# Analysis of *Terrain-Adaptive Wheel Speed Control on the Curiosity Mars Rover: Algorithm and Flight Results*

Robert Vandemark Diane Ngo

2020
November

# Contents

# Abstract

Throughout recent decades, the topic of space exploration is desirable towards scientists and researchers. Mars is the nearest planet that is possible to explore in great detail. There are many challenges pertaining to exploring unknown territory, through long distance, and difficulty of remotely operating robots. NASAs approach to exploring and researching Mars was through the Sample Return Rover, created in 1997 by the JPL Jet Propulsion Laboratory. The scope of this project focuses on the Mars Rover, its wheel-slip detection (traction control), and path-planning towards a desired object.

# Introduction

The Mars Rover is a Sample Return Rover designed by the JPL NASA Laboratory. It consists of a chassis with a central revolute joint on both the left and right sides of the robot. The central revolute joint is connected to two links (on each side), for the front and rear wheels. Each respective link is connected to another link downward that is connected to a wheel. The front links are revolute so that the robot can turn left and right. Note that the original paper is based off of the Curiosity Rover, which has six wheels. The focus on this paper will be on the Sample Return Rover, which has four wheels.

Primitive designs of this robot had a large amount of damage to the rovers wheels. This wheel damage reduced the longevity of the Mars Rover mission by a great amount. NASA had to counteract this damage through researching the cause of the wheel damage. The rover did not properly avoid terrain obstacles nor did it deal with traction loss. The goal for this project is to recreate the traction control algorithm and simulate the results.

The algorithm for the traction control is a velocity-based algorithm. One wheel on the rover will be rotating much faster than the others. The traction control system then applies a brake to that wheel to reduce its slip and then reducing wheel slip. NEED TO FIX !

# Algorithms

## Introduction

The *Traction Control Algorithm* (TCA) implementation for the *Curiosity Mars Rover* (CMR) is one that heavily utilizes geometry and a rigid-body model assumption, while also placing an emphasis on relying on minimal assumptions/data about the environment and avoiding computationally complex calculations on input data from sensors, cameras, etc. There were various reasons as to why some of these assumptions/choices were made and why the agreed upon implementation was deemed the best choice for the given scenario. [4] Figure 3.1 shows a testbed version of the CMR.

Figure 3.1: The Scarecrow Testbed Rover, a Rover Kinematically Similar to the *Curiosity Mars Rover* [4]

## Design Limitations and Considerations

It is important to note that the need for this TCA was not discovered until after the flight system was actively engaging in its mission on the Martian surface, while ground control was observing telemetry of its use. Therefore, it was impossible to make any physical modifications to the CMR,

only software could be remotely flashed to it.

However, there are still ramifications to having this additional routine run on the CMR. The limited computational resources available to do so must be considered carefully, so as to not interfere with existing processes being ran, and the implementation chosen has to have enough resources to perform the task it needs to as well. The team responsible for solving the problem at hand clarified some of these issues and how it restricted their choices for strategies to solve the problem. For example, the rover "does not include force or torque sensors on the mobility subsystem, nor can it measure slip with high enough frequency to be able to react to it." [4]

Some of the characteristics and assumptions of the CMR and its TCA implementation are discussed in the following sections.

## Ackermann Steering Model

Modeling vehicles using the Ackermann steering model is a common practice, including for the CMR. Even thought it adds slightly more complex geometric modeling, the mechanical steering system can be implemented relatively easily, and there are benefits to doing so.

Figure 3.2 illustrates a basic vehicle that is modeled using Ackermann steering, where the left image has the wheels positioned such that the vehicle will move straight, while the right image would cause the vehicle to turn counter-clockwise. Important to note is that when in a turning position, the front wheels are not turned to the same angle as one another, because of the nonzero distance between them. They are positioned such that the direction that both wheels are pointing are normal to a common center point called the *Instantaneous Center of Curvature* (ICC), so that when the vehicle turns, the left and right wheels follow two different circular arcs, but both of their centers are at the ICC.

