Numerical Efficiency of Inverse Simulation Methods applied to a Wheeled Rover

Thaleia Flessa Euan McGookin Douglas Thomson Kevin Worrall

Division of Aerospace Sciences, School of Engineering
University of Glasgow, UK, G12 8QQ

Abstract

A control method based on Inverse Simulation is applied to a four wheel rover. The method calculates the required inputs to achieve a desired, specified response; a trajectory in this case. Inverse Simulation considers the complete system dynamics to calculate the control input using an iterative, numerical Newton – Raphson scheme. Two methods for applying Inverse Simulation are presented, one based on a Differentiation scheme and one on Integration. The paper provides an insight into how the scheme formulation and selected parameters affect both methods' performance when applied to a rover. The selection of system outputs to control, their effect on each scheme's Jacobian, whether it is square or over-determined and the best method to factorize this Jacobian are investigated. The influence of the discretisation step and the convergence tolerance is also examined using two different sets for both schemes and in conjunction with the type of Jacobian used. The comparison is made in terms of the resulting trajectory, the execution time, and the quality of the calculated control input.

Keywords: inverse, simulation, control, navigation, model based, numerical, wheeled vehicle, rover

1 Introduction

A novel method based on Inverse Simulation is used for planetary rover guidance and control. Inverse simulation uses a mathematical model that is representative 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 and represents their time history. Inverse Simulation is a model based, numerical, iterative process where step changes in the various controls are applied until the predicted response matches the desired response (Thomson and Bradley, 2006). Applied to rover navigation, the desired response is a specified, safe trajectory to a goal destination (Worrall *et al.*, 2015a; Worrall *et al.*, 2015b). Inverse Simulation is a novel way of addressing the issue of given a specified, safe path, what are the required control inputs for the rover to reach the destination goal through this path. The method can be applied (a) in situ: given a series of waypoints or a defined trajectory, the rover can calculate the necessary control inputs or (b) offline: operators define the trajectory, the control inputs are calculated and then sent to the rover.

Applications for Inverse Simulation are predominantly within the flight dynamics domain and the application to rotorcraft flight control is a major area. In these particular cases Inverse Simulation is used to produce the required control signals for specific flight manoeuvres (Hess and Gao, 1993; Murray-Smith, 2000; Thomson and Bradley, 2006) and (Avanzini *et al.*, 2013) also introduces a predictive element. The method has also been applied to unmanned aerial vehicles (Murray-Smith and McGookin, 2015) and autonomous underwater vehicles (Murray-Smith *et al.*, 2008). Inverse Simulation has also been used as a model validation method (Murray-Smith, 2000; Thomson and Bradley, 2006). Previous research has demonstrated the potential for Inverse Simulation as a guidance and control method for wheeled rovers (Worrall *et al.*, 2015a; Worrall *et al.*, 2015b).

Planetary rover navigation so far has been achieved using a combination of non-, semi- and fully autonomous methods (Bajracharya *et al.*, 2008). The NASA Mars Exploration Rovers (MER) use a combination of three main driving modes with varying degrees of autonomy. The first mode involves the rover executing a sequence of commands to follow a defined course of waypoints towards specific goal coordinates. In this mode the rover only performs basic safety checks (Biesiadecki *et al.*, 2007). The second mode is semi-autonomous navigation during which the rover is given a set of waypoints towards specific goal coordinates and uses its on-board capabilities for hazard avoidance and for planning a path towards the goal. A special case is when the rover drives towards an area that is unknown to the operators (Biesiadecki *et al.*, 2007; Bajracharya *et al.*, 2008). In this case the rover has to choose the waypoints for a safe path towards the goal and then drive along this path; this is fully autonomous navigation. The third mode is visual odometry: the rover uses images from the on-board cameras to accurately estimate and update its position

(Cheng *et al.*, 2005; Biesiadecki *et al.*, 2007). A similar combination of these driving modes is used for the Curiosity rover and autonomous navigation is used to plot a safe path towards an area unknown to the operators (Bakambu *et al.*, 2012). The fully autonomous and visual odometry modes are used when the rover moves into areas that are not visible to the operators (Cheng *et al.*, 2005; Biesiadecki *et al.*, 2007; Bajracharya *et al.*, 2008). The developers of the ExoMars mission have addressed the issue of control, navigation and autonomy by including an element of autonomous control (Silva *et al.*, 2013) and by conducting field experiments (Woods *et al.*, 2014). Another issue is the computational complexity of the algorithms that are running on-board. The MER and Curiosity rovers use special space qualified and radiation hardened microprocessors whose computational capabilities have been exceeded by more than two orders of magnitude by the average desktop computer (Howard *et al.*, 2012). Furthermore, the navigation algorithms must also be tested using a wide range of parameters, which is best done using simulation (Madison *et al.*, 2007).

