magnitude, opposite sign) and when it moves forward the control signal for each side are equal.

A difference between the Differentiation and the Integration is the smoothness of the calculated control signals; while they are equal between the two methods, those produced by Integration are smoother. The low amplitude, high frequency oscillations observed in the control input results from Differentiation becomes more pronounced as the complexity (i.e., drive distance, duration and heading changes) of the trajectory increases, such as the Long Arc trajectory, where a snapshot from Figure 6.17 and Figure 6.18 is shown in Figure 6.19 and Figure 6.20.

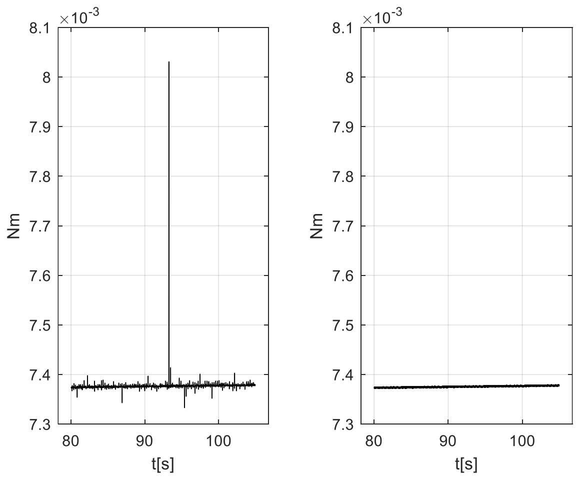


Figure 6.19: Long Arc left side control Differentiation (L) and Integration (R)

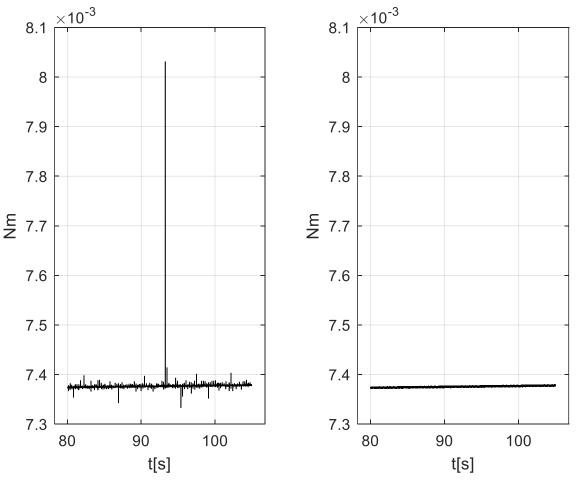


Figure 6.20: Long Arc right side control Differentiation (L) and Integration (R)

This behaviour for the Differentiation Inverse Simulation is because the convergence is based on the derivative and both the states, and the outputs and the Jacobian is not square, whereas the Integration requires only the outputs, and the Jacobian is square. It is in line with previous observations in Section 3.2.

6.4.2 Tabulated Trajectory Results

The performance of the Inverse Simulation algorithms is assessed using the absolute errors[8] between the actual and desired surge velocity u m/s, sway velocity v m/s and yaw velocity r rad/s. The actual velocities u, r, and v are the result of the application of the control input calculated by the Inverse Simulation to the rover in a standard forward simulation. Therefore, these errors measure the performance of the Inverse Simulation in calculating the control input and achieving the desired trajectory profile, i.e., the global error as defined in Section 3.2. This type of error is a true indication of the Inverse Simulation accuracy. The sway velocity v is not used for the Integration method but its error in the forward simulation is included for comparison; in all cases, the desired sway velocity is set to zero. In this assessment, two things are of interest for each type of error: the centre and the dispersion of the error values.

[8] This is the error $|u - u_d|$ and so on, see also Appendix H.

- The mean of the error ($\bar{e}$) and the maximum error ($e_{\max}$). These values provide the central tendency and the scale of the error.

- The dispersion of the data is measured by the standard deviation (σ_e) and is a reliable measure of the scale and dispersion (Heumann *et al.*, 2016).

In addition to these, for each trajectory, the final position in the global coordinate system and heading error are presented. The runtime[9] t_r to calculate the inputs is also shown for a relative comparison between Differentiation and Integration. The results of the Inverse Simulation errors are in Table 6.3.

Table 6.3: Inverse Simulation Results

	Left Right Drive: 3m		**Arc** Drive: 0.73m		**Long Arc** Drive: 16.90m	
Errors	Differentiation	Integration	Differentiation	Integration	Differentiation	Integration
e_{X_e} $[m]$	$1.61\ 10^{-3}$	$1.55\ 10^{-3}$	$1.80\ 10^{-3}$	$1.80\ 10^{-3}$	$8.66\ 10^{-3}$	$1.10\ 10^{-2}$
e_{Y_e} $[m]$	$1.31\ 10^{-3}$	$1.22\ 10^{-3}$	$2.71\ 10^{-3}$	$2.78\ 10^{-3}$	$2.54\ 10^{-2}$	$2.42\ 10^{-2}$
e_ψ $[rad]$	$5.10\ 10^{-5}$	$5.15\ 10^{-6}$	$6.51\ 10^{-5}$	$1.26\cdot10^{-6}$	$1.75\ 10^{-4}$	$2.39\ 10^{-5}$
$\max(e_u)$ $\left[\frac{m}{s}\right]$	$2.92\ 10^{-5}$	0	$3.26\ 10^{-5}$	0	$1.32\ 10^{-4}$	0
$\bar{e}_u$ $\left[\frac{m}{s}\right]$	$4.36\ 10^{-6}$	0	$3.64\ 10^{-6}$	0	$7.46\ 10^{-6}$	0
σ_{e_u}	$4.01\ 10^{-6}$	0	$5.22\ 10^{-6}$	0	$7.15\ 10^{-6}$	0
$\max(e_v)$ $\left[\frac{m}{s}\right]$	$2.07\ 10^{-6}$	0	$1.98\ 10^{-6}$	$1.97\ 10^{-6}$	$9.83\ 10^{-6}$	$9.04\ 10^{-6}$
$\bar{e}_v$ $\left[\frac{m}{s}\right]$	0	0	0	0	$1.06\ 10^{-6}$	$1.07\ 10^{-6}$
σ_{e_v}	0	0	0	0	$1.59\ 10^{-6}$	$1.52\ 10^{-6}$
$\max(e_r)$ $\left[\frac{rad}{s}\right]$	$5.42\ 10^{-5}$	0	$1.81\ 10^{-5}$	0	$1.16\ 10^{-4}$	0
$\bar{e}_r$ $\left[\frac{rad}{s}\right]$	$4.62\ 10^{-6}$	0	$4.42\ 10^{-6}$	0	$5.45\ 10^{-6}$	0
σ_{e_r}	$3.95\ 10^{-6}$	0	$2.68\ 10^{-6}$	0	$6.51\ 10^{-6}$	0
t_r $[s]$	9.50	16.02	6.25	9.21	54.15	66.39

[9] The simulations were conducted on a system with the following specifications: Intel Core 2 Duo @2.50 GHz, 4GB RAM.

The Left-Right trajectory shows the two basic movements of the rover, a forward movement, and a turn on the spot. The Arc trajectory demonstrates its capability to move between closely spaced waypoints. The Long Arc trajectory is the longest in distance and duration and requires the rover's heading to adapt throughout. Any values smaller than 10^{-6} are rounded off to zero, recalling that the Inverse Simulation tolerance is set to $5\ 10^{-7}$. The data provided show that the Inverse Simulation control input when applied to the forward system, results in velocities that converge to the desired ones with very good accuracy.

The maximum error between the desired and actual u, v and r is always in the range of 10^{-5}. The only exception is the Long Arc trajectory, which is the longest drive, and the maximum error is in the range of 10^{-4} for u and r. The mean error between the desired and actual u is always in the range of 10^{-6}, the same for the mean error between the desired and actual r and v. The errors increase as the drive distance and the drive duration increase. Integration has smaller average and maximum errors than those of Differentiation for all cases, even though the sway velocity v is not used for converging to the control input.

The final position errors are always less than $1\ \mathrm{mm}$ for the shorter trajectories and for the Long Arc trajectory where the position error is in the order of $1 - 2\ \mathrm{cm}$. It is also worth considering that the Arc trajectory has a duration of $25.9\ \mathrm{s}$ and a drive distance of $0.73\ \mathrm{m}$, whereas the Long Arc trajectory has a duration of $208.55\ \mathrm{s}$ and a drive distance of $16.90\ \mathrm{m}$. In practice, a rover would not be expected to travel such a distance without any other correction. Overall, both methods have comparable results and the final position and heading errors are small.

The runtime for Integration is larger than the time required for Differentiation, especially for the longer trajectory; the greater accuracy comes at the expense of execution time but also with a simpler algorithm. This also confirms previous observations in Section 3.2 that the rate of convergence of Integration is overall slower than Differentiation, a difference that can be up to an order of magnitude. The Differentiation algorithm converges using a numerical differentiation scheme and the system's state and output equations. The Integration algorithm converges using a numerical integration scheme and the convergence is based on the difference between the desired and actual response. Differentiation looks toward the next time point and anticipates it, whereas Integration looks toward the current and previous time point. Therefore, the key points are a) Integration produces smoother signals b) Differentiation has a shorter runtime c) Integration is more accurate in comparison with Differentiation, but in practice both perform very well for trajectory following

6.5 Summary of Results

Inverse Simulation has been successfully applied for trajectory tracking to a mobile wheeled robot. The method is applied to an experimentally validated model and the calculated control inputs consider the system limitations. A non-square Jacobian that uses a reduced number of states and the remaining states are estimated was used for Differentiation with good results. The Differentiation scheme applied to the rover uses a Jacobian that is not square and estimates some of the states, instead of updating them all, while also taking advantage of the physics of the problem. This is a novel approach since Differentiation was until now applied to systems with a square Jacobian.

The feasibility of Inverse Simulation as a guidance and control method is shown by calculating the nominal control inputs for different trajectories, including a Long Arc test. The results show that Inverse Simulation is an appropriate method for generating control inputs. The accuracy of both algorithms is high and given a desired path the control inputs generated by the Inverse Simulation algorithms can be used to guide the rover along this path, as shown by the small errors between the actual and the desired surge velocity (u), yaw velocity (r), and sway velocity (v), which are in the range of 10^{-5} or less. The issue of runtime required for the control inputs was also examined. Differentiation calculates the control inputs faster than Integration, but they are also less smooth, especially for more complex trajectories. This is due to the equations used within the Differentiation algorithm, which converges using the derivative and the system states and outputs, and the usage of a non-square Jacobian. Compared to Integration, Differentiation is more complex since it depends on the number of states as well as the number of inputs and outputs. This is seen here, where there are six states, with only two of them directly controllable and a Jacobian that is not square. If the desired outputs were changed, this would mean that Eq.(6.6) to (6.13) would have to change. In contrast to this, for Integration, only the error function, Eq.(6.17) and its Jacobian, Eq.(6.18) would change.

Chapter 7 Application of Inverse Simulation to the Rover II: Non-Linear Model with Motor Dynamics

So far, the control inputs are the torque for the left and right sides. The Lynxmotion 4WD3 rover DC motors are wired in parallel, and each side receives the same voltage input. Therefore, when including the motor dynamics, the system control inputs are now a voltage V_1 to the left side and a voltage V_2 to the right side:

$$\mathbf{V} = \begin{bmatrix} V_1 & V_2 \end{bmatrix}^T \tag{7.1}$$

Since each side receives the same voltage input, the equations are written for the left (i=1) and right side (i=2) and were presented in Section 5.1.4. Each motor generates a torque (τ_i) that is proportional to the current (I_i), which depends on the voltage (V_i) and the motor efficiency (η_i). The values for the motor parameters are in Appendix A.

$$\frac{dI_i}{dt} = \frac{-\mathrm{R_a} I_i - \mathrm{K_e}\omega_i + V_i}{\mathrm{L_a}},\ i = [1,2] \tag{7.2}$$

$$\frac{d\omega_i}{dt} = \frac{\mathrm{K_t} I_i - \mathrm{b}\omega_i - \xi\omega_i}{\mathrm{J_m}},\ i = [1,2] \tag{7.3}$$

$$\tau_i = \eta_\mathrm{i} \mathrm{K_t} I_i,\ i = [1,2] \tag{7.4}$$

The force per wheel is calculated by Eq.(7.5), where $\mathrm{r_w}$ is the wheel radius. Because each side receives the same voltage input, the front left and back left wheels produce the same force ($F_{fl} = F_{bl}$) and so do the front right and back right wheel ($F_{fr} = F_{br}$). Thus, the motor dynamics are connected to the system dynamics via the wheel force.

$$F_i = \frac{\tau_i}{\mathrm{r_w}} \tag{7.5}$$

There are six rover states, two desired outputs (u_d, r_d) and two system outputs to control (u, r). The two control inputs to identify are now the input voltages, one for each side: $\mathbf{V} = [V_1 \quad V_2]$. To the six states for the rover, four additional motor states are added: (I_1, ω_1) for the left side and (I_2, ω_2) for the right side. The motors on each side are actuated with the same voltage and have the same efficiency and thus the same current and motor speed.

Each time the system dynamics need to be calculated, the estimated control input at each point in time is first used to find ($I_1, \omega_1, I_2, \omega_2$), then the torque from Eq.(7.4) and then the wheel force, Eq.(7.5). From this point onwards, the rover dynamics and kinematics are calculated as in Section 5.1.3. When the motor dynamics are also considered in the model, a few modifications to the Inverse Simulation algorithm are needed.

7.1 Differentiation

At t_i given the desired output $\mathbf{g}_d(t_i)$ from Eq.(6.4), the functions $\mathbf{F}_1$ and $\mathbf{F}_2$ are defined to find the value of the input $\mathbf{V} = [V_1 \quad V_2]$ and the rover states $\mathbf{q}$ respectively. The size of $\mathbf{F}_1$ is 6×1, Eq.(6.6) and of $\mathbf{F}_2$ is 3×1, Eq.(6.11). Their sizes remain the same since they refer to the rover dynamics, which have not changed. There are still the same three desired outputs (u, v, r) and the two voltage inputs to find. The Jacobian, Eq.(6.12) also retains the same form, but instead of torques, $\mathbf{F}_1$ and $\mathbf{F}_2$ are perturbed in terms of the voltage inputs:

$$\mathbf{J} = \begin{bmatrix} \frac{\delta F_1}{\delta u} & \frac{\delta F_1}{\delta v} & \frac{\delta F_1}{\delta r} & \frac{\delta F_1}{\delta V_1} & \frac{\delta F_1}{\delta V_2} \\ \frac{\delta F_2}{\delta u} & \frac{\delta F_2}{\delta v} & \frac{\delta F_2}{\delta r} & \frac{\delta F_2}{\delta V_1} & \frac{\delta F_2}{\delta V_2} \\ \frac{\delta F_6}{\delta u} & \frac{\delta F_6}{\delta v} & \frac{\delta F_6}{\delta r} & \frac{\delta F_6}{\delta V_1} & \frac{\delta F_6}{\delta V_2} \\ \frac{\delta g_1}{\delta u} & \frac{\delta g_1}{\delta v} & \frac{\delta g_1}{\delta r} & \frac{\delta g_1}{\delta V_1} & \frac{\delta g_1}{\delta V_2} \\ \frac{\delta g_2}{\delta u} & \frac{\delta g_2}{\delta v} & \frac{\delta g_2}{\delta r} & \frac{\delta g_2}{\delta V_1} & \frac{\delta g_2}{\delta V_2} \\ \frac{\delta g_3}{\delta u} & \frac{\delta g_3}{\delta v} & \frac{\delta g_3}{\delta r} & \frac{\delta g_3}{\delta V_1} & \frac{\delta g_3}{\delta V_2} \end{bmatrix} \tag{7.6}$$

The Jacobian is calculated using finite differences and a first-order, forward difference formula is used. Then, Eq.(6.13) (with $V_{1,n-1}$ instead of $\hat{\tau}_{1,n-1}$ etc.) is solved using the factorize command from the MATLAB package presented at (Davis, 2013) and the reduced state vector and the full control input vector are found:

$$\begin{gathered}
\mathbf{Q}_n\big|_{t_i} = \begin{bmatrix} \mathbf{q}_r \\ \hat{\boldsymbol{\tau}} \end{bmatrix}\Bigg|_{t_i} \\
\mathbf{q}_r = \begin{bmatrix} u_n \\ v_n \\ r_n \end{bmatrix},\ \hat{\boldsymbol{\tau}} = \begin{bmatrix} V_{1,n} \\ V_{2,n} \end{bmatrix}
\end{gathered} \tag{7.7}$$

The full state vector for the rover with the updated values u, v, r, V_1, V_2 is now as in Eq.(6.15), repeated below for clarity:

$$\mathbf{q}_n(t_i) = \begin{bmatrix} u_n & v_n & w_n & p_n & q_n & r_n \end{bmatrix}(t_i) \tag{7.8}$$

Then, $\mathbf{F}_1$ and $\mathbf{F}_2$ from Eq.(6.6), Eq.(6.11) are calculated using $\mathbf{q}_n(t_i)$. If both $\mathbf{F}_1$ and $\mathbf{F}_2$ are close to zero, within a certain tolerance, the algorithm converges. At this point, the new dynamics come into play to calculate and integrate the vector $\dot{\mathbf{q}}(t_{i+1})$ using the updated control vector and rover states $\mathbf{q}_n(t_i)$. If convergence is not achieved, the vector state for the next iteration $n+1$ and the input is from Eq.(6.14).

Overall, the Differentiation algorithm remains the same as before. This is because the addition of the motor dynamics may add some complexity, but the number of inputs and the system states are the same. The additional motor dynamics equations are connected to the system dynamics via the wheel force only. Therefore, when the rover dynamics come into play the forces and moments are calculated from the wheel forces and the rover model remains the same.

7.2 Integration

From the state equation Eq.(6.2) and the output equation Eq.(6.3) there are two desired outputs (u_d, r_d) and two system outputs (u, r) to control over the time interval $\mathbf{T}$. There are two changes: (a) the calculated input vector corresponds to input voltages instead of torques and (b) the motor dynamics Eq.(7.2) to Eq.(7.5) are added to the model. In terms of programming the algorithm, virtually no changes are needed, since the number of inputs and outputs is the same as before and the model dynamics are called as an external function (see also the general algorithm setup in Section 4.1.1.2). The control input is $\hat{\mathbf{V}} = \left[V_1 \quad V_2 \right]$ instead of $\hat{\boldsymbol{\tau}}$, to indicate that this is now a voltage input. The error function is:

$$\mathbf{f_e} = \mathbf{y}(t_i) - \mathbf{g_d}(t_i) = \mathbf{g}\left(\mathbf{q}(t_i), \hat{\mathbf{V}}(t_{i-1})\right) = \begin{bmatrix} u(t_i) - u_d(t_i) \\ r(t_i) - r_d(t_i) \end{bmatrix} \tag{7.9}$$

The actual output $\mathbf{y}$ is from Eq.(6.3) and the desired output $\mathbf{g}_d$ is from Eq.(6.4). The Jacobian $\mathbf{J}_e$ of the error function $\mathbf{f}_e$ in Eq.(6.17) is square and has a size of 2×2 since there are two inputs and two outputs.

$$\mathbf{J_e} = \frac{\delta \mathbf{y}}{\delta \hat{\mathbf{V}}} = \begin{bmatrix} \dfrac{\delta y_1}{\delta \hat{V}_1} & \dfrac{\delta y_1}{\delta \hat{V}_2} \\ \dfrac{\delta y_2}{\delta \hat{V}_1} & \dfrac{\delta y_2}{\delta \hat{V}_2} \end{bmatrix} \tag{7.10}$$

Eq.(7.10) is calculated using central finite differences and using the perturbation matrix as outlined in Section 6.2 with voltages instead of torques for the control inputs. The algorithm continues as in Section 6.2. Finally, the new control inputs are calculated using Eq.(6.22), where $\hat{\boldsymbol{\tau}}_n(t_{i-1})$ represents the control input voltages and n is the current Newton-Raphson inner iteration. Using these new input estimates, the outputs are updated and checked for convergence. If the error from Eq.(6.23) is within the acceptable range, then the calculated control input vector $\hat{\mathbf{V}}_n$ is the required for this time instance, $\mathbf{V}_n(t_{i-1}) = \hat{\mathbf{V}}_n(t_{i-1})$.

$$\begin{aligned} \hat{\mathbf{V}}_n(t_{i-1}) &= \hat{\mathbf{V}}_{n-1}(t_{i-1}) - \mathbf{J}_e^{-1}\left(\mathbf{x}_{n-1}(t_i), \hat{\mathbf{V}}_{n-1}(t_{i-1})\right) \cdot \mathbf{f}_e\left(\mathbf{x}_{n-1}(t_i), \hat{\mathbf{V}}_{n-1}(t_{i-1})\right) \\ \mathbf{f}_{\mathbf{e},n} &= \left. \begin{bmatrix} u\left(\hat{\mathbf{V}}_n\right) - u_d \\ r\left(\hat{\mathbf{V}}_n\right) - r_d \end{bmatrix} \right|_{t_{i-1}} \end{aligned} \tag{7.11}$$

The MATLAB backslash operator (\) was selected for solving Eq.(7.11).