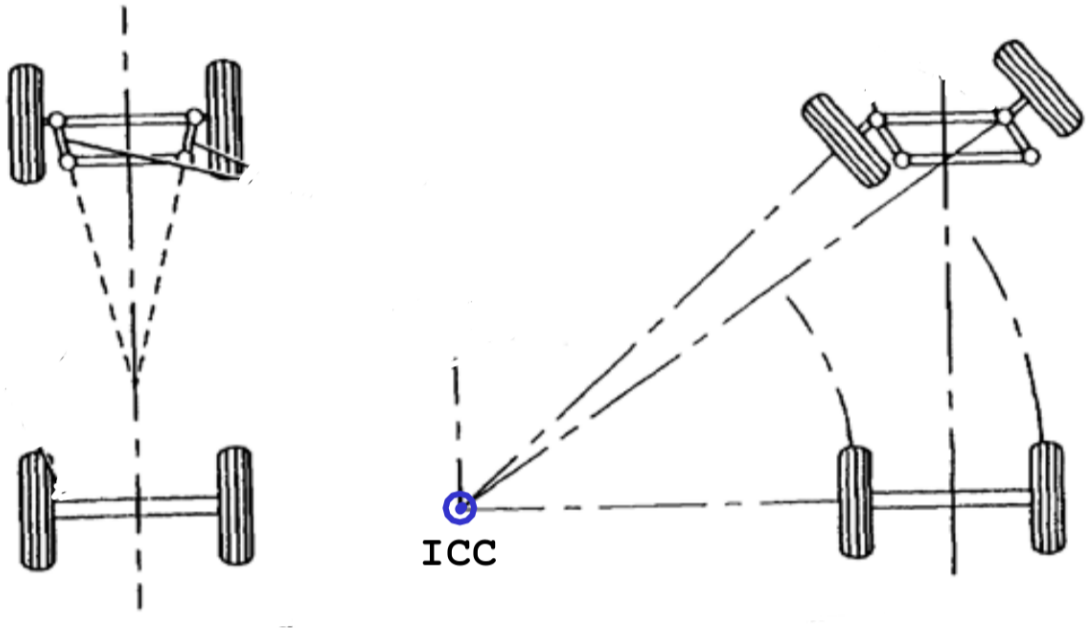


Figure 3.2: A Vehicle with Ackermann Steering Going Straight (Left) and in a Turning Position(Right)

The benefit to this is that, assuming all the wheels' velocities are coordinated and in an ideal world, at no point while traveling in these flat, circular arcs will a wheel slip. Less slippage means less unpredictable behavior, such as a wheel slipping over a rough surface. However, the assumption that the CMR would only be traversing terrain that could be modeled as flat is what led to the need for the TCA, as excessive slippage occurred due to the reality that the terrain was non-negligible. As one wheel would traverse over a rock and then back down, it traveled a longer distance.

The Ackermann steering model by itself does not account for this extra distance and this caused the wheel slippage, which damaged them at an alarming rate. This is how the TCA can be used to improve the model, so that when rough terrain is being traversed, the wheel(s) doing so can be sped up accordingly.

## Rigid-Body Model Assumption

The rigid-body model assumption says that a body of some sturdy material can be assumed to maintain it shape, and that a force/torque that it can expect to encounter will not deform it to a degree that needs to be accounted for. This is not true in reality, but in scenarios like these, the deformation can be negligible.

Making this assumption can be extremely beneficial when implementing an algorithm like TCA because it heavily simplifies the math needed to represent vectors in different coordinate frames.

## Coordinate Frame Notation

Coordinate frames are useful in robotics for the purpose of keeping track of the robot positions, orientations, and velocities in space. These frames can be translated from a reference to another

frame, such as from the base robot frame to each individual leg frame. Robotics often uses Cartesian coordinates, so the coordinate frames are created with an origin, x, y, and z axes.

To be able to show the relative position and orientation of one frame to another, geometric relationships between them need to be found. A rotation matrix can be used to show the relative orientation between two frames. For example, the orientation of the 1st frame $o_1x_1y_1z_1$ with respect to the base frame $o_0x_0y_0z_0$ can be written with the rotation matrix:

$$R_1^0 = [x_1^0|y_1^0|z_1^0] \tag{3.1}$$

One way to compute the rotation matrix is with the entries being in terms of angle $\theta$. The second way is to project the 1st frame on the 0 frame which uses the dot product. The dot products with respect of the 1st frame onto the 0 frame can be shown as

$$x_1^0 = \begin{bmatrix} x_1 \cdot x_0 \\ x_1 \cdot y_0 \\ x_1 \cdot z_0 \end{bmatrix}, y_1^0 = \begin{bmatrix} y_1 \cdot x_0 \\ y_1 \cdot y_0 \\ y_1 \cdot z_0 \end{bmatrix}, z_1^0 = \begin{bmatrix} z_1 \cdot x_0 \\ z_1 \cdot y_0 \\ z_1 \cdot z_0 \end{bmatrix} \tag{3.2}$$