The paper investigates the selection of outputs to control and the parameters that affect the application and execution time of Inverse Simulation to a four wheeled rover. The control inputs are calculated from Inverse Simulation, applied to the rover and the resulting trajectory is compared with the desired.

2 Methodology of Inverse Simulation

Inverse Simulation has two main requirements: a desired 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. The desired trajectory is described using the model's outputs. There are two main implementations of Inverse Simulation for finding the control inputs given a desired output: Differentiation (Hess and Gao, 1993; Murray-Smith, 2000; Thomson and Bradley, 2006; Murray-Smith and McGookin, 2015) and Integration (Hess and Gao, 1993; Thomson and Bradley, 2006; Avanzini *et al.*, 2013; Worrall *et al.*, 2015a; Worrall *et al.*, 2015b). The basic framework for each is similar and uses a numerical Newton – Raphson algorithm; what differs is the method of convergence to the control signal. In Differentiation, a numerical differentiation scheme is used and the convergence is based on the system's state and output equations. In Integration, a numerical integration scheme is used and the convergence is based on whether the system's output matches the desired. An alternative approach to Inverse Simulation uses a modified version of the Integration scheme and search based optimisation (Lu *et al.*, 2008).

2.1 Implementation

A general non-linear system is used where $\boldsymbol{f}\in\mathbf{R}^{\mathbf{m}}$ are the state equations, $\boldsymbol{g}\in\mathbf{R}^{\mathbf{p}}$ are the output equations, $\boldsymbol{u}\in\mathbf{R}^{\mathbf{q}}$ is the control input vector, $\boldsymbol{x}\in\mathbf{R}^{\mathbf{m}}$ is the state variable vector and $\boldsymbol{y}\in\mathbf{R}^{\mathbf{p}}$ is the output vector. The desired output to control is $\boldsymbol{g_d}\in\mathbf{R}^{\mathbf{p}}$.

$$\dot{\mathbf{x}}=\mathbf{f}(\mathbf{x},\mathbf{u}),\ \mathbf{y}=\mathbf{g}(\mathbf{x},\mathbf{u}) \tag{1}$$

For the Differentiation method, (1) is discretised N times over a time interval T, where dt is the discretisation step. The unknowns in (2) are the states $\boldsymbol{x}$ and the input $\boldsymbol{u}$ at t_i. The known variables are the desired output $\boldsymbol{g_d}$ and the states, control and output from the previous discretisation step t_{i-1}. Then, the functions $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$ in (3) are defined to find the values of input $\boldsymbol{u}$ and the states $\boldsymbol{x}$ for the given output $\boldsymbol{g_d}$. The system in (3) is solved using the Newton - Raphson method to update $\boldsymbol{u}$ and $\boldsymbol{x}$ until their values are such that $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$ are both equal to zero within a certain tolerance. The updated equations are in (4) and $\boldsymbol{J}$ is the Jacobian of the system from (3).

$$\begin{aligned}&\frac{\mathbf{x}(t_i)-\mathbf{x}(t_{i-1})}{dt}=\mathbf{f}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right),\ dt=t_i-t_{i-1}\\&\mathbf{y}(t_i)=\mathbf{g}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right)\end{aligned} \tag{2}$$

$$\begin{aligned}&\mathbf{F}_1=\mathbf{f}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right)-\frac{\mathbf{x}(t_i)-\mathbf{x}(t_{i-1})}{dt}\\&\mathbf{F}_2=\mathbf{g}\left(\mathbf{x}(t_i),\mathbf{u}(t_i)\right)-\mathbf{g}_d(t_i)\end{aligned} \tag{3}$$

$$\begin{bmatrix}\mathbf{x}_n\\\mathbf{u}_n\end{bmatrix}(t_i)=\begin{bmatrix}\mathbf{x}_{n-1}\\\mathbf{u}_{n-1}\end{bmatrix}-\mathbf{J}^{-1}\cdot\begin{bmatrix}\mathbf{F}_1(\mathbf{x}_{n-1},\mathbf{u}_{n-1})\\\mathbf{F}_2(\mathbf{x}_{n-1},\mathbf{u}_{n-1})\end{bmatrix}(t_i) \tag{4}$$

For the Integration approach the state and output equations from (1) are again discretised and dt is the discretisation step. The state equations are integrated at t_i. An error function between the current output and the desired $\boldsymbol{g_d}$ is defined in (6). Equation (6) is solved for $\boldsymbol{u}$ using the Newton – Raphson method and the iterative relationship in (7). $\boldsymbol{J_e}$ is the Jacobian of the error function $\boldsymbol{f_e}$ or equivalently the Jacobian of the system outputs when perturbing the inputs.