7.3 Results

The results of the Inverse Simulation errors for Differentiation and Integration for the Arc and Long Arc trajectory are shown in Table 7.1. Compared with the results in Table 6.3, the execution time has increased and so have the errors. The Arc trajectory is a test for the rover moving between closely spaced waypoints and the Long Arc trajectory involves traversing several meters with pose alterations and examines the error built over time over a demanding path.

Table 7.1: Inverse Simulation Results with Motor Dynamics

	Arc Drive: 0.73m		**Long Arc** Drive: 16.90m	
Errors	Differentiation	Integration	Differentiation	Integration
e_{X_e} $[m]$	$4.65\ 10^{-3}$	$1.81\ 10^{-3}$	$1.80\ 10^{-2}$	$1.09\ 10^{-2}$
e_{Y_e} $[m]$	$3.84\ 10^{-4}$	$2.78\ 10^{-3}$	$3.55\ 10^{-2}$	$2.42\ 10^{-2}$
e_{ψ} $[rad]$	$1.29\ 10^{-3}$	0	$1.69\ 10^{-3}$	$4.58\ 10^{-6}$
$\max(e_u)\ \left[\frac{m}{s}\right]$	$1.16\ 10^{-2}$	0	$1.15\ 10^{-2}$	0
$\bar{e}_u\ \left[\frac{m}{s}\right]$	$4.85\ 10^{-4}$	0	$5.97\ 10^{-5}$	0
σ_{e_u}	$9.36\ 10^{-4}$	0	$3.36\ 10^{-4}$	0
$\max(e_v)\ \left[\frac{m}{s}\right]$	$6.20\ 10^{-4}$	$1.97\ 10^{-6}$	$6.20\ 10^{-4}$	$9.04\ 10^{-6}$
$\bar{e}_v\ \left[\frac{m}{s}\right]$	$5.16\ 10^{-6}$	0	$1.62\ 10^{-6}$	$1.07\ 10^{-6}$
σ_{e_v}	$2.07\ 10^{-5}$	0	$7.58\ 10^{-6}$	$1.52\ 10^{-6}$
$\max(e_r)\ \left[\frac{rad}{s}\right]$	$9.74\ 10^{-3}$	0	$9.74\ 10^{-3}$	0
$\bar{e}_r\ \left[\frac{rad}{s}\right]$	$5.96\ 10^{-4}$	0	$7.77\ 10^{-5}$	0
σ_{e_r}	$1.12\ 10^{-3}$	0	$4.41\ 10^{-4}$	0
$t_r\ [s]$	8.40	11.78	66.10	70.63

From the errors in Table 7.1 and compared with the results in Table 6.3 for the case without motor dynamics, the inclusion of the motor dynamics increases the errors. This is because more calculations are needed, and any errors are propagated.

For the voltage control input, Figure 7.1 and Figure 7.2, the voltage is far less smooth compared with the torque control inputs calculated without the inclusion of motor dynamics, Figure 6.13, Figure 6.14. This is because the current is obtained by integrating Eq.(7.2), the voltage from Eq.(7.3) and then the torque and the wheel force in Eq.(7.5) are calculated.

The Differentiation method has low amplitude oscillations superimposed on the control input; Figure 7.3, Figure 7.4 show a snapshot to show the oscillations in the control input. This is due to the less stable nature of Differentiation. The Integration method provides a far smoother control input.

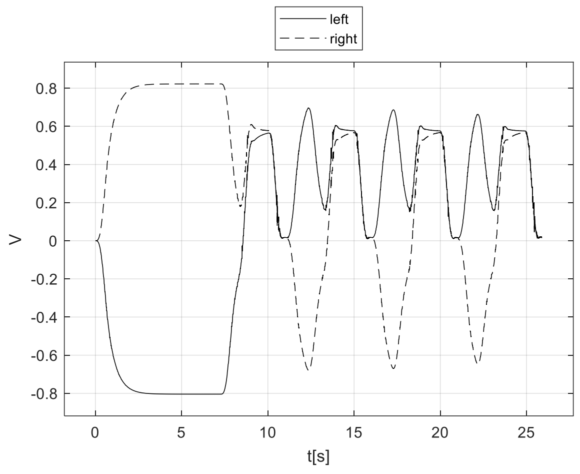


Figure 7.1: Voltage Control Input for Arc (Differentiation)

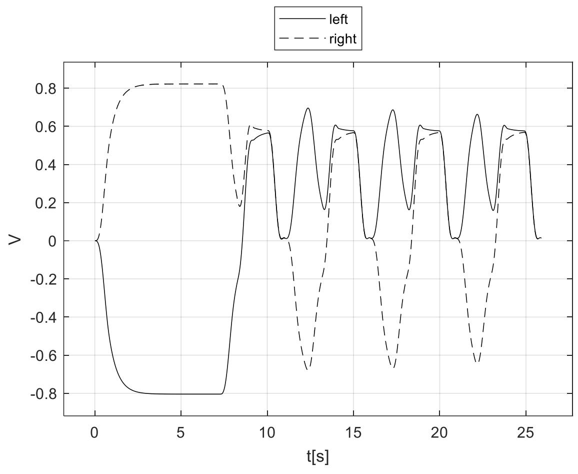


Figure 7.2: Voltage Control Input for Arc (Integration)

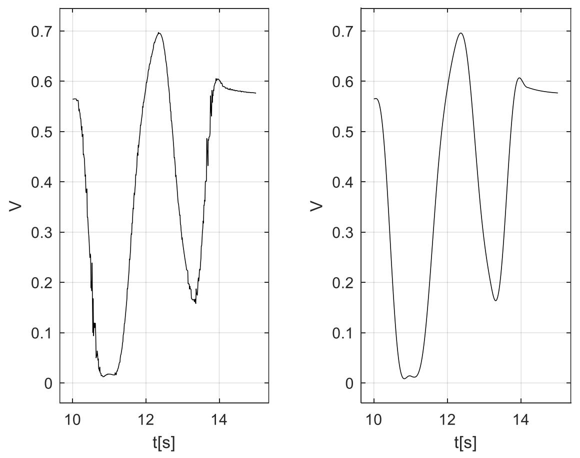


Figure 7.3: Arc left side control Volt Differentiation (L) and Integration (R)

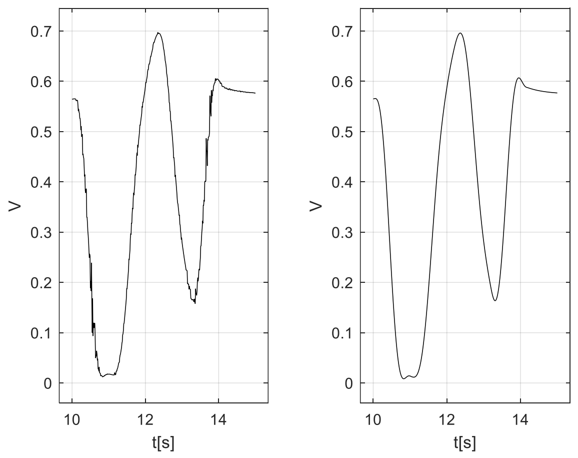


Figure 7.4: Arc right side control Volt Differentiation (L) and Integration (R)

Figure 7.5 and Figure 7.6 show the torque equivalent for the calculated voltages. Figure 7.7 and Figure 7.8 show a snapshot to indicate the oscillations in the control input.

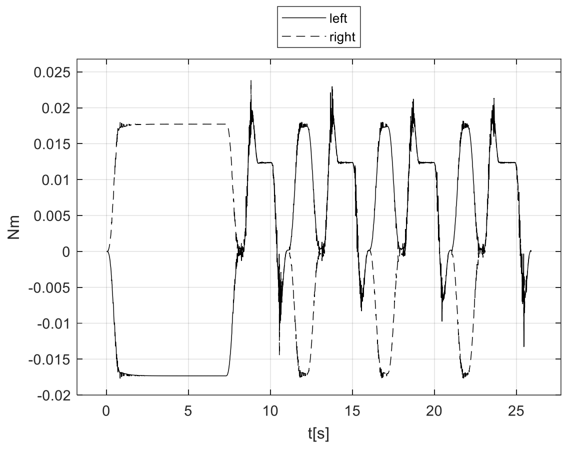


Figure 7.5: Torque control input for Arc (Differentiation)

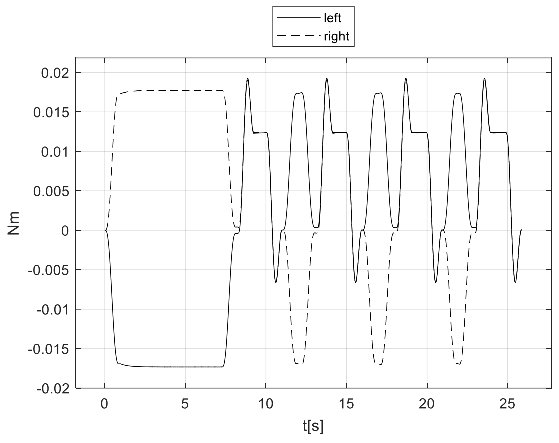


Figure 7.6: Torque control input for Arc (Integration)

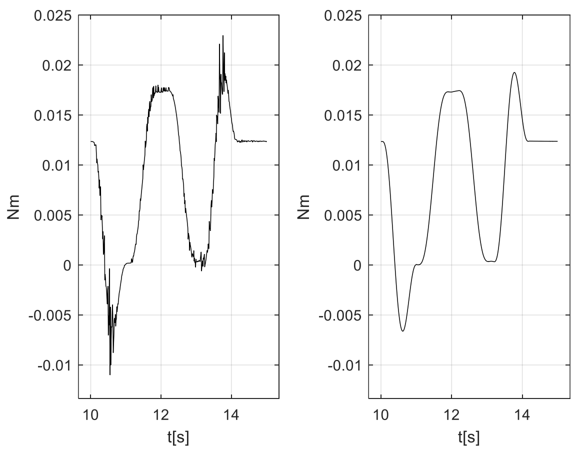


Figure 7.7: Arc left side control Torque Differentiation (L) and Integration (R)

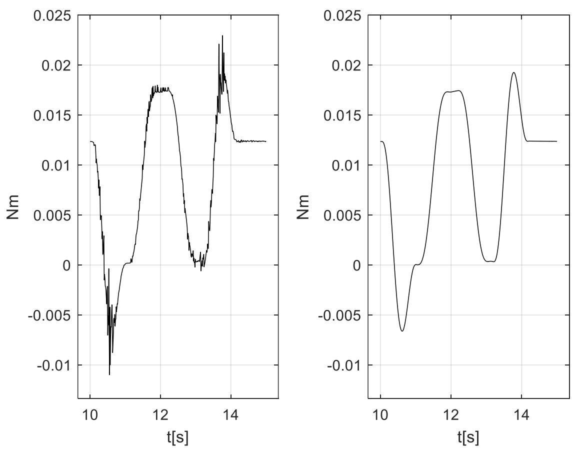


Figure 7.8: Arc left side control Torque Differentiation (L) and Integration (R)

The oscillations are more pronounced here for the Differentiation, this is because the torque is calculated from the current and volt and more steps are required, thus any errors add up. The required torque has increased; the maximum torque is now $0.019\ \mathrm{Nm}$, whereas before it was $0.012\ \mathrm{Nm}$. This is due to the fact the motor dynamics include the motor efficiency η, losses from the electronic and mechanical components and the base friction coefficient ξ at each side. The voltage and torques calculated using the motor dynamics are more realistic because they include the motor losses. Overall, the number of states, inputs and outputs remained the same for both methods and so no modifications were needed to the Inverse Simulation algorithms themselves, only to the file that contains the system dynamics.

Chapter 8 Inverse Simulation Algorithm Tuning for Improved Performance

In the previous chapters, the Differentiation and Integration algorithms for Inverse Simulation were presented and then applied to the rover. Both algorithms require the numerical evaluation of a Jacobian to solve a linear system at each iteration at each point in time and depend on a set of numerical parameters, the most important being the time step dt and the convergence tolerance tol. In this chapter, a closer look is taken at the calculation of the Jacobian for the specific application to the rover model and the numerical parameters that were selected. The relevant mathematical background is in Appendix G and H and this Chapter expands on the general tuning recommendations from Section 4.5. For each case, the average, and the standard deviation of the absolute error for the surge and yaw velocity are shown, as these are the two desired outputs for Inverse Simulation.

8.1 Calculation of the Jacobian: Application to the Rover

The Jacobian inverse was calculated using the factorize command from (Davis, 2013) and the backslash operator (\) for the Differentiation and Integration Algorithm respectively. This section examines more in depth the reason for these choices.

For the Differentiation method from Eq.(6.8) the system is solved for $\delta\mathbf{u}$:

$$\underbrace{\begin{bmatrix} \dfrac{\delta \mathbf{F}_1}{\delta \mathbf{q}}(t_i) & \dfrac{\delta \mathbf{F}_1}{\delta \hat{\boldsymbol{\tau}}}(t_i) \\ \dfrac{\delta \mathbf{F}_2}{\delta \mathbf{q}}(t_i) & \dfrac{\delta \mathbf{F}_2}{\delta \hat{\boldsymbol{\tau}}}(t_i) \end{bmatrix}}_{\substack{\mathbf{J} \\ \downarrow \\ \mathbf{A}}} \underbrace{\begin{bmatrix} \mathbf{q}_n(t_i) - \mathbf{q}_{n-1}(t_i) \\ \hat{\boldsymbol{\tau}}_n(t_i) - \hat{\boldsymbol{\tau}}_{n-1}(t_i) \end{bmatrix}}_{\substack{\delta\mathbf{u} \\ \downarrow \\ \mathbf{x}}} = -\underbrace{\begin{bmatrix} \mathbf{F}_1\left(\mathbf{q}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_i)\right) \\ \mathbf{F}_2\left(\mathbf{q}_{n-1}(t_i), \hat{\boldsymbol{\tau}}_{n-1}(t_i)\right) \end{bmatrix}}_{\substack{\mathbf{F} \\ \downarrow \\ \mathbf{b}}} \tag{8.1}$$

The Jacobian in Eq.(8.1) considers only the controllable states u, r and also v is taken into account and so its size is 6x5. There are more outputs (u, v, r) than inputs (τ_1, τ_2) and so from the control perspective, this is at first glance an under-actuated system. From the linear algebra perspective, this is an overdetermined system since there are more equations than unknowns. In this case, the third output is coupled to the other two and so the actual independent variables are four, which also means that the rank of $\mathbf{J}$ is four and is column rank deficient. Therefore, the third output is associated with the inputs due to the physics of the problem and is very much constrained by this. So, if the system is consistent, of the five unknowns $(u, v, r, \hat{\tau}_1, \hat{\tau}_2)$ the two inputs $(\hat{\tau}_1, \hat{\tau}_2)$ and two of the three outputs (u, r) can be found and the remaining output is coupled to the other two.

For the Integration method, the case is somewhat simpler. The system to solve at each iteration n, at each point in time t_i, from Eq.(4.11) is square and if it is consistent both inputs can be uniquely identified. Otherwise, a least square solution must be found.

$$\mathbf{J_e} \cdot \delta\hat{\boldsymbol{\tau}}_n = -\mathbf{f}_e \tag{8.2}$$

Since both Inverse Simulation methods are developed in MATLAB, the focus is on selecting one of the several methods available at the linear algebra library. The methods that are available in the MATLAB linear algebra library and are suitable for square and non-square systems are detailed in Appendix G. Table 8.1 summarizes the available methods; more details on their implementation are in Appendix G.

Table 8.1: MATLAB Factorisation methods

Command	Comments
inv()	Built-in function. Suitable only for square systems of full rank, is inaccurate and slow.
pinv() Built-in function	Built-in function. Suitable for non-square systems. Calculates the Moore-Penrose inverse using SVD (singular value decomposition)
backslash (\) operator Built-in function	Built-in function. Suitable for square or overdetermined systems with full column rank. Fast, accurate but cannot reuse factorisation.
factorize	Additional MATLAB package, freely available. Suitable for square, over/under determined and rank deficient systems. Selected the best factorisation method between LU, QR, SVD, COD, Cholesky.

Overall, selecting the best method at each iteration n at every time point t_i. of the Inverse Simulation algorithm is not a straightforward task. Each time the individual requirements for each factorisation method must be checked and the best method selected. Additionally, always a least-squares solution is the minimum requirement for even when the system is not consistent, for example, due to numerical errors, unsuitable initial guess, or unsuitable dt. To further illustrate these choices, tests were performed for the Differentiation and the Integration method using the parameters in Table 6.2 for the Long Arc trajectory, Figure 6.6. The performance was assessed using the absolute errors and standard deviation between the actual and desired surge velocity u m/s and yaw velocity r rad/s when the Inverse Simulation input was applied to the system. The execution time s required to calculate the inputs is also shown for a relative comparison between the different methods. Any values smaller than 10^{-6} are rounded off to zero, recalling that the Inverse Simulation tolerance is $5\ 10^{-7}$.

Table 8.2 shows the results of the Differentiation scheme. The backslash method fails as expected: the system is not square, and **J** is (column) rank deficient (the rank is four and there are five columns). Between pinv(**J**) and factorize(**J**), the factorize command is superior in terms of errors and is the one selected, at the expense of increased execution time. The method used by factorize(**J**) is the complete orthogonal decomposition, which is the best and most efficient for rank deficient systems – hence the reduced errors in u and r when applying the calculated control input u from Eq.(8.1) in a standard, forward simulation.

Table 8.2: Factorisation for Differentiation (Long Arc)

	backslash (\)	pinv(J)	factorize(J)
$\bar{e}_u\left[\frac{m}{s}\right]$	-	$1.58\ 10^{-5}$	$7.46\ 10^{-6}$
σ_{e_u}	-	$2.11\ 10^{-5}$	$7.15\ 10^{-6}$
$\bar{e}_r\left[\frac{rad}{s}\right]$	-	$9.23\ 10^{-5}$	$5.45\ 10^{-6}$
σ_{e_r}	-	$2.14\ 10^{-5}$	$6.51\ 10^{-6}$
$t_r\,[s]$	-	44.52	54.15

Table 8.3 shows the results of the Integration scheme. It is worth noting that the results are identical for all three methods, regardless of rounding to zero for any value less than 10^{-6}. The only difference is the execution time. This is expected

since this is a full rank, square system so there is only one solution. Furthermore, there is no need to check the rank and select the best factorisation to ensure a least square solution, as factorize does. The backslash method is selected due to its inherent superiority for square systems, which makes the method more robust. This benefit does come at the expense of increased execution time, compared with the inv command.

Table 8.3: Factorisation for Integration (Long Arc)

	backslash (\)	inv(J)	factorize(J)
$\overline{error_u}\left[\frac{m}{s}\right]$	0	0	0
$\sigma_{error(u)}$	0	0	0
$\overline{error_r}\left[\frac{rad}{s}\right]$	0	0	0
$\sigma_{error(r)}$	0	0	0
$t_{exec}[s]$	66.39	59.68	96.91

8.2 Influence of parameters on results: Application to the Rover

So far, all the results using the rover model are based on an initial set of parameters that were selected to ensure accuracy and decreased execution time; they are in Table 6.2. Additionally, when selecting the time step dt, the time constant of the motors was considered. Furthermore, to improve the Differentiation algorithm results, in terms of accuracy the sway velocity was used as an additional output.

In this section, the inclusion of the sway velocity and the influence of the numerical parameters dt and convergence tolerance (tol) are investigated. These parameters are by necessity specific to the type of system and model used.

8.2.1 Effect of time step dt

The variation of the time step dt is motivated by previous observations discussed in Section 4.2.5 that (a) reducing the time step positively affects the accuracy of the results, and (b) too small a dt can have a negative effect by exciting possible uncontrolled states and increasing the high frequency, low amplitude oscillations superimposed on the calculated control input, particularly for the Differentiation method. Note that these oscillations are also dependent on redundancy issues and using a non-square Jacobian. The initial time step dt was set to 0.01. In practice, when applied to the rover the time step becomes 0.05 s because this is more representative of the control signals the rover controller can generate. Three different cases for the dt were tested: 0.001, 0.05 and the default 0.01 s. Table 8.4 presents the results of the Inverse Simulation Differentiation Method.