There are three basic rotation matrices- about the x-axis, y-axis, and z-axis. With respect to angle $\theta$, these rotation matrices are defined in Equations 3.3, 3.4, 3.5. These 3 rotations are also called the roll($\phi$), pitch ($\theta$), and yaw($\psi$) matrices. The order of rotation is through yaw (x,($\psi$)),pitch (y,($\theta$)), and roll (z,($\phi$).

$$R_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.3}$$

$$R_{y,\theta} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.4}$$

$$R_{z,\theta} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

After finding both the position and orientation of a frame, they can be put into a homogeneous transformation matrix. This can comprise of a series of rigid motions, which are pure translations and pure rotations only. The transformation matrix consists of the rotation between the frames (top 3x3), the distance (right 3x1 column vector), and the bottom row being matrix identity to allow a series of multiplications. Figure 3.6 shows the homogeneous transformation matrix.

$$H_n^0 = \begin{bmatrix} R_n^0 & d_n^0 \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

The rover, like any robot, has coordinate frames to define its position and rotation matrices. Figure 3.3 shows these frames from the left side of the robot. A top view is also provided as Figure 3.4.

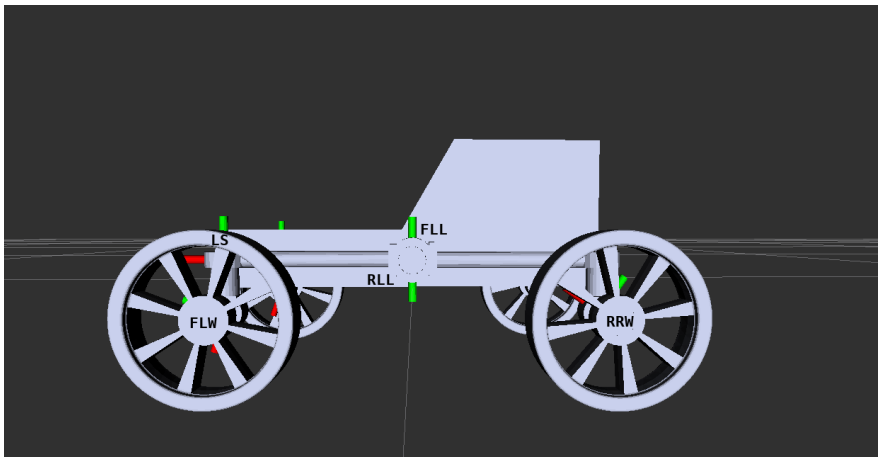


Figure 3.3: Side View of the Robot Model and its Coordinate Frames

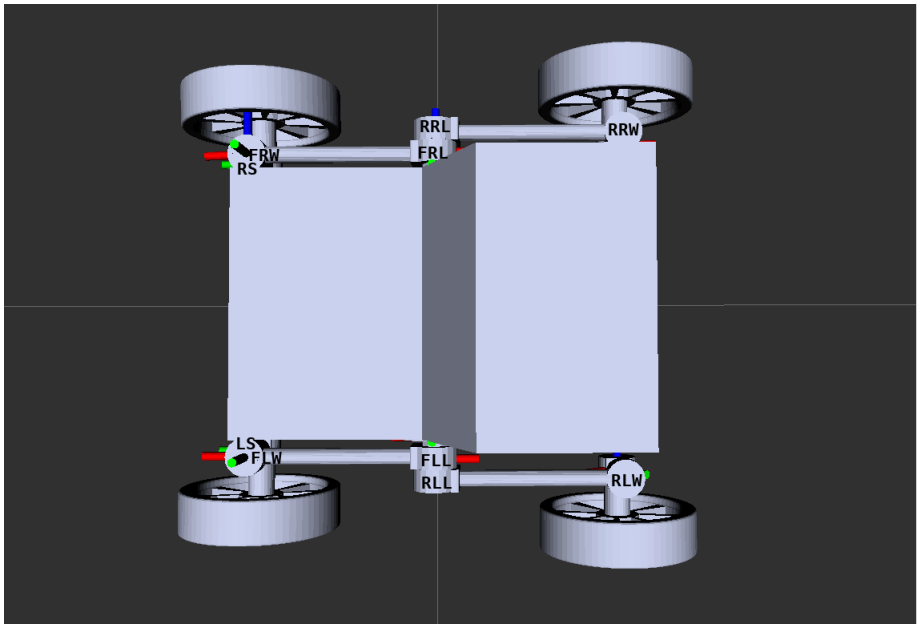


Figure 3.4: Side View of the Robot Model and its Coordinate Frames

A summary of the acronyms for the coordinate frames used in Figures 3.3 and 3.4 are seen below in Table 3.1.

