Numerical Stability of Inverse Simulation Algorithms Applied to Planetary Rover Navigation

Thaleia Flessa, *Student Member, IEEE,* Euan McGookin, *Member, IEEE,* and Douglas Thomson

***Abstract*— Extending the navigational capability of planetary rovers is essential for increasing the scientific outputs from such exploratory missions. In this paper a navigation method based on Inverse Simulation is applied to a four wheel rover. The method calculates the required control inputs to achieve a desired, specified response. Here this is a desired trajectory defined as a series of waypoints. Inverse Simulation considers the complete system dynamics of the rover to calculate the control input using an iterative, numerical Newton – Raphson scheme. The paper provides an insight into the numerical parameters that affect the performance of the method. Also, the influence of varying the timestep and the convergence tolerance is examined in terms of the quality of the calculated control input and the resulting trajectory, as well as the execution time. From this analysis a set of parameters and recommendations to successfully apply Inverse Simulation to a rover is presented.**

I. Introduction

In this study a novel method based on Inverse Simulation is used for autonomous rover guidance and navigation. Inverse simulation uses a mathematical model that is representative of the system and calculates the control inputs necessary for the rover 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 [1]. Applied to rover navigation, the desired response is a trajectory or path to a goal destination [2], [3].

The operation of a planetary rover presents several challenges: time delays, terrain uncertainty, limited communication bandwidth and high latency, inability to repair hardware, system degradation [4]. To extract the maximum scientific return, the rover must be able to efficiently and safely navigate the terrain. The navigation capabilities are essential to the overall success of the mission [4], [5] and Inverse Simulation addresses this particular issue.

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 maneuvers [1, 6, - 8] and [9] also introduces a predictive element. The method has also been applied to unmanned aerial vehicles [10] and autonomous underwater vehicles [11]. Inverse Simulation has also been used as a model validation method [1], [8]. Previous research has demonstrated the potential for Inverse Simulation as a guidance and control method for wheeled rovers [2], [3].

Planetary rover navigation so far has been achieved using a combination of non-, semi- and fully autonomous methods [12]. 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 specified course of waypoints towards specific goal coordinates. In this mode the rover only performs basic safety checks [13]. 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 [12], [13]. 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 [12], [13]. The third mode is visual odometry: the rover uses images from the on-board cameras to accurately estimate and update its position [13], [14]. 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 [15]. The fully autonomous and visual odometry modes are used when the rover moves into areas that are not visible to the operators [12 - 14]. The developers of the ExoMars mission have addressed the issue of navigation and autonomy by including an element of autonomous control within the guidance system [16] and by conducting field experiments to test the current long-range navigation capabilities [17].

This paper focuses on the application of Inverse Simulation to a four wheeled rover as a method of guidance and navigation and investigates the parameters that affect its successful application. In Section II, the Inverse Simulation method is presented and in Section III the mathematical model of the rover and the trajectory generation are shown. Section IV presents the results of the method applied to the rover and a discussion of the parameters that affect its application. Finally, the conclusions are in Section V.

II. Inverse Simulation Methodology

There are two main approaches in implementing Inverse Simulation: Differentiation [1, 6 - 8, 10] and Integration [1 -

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

T. Flessa is a PhD Student at the Division of Aerospace Sciences, School of Engineering, University of Glasgow, Glasgow (e-mail: t.flessa.1@research.gla.ac.uk).

E. McGookin (e-mail: Euan.McGookin@glasgow.ac.uk) and D. Thomson (e-mail: Douglas.Thomson@glasgow.ac.uk) are both Senior Lectures at the Division of Aerospace Sciences, School of Engineering, University of Glasgow, Glasgow.

3, 6 - 8, 9]. The basic framework for each is similar and both use 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. Both methods use a Jacobian and care must be taken when trying to find its inverse or a suitable factorization. For this reason systems where the number of inputs is equal to or greater than the number of outputs are preferred candidates, as these result in square or over-actuated systems [1, 6 - 8]. An alternative involves derivative – free methods, such as [11], [18] which employ Nedler – Mead optimization. However this is outside the scope of this paper.