Table 8.4: Differentiation results for different dt (Arc trajectory)

Errors	dt = 0.001 s	dt = 0.01 s (default)	dt = 0.05 s
e_{X_e} $[m]$	3.82 10^{-4}	1.80 10^{-3}	9.89 10^{-3}
e_{Y_e} $[m]$	5.11 10^{-4}	2.71 10^{-3}	2.30 10^{-2}
$\bar{e}_u$ $\left[\frac{m}{s}\right]$	7.14 10^{-6}	3.64 10^{-6}	1.00 10^{-4}
σ_{e_u}	5.97 10^{-6}	5.22 10^{-6}	2.33 10^{-4}
$\overline{e_r}$ $\left[\frac{rad}{s}\right]$	6.07 10^{-6}	4.42 10^{-6}	1.07 10^{-5}
σ_{e_r}	3.91 10^{-6}	2.68 10^{-6}	1.40 10^{-5}
t_r $[s]$	63.34	6.25	1.95

For the Inverse Simulation Differentiation method reducing the time step increases the accuracy in achieving the desired outputs up to a point. In Table 8.4 between 0.01 s and 0.001 s the results are very similar and of the same order for the average error and standard deviation of the surge and yaw velocity; this indicates that the dt should not be further educed to potentially increase the accuracy. Additionally, the execution time is ten times longer for 0.001 s. As the dt increases above 0.01 s, the errors increase but are still within reasonable bounds and the execution time decreases.

Table 8.5: Integration results for different dt (Arc trajectory)

Errors	dt = 0.001 s	dt = 0.01 s (default)	dt = 0.05 s
$e_{X_e}\ [m]$	3.15 10^{-5}	1.80 10^{-3}	1.13 10^{-2}
$e_{Y_e}\ [m]$	3.58 10^{-4}	2.78 10^{-3}	3.00 10^{-2}
$\overline{e}_u\ \left[\frac{m}{s}\right]$	0	0	0
σ_{e_u}	0	0	0
$\overline{e}_r\ \left[\frac{rad}{s}\right]$	0	0	0
σ_{e_r}	0	0	0
$t_r\ [s]$	92.88	9.21	3.16

For the Inverse Simulation Integration method reducing the time step increases the accuracy. This can be seen in Table 8.5: as the dt increases, the final errors in the position (X_E, Y_E) increase and the execution time decreases. Consider also that for the Integration method, the calculation of the control signals is based on an integral scheme approximated by a first-order Euler integration which is proportional to the step size (Kreyszig, 2014).

If a value of dt is selected that is too small, the prominence of high frequency low amplitude oscillations superimposed on the calculated control input is observed for the Differentiation method, as was also discussed in Section 3.2. Because of the way numerical differentiation works, these oscillations are affected by the increased round off errors. These may be very small but tend to dominate instead of the truncation errors for very small dt. The high frequency, low amplitude oscillations for the Differentiation method can be seen in Figure 8.1, especially between $0-9\ \mathrm{s}$ and around $20\ \mathrm{s}$ and $25\ \mathrm{s}$. In Figure 8.2, where the dt is increased, these oscillations are no longer present. Also, there is no such issue for the control inputs calculated by the Integration method, Figure 8.3, Figure 8.4.

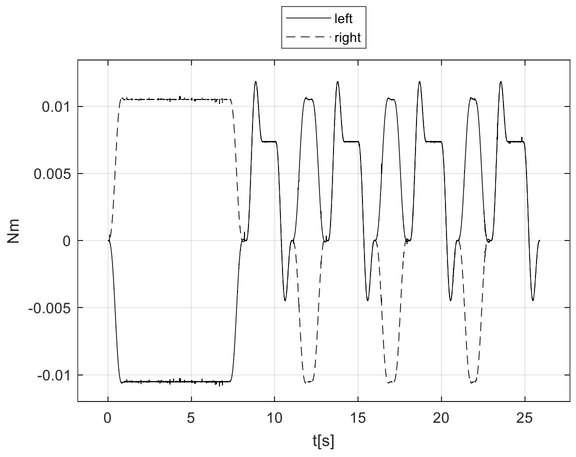


Figure 8.1: Arc Control Input, Differentiation (dt = 0.001)

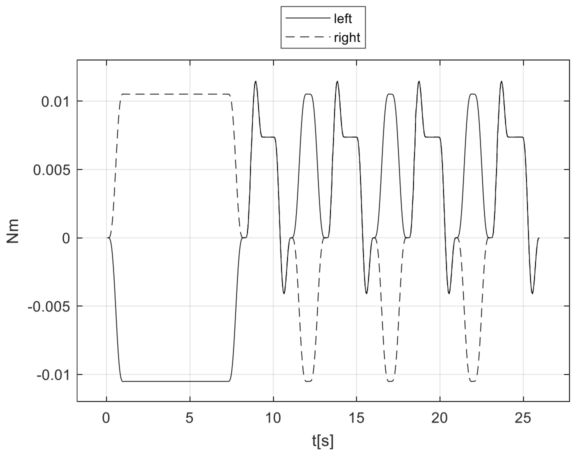


Figure 8.2: Arc Control Input, Differentiation (dt = 0.05)

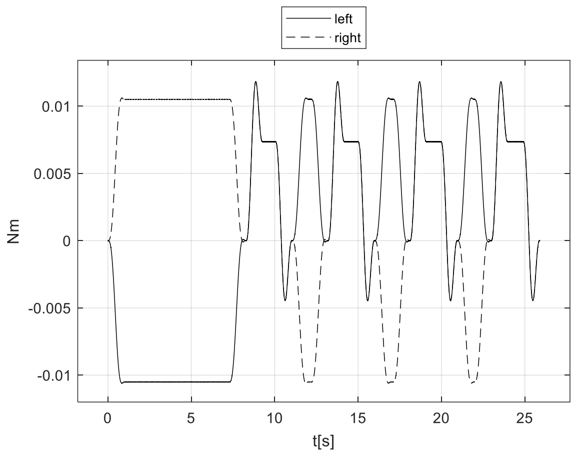


Figure 8.3: Arc Control Input, Integration (dt = 0.001)

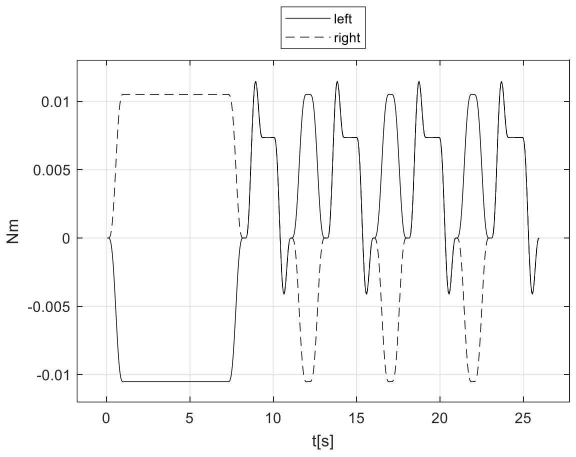


Figure 8.4: Arc Control Input, Integration (dt = 0.05)

8.2.2 Effect of convergence tolerance

In this work, the approximate absolute error E_α is used for the convergence of the Differentiation method and the Integration method, Eq.(6.23). The absolute error provides a good way to frame the result using the tolerance, since the desired quantity is known and, in some cases, the desired can also be zero (for example, the desired sway velocity is always zero).

The convergence tolerance tol for the Newton-Raphson scheme is based on the maximum acceptable error for $\mathbf{Q}_n\big|_{t_i}$ from Eq.(6.14) for the Differentiation method and $\mathbf{f}_{\mathbf{e},n}$ from Eq.(6.23) for the Integration method. The vector $\mathbf{Q}_n\big|_{t_i}$ contains the states (and also desired outputs) u,v,r as well as the control inputs $\hat{\tau}_1$, $\hat{\tau}_2$ to achieve these. The error $\mathbf{f}_{\mathbf{e},n}$ is simply the difference between the current estimate for u and r and their desired values.

The initial tolerance was set to $5\ 10^{-7}$. This is the tolerance for the convergence of the Newton-Raphson scheme. Based on this, the control inputs were calculated. Then, these inputs were passed on to a forward simulation and the resulting trajectory errors are the error between the actual and the desired u, v, r. Therefore, increasing or decreasing the convergence tolerance, affects the convergence to the control inputs and then through the usage of the control inputs in the forward simulation the actual results. The initial tolerance was selected while considering the scale of the desired outputs: the maximum surge velocity $u_{\max} = 0.1\ \frac{m}{s}$ and the maximum yaw velocity $r_{\max} = 10 \frac{\deg}{s}$.

First, the tolerance is set to $5\ 10^{-8}$ and all other parameters are as in Table 6.2 (dt is set to 0.01 s). Table 8.6 shows the results.

Table 8.6: Inverse Simulation Results for tol = $5\ 10^{-8}$

	Left Right		Arc	
Errors	**Differentiation**	**Integration**	**Differentiation**	**Integration**
$\overline{error_u}\left[\frac{m}{s}\right]$	$4.73\ 10^{-6}$	0	$3.64\ 10^{-6}$	0
$\sigma_{error(u)}$	$5.25\ 10^{-6}$	0	$5.22\ 10^{-6}$	0
$\overline{error_r}\left[\frac{rad}{s}\right]$	$4.57\ 10^{-6}$	0	$4.42\ 10^{-6}$	0
$\sigma_{error(r)}$	$3.69\ 10^{-6}$	0	$2.68\ 10^{-6}$	0
$t_r\,[s]$	10.84	12.01	8.48	9.05

Compared to the results for tol $5\ 10^{-7}$ in Table 6.3, there are almost no differences in the errors and Inverse Simulation execution time. This shows that the selected combination of tolerance $5\ 10^{-7}$ and time step 0.01 s are sufficient and there is no need to set a lower tolerance.

Then, the tolerance is increased to $5\ 10^{-5}$ and all other parameters are as in Table 6.2 (dt is set to 0.01 s). Table 8.7 shows the results. The Differentiation method produces control input results for all the test trajectories in Table 6.1, with some slightly increased errors and execution time. This shows that the method converges but a bit slower. Integration produces results only for the Forward and

Left-Right test and fails to converge for the other, such as the Arc trajectory. At first, this may seem surprising; however, it is worth looking at what exactly each method uses to converge to the desired control inputs in the Newton Raphson scheme.

Table 8.7: Inverse Simulation Results for tol = 5 10^{-5}

	Left Right		Arc	
Errors	**Differentiation**	**Integration**	**Differentiation**	**Integration (with sway)**
$\overline{error_u}\left[\frac{m}{s}\right]$	4.35 10^{-6}	1.95 10^{-5}	3.63 10^{-6}	1.92 10^{-5}
$\sigma_{error(u)}$	4.01 10^{-6}	1.13 10^{-6}	5.22 10^{-6}	1.01 10^{-5}
$\overline{error_r}\left[\frac{rad}{s}\right]$	4.61 10^{-6}	2.21 10^{-5}	4.42 10^{-6}	2.78 10^{-5}
$\sigma_{error(r)}$	3.95 10^{-6}	1.31 10^{-6}	2.64 10^{-6}	1.27 10^{-5}
$t_r\,[s]$	14.31	18.72	8.48	11.55

The Integration method converges using the difference between the actual and the desired output. In this case, this is represented by the error function $\mathbf{f}_{\mathbf{e},n}$ from Eq.(6.23) for converging and it contains the desired outputs u and r.

The Differentiation method uses $\mathbf{Q}_n\big|_{t_i}$ from Eq.(6.14) for converging and it contains the states (and also desired outputs) u, v, r as well as the control inputs ($\hat{\tau}_1$, $\hat{\tau}_2$) to achieve these.

In this way, while Integration has a simpler scheme that results in a square system, for a larger tolerance and without adjusting the time step, it fails. The Differentiation method, by virtue of using more system parameters for convergence, can tolerate a little better the loss of fidelity by a larger accepted error tolerance.

One way to remedy this is by using the Integration method with the addition of the sway velocity. This sacrifices somewhat the method's simplicity, but the sway velocity acts as an additional constraint for u and r. In that case, Integration produces results for all test trajectories, with an increased execution time and errors (but still very acceptable), Table 8.7 ($\mathrm{tol} = 5\cdot10^{-5}$) compared with Table 6.3 ($\mathrm{tol} = 5\cdot10^{-7}$) or even Table 8.6 ($\mathrm{tol} = 5\cdot10^{-8}$).

Additionally, the maximum desired yaw velocity is $r_{\max} = 10\ \frac{\deg}{s} = 0.17\ \frac{rad}{s}$. Using a tolerance in the order of 10^{-5} may not be the best choice anyway when the maximum desired value is in the order of 10^{-2}.

8.2.3 Effect of sway velocity

The motivation for adding the sway velocity to the Differentiation method was to reduce the errors. This is illustrated in Table 8.8, which shows the results for the Forward 1m test for the Differentiation method with and without the inclusion of the sway velocity. Table 8.8 also shows the results for the Integration method with and without the inclusion of the sway velocity.

As was discussed in Section 5.1.2.1, the sway velocity is strongly coupled to the surge and yaw velocity via the slip angle, Eq.(5.10), Eq.(5.11), Eq.(5.13). This interaction provides an indirect control of the sway and acts as an additional constraint when solving for the control input. Furthermore, because the assumption was made that the terrain is not significantly uneven or soft and that the robot moves on a planar, smooth, and rigid terrain, the desired sway velocity can be set to zero. In this way, the system's properties are utilised to improve the performance of Inverse Simulation.

Table 8.8: Effect of sway velocity in Inverse Simulation Results

	Forward 1m				Arc	
Errors	Differentiation without sway	Differentiation with sway	Integration without sway	Integration with sway	Integration without sway	Integration with sway
$\overline{error_u}\left[\frac{m}{s}\right]$	1.57 10^{-4}	4.43 10^{-6}	0	0	0	0
$\sigma_{error(u)}$	3.32 10^{-4}	2.98 10^{-6}	0	0	0	0
$\overline{error_r}\left[\frac{rad}{s}\right]$	1.25 10^{-6}	3.39 10^{-6}	0	0	0	0
$\sigma_{error(r)}$	1.66 10^{-6}	3.54 10^{-6}	0	0	0	0
$t_r\,[s]$	5.06	3.42	3.10	4.23	9.21	17.78

For the Differentiation method, the inclusion of the sway velocity is beneficial for the surge velocity errors, there is a difference of magnitude of 10^2 between the two cases and has no serious effect on the yaw velocity. Considering that the algorithm with the sway velocity is faster, i.e., converges faster, the inclusion of sway is beneficial even for this very simple test. It could be argued that an error of 10^{-3} is acceptable, however considering that the error increases with the complexity of the trajectory as seen in Table 6.3, this is not a good choice.

In comparison, including the sway velocity for the Integration method only increases the execution time. The results are identical with or without the sway for all trajectories (with the other simulation parameters as in Table 6.2), regardless of rounding to zero any value less than 10^{-6}. The only difference is the execution time. This is expected, since without the inclusion of the sway, this is a 2x2 square system with a full rank of 2, so there is only one solution. With the inclusion of sway, the system is now 3x2 and is overdetermined, but its rank is still equal to 2. This means that the system again has a unique solution which is of course always the same (Davis, 2013). Furthermore, the system is still solved with the \ (backlash) operator even though the Jacobian has changed.

Table 8.8 also shows the results for the Arc trajectory, which fails for the Differentiation method without the inclusion of the sway velocity. For the Integration method, including the sway velocity again has no effect, beneficial or not, apart from increasing the execution time.

8.3 Summary of Results: Inverse Simulation Tuning: Recommendations for the rover

In this chapter, the parameters that affect the application of the Inverse Simulation are investigated. The calculation and inversion of the Jacobian, the selection of outputs, the time step dt and convergence tolerance tol significantly affect the method's properties and are examined.

First, the inversion of the Jacobian was examined. This is a crucial part of the Inverse Simulation algorithm, and the validity of the results depends to a great extent on its correct execution. Four different methods were examined, the built-in MATLAB functions inv(), pinv(), the backslash (\) operator and the factorize command (Davis, 2013). These methods were tested in MATLAB: the factorize command was selected for the Differentiation method and the backslash (\) operator for the Integration method. The selection was based not only on the numerical results but also on what type of linear system is solved for each method.

For the Differentiation method there are infinite solutions because the system is overdetermined (6x5) and rank deficient (rank is 4). The third output v is associated with the other two outputs u & r due to the physics of the problem and is very much constrained by this. So, of the five unknowns $(u, v, r, \hat{\tau}_1, \hat{\tau}_2)$ the two inputs $(\hat{\tau}_1, \hat{\tau}_2)$ and two of the three outputs (u, r) can be found and the remaining output is coupled with the other two. Using the factorize command ensures that the input and outputs are always a least-squares solution that minimizes the error.

For the Integration method, the case was simpler. The system to solve is square and consistent so both inputs can be uniquely identified. The backslash (\) operator ensures that the correct solution is always identified. Even in the case of modifying the system to add the sway velocity, which results in a non-square overdetermined system (size is 3x2), the solution is still the correct one.

A small dt results in high frequency, low amplitude oscillations in the control input. These oscillations are more evident if the system is overdetermined, as is here for Differentiation. Moreover, if the dt is too small, it introduces additional, rounding errors due to the way Differentiation works and these are not easily corrected. A compromise between a dt that can adequately follow the system as it evolves without inducing oscillations and additional errors is needed. In contrast to this, for the Inverse Simulation Integration method reducing the time step increases the accuracy.

The effect of the convergence tolerance tol was then examined. When the tolerance was set to a lower threshold $\text{tol} = 5\ 10^{-8}$, instead of $\text{tol} = 5\ 10^{-7}$, there were almost no differences in the errors and Inverse Simulation execution time. This shows that the selected combination of tolerance $\text{tol} = 5\ 10^{-7}$ and time step 0.01 s are sufficient. To further investigate the effect of the tolerance, it was then set to a higher threshold at $5\ 10^{-5}$. The result was that the Differentiation method by virtue of using more system parameters for convergence can tolerate a little better the loss of fidelity by a larger accepted error tolerance. Integration on the other hand did not produce results for the more complex trajectory. One way to remedy this is by using the Integration method with the addition of the sway velocity. This sacrifices somewhat the method's simplicity, but the sway velocity acts as an additional constraint for u and r. In that case, Integration produces results for all test trajectories, with an increased execution time and errors (but still very acceptable).

The convergence tolerance also depends on the scale of the desired outputs. The maximum desired yaw velocity is $r_{\max} = 10\ \frac{\deg}{s} = 0.17\ \frac{rad}{s}$ so using a tolerance in the order of 10^{-5} may not be the best choice anyway. The Integration method converges using error between the actual and desired output, so when the maximum yaw velocity is in the order of 10^{-4}, a lower threshold should be chosen in any case. Nonetheless, by adding the sway velocity to the Integration method, it produced good results even when using such a marginal tolerance.

Overall, the combination of $\mathrm{dt} = 0.01\mathrm{s}$ and a $\mathrm{tol} = 5\ 10^{-7}$ produces the best results. For simplicity and overall stability, the Integration scheme is more appropriate. For decreased execution time, Differentiation is preferred, at the expense of slightly larger errors, an overdetermined system that requires special handling and less smooth control signals.

The effect of adding the sway velocity v, which is strongly coupled with u, r, as an additional desired output was also examined. For Differentiation, using v as output is beneficial from the start, even for the simpler trajectories. In contrast to this, for the Integration method, including the sway velocity has no effect, apart from increasing the execution time. Integration with or without the sway produces the same results for all trajectories. This is because the rank of the Jacobian is always 2 and this means that the system has a unique solution. Furthermore, the fact that the sway is not necessary for the Integration method confirms previous results that Integration is more stable overall.

Chapter 9 Conclusions and Future Work

The four main aims of this thesis were to (a) develop Inverse Simulation for a general state-space system and establish a methodology for its application, using two different approaches (Integration, Differentiation) to converge to the control inputs given a desired output, (b) examine the application of Inverse Simulation in terms of the parameters that affect its performance & type of solution it provides, (c) apply Inverse Simulation for output tracking to a four wheeled rover model and (d) examine the parameters that affect the performance of Inverse Simulation when applied to the rover, within the general framework established in the thesis.

In support of these aims, a review of the current state of the art of rover design and control methodologies was conducted, a suitable model and trajectory for the rover were presented, a review of previous Inverse Simulation applications was done and Inverse Simulation application examples and comparison with common controllers were also presented.

Overall, this work provides a general methodology and theoretical background for the Inverse Simulation using two different methods, Differentiation, and Integration. The parameters that affect its performance were investigated using tools from numerical analysis and linear algebra. These parameters were identified as the number of inputs and outputs, what can be considered a desired output, the time step dt, and the convergence tolerance for the numerical scheme. Both methods converge using a Newton-Raphson numerical scheme that employs the Jacobian. Particular attention was given to the formulation of the Jacobian and how can an efficient solution be ensured.