| Label | Name |
|---|---|
| FLL | Front Left Leg |
| FRL | Front Right Leg |
| RLL | Rear Left Leg |
| RRL | Rear Right Leg |
| FLW | Front Left Wheel |
| FRW | Front Right Wheel |
| RLW | Rear Left Wheel |
| RRW | Rear Right Wheel |
| LS | Left Steering |
| RS | Right Steering |

Table 3.1: Table of Acronyms for Model Axes

The rover model can be drawn to show its geometry in a simple manner. Depicted from its side, there is the pivot joint from the chassis in the center. Two legs are connected to the the center pivot joint. The front leg also has a revolute joint at its end to be able to steer the front wheels.

The difference between this model and the Curiosity Rover is that the Curiosity Rover has a third leg, which can be defined as the rocker joint and bogie joints. The legs are designed much differently as compared to the early model of the Sample-Return Rover. The SRR only has the center pivot joint.

As seen in Figure 3.5,the diagram is of the left side of the rover, and the front is facing left. Wheel centers are denoted with $A_i$. The lengths of each leg are shown with $l_i$. Angles of specific areas are denoted by either $\psi$, k, or $\delta$.

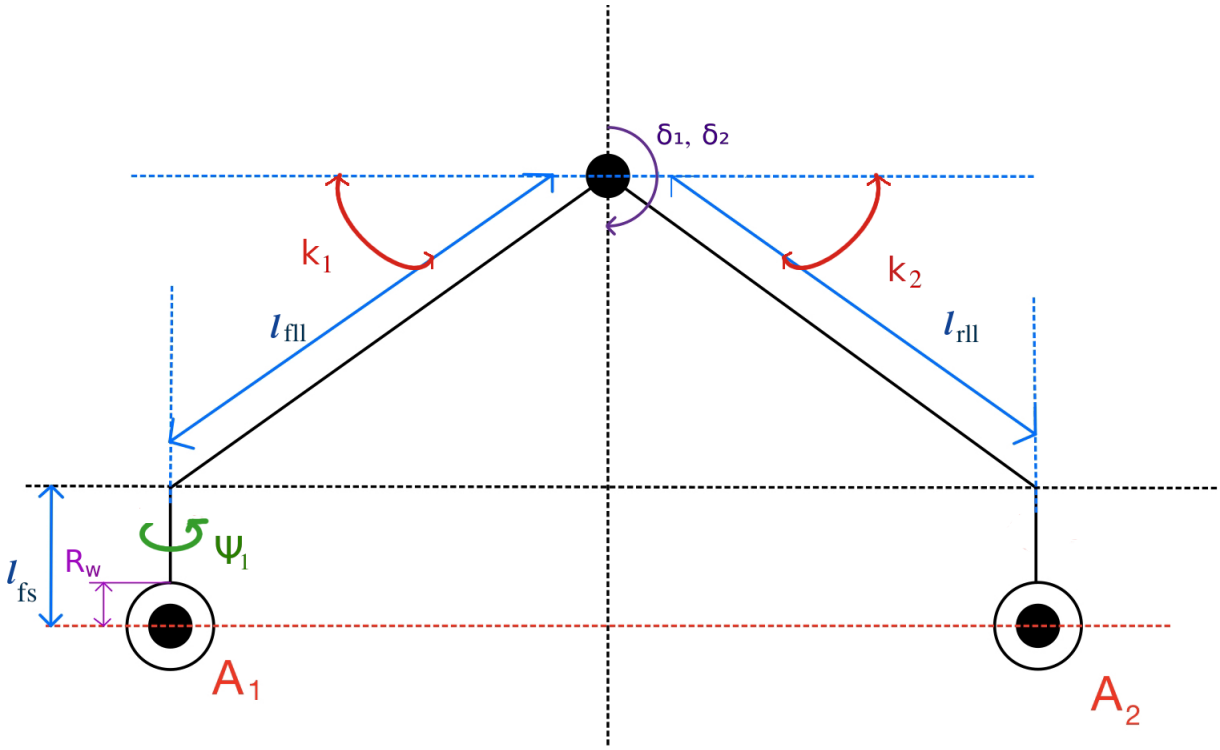


Figure 3.5: Side Diagram of the Rover Model on Flat Ground

| Symbol | Description |
|---|---|
| A1 | Wheel 1 - Front Left |
| A2 | Wheel 2 - Rear Left |
| $R_w$ | Wheel Radius |
| $\psi_1$ | Front Steering Angle |
| $k_1$ | Angle between Front Leg and Chassis x-axis on Flat Ground |
| $k_2$ | Angle between Rear Leg and Chassis x-axis on Flat Ground |
| $l_{fs}$ | Front Steering Leg Length |
| $l_{fll}$ | Front Left Leg Length |
| $l_{rll}$ | Rear Left Leg Length |
| $\delta_1$ | Front Left Leg Pivot Angle |
| $\delta_2$ | Rear Left Leg Pivot Angle |

Table 3.2: Description of Symbol Nomenclature

# Algorithms

## Overview

The following subsections explains the TCA in more detail, including its data flow, how this data is acquired, and the derivations for concepts integral to its implementation.

## Inputs and Outputs

The TCA operates on a number of configuration parameters, values which do not change over time. Most of these have to do with the size and shape of the vehicle (i.e., the lengths of links between joints), which obviously is staticas time passes. The algorithm also relies on inputs that are to be updated every time step (each time the algorithm calculates its outputs), which are summarized in Tables 3.3 and 3.5 for an implementation onboard the CMR and *Sample-Return Rover* (SRR), respectively.