A. Implementation of Inverse Simulation

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 $\mathbf{y}\in\mathbf{R}^{\mathbf{p}}$ is the output vector. The desired output 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 discretized N times over a time interval T into the form shown in (2), where *dt* is the discretization step.

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

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 discretization step t_{i-1}.

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 until the values of $\boldsymbol{u}$ and $\boldsymbol{x}$ 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 in (3).

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

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

For the Integration approach the state and output equations from (1) are again discretized and *dt* is the discretization step. The state equations are integrated at t_i.

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

An error function between the current output and the desired $\boldsymbol{g_d}$ is defined in (6).

$$\mathbf{f}_{\mathbf{e}}=\mathbf{g}\left(\mathbf{x}(t_i),\mathbf{u}(t_{i-1})\right)-\mathbf{g}_{\mathbf{d}}(t_i) \tag{6}$$

Equation (6) is solved for $\boldsymbol{u}$ using the Newton – Raphson method and the iterative relationship (7), where $\boldsymbol{J_e}$ is the Jacobian of the error function $\boldsymbol{f_e}$ or equivalently the Jacobian of the system outputs when perturbing the inputs.

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

B. Numerical Properties of Inverse Simulation

Each approach has advantages and disadvantages, which are usually identified as the following [1, 6 - 8, 18]: (a) The Integration method can use any representative model of the system as long as the outputs and inputs remain the same. 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 [1, 6, 7, 18]. The authors of [1] examine the stability properties of the method in this context. When using the Differentiation method, it has been observed that there are oscillations in the response of the uncontrolled states (constraint oscillations) [1]. 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 [18].

Also from [1] it has been observed that there are low amplitude, high frequency oscillations superimposed on the calculated control input when using the Integration method. Overall, the high frequency, low amplitude oscillations in the control are due to several reasons [1, 6 - 8, 18]: redundancy issues, non-square Jacobian and multiple solutions, several local minima of the error function from (7). These low amplitude, high frequency oscillations are increased when the discretization step *dt* is too small, as it could excite the uncontrollable states [18]. Nonetheless, a relatively small *dt* can have a positive effect because it captures the changes in the system dynamics [18] and this may reduce or even remove them [7], [18]. It is a case of compromising between adequately following the system as it evolves over time and possibly exciting the uncontrollable states. A different approach uses the two timescale method [1], [18].

III. Rover Model and Trajectory Generation

Inverse Simulation used for rover navigation 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 [2], [3]. The method can be applied in situ: given a series of waypoints or a defined trajectory, the rover can calculate the necessary control inputs or offline: operators define the trajectory, the control inputs are calculated and then sent to the rover.

A. Rover Model

The model of the rover has been presented in [2, 3, 19] and has been experimentally validated [19]. It is briefly described here for completeness. Each side has two wheels and the wheels at each side provide the same torque input. The dynamics are described by (8), where $\boldsymbol{\nu}$ 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).

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

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

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

In (9) u, v, w are the surge, sway and heave velocities respectively 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, the remaining forces and moments are the unmatched dynamics.

B. 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 0.1 m/s, analogous to that of operating rovers [5], and the rotational velocity is 0.1 rad/s. 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 [1], [3]. 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 vector $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$.

IV. Inverse Simulation Results

A series of waypoints is first defined and then a trajectory between them is generated as in Section III. 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 Forward (11s) and Rotate on the Spot (12s) tests are the basic movements. The Long Distance test (Fig. 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.