The main benefit of having developed this general methodology is that Inverse Simulation can now be used for a wide range of applications, so long as the system can be formulated in a standard state-space form and at least the system’s inputs and outputs are available. This can be used as a framework and stepping point for any future developments of Inverse Simulation algorithms.

Furthermore, in this work Inverse Simulation was applied for the first time for the guidance and control of a four wheeled, differentially driven rover. The principle is that the time history for the desired trajectory is used for the Inverse Simulation to produce the required control inputs. These control inputs are nominal for the given model and are applied without additional correction to the rover to achieve the desired trajectory. These inputs resulted in accurate position and orientation control of the rover while considering the limitations of the rover model and the actuators used. Using the definition of GNC from Chapter 1, Inverse Simulation is used for guidance by providing the changes in velocity, rotation, and acceleration for following a desired trajectory and for control by using these inputs to execute the desired trajectory in a forward simulation. Inverse Simulation addresses the need for incorporating the dynamic model into the guidance and control system for increased accuracy. In the next section, the main conclusions relevant to each aim per chapter are expanded.

9.1 Conclusions

In **Chapter 2** the relevant state of the art of rover designs is established and a system taxonomy is proposed, based on the following characteristics: mobility type, steering configuration, suspension, and chassis articulation. This system taxonomy is an original result of this thesis and can be used independently to support the design and classification of such systems. The baseline design is four wheels, all-wheel drive, and passive suspension, with a differential for steering. Also in **Chapter 2**, a review of the most common control methods for mobile rover control is presented. The control strategy of a rover is also investigated and there are two main approaches: (a) the rover can go to a given location by executing a pre-defined path without any corrections or (b) the rover can also navigate autonomously to a given location by sensing the environment and making its own decisions.

Chapter 3 is a review of the existing applications of Inverse Simulation. The two main implementations (Differentiation and Integration) and the application considerations from previous experience are presented.

Chapter 4 builds on the review to take a general approach and examine in depth the two main implementations of Inverse Simulation for the standard state-space model. The algorithms are for the general case of an unequal number of inputs and outputs. In **Chapter 4** the **first aim** of this work is achieved: a general methodology and theoretical background for the Inverse Simulation using the two different methods, Differentiation, and Integration, is developed. The **second aim** is also achieved: to examine the application of Inverse Simulation in terms of the parameters that affect its performance and the type of solution it provides. In **Chapter 4** the parameters that affect the Inverse Simulation performance are identified as (a) the time step dt, (b) the convergence tolerance tol, (c) the number of inputs and outputs, (d) the selection of the desired output. The first two are the numerical parameters and the last two depend on the type of system used.

A state-space model is used with m state equations, p output equations and k control inputs. Both methods converge using a Newton-Raphson numerical scheme that employs the Jacobian. The two approaches differ in how they converge to the control input. Differentiation converges using a scheme based on the state derivative and output; its Jacobian has a size of $(m+p)\times(m+k)$. The Integration method converges based on whether the system's output matches the desired; its Jacobian has a size of $(p\times k)$. Integration has the benefit of being decoupled from the system dynamics and is simpler to set up. The usage of both states and outputs means that Differentiation tends to converge faster than Integration, a difference that can be up to an order of magnitude.

Because both methods use an iterative Newton-Raphson scheme to solve the system of algebraic equations, the problem is essentially transformed into a set of linear equations. For the general case when $k \neq p$ the Jacobian needs to be factorised to achieve the best available solution, which is defined as the least square solution. The factorisation of the Jacobian at each iteration depends on its size and rank, which is not trivial to estimate. To solve the problem and to provide a computationally efficient solution that is also numerically stable (as defined in Appendix G) it is recommended to carefully select the factorisation implementation to be used and Appendix G provides a detailed background.

The time step dt should correspond to the physical limitations and response times of the system. Reducing the time step positively affects the accuracy of the results and increases the execution time. When using the Differentiation method, when the dt is reduced too much, there is a point where the truncation error is reduced but the round off error starts to dominate, which is detrimental to the method's accuracy. The Integration method does not have these issues and the error can be reduced by decreasing the step size, which is why it is considered more stable. Additionally, too small a dt can have a negative effect by exciting possible uncontrolled states and increasing the high frequency, low amplitude oscillations superimposed on the calculated control input, particularly for the Differentiation method.

Differentiation and Integration converge using a specified tolerance value, tol. The tolerance value depends on the scale of the actual and the desired solution and frames the result within the interval of $\pm\,\mathrm{tol}$, thus should be chosen depending not only on the acceptable error but also on the scale of the values calculated.

The C^{i} continuity order of the desired output needs to be carefully selected so that it is a realistic representation, and the output is sufficiently smooth, but this is not constrained by issues such as the system's relative degree. In fact, compared with feedback linearisation, Inverse Simulation is a more general method that can be used for MIMO systems that are not control affine and are not square. Inverse Simulation depends on the system model, but there is no analytical inversion and so Inverse Simulation can handle model changes better, especially Integration.

Also in **Chapter 4**, the linear case of Inverse Simulation is developed. The linear case is the simplest case of the Differentiation method and shows that the desired output selection may affect the system stability. This is not the case for Integration. For the linear case, the relationship between output controllability and Inverse Simulation is established, namely that $p = k$. This corresponds to the general case of a square system that has a unique solution.

Application examples of the linear case and Integration are given using a mass spring damper (MSD) and an active quarter car model (QCA). The system response and control effort are compared with that of a PID controller for the MSD. For the QCA model, the system response from applying a PID controller was the desired output for Inverse Simulation. Integration provided consistently good results in terms of accurately following the desired output. A third application example is given where Inverse Simulation is used to determine the road disturbance, using the passive quarter car model. This shows how the Inverse Simulation algorithm can be generally used to determine an input given an output, without that input being necessarily a control input, thus further extending the method's validity.

In **Chapter 5** the main requirements for applying Inverse Simulation to the rover are presented: a model and a suitable output. The kinematic and dynamic model of an experimentally validated rover moving on planar, smooth, and rigid terrain is presented. The rover adheres to the baseline design from Chapter 2 and uses differential steering, thus it can turn on the spot. The trajectory is based on waypoints and is generated using a 6th order polynomial that constitutes a series of forward (surge) movements followed by on-the-spot turns (yaw rotations).

In **Chapter 6** the **fourth aim** of this work is achieved: Inverse Simulation is applied for the first time for the control of a four-wheeled, differentially driven rover. The Differentiation method uses a non-square Jacobian with a reduced number of states; the remaining states are estimated. This is a novel approach that utilises the physics of the problem since Differentiation was until now applied to systems with a square Jacobian. The inputs to determine are reduced to two (left and right-side motor torque). Three outputs are used, surge (u), yaw (r) and sway velocity (v). The sway velocity is set to zero and is included because it improves the performance of the method. The Jacobian is also reduced to a 6×5 by perturbing only the controllable states (u, r, v) and the scheme estimates the remaining states. This is an overdetermined system and from a control perspective, at first glance, under-actuated. The sway however is matched dynamically to the surge and yaw velocity via the slip angle, thus of the three outputs, only two are independent and are matched to the number of control inputs. For the Integration method, things are somewhat simpler: two inputs and two outputs (u, r) that result in a square 2×2 Jacobian.

The feasibility of Inverse Simulation as a guidance and control method that can produce nominal control signals is shown by calculating the nominal control inputs for different trajectories: moving between closely spaced waypoints in the Arc trajectory ($0.73\ \mathrm{m}$ drive distance, $25.9\ \mathrm{s}$) and longer distances with several pose changes, such as the Long Arc ($16.91\ \mathrm{m}$ drive distance, $208.55\ \mathrm{s}$) and Valley test ($13.44\ \mathrm{m}$ drive distance, $203.70\ \mathrm{s}$). The errors between the actual and the desired surge velocity (u), yaw velocity (r), and sway velocity (v), are in the range of 10^{-5}

or less. Integration has smaller average and maximum errors than those of Differentiation, even though the sway velocity v is not used for converging to the control input. Differentiation also calculates the control inputs faster than Integration by about 20%.

In **Chapter 7** the rover model is augmented with the inclusion of the motor dynamics. This is part of the **fourth aim**. The inputs are now two voltages, one for each side, instead of torques. The motor dynamics are connected to the system dynamics via a single equation and the number of states, inputs and outputs remain the same for both methods. Thus, no modifications were needed to the Inverse Simulation algorithms themselves.

In **Chapter 8** the parameters and solution type of Inverse Simulation are further investigated, motivated by the previous analysis and the application to the rover. Thus, the **second** and **fourth aims** are achieved. For the rover application, the following conclusions are reached: When using Integration, there is a unique solution and the most efficient way to find it is the backslash (\) operator. For Differentiation, the factorize command from (Davis, 2013) is used to provide the best, least-squares solution. Both algorithms are formulated with and without the addition of the sway velocity. For Differentiation, using the sway velocity is beneficial, even for the simpler trajectories. For the Integration method, including the sway velocity only increases the execution time without improving the accuracy of the results.

For the Differentiation method reducing the time step increases the accuracy only up to a point. Additionally, the prominence of high frequency, low amplitude oscillations superimposed on the calculated control input is increased for the Differentiation method. These oscillations are affected by the increased round off errors that tend to dominate for very small dt. For Integration, reducing dt increases the accuracy. For both cases, the accuracy gains are offset by increased computation time. The time step selected should be between 0.01 and 0.05 s, which is in line with what the rover actuators can achieve.

When varying the tolerance, while Integration has a simpler scheme that results in a square system, for a larger tolerance and without adjusting the time step at all, it fails. The Differentiation method by virtue of using more system parameters for convergence can tolerate a little better the loss of fidelity by a larger accepted error tolerance.

9.2 Future Work

In this section, the following further pathways of development of Inverse Simulation are proposed.

9.2.1 Combination with another control method

Inverse Simulation calculates the nominal signals given a desired output and a model. A natural question would be, what happens when we are no longer in the nominal case. In this case, the Inverse Simulation nominal control signals can now be used for further development of control algorithms. Inverse Simulation can be applied either in situ (the rover calculates the necessary control inputs given a defined trajectory) or offline. In the latter case, the trajectory and the control inputs are defined elsewhere, they are uploaded to the rover which then executes the trajectory. Two different approaches are proposed.

In the first approach, Inverse Simulation acts on the difference δy between the actual system output $y(t_i)$ and the desired $y_d(t_i)$. This means that the Inverse Simulation algorithm runs online, in real-time and utilizes as an output the difference between the actual and the desired, thus using the feedback action from the system. The resulting control signal is aimed to correct the difference δy, Figure 9.1. This approach has been used by (Avanzini *et al.*, 2013) for rotorcraft control, using a scheme that resembles that of a model predictive scheme. A simplified model was used to perform the Inverse Simulation and then, for the forward simulation from which the actual system output was obtained, the full rotorcraft model was used (Avanzini *et al.*, 2013). The Integration method was used to take advantage of the fact that it required the outputs and inputs and that the full state vector may not be available in a practical application.

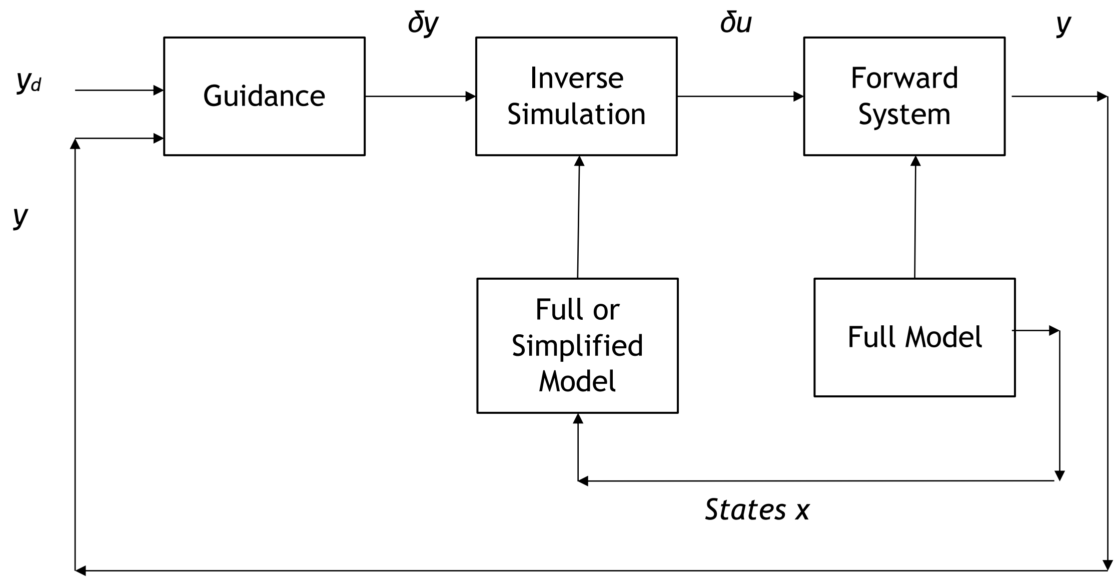


Figure 9.1: Inverse Simulation Online Scheme

The second approach is that the Inverse Simulation inputs are used to drive the system without any additional correction unless a predefined error exceeds a threshold. In that case, an additional controller could be used. The Inverse Simulation nominal inputs can be calculated online or offline to preserve computational resources and uploaded to the system.

This scheme is based on the idea of expected perception in (Cauli *et al.*, 2016). Expected perception control systems utilise the system's internal model and its interaction with the environment. The control system monitors the error between the predicted and the actual data. If the error is small, the system may skip any corrective action, thus saving computational and energy resources. If the error is large, the system will implement a corrective action through feedback. The internal model can be (Cauli *et al.*, 2016) a forward model (predict future data from current), an environmental model (predict dynamics of external objects) or an inverse model (finds the actions to obtain a desired response). Expected perception control systems have been used in robotic arms, grasping tasks, or to locate unexpected objects and they use either forward or environmental models (Cauli *et al.*, 2016).

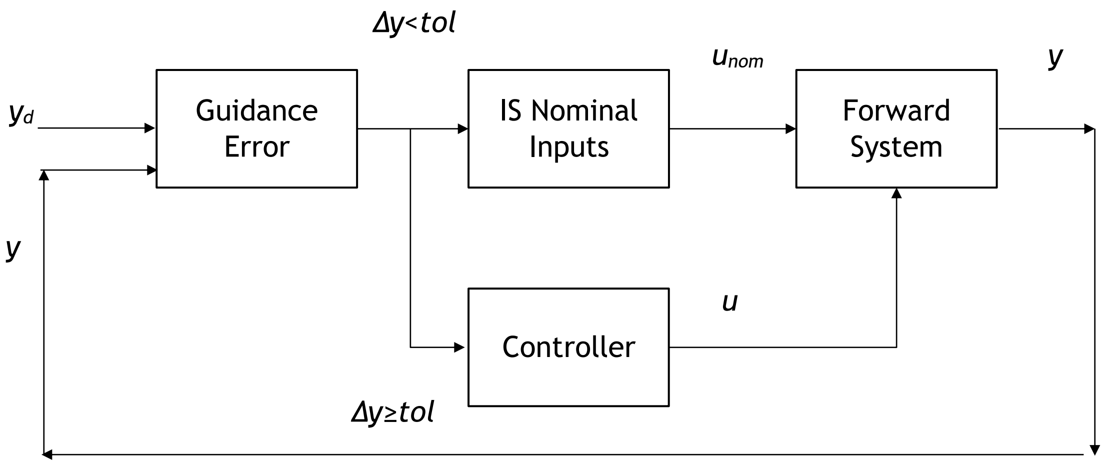


Figure 9.2: Inverse Simulation Expected Perception

In the context of this work, the internal model is the inverse as defined by Inverse Simulation and provides the nominal inputs for the desired outputs. While the actual outputs as obtained by the nominal inputs in a forward simulation do not exceed the error tolerance, then there is no need for additional correction.

9.2.2 Existence of Solution and Initial Parameters

In Chapter 4, the observation was made that both methods use an iterative Newton – Raphson scheme to solve the system of algebraic equations, over a time interval discretised with an appropriate time step dt. This led to investigating the number of inputs, outputs and states, the size of the Jacobian, its dependence on the number of inputs and outputs and how this affects the solution in Chapters 4 & 7.

Viewing Inverse Simulation as fundamentally trying to find a solution using the Newton-Raphson method at each t_i, it can be reduced to Eq.(9.1). The convergence rate is at best quadratic locally in an area around the initial starting point $\mathbf{x}_0$ (Kreyszig, 2014).

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \left[\mathbf{F}'\left(\mathbf{x}_{n-1}\right)\right]^{-1}\mathbf{F}\left(\mathbf{x}_{n-1}\right) \tag{9.1}$$

At this point, another set of questions arises. First, is it possible to know in advance that Eq.(9.1) has a solution at least locally, i.e. in an area around the initial starting point, and can that area be calculated? Second, under which conditions for the Jacobian $\mathbf{F}(\mathbf{x}_0), \mathbf{F}(\mathbf{x}_n)$ will Eq.(9.1) converge?

Knowing the existence of a solution in advance and the area where it may be found would mean that the Inverse Simulation will indeed find the solution if the algorithm is set up appropriately. It would also indicate whether the desired output is feasible. These questions do not deal directly with the properties discussed in this work; rather they attempt to see if there is such a solution, to begin with. Of course, the Jacobian and so the number of inputs, outputs and states do matter, as well as how well the Newton-Raphson algorithm is set up in terms of the time step and the convergence tolerance. After all, it is the same problem viewed in a different way.

The questions of existence, conditions and error bounds can be answered by the Kantorovich theorem, which states that: "The iterative Newton method applied to a general system of nonlinear equations, converges to a solution near an initial point $\mathbf{x}_0$, provided that the Jacobian of the system satisfies a Lipschitz condition[10] near $\mathbf{x}_0$ and its inverse at $\mathbf{x}_0$ satisfies certain boundness conditions. The theorem also gives computable error bounds for the iterates" (Tapia, 1971).

This theorem is both an existence and a convergence theorem for nonlinear equations, without the need to find the actual solution (Polyak, 2006). References (Ortega, 1968; Tapia, 1971; Argyros, 2008) provide formulations for the Kantorovich theorem for square systems The Kantorovich theorem has also been expanded for overdetermined and underdetermined systems (Galántai, 2000; Polyak, 2006; Argyros, 2008). Applied to Inverse Simulation, a square system corresponds to the case of an equal number of inputs and outputs and a non-square system (over or underdetermined) to the more general case, as detailed in Chapter 3. The benefit of using this approach is that based on the values of the Jacobian and its inverse (or factorised inverse) at the starting point, the area and order of convergence can be found as well as an upper error bound for the iterates (Ortega, 1968; Tapia, 1971; Argyros, 2008).

[10] A function $f:V \to E$ satisfies a Lipschitz condition (is Lipschitz-continuous) on the open set $U \subseteq V$ if there is a positive constant L (called the Lipschitz constant) such that $\left\| f(x) - f(y) \right\| \le L \left\| x - y \right\|$ for all $x, y \in U$ (Lang, 1997). This is a weaker condition than saying a function is differentiable (Lang, 1997).

The error bound for the current iteration n that is given by $\|\mathbf{x}_d - \mathbf{x}_n\| \le \alpha_n$, where $\mathbf{x}_d$ is the solution (the desired output for Inverse Simulation), and the maximum error is $\|\mathbf{x}_d - \mathbf{x}_0\| \le \alpha_0$. The values of α_0, α_n can be calculated as in (Ortega, 1968; Tapia, 1971; Argyros, 2008) Then, the value of α_0 can be used as an upper estimate for the convergence tolerance of Inverse Simulation.

The way this approach could be used is to examine if the initial starting point at the start of each iteration, would lead to a solution at each. The main issue is, as always, the calculation of the Jacobian and its inverse. Of course, both the Differentiation and the Integration algorithm require the calculation of the Jacobian anyway, so this will not be an additional calculation. For example, when applied to the rover, this could be used to check if the waypoints are feasible given the model. The works that developed and applied the Kantorovich theorem assume that the Jacobian in its analytical form is accessible, which is far from straightforward. An alternative worth investigating is assuming that at least a good approximation of the Jacobian and its reverse (instead of its analytical form) is also suitable.

9.2.3 Other Considerations

In this work, the calculation of the Jacobian is required. An alternative to numerical formulas using divided differences is Automatic Differentiation. Automatic differentiation uses "exact formulas along with floating-point values, instead of expression strings as in symbolic differentiation, and it involves no approximation error as in numerical differentiation using difference" (Neidinger, 2010). The method has been used in computational fluid dynamics, atmospheric sciences, engineering design optimization and machine learning (Baydin *et al.*, 2015). The benefit of this approach is that the accuracy is guaranteed, it works well in iterative solvers and is easy to generalize to higher derivatives. This approach is suitable for both the Differentiation and the Integration methods. Also observed were low amplitude, and high frequency oscillations present in the control signal, primarily when using the Differentiation method. If these are deemed to be problematic for application particular implementation, a filter could be used to remove these oscillations before applying the inputs to the system.