| **Input Value** | **Description** |
|---|---|
| $\vec{\omega}$ | The 3x1 angular velocity vector around the base coordinate frame of the vehicle |
| $\vec{\delta} = [\beta, \rho_1, \rho_2]^T$ | The 3x1 vector of joint positions describing the rotation of the bogie joint and each rocker joint |
| $\dot{\vec{\delta}} = \left[\dot{\beta}, \dot{\rho_1}, \dot{\rho_2}\right]^T$ | The 3x1 vector of joint velocities describing the rotational speed of the bogie joint and each rocker joint |
| $\vec{\Psi}$ | The 6x1 vector of joint positions describing the steering angle of each wheel |

Table 3.3: Summary of the Inputs for the Algorithm Onboard the Curiosity Mars Rover

| Input Value | Description |
|---|---|
| $\vec{\dot{\theta}}$ | The 6x1 vector of joint speeds to be commanded to each of the vehicle's wheels |

Table 3.4: Summary of the Inputs for the Algorithm Onboard the Curiosity Mars Rover

When considering the algorithm for the SRR, the inputs and outputs will only change because the number of suspension joints and wheels differs from the CMR. Otherwise, the idea is the same. The inputs and outputs for an SRR implementation are summarized in Tables 3.5 and 3.6.

| Input Value | Description |
|---|---|
| $\vec{\omega}$ | The 3x1 angular velocity vector around the base coordinate frame of the vehicle |
| $\vec{\delta}$ | The 4x1 vector of joint positions describing the rotation of the joints corresponding to each of the four legs' pivot points |
| $\vec{\dot{\delta}}$ | The 4x1 vector of joint velocities describing the rotational speed of the joints corresponding to each of the four legs' pivot points |
| $\vec{\Psi}$ | The 4x1 vector of joint positions describing the steering angle of each wheel |

Table 3.5: Summary of the Inputs for the Algorithm Onboard the Curiosity Mars Rover

| Input Value | Description |
|---|---|
| $\vec{\dot{\theta}}$ | The 6x1 vector of joint speeds to be commanded to each of the vehicle's wheels |

Table 3.6: Summary of the Inputs for the Algorithm Onboard the Curiosity Mars Rover

### Measuring the Inputs

As seen in Tables 3.3 and 3.5, the TCA has four inputs, each of which are measured/calculated each time step. The CMR was equipped with encoders on each joint, so because the inputs $\vec{\delta}$ and $\vec{\Psi}$ are joint position measurements, their values are gathered directly from each of the respective encoders. Absolute encoders are able to output a position relative to some coordinated "home/zero" position, which is what are utilized here.