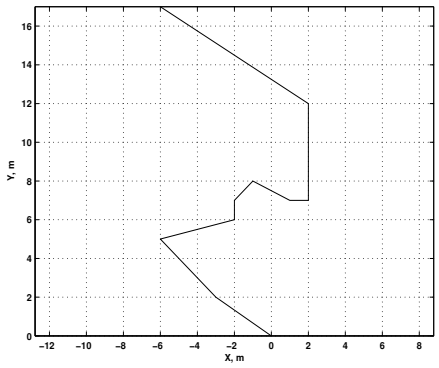

Figure 1. Long Distance test trajectory

For the Integration method [2, 3] the outputs to control are the surge velocity u and the rotational velocity r. There are two inputs and two outputs and so the Jacobian of the error (7) has a size of 2x2 and its rank is 2. For the Differentiation method, it was observed during the initial simulations that including as an additional output to control the sway velocity, the overall results are improved. There are three desired outputs: u_d, r_d as before and v_d, which is set to zero. The sway velocity v is strongly coupled to u and r [19] and adding it as a desired output that is set to zero acts as an additional constraint for u and r. For the Jacobian (4), only the controllable states u, r and additionally v are taken into account and so its size is 6x5 and its rank is 5. The remaining states for (4) are estimated after the scheme has converged 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) [20].

For each method the following parameters need to be assigned values: dt, the tolerance for the convergence of Newton-Raphson, the initial starting values for the control inputs and the maximum number of iterations. The dt and the tolerance are varied, as these have the greatest effect on the results. Their influence on the time required to generate the control inputs is also examined. For dt, the physical properties of the system must also be taken into account. Here, this is the timestep of the motors that provide the torque input and dt should be in the range of 0.05s [3]. When changing dt, the trajectory generation between the waypoint changes, to adapt to the different timestep and generate an appropriate time history. The convergence tolerance affects the overall quality of the calculated input: too small and equations (4), (7) do not converge, too large and the

accumulated error increases. The other parameters will be the same throughout for both methods. The rover starts from rest, i.e. the motor torque is zero. Here a very small value is assumed for the initial torque estimate: 2.5×10^{-7} Nm for each side. The maximum number of iterations is 30, sufficient to converge but not big enough to overly increase the execution time. Both methods are tested on a Core 2 Duo T9300, 2.50 GHz, 4 GB RAM system running MATLAB 2014b, 64 bit,

A. *Baseline Simulation*

The initial parameters are dt = 0.01s and a tolerance of 5×10^{-7}. Table I shows the errors accumulated for the Long Distance test. The main difference is the calculation time. Fig. 1 shows the control inputs generated for Differentiation and Fig. 2 for Integration. The left side control is signified by the solid line and the right side by the dashed line (same for Fig. 4, 5). The right side control signals are symmetrical to those of the left 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. around 150, 250s in Fig. 1, 2. The oscillations from Differentiation have a small magnitude and high frequency and are due to the fact that scheme uses a redundant system which may have multiple solutions, in line with previous observations discussed in Section II.

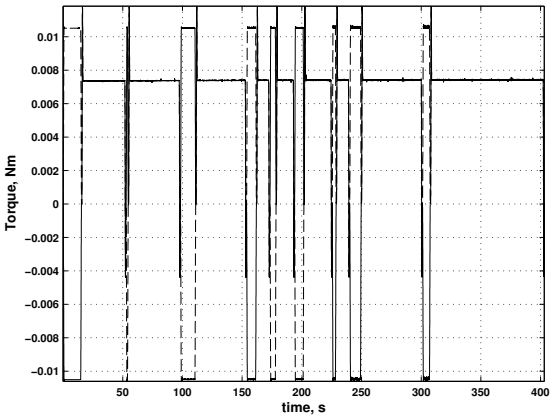

Figure 2. Differentiation: Control Input, Long Distance (0.01s, 5×10^{-7})

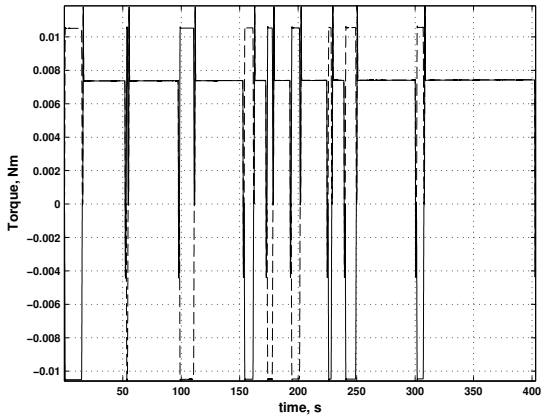

Figure 3. Integration: Control Input, Long Distance (0.01s, 5×10^{-7})