Another thing for future consideration is augmenting the model used in this work. This could be done by including terramechanics, a suspension and simulating the movement of the rover on a slope. The last case would be particularly interesting for Differentiation because Differentiation uses a reduced Jacobian with only u, r, and v – the states that can be directly controlled. A further point to consider is what would constitute a desired output; for example, when moving on a slope it would make sense to constrain the permissible tilt.

In closing, this thesis (a) provides the tools to move forward in applying Inverse Simulation to a wide range of applications, such as control, fault detection, disturbance detection, and output feasibility, and (b) applies Inverse Simulation for output tracking to a four wheeled rover model. Inverse Simulation calculates the changes in velocity, rotation, and acceleration for following a desired trajectory and for control by using these inputs to execute the desired trajectory. Hence, Inverse Simulation is a method for guidance and control.

List of References

Argyros, I. K. (2008) *Convergence and Applications of Newton-type Iterations*. 1st edn. New York, NY: Springer New York. doi: 10.1007/978-0-387-72743-1.

Arvidson, R. E., Iagnemma, K. D., Maimone, M., *et al.* (2017) 'Mars Science Laboratory Curiosity Rover Megaripple Crossings up to Sol 710 in Gale Crater', *Journal of Field Robotics*, 34(3), pp. 495–518. doi: 10.1002/rob.21647.

Aström, K. J. and Kumar, P. R. (2014) 'Control: A perspective', *Automatica*, 50(1), pp. 3–43. doi: 10.1016/j.automatica.2013.10.012.

Avanzini, G. and Matteis, G. de (2001) 'Two-Timescale Inverse Simulation of a Helicopter Model', *Journal of Guidance, Control, and Dynamics*, 24(2), pp. 330–339. doi: 10.2514/2.4716.

Avanzini, G., Matteis, G. De and Torasso, A. (2010) 'Modelling issues in helicopter inverse simulation', in *36th European Rotorcraft Forum*. Paris, France.

Avanzini, G., De Matteis, G. and Torasso, A. (2017) 'Assessment of Helicopter Model Accuracy Through Inverse Simulation', *Journal of Aircraft*, 54(2), pp. 535–547. doi: 10.2514/1.C033847.

Avanzini, G., Thomson, D. G. and Torasso, A. (2013) 'Model Predictive Control Architecture for Rotorcraft Inverse Simulation', *Journal of Guidance, Control, and Dynamics*, 36(1), pp. 207–217. doi: 10.2514/1.56563.

Bagiev, M., Thomson, D. G., Anderson, D. and Murray-Smith, D. (2012) 'Predictive inverse simulation of helicopters in aggressive manoeuvring flight', *The Aeronautical Journal*, 116(1175), pp. 87–98. doi: 10.1017/S0001924000006631.

Bajracharya, M., Maimone, M. W. and Helmick, D. (2008) 'Autonomy for Mars Rovers: Past, Present, and Future', *Computer*, 41(12), pp. 44–50. doi: 10.1109/MC.2008.479.

Barfoot, T., Furgale, P., Stenning, B., *et al.* (2011) ‘Field testing of a rover guidance, navigation, and control architecture to support a ground-ice prospecting mission to Mars’, *Robotics and Autonomous Systems*, 59(6), pp. 472–488. doi: 10.1016/j.robot.2011.03.004.

Bartlett, P. W., Wettergreen, D. and Whittaker, W. (2008) ‘Design of the Scarab Rover for Mobility & Drilling in the Lunar Cold Traps’, in *9th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*. Hollywood, USA, pp. 3–6.

Bartsch, S., Birnschein, T., Cordes, F., *et al.* (2010) ‘SpaceClimber: Developement of a Six-Legged Climbing Robot for Space Exploration’, in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics),*. Munich, Germany, pp. 1–8.

Bartsch, S., Birnschein, T., Römmermann, M., Hilljegerdes, J., Kühn, D. and Kirchner, F. (2012) ‘Development of the six-legged walking and climbing robot SpaceClimber’, *Journal of Field Robotics*, 29(3), pp. 506–532. doi: 10.1002/rob.21418.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A. and Siskind, J. M. (2015) ‘Automatic differentiation in machine learning: a survey’, *Journal of Machine Learning Research*, 18, pp. 1–43. Available at: http://arxiv.org/abs/1502.05767.

Belo, F. A. W., Birk, A., Brunskill, C., *et al.* (2012) ‘The ESA Lunar Robotics Challenge: Simulating operations at the lunar south pole’, *Journal of Field Robotics*, 29(4), pp. 601–626. doi: 10.1002/rob.20429.

Biesiadecki, J. J., Leger, P. C. and Maimone, M. W. (2007) ‘Tradeoffs Between Directed and Autonomous Driving on the Mars Exploration Rovers’, *The International Journal of Robotics Research*, 26(1), pp. 91–104. doi: 10.1177/0278364907073777.

Biesiadecki, J. J. and Maimone, M. W. (2006) ‘The Mars Exploration Rover Surface Mobility Flight Software: Driving Ambition’, in *2006 IEEE Aerospace Conference*. Big Sky, MT, USA: IEEE, pp. 1–15. doi: 10.1109/AERO.2006.1655723.

Blajer, W., Graffstein, J. and Krawczyk, M. (2009) ‘Modeling of Aircraft Prescribed Trajectory Flight as an Inverse Simulation Problem’, in *Modeling, Simulation and Control of Nonlinear Engineering Dynamical Systems*. Dordrecht: Springer Netherlands, pp. 153–162. doi: 10.1007/978-1-4020-8778-3_14.

Blažič, S. (2011) ‘A novel trajectory-tracking control law for wheeled mobile robots’, *Robotics and Autonomous Systems*, 59(11), pp. 1001–1007. doi: 10.1016/j.robot.2011.06.005.

Botha, T. R. and Schalk Els, P. (2015) ‘Rough terrain profiling using digital image correlation’, *Journal of Terramechanics*, 59, pp. 1–17. doi: 10.1016/j.jterra.2015.02.002.

Bottasso, C. L. and Ragazzi, A. (2001) ‘Deferred-Correction Optimal Control with Applications to Inverse Problems in Flight Mechanics’, *Journal of Guidance, Control, and Dynamics*, 24(1), pp. 101–108. doi: 10.2514/2.4681.

Bradley, R., Padfield, G. D., Murray-Smith, D. J. and Thomson, D. G. (1990) ‘Validation of helicopter mathematical models’, *Transactions of the Institute of Measurement and Control*, 12(4), pp. 186–196. doi: 10.1177/014233129001200405.

Brockett, R. W. (1983) ‘Asymptotic stability and feedback stabilization’, in Brockett, R. W., Millman, R. S., and Sussmann, H. J. (eds) *Differential Geometric Control Theory*. Boston, MA, USA: Birkhauser, pp. 181–208. doi: 10.1.1.324.9912.

Campion, G. and Chung, W. (2008) ‘Wheeled Robots’, in Siciliano, B. and Khatib, O. (eds) *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer, pp. 391–410. Available at: https://link.springer.com/referenceworkentry/10.1007/978-3-540-30301-5_18.

Carsten, J., Rankin, A., Ferguson, D. and Stentz, A. (2009) ‘Global planning on the Mars Exploration Rovers: Software integration and surface testing’, *Journal of Field Robotics*, 26(4), pp. 337–357. doi: 10.1002/rob.20287.

Cauli, N., Falotico, E., Bernardino, A., Santos-Victor, J. and Laschi, C. (2016) ‘Correcting for changes: expected perception-based control for reaching a moving target’, *IEEE Robotics & Automation Magazine*, 23(1), pp. 63–70. doi: 10.1109/MRA.2015.2505958.

Celi, R. (2000) ‘Optimization-Based Inverse Simulation of a Helicopter Slalom Maneuver’, *Journal of Guidance, Control, and Dynamics*, 23(2), pp. 289–297. doi: 10.2514/2.4521.

Chapra, S. C. and Canale, R. (2001) *Numerical Methods for Engineers: With Software and Programming Applications*. 4th edn. McGraw-Hill Higher Education.

Cheein, F. A. and Scaglia, G. (2014) ‘Trajectory Tracking Controller Design for Unmanned Vehicles: A New Methodology’, *Journal of Field Robotics*, 31(6), pp. 861–887. doi: 10.1002/rob.21492.

Chhaniyara, S., Brunskill, C., Yeomans, B., *et al.* (2012) ‘Terrain trafficability analysis and soil mechanical property identification for planetary rovers: A survey’, *Journal of Terramechanics*, 49(2), pp. 115–128. doi: 10.1016/j.jterra.2012.01.001.

Chu, L. E., Brown, K. M. and Kriechbaum, K. (2017) ‘Mars 2020 sampling and caching subsystem environmental development testing and preliminary results’, *IEEE Aerospace Conference Proceedings*, (Figure 2), pp. 1–10. doi: 10.1109/AERO.2017.7943564.

Connors, J. and Elkaim, G. (2007) ‘Analysis of a Spline Based, Obstacle Avoiding Path Planning Algorithm’, in *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*. IEEE, pp. 2565–2569. doi: 10.1109/VETECS.2007.528.

Cook, G. (2011) *Mobile Robots: Navigation, Control and Remote Sensing*. 1rst edn. New Jersey, USA: Wiley-IEEE Press.

Correal, R., Pajares, G. and Ruz, J. J. (2016) ‘Autonomy for ground-level robotic space exploration: framework, simulation, architecture, algorithms and experiments’, *Robotica*, 34(02), pp. 274–305. doi: 10.1017/S0263574714001428.

Davis, T. A. (2013) ‘Algorithm 930: FACTORIZE: An Object-Oriented Linear System Solver for MATLAB’, *ACM Transactions on Mathematical Software*, 39(4), pp. 1–18. doi: 10.1145/2491491.2491498.

Dawkins, J. J. (2014) ‘Model based off-road terrain profile estimation’, in *2014 American Control Conference*. Portland, Oregon, USA: IEEE, pp. 2792–2797. doi: 10.1109/ACC.2014.6859189.

de Divitiis, N. (1999) ‘Inverse Simulation of Aeroassisted Orbit Plane Change of a Spacecraft’, *Journal of Spacecraft and Rockets*, 36(6), pp. 882–889. doi: 10.2514/2.3507.

Dixon, J. C. (2007) *The Shock Absorber Handbook, The Shock Absorber Handbook: Second Edition*. Chichester, UK: John Wiley & Sons, Ltd (Wiley-Professional Engineering Publishing Series). doi: 10.1002/9780470516430.

Du, X. (2013) ‘Inverse simulation under uncertainty by optimization’, *Journal of Computing and Information Science in Engineering*, 13(2), pp. 1–8. doi: 10.1115/1.4023859.

Dubins, L. (1957) ‘On Curves of Minimal Length with a Constraint on Average Curvature and with Prescribed Initial and Terminal Positions and Tangents’, *American Journal of Mathematics*, 79, pp. 497–516.

Ferguson, K. and Thomson, D. (2016) ‘Maneuverability Assessment of a Compound Helicopter Configuration’, *Journal of the American Helicopter Society*, 61(1), pp. 1–15. doi: 10.4050/JAHS.61.012008.

Forbes-Spyratos, S., Jahn, I. H., Preller, D. and Smart, M. (2014) ‘Inverse Simulation for Hypersonic Vehicle Analysis’, in *19th AIAA International Space Planes and Hypersonic Systems and Technologies Conference*. Reston, Virginia, USA: American Institute of Aeronautics and Astronautics, pp. 1–19. doi: 10.2514/6.2014-2954.

Fossen, T. I. (2002) *Marine Control Systems*. 1rst edn. Trondheim, Norway: Marine Cybernetics.

Fossen, T. I. (2011) *Handbook of Marine Craft Hydrodynamics and Motion Control*. 1st edn. Chichester, UK: John Wiley & Sons, Ltd. doi: 10.1002/9781119994138.

Foster, L. V and Davis, T. A. (2013) 'Algorithm 933: Reliable Calculation of Numerical Rank, Null Space Bases, Pseudoinverse Solutions, and Basic Solutions using SuiteSparseQR', *ACM Transactions on Mathematical Software*, 40(1), pp. 1–23. doi: 10.1145/2513109.2513116.

Galántai, A. (2000) 'The theory of Newton's method', *Journal of Computational and Applied Mathematics*, 124(1–2), pp. 25–44. doi: 10.1016/S0377-0427(00)00435-0.

Garcia, E., Jimenez, M. A., De Santos, P. G. and Armada, M. (2007) 'The evolution of robotics research', *IEEE Robotics & Automation Magazine*, 14(1), pp. 90–103. doi: 10.1109/MRA.2007.339608.

Ghotbi, B., González, F., Kövecses, J. and Angeles, J. (2016) 'Mobility evaluation of wheeled robots on soft terrain: Effect of internal force distribution', *Mechanism and Machine Theory*, 100, pp. 259–282. doi: 10.1016/j.mechmachtheory.2016.02.005.

Gray, G. J. (1992) *Development and Validation of Nonlinear Models for Helicopter Dynamics*. University of Glasgow. Available at: https://core.ac.uk/download/pdf/293063277.pdf.

Gu, Y., Ohi, N., Lassak, K., *et al.* (2017) 'Cataglyphis: An autonomous sample return rover', *Journal of Field Robotics*, (May 2016), pp. 1–27. doi: 10.1002/rob.21737.

Hess, R. A. and Gao, C. (1993) 'A Generalized Algorithm for Inverse Simulation Applied to Helicopter Maneuvering Flight', *Journal of the American Helicopter Society*, 38(4), pp. 3–15. doi: 10.4050/JAHS.38.3.

Hess, R. A., Wang, S. H. and Gao, C. (1991) 'Generalized technique for inverse simulation applied to aircraft maneuvers', *Journal of Guidance, Control, and*

Dynamics, 14(5), pp. 920–926. doi: 10.2514/3.20732.

Heumann, C., Schomaker, M. and Shalabh (2016) *Introduction to Statistics and Data Analysis*, *Springer International Publishing*. Springer International Publishing. doi: 10.1007/978-3-319-46162-5.

Heverly, M., Matthews, J., Lin, J., *et al.* (2013) 'Traverse Performance Characterization for the Mars Science Laboratory Rover', *Journal of Field Robotics*, 30(6), pp. 835–846. doi: 10.1002/rob.21481.

Higham, N. J. (2002) *Accuracy and stability of numerical algorithms*. 2nd edn. Philadelphia, USA: SIAM (Society for Industrial and Applied Mathematics). doi: 10.2307/2669725.

Howard, T. M. and Kelly, A. (2007) 'Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots', *The International Journal of Robotics Research*, 26(2), pp. 141–166. doi: 10.1177/0278364906075328.

Howard, T. M., Morfopoulos, A., Morrison, J., *et al.* (2012) 'Enabling continuous planetary rover navigation through FPGA stereo and visual odometry', in *2012 IEEE Aerospace Conference*. Big Sky, MT, USA: IEEE, pp. 1–9. doi: 10.1109/AERO.2012.6187041.

Hwang, C.-L. and Wu, H.-M. (2013) 'Trajectory tracking of a mobile robot with frictions and uncertainties using hierarchical sliding-mode under-actuated control', *IET Control Theory & Applications*, 7(7), pp. 952–965. doi: 10.1049/iet-cta.2012.0750.

InSight Mission Overview (2018) *Nasa*. Available at: https://mars.nasa.gov/insight/mission/overview/ (Accessed: 15 December 2018).

Ireland, M., Flessa, T., Thomson, D. and McGookin, E. (2017) 'Comparison of Nonlinear Dynamic Inversion and Inverse Simulation', *Journal of Guidance, Control, and Dynamics*, 40(12), pp. 3307–3312. doi: 10.2514/1.G002875.

Ireland, M. L., Worrall, K. J., Mackenzie, R., Flessa, T., McGookin, E. W. and Thomson, D. G. (2017) ‘A Comparison of Inverse Simulation-Based Fault Detection in a Simple Robotic Rover with a Traditional Model-Based Method’, in *19th International Conference on Autonomous Robots and Agents (ICARA 2017)*. Madrid, Spain. doi: doi.org/10.5281/zenodo.1129652.

Ireland, M., Mackenzie, R., Flessa, T., Worrall, K. J., Thomson, D. and McGookin, E. (2017) ‘Inverse Simulation as a Tool for Fault Detection and Isolation in Planetary Rovers’, in *10th International ESA Conference on Guidance, Navigation and Control Systems (GNC)*. Salzburg, Austria. Available at: http://eprints.gla.ac.uk/136302/.

Isidori, A. (1995) *Nonlinear Control Systems*. London: Springer London (Communications and Control Engineering). doi: 10.1007/978-1-84628-615-5.

Isidori, A. (1999) *Nonlinear Control Systems II*. London: Springer London (Communications and Control Engineering). doi: 10.1007/978-1-4471-0549-7.

Kanayama, Y., Kimura, Y., Miyazaki, F. and Noguchi, T. (1990) ‘A stable tracking control method for an autonomous mobile robot’, in *1990 IEEE International Conference on Robotics and Automation*. Cincinnati, OH, USA: IEEE Comput. Soc. Press, pp. 384–389. doi: 10.1109/ROBOT.1990.126006.

Karelahti, J., Virtanen, K. and Öström, J. (2008) ‘Automated Generation of Realistic Near-Optimal Aircraft Trajectories’, *Journal of Guidance, Control, and Dynamics*, 31(3), pp. 674–688. doi: 10.2514/1.31159.

Keesman, K. J. (2011) *System Identification*. London: Springer London (Advanced Textbooks in Control and Signal Processing). doi: 10.1007/978-0-85729-522-4.

Khalil, H. K. (2003) *Nonlinear Systems International Edition*. 3rd edn. New Jersey, USA: Pearson Education.

Kim, C. J., Lee, S. H. and Hur, S. W. (2020) ‘Kinematically Exact Inverse-Simulation Techniques with Applications to Rotorcraft Aggressive-Maneuver Analyses’, *International Journal of Aeronautical and Space Sciences*, 21(3), pp.

790–805. doi: 10.1007/s42405-020-00249-8.

Kolter, J. Z. and Ng, A. Y. (2009) 'Task-space trajectories via cubic spline optimization', in *2009 IEEE International Conference on Robotics and Automation*. Kobe, Japan: IEEE, pp. 1675–1682. doi: 10.1109/ROBOT.2009.5152554.

Kreyszig, E. (2014) *Advanced Engineering Mathematics*. 10th edn. John Wiley & Sons, Ltd. Available at: https://www.wiley.com/.

Lang, S. (1997) *Undergraduate Analysis*. New York, NY: Springer New York (Undergraduate Texts in Mathematics). doi: 10.1007/978-1-4757-2698-5.

LaValle, S. M. (2006) *Planning Algorithms*, *Planning Algorithms*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511546877.

Lekkas, A. and Fossen, T. (2013) 'Line-of-sight guidance for path following of marine vehicles', in Gal, O. (ed.) *Advanced in Marine Robotics*. 1st edn. Lambert Academic, pp. 1–29.

Lekkas, A. M. and Fossen, T. I. (2014) 'Integral LOS Path Following for Curved Paths Based on a Monotone Cubic Hermite Spline Parametrization', *IEEE Transactions on Control Systems Technology*, 22(6), pp. 2287–2301. doi: 10.1109/TCST.2014.2306774.

Lin, K. C. and Lu, P. (1995) 'Inverse Simulation - An Error Analysis', *SIMULATION*, 65(6), pp. 385–392. doi: 10.1177/003754979506500602.

Lindemann, R. A., Bickler, D. B., Harrington, B. D., Ortiz, G. M. and Voothees, C. J. (2006) 'Mars exploration rover mobility development', *IEEE Robotics & Automation Magazine*, 13(2), pp. 19–26. doi: 10.1109/MRA.2006.1638012.

Lindemann, R. A. and Voorhees, C. J. (2005) 'Mars Exploration Rover Mobility Assembly Design, Test and Performance', in *2005 IEEE International Conference on Systems, Man and Cybernetics*. Waikoloa, HI, USA: IEEE, pp. 450–455. doi: 10.1109/ICSMC.2005.1571187.

Liu, S. and Sun, D. (2011) ‘Optimal motion planning of a mobile robot with minimum energy consumption’, in *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. Budapest, Hungary: IEEE, pp. 43–48. doi: 10.1109/AIM.2011.6027010.