$$\begin{aligned}&\mathbf{x}(t_i)=\int_{t_{i-1}}^{t_i}\dot{\mathbf{x}}(\tau_i)\,d\tau+\mathbf{x}(t_{i-1})\\&\mathbf{y}(t_i)=\mathbf{g}\left(\mathbf{x}(t_i),\mathbf{u}(t_{i-1})\right)\end{aligned} \tag{5}$$

$$\mathbf{f_e}=\mathbf{g}\left(\mathbf{x}(t_i),\mathbf{u}(t_{i-1})\right)-\mathbf{g_d}(t_i) \tag{6}$$

$$\mathbf{u}_n(t_{i-1})=\mathbf{u}_{n-1}-\mathbf{J}_e^{-1}(\mathbf{x}_{n-1},\mathbf{u}_{n-1})\cdot\mathbf{f_e}(\mathbf{x}_{n-1},\mathbf{u}_{n-1}) \tag{7}$$

2.2 Numerical Properties

Both implementations use a Jacobian and care must be taken when trying to find its inverse or a suitable

factorization. For the Differentiation method, from Eq. (4) the dimension of the Jacobian $\boldsymbol{J}$ is $[m+p]\times[m+q]$. For the Integration method from Eq. (7) the dimension of the Jacobian $\boldsymbol{J_e}$ is $[p]\times[q]$. If there is an equal number of inputs and outputs ($p=q$), then the Jacobian is a square matrix. If however, the number of inputs and outputs is not equal, then factorization methodologies such as LU, QR or Cholesky decomposition or the Moore–Penrose pseudoinverse (Strang, 2009; Davis, 2013), can be used. When there are more outputs than inputs ($p>q$), this results in an over-determined system and the pseudoinverse or factorization can be used. In that case, the calculated outputs are a least-square fit to the desired outputs and not necessarily a good one. For this reason, systems where the number of inputs is equal to or greater than the number of outputs are preferred candidates (Hess and Gao, 1993; Murray-Smith, 2000; Thomson and Bradley, 2006; Murray-Smith *et al.*, 2008).

Each approach has advantages and disadvantages, which are usually identified as the following (Hess and Gao, 1993; Murray-Smith, 2000; Thomson and Bradley, 2006; Lu *et al.*, 2008): (a) The Integration method can use any representative model of the system. Differentiation requires both the states and the outputs and any change in the model results in a reformulation of the algorithm. Therefore, the Differentiation method is more time consuming to set up and maintain, whereas for Integration the model can be modified more easily, (b) The Integration method has a convergence rate that is up to an order of magnitude larger than that of the Differentiation method but it is generally more stable; what is gained in flexibility and stability, is lost in computing time. The numerical properties of Inverse Simulation have been examined mostly when the method is applied to flight dynamics (Hess and Gao, 1993; Thomson and Bradley, 2006; Lu *et al.*, 2008). It was observed that there are oscillations in the response of the uncontrolled states (constraint oscillations) (Thomson and Bradley, 2006; Lu *et al.*, 2008). However, these oscillations depend more on the dynamical properties of the system, its uncontrollable states and zero dynamics as well as on the discretisation step *dt*, rather than on the method used. They are also significantly reduced when a larger *dt* is used (Lu *et al.*, 2008). Also from (Thomson and Bradley, 2006) it was observed that there are low amplitude, high frequency oscillations superimposed on the calculated control input. These oscillations are due to several reasons (Hess and Gao, 1993; Murray-Smith, 2000; Thomson and Bradley, 2006; Lu *et al.*, 2008): redundancy issues, non-square Jacobian and multiple solutions, several local minima of the error function from (4), (7). The oscillations increase when the discretisation step *dt* is too small, as it could excite the uncontrollable states (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 (Lu *et al.*, 2008).

3 Rover Model and Trajectory Generation

Inverse Simulation requires a mathematical model of the system and a desired response, which is a trajectory. First, a path to the destination is determined as a series of waypoints. This information provides the desired trajectory for the Inverse Simulation, which in turn generates the required guidance commands (control inputs) to follow the trajectory (Worrall *et al.*, 2015a; Worrall *et al.*, 2015b).