TABLE I. LONG DISTANCE (1)

Long Distance: *dt* 0.01, tolerance 5×10^{-7}		
	Differentiation	***Integration***
mean position error (m)	-0.0009	0.0008
σ position error	0.0008	0.0004
mean heading error (rad)	0.0003	0.0003
σ heading error	0.0007	0.0015
execution time (s)	57.33	117.08

B. *Effect of varying the time increment dt*

For the Long Distance test, between dt = 0.001 and dt = 0.01 the results are very similar in terms of error, with the Integration performing slightly better. The main difference is an increase in execution time, the Integration requires now 1788.79s, which is more than three times that of Differentiation at 501.45s and a scale of magnitude larger compared to Table I. For dt = 0.05 the results are in Table II, there are slightly bigger errors than in Table I but still small.

TABLE II. LONG DISTANCE (2)

Long Distance: *dt* 0.05, tolerance 5×10^{-7}		
	Differentiation	***Integration***
mean position error (m)	-0.0058	0.0044
σ position error	0.0038	0.0023
mean heading error (rad)	-0.0002	0.0038
σ heading error	0.0035	0.0093
execution time (s)	11.65	15.45

In Table II, the execution time is much reduced, which is important for using the method on-line. Integration performs better in terms of the position error and Differentiation slightly belter for the heading error. The position error for Integration is in both cases in the range of 10^{-3} while for Differentiation this changes from 10^{-4} to 10^{-3}. Moreover, the standard deviation of both the position and the heading error is larger, which means that during the movement, the rover has some sharper deviations for the desired position and heading than before (Table I). By increasing dt to 0.05 the high frequency, low amplitude oscillations in the control input decrease for both methods, Fig. 4, 5. The effect of the excitation of the uncontrollable states is reduced and the rover follows the desired trajectory accurately.

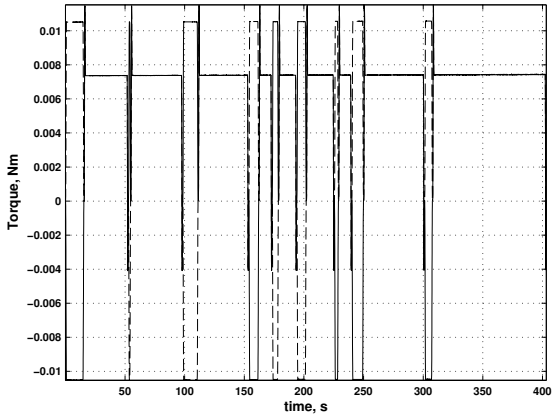

Figure 4. Differentiation: Control Input, Long Distance (0.05s, 5×10^{-7})

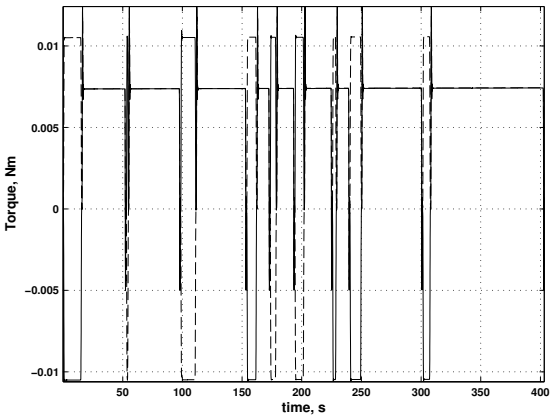

Figure 5. Integration: Control Input, Long Distance (0.05s, 5×10^{-7})