Liu, S. and Sun, D. (2014) ‘Minimizing Energy Consumption of Wheeled Mobile Robots via Optimal Motion Planning’, *IEEE/ASME Transactions on Mechatronics*, 19(2), pp. 401–411. doi: 10.1109/TMECH.2013.2241777.

Lu, L., Murray-Smith, D. J. and McGookin, E. W. (2007) ‘Investigation of inverse simulation for design of feedforward controllers’, *Mathematical and Computer Modelling of Dynamical Systems*, 13(5), pp. 437–454. doi: 10.1080/13873950701344023.

Lu, L., Murray-Smith, D. J. and Thomson, D. G. (2008) ‘Issues of numerical accuracy and stability in inverse simulation’, *Simulation Modelling Practice and Theory*, 16(9), pp. 1350–1364. doi: 10.1016/j.simpat.2008.07.003.

De Luca, A., Oriolo, G. and Vendittelli, M. (2001) ‘Control of Wheeled Mobile Robots: An Experimental Overview’, in Nicosia, S., Siciliano, B., Bicchi, A., and Valigi, P. (eds) *Lecture Notes in Control and Information Sciences*. 1st edn. Berlin, Heidelberg: Springer, pp. 181–226. doi: 10.1007/3-540-45000-9_8.

Lynxmotion (2018). Available at: http://www.lynxmotion.com/driver.aspx?Topic=oldassem (Accessed: 15 December 2018).

Lu, L., Murray-Smith, D. J. and Thomson, D. G. (2007) ‘Sensitivity-Analysis Method for Inverse Simulation Application’, *Journal of Guidance, Control, and Dynamics*, 30(1), pp. 114–1215. doi: 10.2514/1.20722.

Madison, R., Jain, A., Lim, C. and Maimone, M. (2007) ‘Performance characterization of a rover navigation algorithm using large-scale simulation’, *Scientific Programming*, 15(2), pp. 95–105.

Maimone, M., Cheng, Y. and Matthies, L. (2007) ‘Two years of Visual Odometry on the Mars Exploration Rovers', *Journal of Field Robotics*, 24(3), pp. 169–186. doi: 10.1002/rob.20184.

Maimone, M., Johnson, A., Cheng, Y., Willson, R. and Matthies, L. (2006) 'Autonomous Navigation Results from the Mars Exploration Rover (MER) Mission', in Ang, M. H. and Khatib, O. (eds) *Experimental Robotics IX. Springer Tracts in Advanced Robotics*. 1st edn. Berlin, Heidelberg: Springer, Berlin, Heidelberg, pp. 3–13. doi: 10.1007/11552246_1.

Maimone, M. W., Biesiadecki, J., Tunstel, E., Cheng, Y. and Leger, C. (2006) 'Surface Navigation and Mobility Intelligence on the Mars Exploration Rovers', in Howard, A. M. and Tunstel, E. W. (eds) *Intelligence for Space Robotics*. San Antonio, TX, USA: TSI Press, pp. 45–70.

Martins, F. N., Celeste, W. C., Carelli, R., Sarcinelli-Filho, M. and Bastos-Filho, T. F. (2008) 'An adaptive dynamic controller for autonomous mobile robot trajectory tracking', *Control Engineering Practice*, 16(11), pp. 1354–1363. doi: 10.1016/j.conengprac.2008.03.004.

Mateo Sanguino, T. de J. (2017) '50 years of rovers for planetary exploration: A retrospective review for future directions', *Robotics and Autonomous Systems*, 94, pp. 172–185. doi: 10.1016/j.robot.2017.04.020.

Michaud, S., Hoepflinger, M., Thueer, T., *et al.* (2008) 'Lesson Learned from Exomars Locomotion System Test Campaign', in *10th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*. Noordwijk, The Netherlands, pp. 1–8.

Moler, C. B. (2004) *Numerical Computing with Matlab*. Society for Industrial and Applied Mathematics. doi: 10.1137/1.9780898717952.

Morin, P. and Samson, C. (2008) 'Motion Control of Wheeled Mobile Robots', in Siciliano, B. and Khatib, O. (eds) *Springer Handbook of Robotics*. 1st edn. Berlin, Heidelberg: Springer, pp. 133–159. doi: 10.1007/978-3-540-30301-5_7.

Muirhead, B. K. (2004) 'Mars rovers, past and future', in *2004 IEEE Aerospace*

Conference. Big Sky, MT, USA: IEEE, pp. 128–134. doi: 10.1109/AERO.2004.1367598.

Murray-Smith, D. J. (2000) 'The inverse simulation approach: a focused review of methods and applications', *Mathematics and Computers in Simulation*, 53(4–6), pp. 239–247. doi: 10.1016/S0378-4754(00)00210-X.

Murray-Smith, D. J. (2014) 'Inverse simulation and analysis of underwater vehicle dynamics using feedback principles', *Mathematical and Computer Modelling of Dynamical Systems*, 20(1), pp. 45–65. doi: 10.1080/13873954.2013.805146.

Murray-Smith, D. J., Lu, L. and McGookin, E. W. (2008) 'Applications of inverse simulation to a nonlinear model of an underwater vehicle.', in *Summer Simulation Multi-Conference 2008 - Grand Challenges in Modelling & Simulation*. Edinburgh, Scotland,.

Murray-Smith, D. J. and McGookin, E. W. (2015) 'A case study involving continuous system methods of inverse simulation for an unmanned aerial vehicle application', *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 229(14), pp. 2700–2717. doi: 10.1177/0954410015586842.

Nakayama, Y. (2018) 'Drag and Lift', in *Introduction to Fluid Mechanics*. Elsevier, pp. 177–201. doi: 10.1016/B978-0-08-102437-9.00009-7.

Neidinger, R. D. (2010) 'Introduction to automatic differentiation and MATLAB object-oriented programming', *SIAM Review*, 52(3), pp. 545–563. doi: 10.1137/080743627.

Nesnas, I. A. D., Matthews, J. B., Abad-Manterola, P., *et al.* (2012) 'Axel and DuAxel rovers for the sustainable exploration of extreme terrains', *Journal of Field Robotics*, 29(4), pp. 663–685. doi: 10.1002/rob.21407.

Ngwangwa, H. M., Heyns, P. S., Breytenbach, H. G. A. and Els, P. S. (2014) 'Reconstruction of road defects and road roughness classification using Artificial

Neural Networks simulation and vehicle dynamic responses: Application to experimental data', *Journal of Terramechanics*, 53, pp. 1–18. doi: 10.1016/j.jterra.2014.03.002.

Nie, C., Pacheco Corcho, X. and Spenko, M. (2013) 'Robots on the Move: Versatility and Complexity in Mobile Robot Locomotion', *IEEE Robotics & Automation Magazine*, 20(4), pp. 72–82. doi: 10.1109/MRA.2013.2248310.

Ogata, K. (2008) *Modern Control Engineering International Edition*. 5th edn. New Jersey, USA: Pearson Education.

Oriolo, G., De Luca, A. and Vendittelli, M. (2002) 'WMR control via dynamic feedback linearization: design, implementation, and experimental validation', *IEEE Transactions on Control Systems Technology*, 10(6), pp. 835–852. doi: 10.1109/TCST.2002.804116.

Ortega, J. M. (1968) 'The Newton-Kantorovich Theorem', *The American Mathematical Monthly*, 75(6), p. 658. doi: 10.2307/2313800.

Öström, J. (2007) 'Enhanced Inverse Flight Simulation for a Fatigue Life Management System', in *AIAA Modeling and Simulation Technologies Conference and Exhibit*. Reston, Virigina: American Institute of Aeronautics and Astronautics. doi: 10.2514/6.2007-6367.

Pacejka, H. B. (2012) 'Basic Tire Modeling Considerations', in *Tire and Vehicle Dynamics*. Elsevier, pp. 59–85. doi: 10.1016/B978-0-08-097016-5.00002-4.

Paden, B., Cap, M., Yong, S. Z., Yershov, D. and Frazzoli, E. (2016) 'A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles', *IEEE Transactions on Intelligent Vehicles*, 1(1), pp. 33–55. doi: 10.1109/TIV.2016.2578706.

Patel, N., Slade, R. and Clemmet, J. (2010) 'The ExoMars rover locomotion subsystem', *Journal of Terramechanics*, 47(4), pp. 227–242. doi: 10.1016/j.jterra.2010.02.004.

Polyak, B. T. (2006) ‘Newton-Kantorovich Method and Its Global Convergence’, *Journal of Mathematical Sciences*, 133(4), pp. 1513–1523. doi: 10.1007/s10958-006-0066-1.

Popp, K. and Schiehlen, W. (2010) *Ground vehicle dynamics*, *Ground Vehicle Dynamics*. doi: 10.1007/978-3-540-68553-1.

Quadrelli, M. B., Wood, L. J., Riedel, J. E., *et al.* (2015) ‘Guidance, Navigation, and Control Technology Assessment for Future Planetary Science Missions’, *Journal of Guidance, Control, and Dynamics*, 38(7), pp. 1165–1186. doi: 10.2514/1.G000525.

Rankin, A., Maimone, M., Biesiadecki, J., Patel, N., Levine, D. and Toupet, O. (2020) ‘Driving Curiosity: Mars Rover Mobility Trends during the First Seven Years’, *IEEE Aerospace Conference Proceedings*, (Lm). doi: 10.1109/AERO47225.2020.9172469.

Rath, J. J., Veluvolu, K. C. and Defoort, M. (2015) ‘Simultaneous Estimation of Road Profile and Tire Road Friction for Automotive Vehicle’, *IEEE Transactions on Vehicular Technology*, 64(10), pp. 4461–4471. doi: 10.1109/TVT.2014.2373434.

Reid, E., Iles, P., Muise, J., *et al.* (2015) ‘The Artemis Jr. rover: Mobility platform for lunar ISRU mission simulation’, *Advances in Space Research*, 55(10), pp. 2472–2483. doi: 10.1016/j.asr.2014.10.025.

Reina, G. and Foglia, M. (2013) ‘On the mobility of all-terrain rovers’, *Industrial Robot: An International Journal*, 40(2), pp. 121–131. doi: 10.1108/01439911311297720.

Renny Simba, K., Uchiyama, N. and Sano, S. (2016) ‘Real-time smooth trajectory generation for nonholonomic mobile robots using Bézier curves’, *Robotics and Computer-Integrated Manufacturing*, 41, pp. 31–42. doi: 10.1016/j.rcim.2016.02.002.

Rodríguez-Seda, E. J., Tang, C., Spong, M. W. and Stipanović, D. M. (2014)

‘Trajectory tracking with collision avoidance for nonholonomic vehicles with acceleration constraints and limited sensing’, *The International Journal of Robotics Research*, 33(12), pp. 1569–1592. doi: 10.1177/0278364914537130.

Rollins, E., Luntz, J., Foessel, A., Shamah, B. and Whittaker, W. (1998) ‘Nomad: A demonstration of the transforming chassis’, in *1998 IEEE International Conference on Robotics and Automation*. Leuven, Belgium: IEEE, pp. 611–617. doi: 10.1109/ROBOT.1998.677040.

Rutherford, S. and Thomson, D. G. (1996) ‘Improved methodology for inverse simulation’, *The Aeronautical Journal*, 100(93), pp. 79–86.

Savaresi, S. M., Poussot-Vassal, C., Spelta, C., Sename, O. and Dugard, L. (2010) *Semi-Active Suspension Control Design for Vehicles*. 1rst edn. Oxford, UK: Elsevier. doi: 10.1016/B978-0-08-096678-6.00014-6.

Schiele, A., Romstedt, J., Lee, C., *et al.* (2008) ‘Nanokhod exploration rover - A rugged rover suited for small, sow-cost, planetary lander mission’, *IEEE Robotics and Automation Magazine*, 15(2), pp. 96–107. doi: 10.1109/MRA.2008.917888.

Schramm, D., Hiller, M. and Bardini, R. (2014) *Vehicle Dynamics*, *SAE Technical Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-540-36045-2.

Sellers, J. J. (2005) *Understanding Space: An Introduction to Astronautics*. 3rd edn. McGraw-Hill.

Shamah, B. (1999) *Experimental Comparison of Skid Steering vs. Explicit Steering for Wheeled Mobile Robot*. Carnegie Mellon University.

Sharma, K. R., Dusek, F. and Honc, D. (2017) ‘Comparitive study of predictive controllers for trajectory tracking of non-holonomic mobile robot’, in *2017 21st International Conference on Process Control (PC)*. Štrbské Pleso, Slovakia: IEEE, pp. 197–203. doi: 10.1109/PC.2017.7976213.

Siegwart, R., Nourbakhsh, I. R. and Scaramuzza, D. (2011) *Introduction to*

Autonomous Mobile Robots. 2nd edn. Cambridge, Massachusetts: MIT Press.

Silva, N., Lancaster, R. and Clemmet, J. (2013) 'ExoMars Rover Vehicle Mobility Functional Architecture and Key Design Drivers', in *12th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*. Noordwijk, The Netherlands.

Skogestad, S. and Postlethwaite, I. (2005) *Multivariable feedback control: analysis and design*. 2nd edn. John Wiley & Sons, Ltd.

Škrjanc, I. and Klančar, G. (2017) 'A comparison of continuous and discrete tracking-error model-based predictive control for mobile robots', *Robotics and Autonomous Systems*, 87, pp. 177–187. doi: 10.1016/j.robot.2016.09.016.

Slotine, J. J. and Li, W. (1991) *Applied Nonlinear Control*. New Jersey, USA: Pearson Education.

Solea, R., Filipescu, A. and Nunes, U. (2009) 'Sliding-mode control for trajectory-tracking of a Wheeled Mobile Robot in presence of uncertainties', in *7th Asian Control Conference*. Hong Kong, China, pp. 1701–1706.

Strang, G. (2009) *Introduction to Linear Algebra*. 4th edn. Wellesley MA: Wellesley-Cambridge Press.

Su, Y. and Cao, Y. (2002) 'A nonlinear inverse simulation technique applied to coaxial rotor helicopter maneuvers', *Aircraft Engineering and Aerospace Technology*, 74(6), pp. 525–533. doi: 10.1108/00022660210420852.

SunSpiral, V., Wheeler, D. W., Chavez-Clemente, D. and Mittman, D. (2012) 'Development and field testing of the FootFall planning system for the ATHLETE robots', *Journal of Field Robotics*, 29(3), pp. 483–505. doi: 10.1002/rob.20410.

Tapia, R. A. (1971) 'The Kantorovich Theorem for Newton's Method', *The American Mathematical Monthly*, 78(4), p. 389. doi: 10.2307/2316909.

Thomson, D. G. (1987) *Evaluation of helicopter agility through inverse solution of the equations of motion*. University of Glasgow.

Thomson, D. G. and Bradley, R. (1990) ‘Prediction of the dynamic characteristics of helicopters in constrained flight’, *The Aeronautical Journal*, 94(940), pp. 344–354. doi: 10.1017/S0001924000023307.

Thomson, D. G. and Bradley, R. (2006) ‘Inverse simulation as a tool for flight dynamics research-Principles and applications’, *Progress in Aerospace Sciences*, 42(3), pp. 174–210. doi: 10.1016/j.paerosci.2006.07.002.

Thueer, T. and Siegwart, R. (2010) ‘Mobility evaluation of wheeled all-terrain robots’, *Robotics and Autonomous Systems*, 58(5), pp. 508–519. doi: 10.1016/j.robot.2010.01.007.

Tian, Y. and Sarkar, N. (2014) ‘Control of a Mobile Robot Subject to Wheel Slip’, *Journal of Intelligent & Robotic Systems*, 74(3–4), pp. 915–929. doi: 10.1007/s10846-013-9871-1.

Della Torre, A., Ercoli Finzi, A., Genta, G., *et al.* (2010) ‘AMALIA Mission Lunar Rover—The conceptual design of the Team ITALIA Rover, candidate for the Google Lunar X Prize Challenge’, *Acta Astronautica*, 67(7–8), pp. 961–978. doi: 10.1016/j.actaastro.2010.05.023.

Tzafestas, S. G. (2018) ‘Mobile Robot Control and Navigation: A Global Overview’, *Journal of Intelligent and Robotic Systems: Theory and Applications*, 91(1), pp. 35–58. doi: 10.1007/s10846-018-0805-9.

Ulamec, S., Espinasse, S., Feuerbacher, B., *et al.* (2006) ‘Rosetta Lander—Philae: Implications of an alternative mission’, *Acta Astronautica*, 58(8), pp. 435–441. doi: 10.1016/j.actaastro.2005.12.009.

Ulamec, S., Fantinati, C., Maibaum, M., *et al.* (2016) ‘Rosetta Lander – Landing and operations on comet 67P/Churyumov-Gerasimenko’, *Acta Astronautica*, 125, pp. 80–91. doi: 10.1016/j.actaastro.2015.11.029.

Wagner, M., Heys, S., Wettergreen, D., *et al.* (2005) ‘Design and control of a passively steered, dual axle vehicle’, in *8th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*. Munich, Germany, pp. 29–36.

Wettergreen, D., Moreland, S., Skonieczny, K., Jonak, D., Kohanbash, D. and Teza, J. (2010) 'Design and field experimentation of a prototype Lunar prospector', *The International Journal of Robotics Research*, 29(12), pp. 1550–1564. doi: 10.1177/0278364910370217.

Wilcox, B. H. (2012) 'ATHLETE: A limbed vehicle for solar system exploration', *IEEE Aerospace Conference Proceedings*, pp. 1–9. doi: 10.1109/AERO.2012.6187269.

Wolek, A. and Woolsey, C. A. (2017) 'Model-Based Path Planning', in Fossen, T. I., Pettersen, K. Y., and Nijmeijer, H. (eds) *Sensing and Control for Autonomous Vehicles*. 1st edn. Cham: Springer International Publishing (Lecture Notes in Control and Information Sciences), pp. 183–206. doi: 10.1007/978-3-319-55372-6_9.

Wong, J. Y. and Huang, W. (2006) '"Wheels vs. tracks" – A fundamental evaluation from the traction perspective', *Journal of Terramechanics*, 43(1), pp. 27–42. doi: 10.1016/j.jterra.2004.08.003.

Woods, M., Shaw, A., Tidey, E., *et al.* (2014) 'Seeker-Autonomous Long-range Rover Navigation for Remote Exploration', *Journal of Field Robotics*, 31(6), pp. 940–968. doi: 10.1002/rob.21528.

Worrall, K. (2010) *Guidance and search algorithms for mobile robots: application and analysis within the context of urban search and rescue*. University of Glasgow.

Worrall, K. J., Thomson, D., McGookin, E. and Flessa, T. (2015) 'Autonomous Planetary Rover Control using Inverse Simulation', in *13th ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA 2015)*. Noordwijk, The Netherlands. Available at: http://eprints.gla.ac.uk/106312/.

Worrall, K. and McGookin, E. W. (2006) 'A Mathematical Model of a LEGO Differential Drive Robot', in *International Control Conference (ICC)*. Glasgow, UK: UKACC (United Kingdom Automatic Control Council). Available at: http://eprints.gla.ac.uk/30835/.

Wright, J., Hartman, F., Cooper, B., Maxwell, S., Yen, J. and Morrison, J. (2006) ‘Driving on Mars with RSVP’, *IEEE Robotics & Automation Magazine*, 13(2), pp. 37–45. doi: 10.1109/MRA.2006.1638014.

Yang, G., Bellingham, J., Dupont, P. E., *et al.* (2018) ‘The grand challenges of Science Robotics’, *Science Robotics*, 3(14), p. eaar7650. doi: 10.1126/scirobotics.aar7650.

Yongguo Mei, Yung-Hsiang Lu, Hu, Y. C. and Lee, C. S. G. (2006) ‘Deployment of mobile robots with energy and timing constraints’, *IEEE Transactions on Robotics*, 22(3), pp. 507–522. doi: 10.1109/TRO.2006.875494.

Yoshida, K., Yoshida, K., Wilcox, B. and Wilcox, B. (2008) ‘Space Robots and Systems’, in Siciliano, B. and Khatib, O. (eds) *Springer Handbook of Robotics*. 1st edn. Berlin, Heidelberg, pp. 1031–1063. Available at: https://link.springer.com/referenceworkentry/10.1007/978-3-540-30301-5_46.

Young, A. (2007) *Lunar and Planetary rovers*. 1st edn, *Book*. 1st edn. Chichester, UK: Springer - Praxis.

Yun, X. and Yamamoto, Y. (1992) *On feedback linearization of mobile robots, Technical Reports (CIS)*. Available at: http://repository.upenn.edu/cgi/viewcontent.cgi?article=1524&context=cis_reports.