3.1 Rover Model

The model of the rover has been presented in (Worrall, 2010; Worrall *et al.*, 2015a; Worrall *et al.*, 2015b) and has been experimentally validated (Worrall, 2010). It is briefly described here for clarity. Each side has two wheels and the wheels at each side provide the same torque input. The dynamics are described by (8), where $\boldsymbol{v}$ is the state velocity vector (9) in the local body frame, $\boldsymbol{\eta}$ is the velocity vector in the global frame and $\boldsymbol{\tau}$ is the input vector (10).

$$\begin{bmatrix}\dot{\mathbf{v}}\\ \dot{\boldsymbol{\eta}}\end{bmatrix}=\begin{bmatrix}\mathbf{M}^{-1}\{\boldsymbol{\tau}-\mathbf{C}(\mathbf{v})\mathbf{v}-\mathbf{D}(\mathbf{v})\mathbf{v}-\mathbf{g}(\boldsymbol{\eta})\}\\ \mathbf{J}_t(\boldsymbol{\eta})\mathbf{v}\end{bmatrix} \quad (8)$$

$$\mathbf{v}=\begin{bmatrix}u & v & w & p & q & r\end{bmatrix}^T \quad (9)$$

$$\boldsymbol{\tau}=\begin{bmatrix}X & Y & Z & K & M & N\end{bmatrix}^T \quad (10)$$

In (9) u, v, w are the surge, sway and heave velocities and p, q, r are the roll, pitch and yaw rates respectively. In (10) X is the surge, Y is the sway and Z is the heave force, K is the roll, M is the pitch and N is the yaw moment. X and N are controllable by two inputs: one torque at each side. The remaining forces and moments are the unmatched dynamics.

3.2 Trajectory Generation

The trajectory is represented as a series of waypoints, each defined by an x-y coordinate with a common origin. A path between each waypoint and the next is calculated, with the robot stopping at each waypoint to turn on the spot to achieve the desired orientation and then move again.

The distance and time to travel between each waypoint is calculated assuming a constant velocity between stages with initial and final acceleration and deceleration transients: the constant forward speed is 0.1ms^{-1}, analogous to that of operating rovers, and the rotational velocity is 0.1rads^{-1}. At each waypoint a check is made to determine 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 is achieved. The path from one waypoint to the next is defined by specifying the acceleration as a 7th order polynomial function of time and is based on that presented in (Thomson and Bradley, 2006; Worrall *et al.*, 2015a). A 7th order polynomial has the benefit of producing smooth trajectory profiles with high order, continuous derivatives. The output is the acceleration time history, which is then integrated to provide the velocities and the displacements. The result is a continuous time history of acceleration, speed and distance between each successive waypoint that fully describes the rover's position and orientation; namely the elements of $\boldsymbol{v}$, $\boldsymbol{\eta}$.

4 Application Results

A series of waypoints are first defined and then a trajectory between them is generated as in Section 3. It is assumed that these waypoints represent a safe, feasible path. Inverse Simulation calculates the control inputs for each trajectory. Then these inputs are applied to the system and it is checked whether the resulting trajectory matches the desired. The following test trajectories were selected. The Long Distance test (Figure 1, 400s) involves several pose changes and will be used as a benchmark to show how the errors built up over time and to compare the different parameters. The Figure of Eight test (Figure 2, 175s) is used to demonstrate a complex path with multiple, sharp turns and how the method copes with successive pose changes.

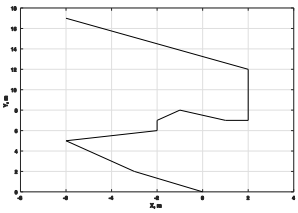

Figure 1. Long Distance.

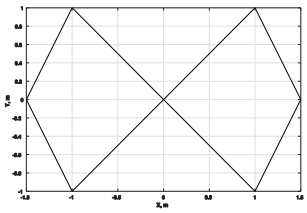

Figure 2. Figure of Eight.

The simulation parameters that need to be assigned values are: discretisation step *dt*, convergence tolerance *tol*, torque input initial estimate, maximum number of iterations for the Newton-Raphson algorithm. For *dt*, the timestep of the motors and the need to adequately follow the system are taken into account (Worrall *et al.*, 2015a). The rover starts from rest (zero motor torque). Here a very small value is set for the initial estimate. The number of iterations is set to ensure convergence without increasing the execution time.

The assessment criteria are the following: (a) mean error and standard deviation between the actual and the desired position x (integrated from u), (b) mean error and standard deviation between the actual and the desired heading angle θ (integrated from r), (c) calculation time, an important measurement for any control algorithm. The position and heading angle represent the rover position in space and hence how wells it follows the desired trajectory.

Table 1. Simulation Parameters.