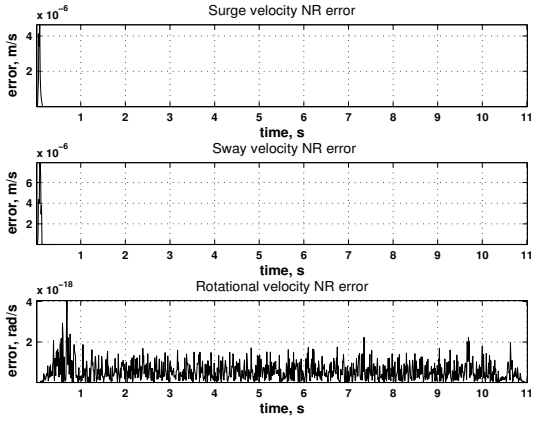

Figure 6. Differentiation: NR Errors, Forward (0.01s, 5×10^{-5})

C. *Effect of varying the tolerance*

For tolerance 5×10^{-5} and 5×10^{-4}, the Integration method produces results only for the Forward test and fails for the Rotate on the Spot and Long Distance tests. The errors between the actual and desired u and r are too big and in an attempt to correct the errors the calculated control increases greatly and $\boldsymbol{J_e}$ in (7) becomes singular. This can be seen from the condition number of $\boldsymbol{J_e}$ for the Rotate on the Spot test (total time 12.0s). For tolerance 5×10^{-5} it starts from 8, by 1.19s the condition number is 1484387.11 and by 1.2s the condition number is infinite. The condition number of $\boldsymbol{J_e}$ for 5×10^{-4} starts from 8, by 0.76s is 3062065.29 and by 0.77s it is infinite. For both cases, $\boldsymbol{J_e}$ becomes ill-conditioned. Table III shows the results for 5×10^{-5}, Forward test.

TABLE III. FORWARD 1M

Forward: *dt* 0.01, tolerance 5×10^{-5}		
	Differentiation	***Integration***
mean position error (m)	-0.00003	0.0009
σ position error	0.00001	0.0003
mean heading error (rad)	-0.00001	-0.0002
σ heading error	0.000008	0.00004
execution time (s)	1.68	3.21

Fig. 6 and 7 show the errors of surge velocity u, sway velocity v and rotational velocity r after (3) and (6) have converged respectively. The desired values of v and r are zero. For Integration, the v error is not used for the scheme's convergence; Differentiation uses v as an additional output. For Differentiation the r error deviates about 10^{-18} rad/s from zero, whereas for Integration it deviates about 10^{-3} rad/s from zero. The v error deviates 10^{-6} m/s from zero and the u error deviates 10^{-6} m/s from zero for Differentiation. For Integration the v error deviates 10^{-5} m/s from zero and the u error 10^{-4} m/s, both at the trajectory's start and end. Since v is strongly coupled with u and r, including it as an output has a corrective effect on the actually controllable states. This effect is particularly evident in the rotational velocity r and looking back at Table I, II and III Integration exhibits bigger errors for the heading when dt is increased and a bigger standard deviation overall. When the tolerance is low enough or when there are no changes in the orientation, this small difference is negligible. As the tolerance increases and the trajectory requires pose changes, it becomes more important to accurately follow the desired outputs.

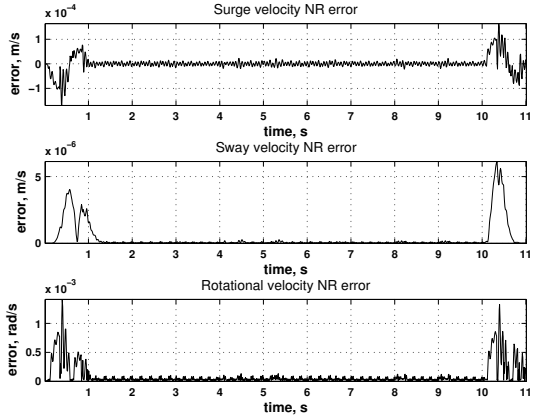

Figure 7. Integration, NR Errors, Forward (0.01s, 5×10^{-5})

D. *Combined effect of dt and tolerance*

Increasing the tolerance to 5×10^{-5} and the dt to 0.05s affects positively the Inverse Simulation: Integration produces results for the Turn and Long Distance tests for tolerance 5×10^{-5} (Table IV) and 5×10^{-4}. This behavior is due to the fact that if dt is too small and the tolerance increases, the errors add up to quickly between the waypoints and are not corrected. By increasing the dt while also increasing the tolerance, the errors are corrected. Referring back to Section II, it is a case of compromising between adequately following the system (a small dt), possibly exciting the uncontrollable states and selecting a tolerance that matches the speed by which the system evolves over time. Moreover, the execution time is now comparable between the two methods and the Differentiation method continues to exhibit smaller errors for r, due to the corrective effect of v. Fig.8 shows the trajectory followed by the rover for the case of dt = 0.05s, tolerance 5×10^{-5}. Both methods match the desired very well.