Some encoders are also capable of measuring the instantaneous velocity of their respective joint, but the team behind this solution for the TCA did not have these available on the CMR to collect the values of $\vec{\dot{\delta}}$. It was clarified that the velocities of these joints were calculated as a time derivative [4]. This means that given the initial and final position $\alpha$ of a joint which rotated over a small length of time $\Delta t$, the velocity of the joint at its final position can be estimated as follows:

$$\dot{\alpha} \approx \dot{\alpha}_{avg} = \frac{\alpha_f - \alpha_i}{\Delta t} \tag{3.7}$$

The angular velocity vector $\vec{\omega}$ of the CMR was calculated using an *Inertial Measurement Unit* (IMU), and more specifically the digital readings off the gyroscope(s) inside of it. A gyroscope, a 3D rendering of which can be seen in Figure 3.6, is a three degree of freedom mechanism which allows for the measurement of independent rotation about the three principal axes. With an IMU mounted to the body of the vehicle, $\vec{\omega}$ is easily measured.

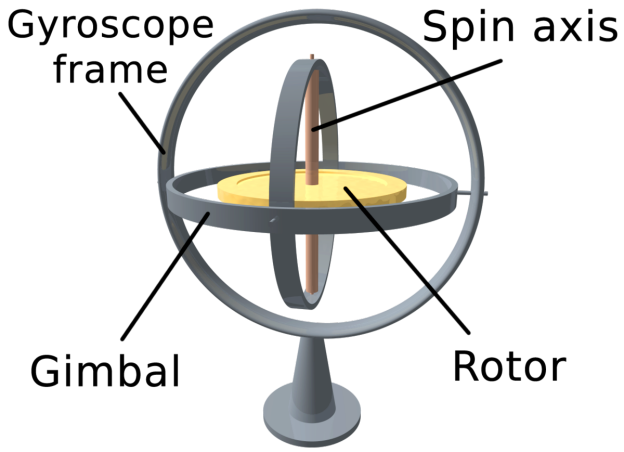


Figure 3.6: A 3D Rendering of a Gyroscope

### Transferring Velocity Between Coordinate Frames

This implementation for TCA is a largely geometric solution in the sense that a majority of the calculations it performs is estimating the linear velocity of each wheel given the knowledge of the vehicle's overall linear and angular velocity and the velocity of the joints intermediate to the vehicle's body and that wheel. Therefore, it is important to understand how these types of vectors interact with one another.

Consider some coordinate frames G and H which are attached to separate rigid-bodies as described in Section 3.4. Now consider some point Q that is resolved in frame H. The motion of Q is trivial to resolve in H, but what if the motion of Q needs to be resolved in G? First consider the case where the frames G and H are coincident and the position of their origins do not change with time, but there is a nonzero rotational velocity of H relative to G called ${}^{G}_{H}\Omega$. Assuming the point Q has no velocity relative to H, the linear velocity induced by this rotation resolved in G can be calculated as follows [1]:

$$ {}^{G}V_{Q} = {}^{G}_{H}\vec{\omega} \times \left({}^{G}_{H}R \cdot {}^{H}Q\right), \text{ when } {}^{H}V_{Q} = 0_{3x1} \tag{3.8} $$

In other words, the linear velocity of Q resolved in G is equal to the cross product of the rotational velocity of H relative to G and the position of Q resolved in G. Now, consider that there is a nonzero linear velocity of Q relative to H. This is simply rotated into the frame G, so the expression expands to the following [1]:

$$ {}^{G}V_{Q} = \left({}^{G}_{H}R \cdot {}^{H}V_{Q}\right) + \left({}^{G}_{H}\vec{\omega} \times \left({}^{G}_{H}R \cdot {}^{H}Q\right)\right) \tag{3.9} $$

In the general case, there are more considerations to take to fully formulate how linear velocities are transferred across frames [1]. However, none of them need be considered for this scenario because they are unique to the fact that G and H are assumed to be attached to separate rigid bodies. In the context of this algorithm, all frames are known to be attached to the same rigid body (the rover). Therefore, when solving for the velocity of a frame relative to another, both attached to the rover, Equation 3.9 will fully define that solution.

# Bibliography


[1] John J. Craig. *Introduction to Robotics: Mechanics and Control*. 2nd ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1989.

[2] S. Hutchinson M. W. Spong and M. Vidyasagar. *Robot Modeling and Control*. 1st ed. John Wiley and Sons, Inc., 2001.

[3] P. S. Schenker et al. "Robotic Autonomy for Space: Cooperative and Reconfigurable Mobile Surface Systems". In: i-SAIRAS 2001. Canadian Space Agency, St. Hubert, Quebec, Canada, 2001.

[4] Olivier Toupet et al. "Terrain-Adaptive Wheel Speed Control on the Curiosity Mars Rover: Algorithm and Flight Results". In: *Journal of Field Robotics* (2018), pp. 699–728. DOI: `http://dx.doi.org/10.1002/rob.21903`.