Parameter	*Set 1*	*Set 2*
dt (s)	0.01	0.05
tol	5×10^{-7}	5×10^{-5}
initial control estimate (Nm)	2.5×10^{-7}	
maximum iterations	30	
MATLAB version	2014b, 64 bit	
hardware	Core 2 Duo T9300, 2.50 GHz, 4 GB RAM	

4.1 Selection of inputs, outputs and Jacobian inversion.

The two controllable outputs are the surge X and the yaw moment N. This is equivalent to controlling the surge and yaw velocities and so the desired outputs are u_d, r_d. There are two control inputs, one torque per side (τ_{left}, τ_{right}). For the Integration method, there are two inputs and two outputs and so the size of the Jacobian (Worrall *et al.*, 2015a; Worrall *et al.*, 2015b) in Eq. (7) is 2x2. For the Differentiation method, it was observed during the initial simulations that including as an additional output to control the sway velocity v, the overall results are significantly improved without sacrificing greatly in execution time. There are three desired outputs: u_d, r_d as before and v_d, which is set to zero. The sway velocity v is not matched dynamically to the actuators of the system and therefore cannot be directly controlled. It is however strongly coupled to u and r (Worrall, 2010) and this interaction can provide indirect control of sway and act as an additional constraint. For the Jacobian, Eq. (4), only the controllable states u, r and also v are taken into account and so its size is 6x5. The remaining states for (4) are estimated after convergence at each t_i. This is an over-determined system and to ensure that the solution is always a least square solution a suitable factorization method is used to find the pseudo-inverse of $\boldsymbol{J}$ and solve (4). There are several methods to find the Jacobian inverse. Table 2 shows the methods in MATLAB that are examined (Davis, 2013). Each method from Table 2 is tested using the Long Distance test and the first set of parameters.

Table 3 shows the results for the Differentiation scheme. The *backslash* method fails because $\boldsymbol{J}$ is (column) rank deficient; this is expected because the outputs to control are u, v and r and v is strongly coupled to u and r. 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 suitable for rank deficient systems. Table 4 shows the results for

the Integration scheme. The *backslash* method is the best for the error and execution time and is the one selected. Integration is slower, which is in line with previous observations (Section 2.2).

Table 2. Inversion Methods.

Method	*Comments*
inv() - built-in function	Suitable only for square systems of full rank, can be very inaccurate.
pinv() - built-in function	Suitable for non-square systems, calculates the Moore–Penrose pseudoinverse using singular value decomposition (SVD).
\ (backslash operator) - built-in function	Suitable for square or over determined systems with full column rank, fast, accurate. Factorization cannot be reused. Suggested MATLAB method.
FACTORIZE (Davis, 2013) - additional package, acts as a wrapper for the built-in MATLAB functions	Selects the most suitable factorization method from LU decomposition, Cholesky decomposition, QR decomposition, SVD (singular value decomposition), COD (complete orthogonal decomposition). Suitable for square, rank deficient and over/under determined systems.

Table 3. Inversion: Differentiation (with sway), Long Distance (set 1).

	\	*pinv(J)*	*factorize(J)*
mean position x error (m)	-	0.00247	0.00082
σ position error	-	0.00211	0.00080
mean heading θ error (rad)	-	0.00123	0.00053
σ heading error	-	0.00076	0.00056
execution time (s)	-	34.12	55.45

Table 4. Inversion: Integration (without sway), Long Distance (set 1).

	\	*inv(J)*	*factorize(J)*
mean position x error (m)	0.00082	0.00082	0.00082
σ position error	0.00040	0.00040	0.00040
mean heading θ error (rad)	0.00073	0.00073	0.00073
σ heading error	0.0013	0.0013	0.0013
execution time (s)	79.19	376.89	317.84

4.2 Scheme comparison

From Tables 3, 4 the main difference between the two schemes is the calculation time. Differentiation performs slightly better for the heading. Figure 3 shows the control inputs generated for Differentiation and Figure 4 for Integration.

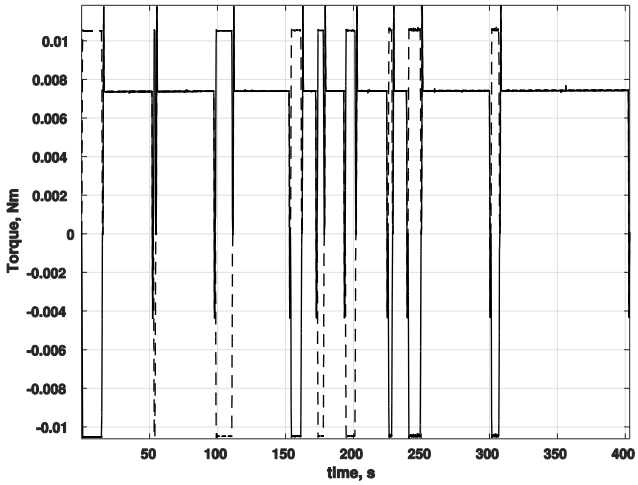

Figure 3. Differentiation Control, Long Distance (set 1).