TABLE IV. LONG DISTANCE (3)

Long Distance: *dt* 0.05, tolerance 5×10^{-5}		
	Differentiation	***Integration***
mean position error (m)	-0.0047	0.0042
σ position error	0.0032	0.0023
mean heading error (rad)	-0.0006	0.0046
σ heading error	0.0036	0.0094
execution time (s)	11.64	12.78

Comparing Table I, II and IV the errors are larger in Table II. Integration exhibits similar errors in Table I, II, IV whereas Differentiation varies, for example the mean position error is -0.0009m in Table I and -0.0047m in Table IV.

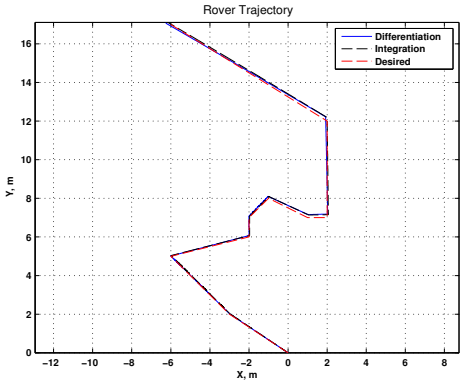

Figure 8. Long Distance Test (0.05s, 5×10^{-5})

V. Conclusion

Inverse Simulation has been successfully applied to a rover and the parameters that affect its application were investigated. These results show that Inverse Simulation can now be extended to rovers, which indicates that the method behaves consistently amongst a number of different systems. The discretization step *dt* and tolerance significantly affect the method's numerical properties and were examined here. A small *dt* results in high frequency, low amplitude oscillations in the control input. These oscillations are also due to the presence of uncontrollable states but are more evident if the system is over-determined, as is here for Differentiation. A compromise between a *dt* that can adequately follow the system as it evolves without exciting the uncontrollable states is needed. Moreover, if the *dt* is too small, it introduces numerical errors which are not corrected if there is a large tolerance. This was seen when the tolerance was increased: Integration failed but Differentiation produced results. This is due to the fact that Differentiation uses as an additional output the sway velocity *v*, an uncontrollable state that is strongly coupled with the controllable states *u*, *r*. This benefit results in an over-determined system that requires special handling when solving the Newton – Raphson equations. For the rover, a *dt* of 0.01s and a tolerance of 5×10^{-7} produce the best results, however the calculation time is large particularly for Integration A *dt* of 0.05 and a tolerance of 5×10^{-5} produces good results with position errors in the mm range, heading errors of less than 0.004 rad and the execution time is significantly reduced. Overall, the Differentiating method is faster than Integration and produces good results, at the expense of slightly larger position errors and the usage of an over-determined system. The Integration method is slower but the associated errors are more consistent. Finally, for both cases the calculated control inputs where within the rover's operational limits.

References

[1] D. G. Thomson, R. Bradley, "Inverse simulation as a tool for flight dynamics research—Principles and applications," *Prog. in Aerospace Sci.*, vol. 42, no. 3, pp. 174-210, May 2006.

[2] K. Worrall, D. G. Thomson, and E. W. McGookin, "Application of Inverse Simulation to a Wheeled Mobile Robot," *6th Int. Conf. on Automation, Robotics and Applicat. (ICARA)*, Queenstown, New Zealand, 2015, pp. 155 – 160.

[3] K. Worrall, D. G. Thomson, E. W. McGookin, and T. Flessa., "Autonomous Planetary Rover Control Using Inverse Simulation," *13th Symp. on Advanced Space Technologies in Robotics and Automation (ASTRA)*, Noordwijk, The Netherlands, May 2015.