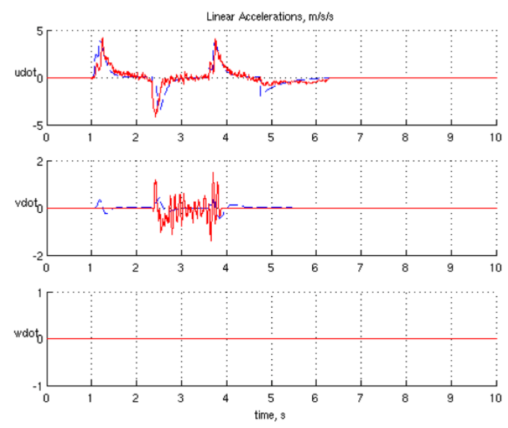
Yun, X. and Yamamoto, Y. (1993) ‘Internal dynamics of a wheeled mobile robot’, in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS ’93)*. IEEE, pp. 1288–1294. doi: 10.1109/IROS.1993.583753.

Zhou, W., Wang, H., Yu, D. and Sun, F. (2017) ‘Error Analysis and Modification of Inverse Simulation for Manually Controlled Rendezvous and Docking’, *Journal of Aerospace Engineering*, 30(1), p. 04016072. doi: 10.1061/(ASCE)AS.1943-5525.0000662.

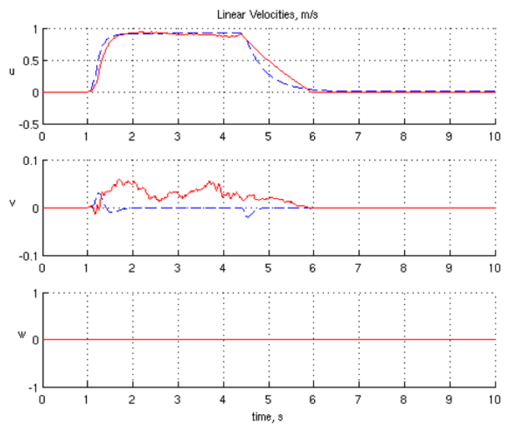
Appendix A: Rover Specifications

Quantity	Symbol	Value
Mass	m	2.148 kg
Wheel Radius	r_w	0.0635 m
Moment arm of wheel	r_m	0.1245 m
Effective area in x-axis	A_x	0.0316 m^2
Effective area in y-axis	A_y	0.0448 m^2
Drag Coefficient	C_d	0.89
Moment of Inertia about x-axis	I_x	0.0140 kgm^2
Moment of Inertia about y-axis	I_y	0.0252 kgm^2
Moment of Inertia about z-axis	I_z	0.0334 kgm^2
Coefficient of friction in x	σ_x	0.22
Coefficient of friction in y	σ_y	1.00
Coefficient of friction in z	σ_z	0.30
Coefficient of friction about x	σ_p	0.35
Coefficient of friction about y	σ_q	0.44
Coefficient of friction about z	σ_r	0.18
Viscous torque	b	0.008 Nm
Moment of inertia of motor	J_m	0.005 kgm^2
Torque constant	K_t	0.35 NmA^{-1}
EMF constant	K_e	0.35 $Vrad^{-1}s^{-1}$
Inductance of circuit	L	0.1 H
Resistance of circuit	R	4 Ω
Gradient for efficiency curve	α	-0.133 A^{-1}
Offset for efficiency curve	γ	0.6
Base friction coefficient on wheel	ξ	0.002 $Nmsrad^{-1}$

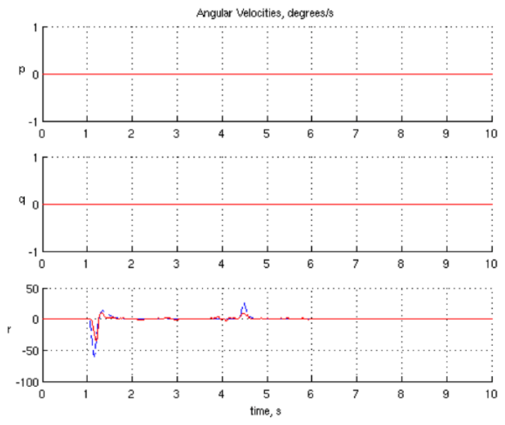
The validation results for the second experiment (drive the robot forward in a straight line and then execute a left turn) are shown below (Worrall, 2010).



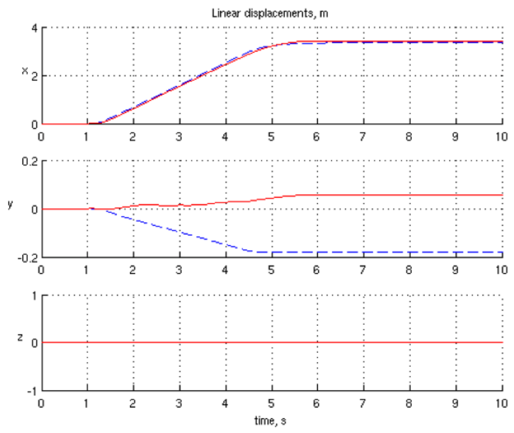
Linnear Accelerations



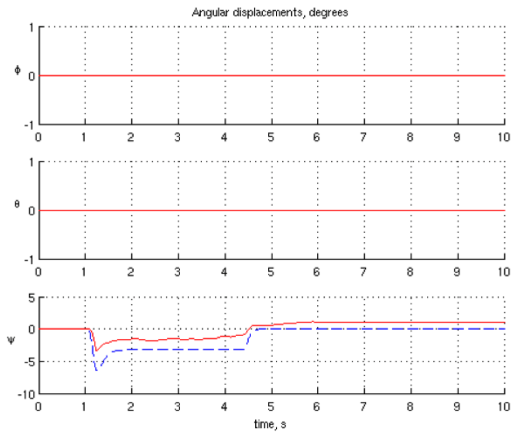
Linear Velocities



Angular Velocities



Linear Displacements



Angular Displacements

Appendix B: Inverse Simulation Differentiation

The algorithm steps for the Inverse Simulation Differentiating are the following.

1. At t_i the desired output $\mathbf{g_d}(t_i)$, the state $\mathbf{x}(t_{i-1})$ & input $\mathbf{u}(t_{i-1})$ are known.

2. Iteration $n=1$: obtain estimates for $\mathbf{x}_0(t_i)$, $\mathbf{u}_0(t_i)$.

3. Iteration $n>1$: $\mathbf{x}_{n-1}(t_i)$, $\mathbf{u}_{n-1}(t_i)$ from the previous iteration $n-1$.

4. Calculate new estimates for $\mathbf{F}_1$ and $\mathbf{F}_2$ using $\mathbf{x}_{n-1}(t_i)$, $\mathbf{u}_{n-1}(t_i)$, $\mathbf{x}(t_{i-1})$ from Eq. (4.1), Eq.(4.2) and Eq.(4.6).

5. Calculate the Jacobian $\mathbf{J}$ of $\mathbf{F}_1$ and $\mathbf{F}_2$. To do this, assume that the values of $\mathbf{x}_{n-1}(t_i)$, $\mathbf{u}_{n-1}(t_i)$ are perturbed by δx and δu respectively.

6. Factorise the Jacobian $\mathbf{J}$ to obtain $\mathbf{J}^{-1}$ and use Eq.(4.7) to find the new estimates $\mathbf{x}_n(t_i)$, $\mathbf{u}_n(t_i)$.

7. Use the new estimates $\mathbf{x}_n(t_i)$, $\mathbf{u}_n(t_i)$ to calculate the new values of $\mathbf{F}_1$ and $\mathbf{F}_2$ from Eq.(4.6).

8. Check if the values of $\mathbf{F}_1$ and $\mathbf{F}_2$ are close enough to zero given a defined tolerance.

9. If NO, then go to step 3 and continue to iteration $n+1$.

10. If YES, then the current estimates $\mathbf{x}_n(t_i)$, $\mathbf{u}_n(t_i)$ are the values necessary to achieve the desired output $\mathbf{g_d}(t_i)$. Set $\mathbf{x}(t_i)=\mathbf{x}_n(t_i)$, $\mathbf{u}(t_i)=\mathbf{u}_n(t_i)$, $\mathbf{y}(t_i)=\mathbf{g}(\mathbf{x}(t_i),\mathbf{u}(t_i))$ and move to t_{i+1}.

Appendix C: Inverse Simulation Integration

The algorithm steps for the Inverse Simulation Integration are the following.

1. At t_i the desired output $\mathbf{g_d}(t_i)$, the state $\mathbf{x}(t_{i-1})$ & input $\mathbf{u}(t_{i-2})$ are known

2. Iteration $n=1$: obtain estimates for $\mathbf{x}_0(t_{i-1})$, $\mathbf{u}_0(t_{i-1})$, $\mathbf{y}_0(t_i)$from Eq.(4.8).

3. Iteration $n>1$: obtain control input $\mathbf{u}_{n-1}(t_{i-1})$ and output $\mathbf{y}_{n-1}(t_i)$ from the previous iteration $n-1$.

4. Calculate the error function $\mathbf{f}_e$ from Eq.(4.10) using $\mathbf{y}_{n-1}(t_i)$.

5. Calculate the Jacobian $\mathbf{J}_e$ of the error function $\mathbf{f}_e$ by perturbing $\mathbf{u}_{n-1}(t_{i-1})$ and then applying it to the system state and output, Eq.(4.1), Eq.(4.8) to get the perturbed system output.

6. Calculate the new control input estimate $\mathbf{u}_n(t_{i-1})$ using Eq.(4.11) and the inverted Jacobian $\mathbf{J}_e^{-1}$.

7. Calculate the new state $\mathbf{x}_n(t_{i-1})$ & new output $\mathbf{y}_n(t_{i-1})$, Eq.(4.1), Eq.(4.8).

8. Calculate the absolute difference between the new output $\mathbf{y}_n(t_{i-1})$and the desired output $\mathbf{g_d}(t_i)$ and check if the difference is within a certain tolerance.

9. If NO, then go to step 3 and continue to iteration $n+1$.

10. If YES, then the current control estimate $\mathbf{u}_n(t_{i-1})$ is what will drive the output to the desired $\mathbf{g_d}(t_i)$. Set $\mathbf{u}(t_{i-1})=\mathbf{u}_n(t_{i-1})$, $\mathbf{x}(t_{i-1})=\mathbf{x}_n(t_{i-1})$, $\mathbf{y}(t_{i-1})=\mathbf{y}_n(t_{i-1})$and move to t_{i+1}.

Appendix D: MSD System & Inverse Simulation Parameters

Quantity	Symbol	Value
Mass	m	1 kg
Spring constant	k	10 Nm^{-1}
Damping coefficient	c	4 Ns^{-1}m
Damping ratio	ζ	0.63
Natural frequency	ω_0	3.16 s^{-1}
Control input	u	N
Matrix **A** eigenvalues	$\lambda_{1,2}$	-2 ± 2.45i
Total simulation time	T	20 s
Time step	dt	0.001 s
Convergence tolerance	tol	10^{-6} m
Maximum iterations	η_{max}	25
Position perturbation (Differentiation)	δx	10^{-5} m
Velocity perturbation (Differentiation)	$\delta x'$	10^{-4} m
Control perturbation (Differentiation)	δu	10^{-3} N
Minimum control perturbation (Integration)	pertu	10^{-4} N
Minimum output perturbation (Integration)	perty	10^{-5} N
PID Gain	K_p	500
PID Gain	K_D	50
PID Gain	K_I	300

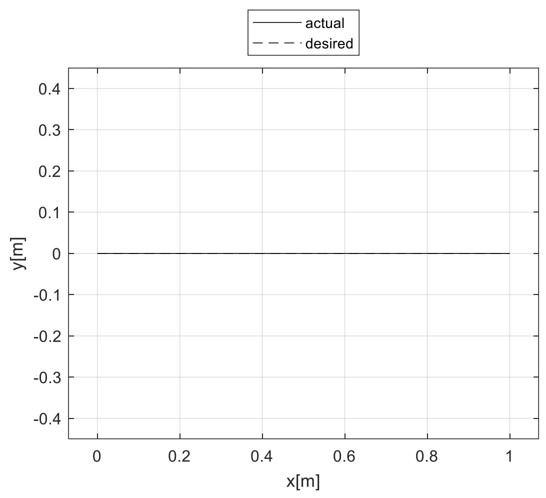
Appendix E: QC System & Inverse Simulation Parameters

Quantity	Symbol	Value
Sprung mass	m_{sq}	250 kg
Sprung mass position	x_1	m
Sprung mass velocity	$x_2 = x_1'$	ms^{-1}
Unsprung mass	m_{us}	30 kg
Unsprung mass position	x_3	m
Unsprung mass velocity	$x_4 = x_3'$	ms^{-1}
Spring stiffness	k_s	2000 Nm^{-1}
Damping coefficient	b_s	1000 Nsm^{-1}
Tyre Stiffness	k_t	200000 Nm^{-1}
Road disturbance	z_r	m
Actuator force	F_s	N
Matrix **A** eigenvalues	$\lambda_{1,2}$ $\lambda_{3,4}$	$-16.9950 \pm 83.289i$ $-1.6716 \pm 8.427i$
Total simulation time	T	10 s
Time step	dt	0.001 s
Maximum iterations	η_{max}	50
Minimum control perturbation	pertu	10^{-4} N
Minimum output perturbation (Active QC)	perty	10^{-8} m
Convergence tolerance (Active QC)	tol	10^{-5} m
PID Gain	K_P	7000
PID Gain	K_D	1000
PID Gain	K_I	10000
LQR control penalty	R	0.001
LQR variation for matrix **Q**	var	20000
Convergence tolerance (Passive QC)	tol	10^{-4} m
Minimum output perturbation (Passive QC)	perty	10^{-7} m

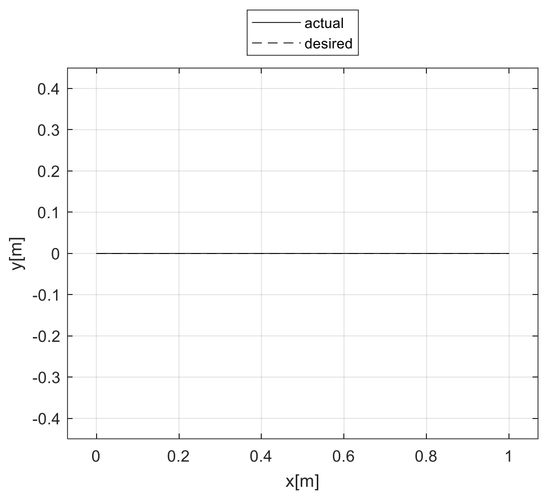
Appendix F: Rover Inverse Simulation Results

	Forward Drive Drive:1m		**Rhombus** Drive: 11.31m		**Valley** Drive: 13.44m	
Errors	Differentiation	Integration	Differentiation	Integration	Differentiation	Integration
$e_{X_e}\ [m]$	8.40 10^{-4}	8.76 10^{-7}	1.24 10^{-2}	1.24 10^{-2}	2.19 10^{-2}	2.13 10^{-2}
$e_{Y_e}\ [m]$	9.06 10^{-6}	5.91 10^{-7}	2.76 10^{-2}	2.79 10^{-2}	4.11 10^{-2}	4.05 10^{-2}
$e_{\psi}\ [rad]$	1.09 10^{-5}	1.19 10^{-6}	3.31 10^{-5}	1.68 10^{-5}	1.47 10^{-4}	2.00 10^{-5}
$\max(e_u)\ \left[\frac{m}{s}\right]$	1.45 10^{-5}	0	6.11 10^{-5}	0	6.25 10^{-5}	0
$\bar{e}_u\ \left[\frac{m}{s}\right]$	4.43 10^{-6}	0	7.98 10^{-6}	0	8.95 10^{-6}	0
σ_{e_u}	3.00 10^{-6}	0	7.36 10^{-6}	0	8.43 10^{-6}	0
$\max(e_v)\ \left[\frac{m}{s}\right]$	0	0	6.08 10^{-6}	6.64 10^{-6}	9.56 10^{-6}	8.56 10^{-6}
$\bar{e}_v\ \left[\frac{m}{s}\right]$	0	0	1.07 10^{-6}	1.62 10^{-6}	1.50 10^{-6}	1.54 10^{-6}
σ_{e_v}	0	0	1.46 10^{-6}	1.54 10^{-6}	1.98 10^{-6}	$2.06 \cdot 10^{-6}$
$\max(e_r)\ \left[\frac{rad}{s}\right]$	2.78 10^{-5}	0	7.23 10^{-5}	0	7.72 10^{-5}	0
$\bar{e}_r\ \left[\frac{rad}{s}\right]$	3.39 10^{-6}	0	4.96 10^{-6}	0	5.21 10^{-6}	0
σ_{e_r}	3.54 10^{-6}	0	4.65 10^{-6}	0	5.00 10^{-6}	0
$t_r\ [s]$	3.42	3.10	33.41	43.32	50.68	70.36

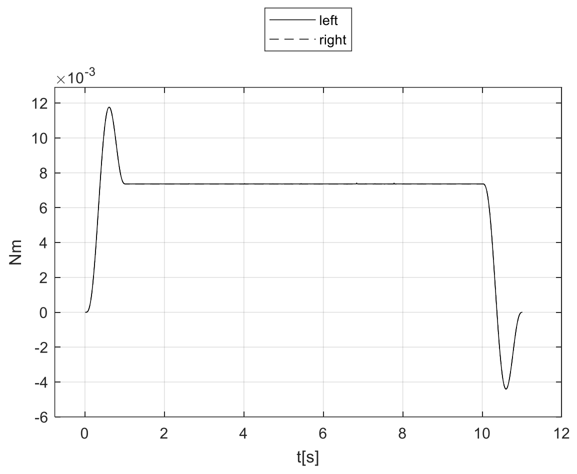
Forward 1m Results: For Differentiation, the final error is $8.40\ 10^{-4}\ \mathrm{m}$ on the global X_e axis, $8.76\ 10^{-4}\ \mathrm{m}$ on the Y_e axis. For Integration, the final error is $8.76\ 10^{-4}\ \mathrm{m}$ on the X_e axis and $5.91\ 10^{-4}\ \mathrm{m}$ on the Y_e axis.



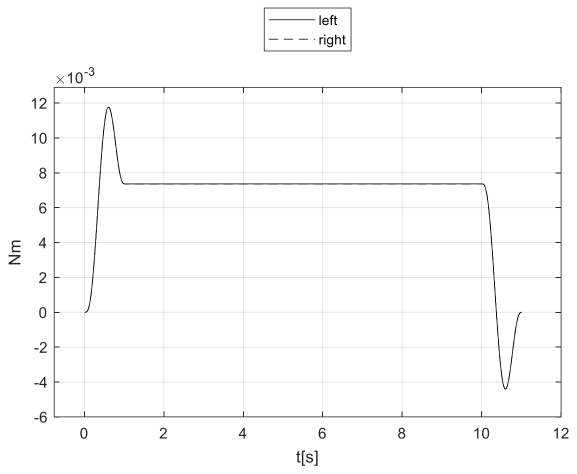
Trajectory Forward 1m (Differentiation)



Trajectory Forward 1m (Integration)

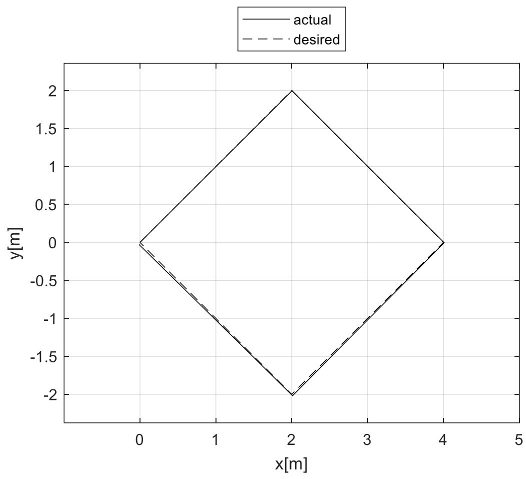


Control Input Forward 1m (Differentiation)

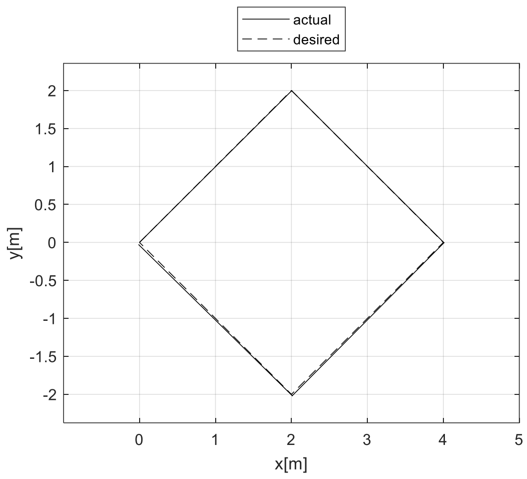


Control Input Forward 1m (Integration)

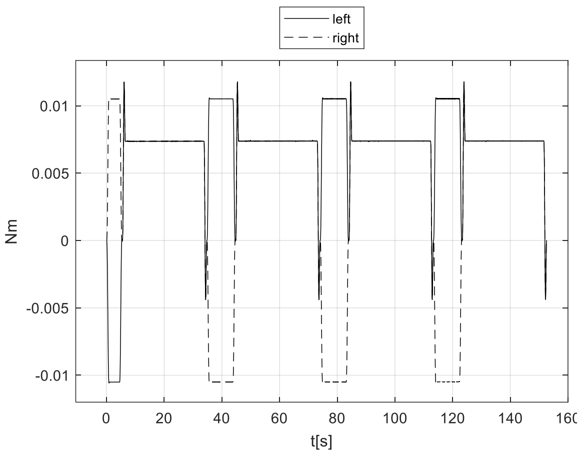
Rhombus: The rover starts from the origin $(0,0)$ and moves clockwise along the trajectory. For Differentiation, the final error is $1.24\ 10^{-2}\ \mathrm{m}$ on the global X_e axis and $2.76\ 10^{-2}\ \mathrm{m}$ on the Y_e axis. For Integration, the final error is $1.23\ 10^{-2}\ \mathrm{m}$ on the X_e axis and $2.79\ 10^{-2}\ \mathrm{m}$ on the Y_e axis. The spikes in the control input occur at the points where the rover executes a sharp turn. Both algorithms produce good results in the case of a closed trajectory with sharp $45\ \deg$ turns.