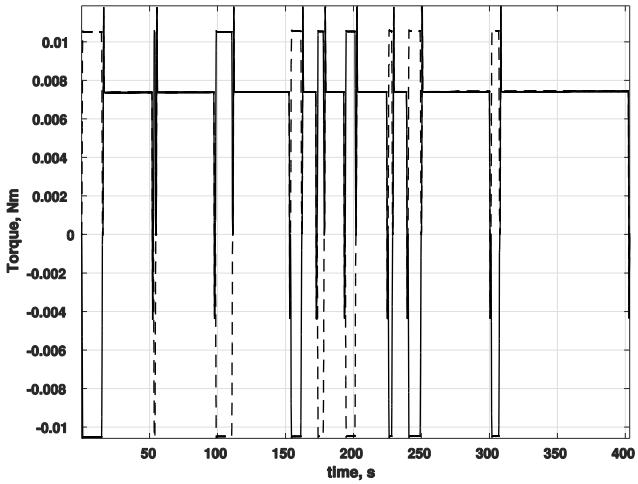

Figure 4. Integration Control, Long Distance (set 1).

The left side control is signified by the solid line and the right side by the dashed line. The left and right signals are symmetrical when the rover is moving forward (e.g. at 100s), which is expected since each side is controlled by one input. When the heading changes there is a momentary spike in the input. The control inputs from Integration are smoother, e.g. between 100 – 150s and at 250s in Figure 3 and 4. The oscillations from Differentiation have a small magnitude and high frequency and are due to the fact that the scheme uses an over-determined system which may have multiple solutions (Section 2.2).

Table 5 shows the results for the Long Distance test, set 2: *dt* is increased and so is the convergence tolerance. The execution time is significantly reduced, and Integration is now faster than Differentiation. Integration performs slightly better in terms of the

position error and Differentiation is better for the heading error. Compared with Tables 3, 4, the standard deviation of both the position and the heading error is larger; the rover has some sharper deviations from the desired position and heading. Figure 5 shows the calculated control input from Differentiation. By increasing the *dt* to 0.05s, the high frequency, low amplitude oscillations in the control input decrease significantly.

Table 5. Long Distance (set 2).

	Differentiation (with sway)	*Integration (without sway)*
mean position x error (m)	0.00468	0.00450
σ position error	0.00314	0.00196
mean heading θ error (rad)	0.00276	0.00736
σ heading error	0.00262	0.00700
execution time (s)	12.19	8.72

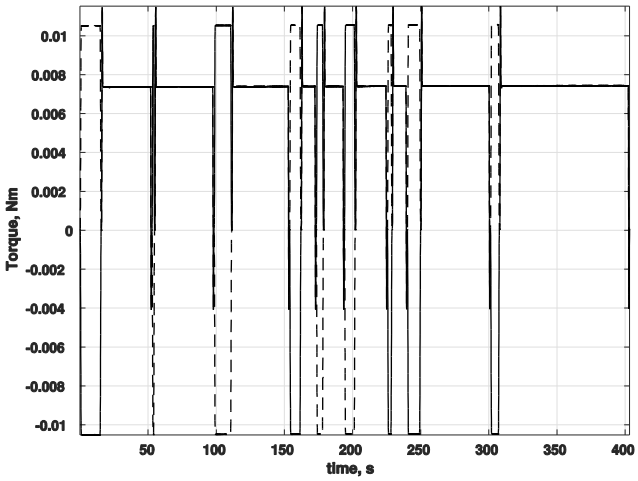

Figure 5. Differentiation Control, Long Distance (set 2).

Table 6 shows the results for the Figure of Eight test. Both methods perform similarly but Differentiation is faster and slightly better for the standard deviation of the heading error.

Figure 6 and 7 show the errors of *u*, *v* and *r* after the Newton–Raphson algorithm for Eq. (3) and (6) has converged at each t_i. The desired value of *v* is set to zero and the desired values of *u* and *r* are the same for both methods. For Differentiation, the error in *r* deviates about 10^{-16} rad/s from zero, whereas for Integration it deviates about 10^{-4} rad/s from zero. The *v* error deviates 10^{-7} m/s from zero and the *u* error deviates 10^{-8} m/s from zero for Differentiation. For Integration, the *v* error deviates 10^{-4} m/s and the *u* error 10^{-5} m/s from zero. At Figure 7, when the heading changes there is a spike in the errors. At Figure 6, the much smaller errors are due to *v* used as an additional output. This effect is particularly evident when comparing the errors in *r*.

Table 6. Figure of Eight (set 1).