[4] M. B. Quadrelli, et al., "Guidance, Navigation, and Control Technology Assessment for Future Planetary Science Missions," *Journal of Guidance, Control, and Dynamics,* vol. 38, no. 7, pp. 1165-1186, 2015.

[5] T. Flessa, E. W. McGookin, and D. G. Thomson, "Taxonomy, Systems Review and Performance Metrics of Planetary Exploration Rovers", *13th Int. Conf. on Control, Automation, Robotics and Vision (ICARCV)*, Marina Bay Sands, Singapore, Dec. 2014, pp. 1554-1559.

[6] R.A. Hess, and C. Gao, "A generalized algorithm for inverse simulation applied to helicopter manoeuvring flight," *Journal of the American Helicopter Society*, vol. 38, no. 4, pp. 3-15, 1993.

[7] S. Rutherford, and D. G. Thomson, "Improved methodology for inverse simulation," *Aeronautical Journal*, vol. 100, no. 2149, pp. 79–86, 1996.

[8] D. J. Murray-Smith, "The inverse simulation approach: a focused review of methods and applications," *Mathematics and Computers in Simulation*, vol. 53, no. 4 – 6, pp. 239-247, 2000.

[9] G. Avanzini, D. G. Thomson, and A. Torasso. "*Model Predictive Control Architecture for Rotorcraft Inverse Simulation*," Journal of Guidance, Control, and Dynamics, vol. 36, no. 1, pp. 207-217, 2013.

[10] D. J. Murray-Smith, and E. W. McGookin, "A case study involving continuous system methods of inverse simulation for an unmanned aerial vehicle application," *Proc. of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 229, no. 14, pp. 2700-2717, 2015.

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

[12] M. Bajracharya, M. W. Maimone, and D. Helmick, "Autonomy for Mars Rovers: Past, Present and Future," *IEEE Computer*, vol.41, no.12, pp. 44-50, Dec. 2008

[13] J. J. Biesiadecki, C. Leger, M. W. Maimone , "Tradeoffs Between Directed and Autonomous Driving on the Mars Exploration Rovers," *Int. Journal of Robotics Research*, vol. 26, no 1, pp. 91-104, 2007.

[14] Y. Cheng, M. W. Maimone, and L. Matthies, "Visual odometry on the Mars exploration rovers - a tool to ensure accurate driving and science imaging," *IEEE Robot. Automat. Mag.*, vol.13, no.2, pp.54-62, June 2006

[15] G. Webster. (2012, Aug. 27). *NASA's Mars Curiosity Debuts Autonomous Navigation* [Online]. Available: http://mars.jpl.nasa.gov/msl/news/whatsnew/index.cfm?FuseAction=ShowNews&NewsID=1514.

[16] N. Silva, R. Lancaster, and J. Clemmet, "ExoMars Rover Vehicle Mobility Functional Architecture and Key Design Drivers", *12th Symp. on Advanced Space Technologies in Robotics and Automation (ASTRA)*, Noordwijk, The Netherlands, May 2013

[17] M. Woods, et al., "Seeker - Autonomous Long-range Rover Navigation for Remote Exploration," *Journal of Field Robotics, vol. 31*, no. 6, pp. 940–968, 2014.

[18] L. Lu, D. J. Murray-Smith, and D. G. Thomson, D. G., "Issues of numerical accuracy and stability in inverse simulation," *Simulation Modelling Practice and Theory*, vol. 16, no. 9, pp. 1350–1364. 2008.

[19] K. Worrall, "Guidance and Search Algorithms for Mobile Robots: Application and Analysis within the Context of Urban Search and Rescue," Ph.D. dissertation, Dept. of Electronics and Electrical Engineering, University of Glasgow, Glasgow, UK, 2008.

[20] T. A. Davis, "Algorithm 930: FACTORIZE: an object-oriented linear system solver for MATLAB," *ACM Transactions on Mathematical Software*, vol. 39, no. 4, pp. 28:1 – 28:18, 2013.