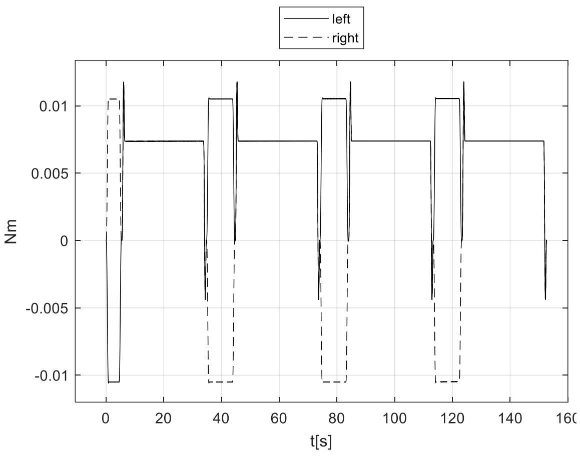
Trajectory Rhombus (Differentiation)



Trajectory Rhombus (Integration)

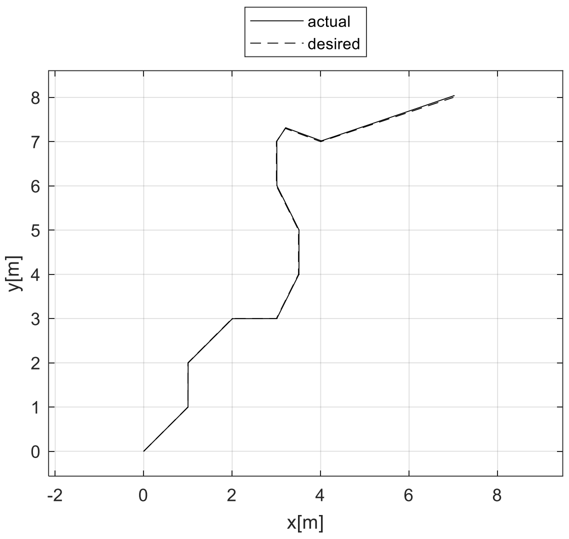


Control Input Rhombus (Differentiation)

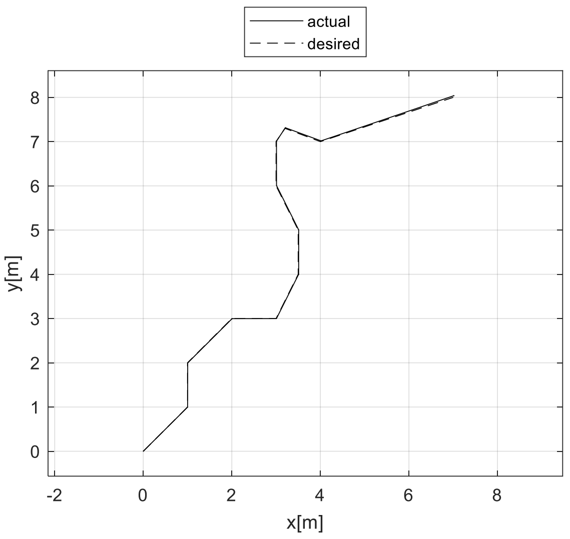


Control Input Rhombus (Integration)

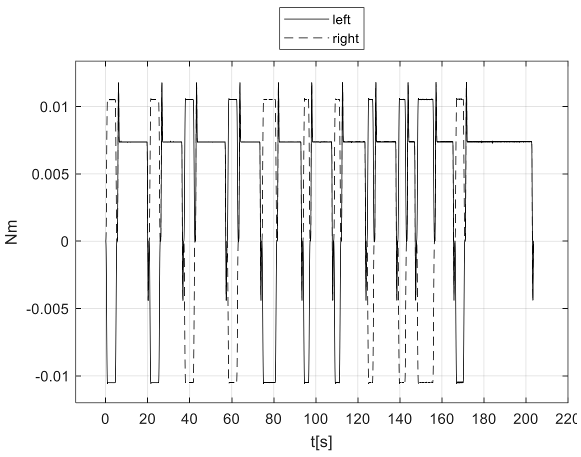
Valley: For Differentiation, the final error is $2.19\ 10^{-2}\ \mathrm{m}$ on the global X_e axis and $4.1\ 10^{-2}\ \mathrm{m}$ on the Y_e axis. For Integration, the final error is $2.13\ 10^{-2}\ \mathrm{m}$ on the X_e axis and $4.05\ 10^{-2}\ \mathrm{m}$ on the Y_e axis. Again, the spikes in the control input occur at the points where the rover executes a sharp turn.



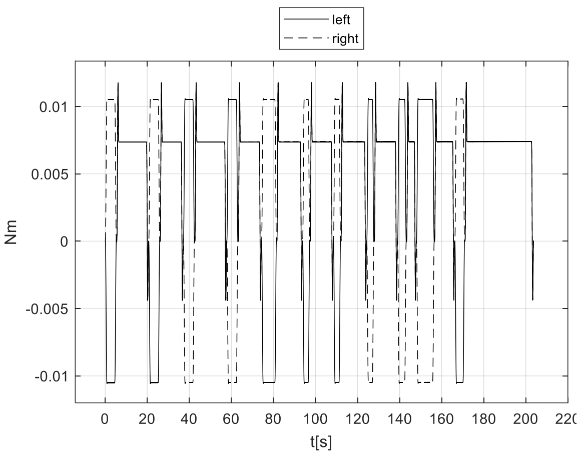
Trajectory Valley (Differentiation)



Trajectory Valley (Integration)



Control Input Valley (Differentiation)



Control Input Valley (Integration)

Appendix G: Solution of a General Linear System, Matrix Factorisation Methods and MATLAB implementation

The Differentiation method results in Eq.(4.7), which is a system of $(\mathrm{m+k})$ algebraic equations. This solution requires some form of factorisation for the Jacobian $\mathbf{J}$ of size $(\mathrm{m+p})\mathrm{x}(\mathrm{m+k})$, where there are m states, k control input variables and p outputs. Similar to the Differentiation algorithm, Integration results in a system of k algebraic equations, Eq.(4.11). The dimension of the Jacobian $\mathbf{J_e}$ is pxk, where k are the control input variables and p the outputs.

The general form of a linear system to be solved is $\mathbf{Ax}=\mathbf{b}$, $\mathbf{A}:nxm, \mathbf{x}:mx1, \mathbf{b}:nx1$ and is either consistent (has at least one solution) or inconsistent (has no solution). To find if it is consistent, the rank of the augmented matrix $\mathbf{A}|\mathbf{b}$ must be the same as the rank of $\mathbf{A}$: $rank(\mathbf{A})=rank(\mathbf{A}|\mathbf{b})$, while always $rank(\mathbf{A})\leq \min(n,m)$ (Strang, 2009). The system has a unique solution if, additionally, the rank of matrix $\mathbf{A}$ is equal to the number of unknowns (m) or equivalently $\mathbf{A}$ has full column rank (Strang, 2009). Otherwise, there are infinite solutions and the free parameters are equal to $m-rank(\mathbf{A})$ (Strang, 2009):

Therefore, a system with $\mathrm{n<m}$ is underdetermined and can never have a unique solution; even if it has full column rank it will be n, so always less than m. An underdetermined system can have either zero or infinite solutions. A system with $\mathrm{n>m}$ is an overdetermined system. If it has full column rank of m there is a unique solution, though in practical problems this is rare and there is no solution or infinite solutions (Strang, 2009).

It often happens that the linear system has no full column rank or even no solution (Higham, 2002; Strang, 2009). In the case of infinite solutions, there is some part of $\mathbf{x}$ that can be defined but there are still free parameters. When solving numerically and since all measurements are never perfect, an equation that cannot be solved is reached and the algorithm stops (Higham, 2002; Strang, 2009). This means that the residual error $\mathbf{e}=\|\mathbf{Ax}-\mathbf{b}\|$ is not zero and an exact solution $\mathbf{x}$ cannot be found.

To overcome all these issues and because for practical problems, we do need a solution; even in the case of an inconsistent system, the best available solution is accepted. This is a least-squares solution and is defined as finding a solution $\mathbf{x}_*$ so that the residual error $\mathbf{e}$ is minimised: $\min\|\mathbf{Ax}_* - \mathbf{b}\|$ (Higham, 2002; Strang, 2009). The least-squares solution $\mathbf{x}_*$ is also the preferred type of solution in the case of infinite solutions, which is usually the case for overdetermined systems (Strang, 2009). The next step is to find the inverse of $\mathbf{A}$ (in the ideal case of a full rank, square system) or more usually a suitable factorisation for $\mathbf{A}$ that will then provide $\mathbf{A}_*$ and thus solve for $\mathbf{x}_*$ (Strang, 2009) so that $\mathbf{x}_* = \mathbf{A}_*\mathbf{b}$.

A factorisation is when the original matrix $\mathbf{A}$ becomes the product of two or more special matrices, which makes evaluating $\mathbf{A}_*$ and solving for $\mathbf{x}_*$ relatively easier (Strang, 2009). A matrix $\mathbf{A}$ does not necessarily have only one unique factorisation but it does have a unique least square solution (Strang, 2009). For example, a square, full rank matrix has the inverse matrix and can also be factorised using other methods, but all result in the same unique solution.

Sometimes, matrix $\mathbf{A}$ is not square, but it does have full column (row) rank. Then, there is a ready formula for $\mathbf{A}_*$, which is the left (right) inverse of $\mathbf{A}$. For practical problems, matrix $\mathbf{A}$ usually does not have full column (row) rank and a more general factorisation is performed (Higham, 2002; Strang, 2009; Davis, 2013).

The factorisation methods that are most often used and are available in mathematical software (such as MATLAB, Mathematica, LAPACK) are presented next, along with a discussion on their numerical stability and their MATLAB implementation.

Choosing the best factorisation method, suitable for the type of matrix $\mathbf{A}$ and based on its dimension and rank, to provide a computationally efficient least-squares solution is as much an art as is a science and is highly specific to the problem at hand (Davis, 2013). Moreover, the use of the commonly available formulas in linear algebra textbooks, such as those for the left (right) inverse (Strang, 2009), is strongly discouraged for numerical computations (Higham, 2002; Davis, 2013; Foster *et al.*, 2013). For example, calculating the inverse matrix using the standard formula is not only more computationally expensive, but also much less stable and prone to numerical errors (Higham, 2002; Davis, 2013). Furthermore, estimating the rank of a non-square matrix in numerical computations is not always straightforward and can be computationally expensive (Strang, 2009; Davis, 2013).

For all these reasons, the decision was made to use the specialised software from (Davis, 2013) that selects the best available factorisation method and then always provides the least square solution at each iteration n at every time point t_i.

There is also another issue to consider: how "good" is the least square solution. Generally, any matrix $\mathbf{A}$ of size $n \times m$ has $rank(\mathbf{A}) = r \le \min(n,m)$. A rank of r means that there are r number of linearly independent rows and columns of A, the rest can be omitted (Strang, 2009). The closer the rank of matrix $\mathbf{A}$ is to $\min(n,m)$ the fewer equations need to be discarded and ideally, one would have $r = n = m$. Intuitively then, the closer r is to n (or m), then the better formulated is the system being solved in terms of its equations, unknowns, and constraints. For the specific problem of Inverse Simulation, the quality of the solution is determined by how well the calculated control input from Eq.(4.7) or Eq.(4.11) achieves the desired output when it is applied in the forward system.

The factorisation methods that are most often used and are available in mathematical software (such as MATLAB, Mathematica, LAPACK) are the following (Higham, 2002; Strang, 2009; Davis, 2013).

LU (lower – upper) decomposition is based on Gaussian elimination and factors a matrix as the product of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$, $\mathbf{A}$=$\mathbf{LU}$. The variation used in numerical computing is decomposing matrix $\mathbf{A}$ into an upper triangular matrix $\mathbf{U}$, a lower triangular matrix $\mathbf{L}$, and a permutation matrix $\mathbf{P}$, $\mathbf{PA} = \mathbf{LU}$. LU and its variations are used for square and non-square matrices.

If matrix $\mathbf{A}$ is symmetric and positive definite[11], then the Cholesky decomposition is faster than the LU and $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. Another option related to the Cholesky decomposition is LDL^T, which decomposes matrix $\mathbf{A}$ as $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^\mathrm{T}$, where $\mathbf{D}$ is a diagonal matrix.

A non-square matrix $\mathbf{A}$ with $\mathrm{n}>\mathrm{m}$ and full column rank is best factorised using the QR method, $\mathbf{A} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix.

If matrix $\mathbf{A}$ is rank deficient, $rank(\mathbf{A}) = r < \min(n,m)$, then the COD (complete orthogonal decomposition) can be used and $\mathbf{A} = \mathbf{U}\mathbf{R}\mathbf{V}^\mathrm{T}$, where $\mathbf{R}$ is and an upper triangular matrix, $\mathbf{U}$ and $\mathbf{V}$ have orthonormal columns.

Any matrix $\mathbf{A}$ can be factorised using SVD (singular value decomposition) and $\mathbf{A} = \mathbf{Q}_1\mathbf{\Sigma}\mathbf{Q}_2^\mathrm{T}$, where $\mathbf{\Sigma}$ is diagonal and $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are orthogonal. The SVD method is more time consuming than other alternative factorisations, but it is also the most reliable, especially for rank deficient problems (Strang, 2009; Davis, 2013).

To start with, for square systems MATLAB has the inv command for calculating the inverse of a square matrix and the backslash operator (\) for solving more general linear systems. Another option for non-square systems is the pinv command that calculates the pseudoinverse that acts as a partial replacement for the matrix inverse and provides a least-squares solution. The \ operator is always preferred instead of the inv command because the backslash calculation is quicker and has a smaller residual error by several orders of magnitude than inv. The backslash \ operator selects between four factorisations: LU, Cholesky, LDL^T, or QR. If one method fails, then it attempts to solve the system using another of these four factorisations and for non-square systems, it always uses the QR factorisation. The backslash operator is a powerful function but cannot guarantee a least-squares solution for underdetermined systems and rank deficient systems and its factorisation cannot be reused (Davis, 2013). Using the individual MATLAB functions for LU, Cholesky, LDL^T, QR, COD, or SVD factorisation is a difficult task

[11] **A** is positive definite if all its eigenvalues have a positive real part (Strang, 2009). From linear stability theory, this is an unstable system and thus undesirable in control problems (Ogata, 2008).

and between different versions of MATLAB, there also may be subtle differences in their implementation. The decomposition command in MATLAB creates a reusable matrix decomposition but still doesn't provide directly a least-squares solution.

The factorisation methods LU, Cholesky, LDL^T, QR, COD and SVD are backwards stable and thus are considered numerically stable, using the definition below (Higham, 2002).

A method for computing an approximate solution $\hat{y}$ to the problem $y = f(x)$ is backwards stable if for a small Δx, $\hat{y} = f(x + \Delta x)$ (Higham, 2002). The value of $\|\Delta x\|$ is the backwards error and its scale depends on the problem being solved (Higham, 2002). If a method is backwards stable, then it is also numerically stable in the sense that $\hat{y} + \Delta y = f(x + \Delta x)$ for small Δx and Δy (Higham, 2002). In this way, a numerically stable solution means that "the computed value of $\hat{y}$ scarcely differs from the value $\hat{y} + \delta y$ that would have been produced by an input $\hat{x} + \delta x$ that is scarcely different from the actual input x. In other words, $\hat{y}$ is almost the right answer for almost the right data" (Higham, 2002).

Another issue to consider is how sensitive the solution $\mathbf{x}$ or $\mathbf{x}_+$ is to small changes in matrix $\mathbf{A}$ or $\mathbf{b}$. For a square matrix $\mathbf{A}$, this is defined by the condition number $c = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$, where $\| \ \|$ is the matrix norm (Strang, 2009):

Ideally, small changes $\delta\mathbf{A}$ or $\delta\mathbf{b}$ should result in small changes in $\mathbf{x}$, which is why it is desirable that the condition number should be small. When solving the linear system $\mathbf{Ax} = \mathbf{b}$, errors enter in two ways.

They begin with an error $\delta\mathbf{A}$ or $\delta\mathbf{b}$ which is then carried over to the solution. The error $\delta\mathbf{b}$ depends on the measurements of $\delta\mathbf{b}$ and on the computer round off error (Higham, 2002; Strang, 2009). In addition to these two errors, the error $\delta\mathbf{A}$ also depends on the method by which $\delta\mathbf{A}$ is factorised to solve the system (Strang, 2009). Therefore, the factorisation method has an important role in determining the overall sensitivity to errors in the solution, that does not depend on the condition number of the problem. The factorisation methods in this Appendix are numerically stable as implemented in MATLAB, and thus have a positive effect in the solution – which is what was expected from a least square solution.

Finally, the condition number and the backwards error can also provide an upper bound for the forward error, which is defined as the difference between the exact solution and the computed solution (Higham, 2002). As a rule of thumb it is that $(\textit{forward error}) \leq c \times (\textit{backward error})$ (Higham, 2002). Estimating the backward error and the condition number is not trivial and the forward error can also be estimated using perturbation theory (Higham, 2002).

The main takeaway from this analysis is that even if the condition number is high, using an appropriate factorisation method provides a least-squares solution that minimises the approximation error and we can be reasonably certain that the rounding errors (such as those in $\delta\mathbf{A}$ or $\delta\mathbf{b}$) do not adversely affect the solution. This, combined with reducing data uncertainty in measurements and a well-formulated problem in terms of its physical qualities, all ensure a good solution.

Appendix H: Error and convergence tolerance

In numerical computations, there are three main sources of errors: truncation, rounding, and data uncertainty (Higham, 2002). Truncation errors result from the use of an approximation for a desired quantity (e.g. Taylor series or more generally an iterative solution of equations) (Higham, 2002). Rounding errors occur due to the way computers represent numerical values (Higham, 2002). Data uncertainty arises from errors in measurement and estimation (Higham, 2002). The assumption here is that the data have been properly acquired and processed. Rounding errors are generally unavoidable due to the finite precision arithmetic used in computers, available memory, and computational time constraints. Therefore, the truncation errors are the errors of interest in this section.

In general, for the desired x_d and its estimate $\tilde{x}$ (scalar or vector), the error can be defined as the absolute $E_a = |x_d - \tilde{x}|$ or the relative $E_r = \frac{E_a}{|x_d|}, |x_d| \neq 0$.

The relative error represents the qualitative side, how accurate is the estimate $\tilde{x}$ relative to the value of the desired x_d and is a measure of the number of significant digits that are correct (Higham, 2002).

The absolute error measures the total error between $\tilde{x}$ and x_d and it represents the quantitative side of the error. The absolute error depends on the magnitude of x_d and frames the result within the interval or tolerance tol: $E_a \leq tol \rightarrow \tilde{x} \in [x_d - tol, x_d + tol]$. It depends on the scale of $\tilde{x}$ and x_d and it should always be stated what it is an error of. If the desired value is not known, which is usually the case, then the error is calculated using the change in the approximate value from one iteration to the next: $\tilde{E}_a = |\tilde{x}_{n+1} - \tilde{x}_n|$.

Errors such as $E_a, \tilde{E}_a$ arise for example when a derivative is approximated using a Taylor expansion, when solving a least-squares problem or when estimating how close to zero the error is for the Newton-Raphson algorithm.