	Differentiation (with sway)	*Integration (without sway)*
mean position x error (m)	0.00034	0.00060
σ position error	0.00036	0.00048
mean heading θ error (rad)	0.00202	0.00203
σ heading error	0.09501	0.11632
execution time (s)	25.94	40.45

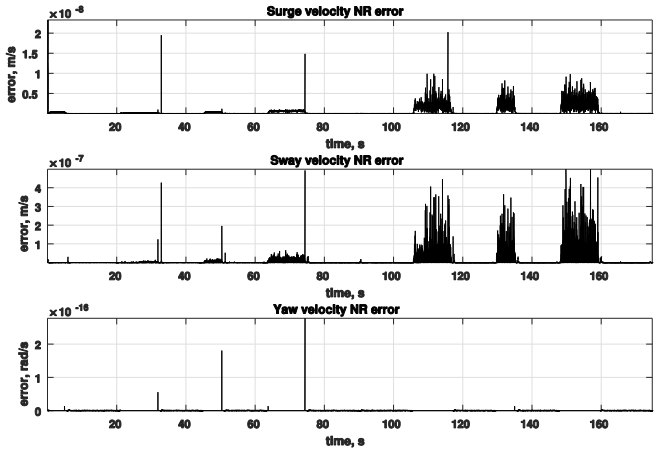

Figure 6. Differentiation NR Errors Fig of Eight (set 1).

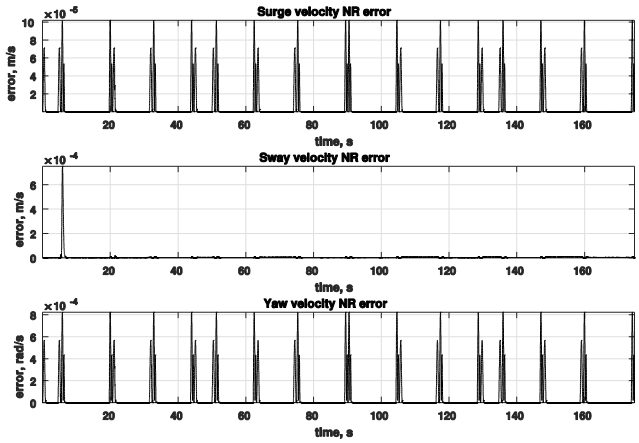

Figure 7. Integration NR Errors Fig. of Eight (set 1).

From Tables 3 - 6, Integration exhibits bigger errors for the heading angle. When the *dt* and the tolerance are small enough or when there are no abrupt orientation changes, this is negligible. As *dt* and the tolerance increase and the trajectory requires sharp heading changes, this difference becomes more important. This can be seen in the failure of Integration for the Figure of Eight test using parameter set 2. The errors in *r* are not corrected, the calculated control in (7) increases and the condition number of $\boldsymbol{J_e}$ by 22.5s (total time 175s) is infinite; the Jacobian is ill-conditioned and cannot be factorized.

4.3 Effect of sway velocity

To reduce the error in r and conversely in θ, the sway velocity v is used as an additional output for Integration. Then, $\boldsymbol{J_e}$ in Eq. (7) has a size of 3x2: three outputs (u, v, r), two inputs and the *factorize* command is used. Table 7 shows the results for the Figure of Eight test, set 2. Both methods produce similar results, however Integration is slower and still has a larger error and standard deviation for the heading. Nonetheless, the usage of v has here a positive effect and enables Integration to converge. When using v in Integration for the Long Distance test, set 2, the execution time increases to 29.71s compared to 8.72s (Table 5) without any error improvements. For the Long Distance test, set 1 (Table 4), the time is greatly increased to 791.99s.

Table 7. Figure of Eight Test (set 2).

	Differentiation (with sway)	*Integration (with sway)*
mean position x error (m)	0.00384	0.00332
σ position error	0.00259	0.00231
mean heading θ error (rad)	0.01047	0.02298
σ heading error	0.00259	0.27941
execution time (s)	5.30	20.71

Table 8. Differentiation: Long Distance (set 1).

	Differentiation (with sway)	*Differentiation (without sway)*
mean position x error (m)	0.00082s	0.002404
σ position error	0.00080	0.001960
mean heading θ error (rad)	0.00053	0.003154
σ heading error	0.00056	0.002864
execution time (s)	55.45	56.26

Table 8 shows the Differentiation results for the Long Distance test (set 1) with and without using v. Without v, $\boldsymbol{J}$ in (4) is square (4x4). The errors increase by two orders of magnitude and the execution time is almost the same: the method converges slower and with larger errors. Compared with Integration (Table 4), the inclusion of v has a greater effect on Differentiation. This confirms previous results, that Integration is more stable. Here, Differentiation performs slightly better but requires an over-determined system and specialized handling.

5 Conclusions

The selection of outputs to control, their effect on the size of the Jacobian and the best factorization method were examined. A square Jacobian is used for Integration and an over-determined for Differentiation. The schemes were compared for varying dt and convergence tolerance. A small dt results in high frequency, low amplitude oscillations in the control input from Differentiation. To remove these, the dt was increased. The effect of sway velocity v, which is strongly coupled with u, r but not directly controlled, was examined. For Differentiation, using v as an output is beneficial from the start. Integration performed well for both parameter sets for the Long Distance test without v. For the Figure of Eight test for a dt of 0.05s and tolerance 5×10^{-5}, including v was necessary. It is worth noting that this test is not a realistic trajectory and is used to test the method's limits. A dt of 0.01s and a tolerance of 5×10^{-7} produce the best results, with increased calculation time. For simplicity and overall stability, the Integration scheme is more appropriate. For decreased execution time, Differentiation is preferred, at the expense of slightly larger position errors and an over-determined system. In all cases, the control inputs from Inverse Simulation where within operational limits.

Acknowledgments

Research supported by grant EPSRC/1369575 from the UK Engineering and Physical Sciences Research Council (EPSRC).

References

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

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

J. N. Bakambu, C. Langley, G. Pushpanathan, W. J. MacLean, and R. Mukherji. Field trial results of planetary rover visual motion estimation in Mars analogue terrain. *Journal of Field Robotics*, 29(3), 413–425, 2012. doi: 10.1002/rob.21409.

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

Y. Cheng, M. W. Maimone, and L. Matthies. Visual Odometry on the Mars Exploration Rovers. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, Waikoloa, HI, USA, pages 903–910, 2005. doi: 10.1109/ICSMC.2005.1571261.

T. A. Davis. Algorithm 930: FACTORIZE: An Object-Oriented Linear System Solver for MATLAB. *ACM Transactions on Mathematical Software*, 39(4), 1–18, 2013. doi: 2491491.2491498.

R. A. Hess and C. Gao. A Generalized Algorithm for Inverse Simulation Applied to Helicopter Maneuvering Flight. *Journal of the American Helicopter Society*, 38(4), 3–15, 1993. doi: 10.2514/3.20732.

T. M. Howard, A. Morfopoulos, J. Morrison, Y. Kuwata, C. Villalpando, L. Matthies, and M. McHenry. Enabling continuous planetary rover navigation through FPGA stereo and visual odometry. In *2012 IEEE Aerospace Conference*, Big Sky, MT, USA, pages 1–9, 2012. doi: 10.1109/AERO.2012.6187041.

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

R. Madison, A. Jain, C. Lim, and M. W. Maimone. Performance characterization of a rover navigation algorithm using large-scale simulation. *Scientific Programming*, 15(2), 95–105, 2007. doi: 10.1155/2007/638280.

D. J. Murray-Smith. The inverse simulation approach: a focused review of methods and applications. *Mathematics and Computers in Simulation*, 53(4–6), 239–247, 2000. doi: 10.1016/S0378-4754(00)00210-X.

D. J. Murray-Smith, L. Lu, and E. W. McGookin. Applications of inverse simulation to a nonlinear model of an underwater vehicle. In *Summer Simulation Multi-Conference 2008 - Grand Challenges in Modelling and Simulation*, Edinburgh, Scotland, 2008.

D. J. Murray-Smith and E. W. McGookin. 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), 2700–2717, 2015. doi: 10.1177/0954410015586842.

N. Silva, R. Lancaster, and J. Clemmet. 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, 2013.

G. Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley MA, 4th ed. 2009.

D. G. Thomson and R. Bradley. Inverse simulation as a tool for flight dynamics research - Principles and applications. *Progress in Aerospace Sciences*, 42(3), 174–210, 2006. doi: 10.1016/j.paerosci.2006.07.002.

M. Woods, E. Tidey, B. Van Pham, L. Simon, R. Mukherji, B. Maddison, G. Cross, A. Kisdi, W. Tubby, G. Visentin, and G. Chong. Seeker-Autonomous Long-range Rover Navigation for Remote Exploration. *Journal of Field Robotics*, 31(6), 940–968, 2014. doi: 10.1002/rob.21528.

K. J. Worrall, D. G. Thomson, E. W. McGookin, and T. Flessa. Autonomous Planetary Rover Control using Inverse Simulation. In *13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2015)*, Noordwijk, The Netherlands, 2015a.

K. J. Worrall. *Guidance and search algorithms for mobile robots: application and analysis within the context of urban search and rescue*. PhD Thesis, University of Glasgow.

K. J. Worrall, D. G. Thomson, and E. W. McGookin. Application of Inverse Simulation to a wheeled mobile robot. In *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, Queenstown, New Zealand, pages 155–160, 2015b. doi: 10.1109/ICARA.2015.7081140.