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

Robert Vandemark Diane Ngo

2020
November

Contents

Abstract

Space exploration continues to become more accessible as technology develops and international interest for it is reignited. Observing Mars is one of the highest priorities at the present and in the near future for a number of reasons, including the fact that it is the closest to us, it may have once had life, has been confirmed to have had liquid water at some point, and decades of intimate exploration has established a solid foundation for a human presence on the "Red Planet". There are many challenges pertaining to exploring unknown territory over long distances. NASA's approach to exploring and researching Mars was through the *Sample-Return Rover* (SRR), the first iteration of which was created in 1997 by the JPL Jet Propulsion Laboratory, and the most recent being launched this year in 2020. The scope of this project focuses on an emergent implementation for a type of traction control algorithm, designed for the *Curiosity Mars Rover* (CMR), as damage to the wheels was developing at an incredibly alarming rate due to the fact that the wheels were slipping over rough, elevated surfaces. This report offers a detailed description of the original problem, the algorithm proposed and implemented as the solution, and discussion about an attempt at simulating the algorithm on the SRR.

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], which is expanded on later in the document. Figure 2.1 shows a testbed version of the CMR.

Figure 2.1: The Scarecrow Testbed Rover, a Rover Kinematically Similar to the *Curiosity Mars Rover* [4]

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 inter-

fere 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 Section 3.1.

This algorithm was chosen to study because of the advantages it has over other possible implementations, some of which were considered by the team that came up with this original TCA solution. Aspects of the *Sample-Return Rover* (SRR) were being modeled for another course, so consideration was taken in applying the concepts of this solution and attempting to utilize them on this kinematically different vehicle, an image of which can be seen in Figure 2.2. A 3D rendering of a model similar to the SRR was generated, as well as a simulation suite that's able to interact with it, including an implementation of the algorithm. At the time of finishing this report, the simulation suite is still in its early stages and has limited functionality.

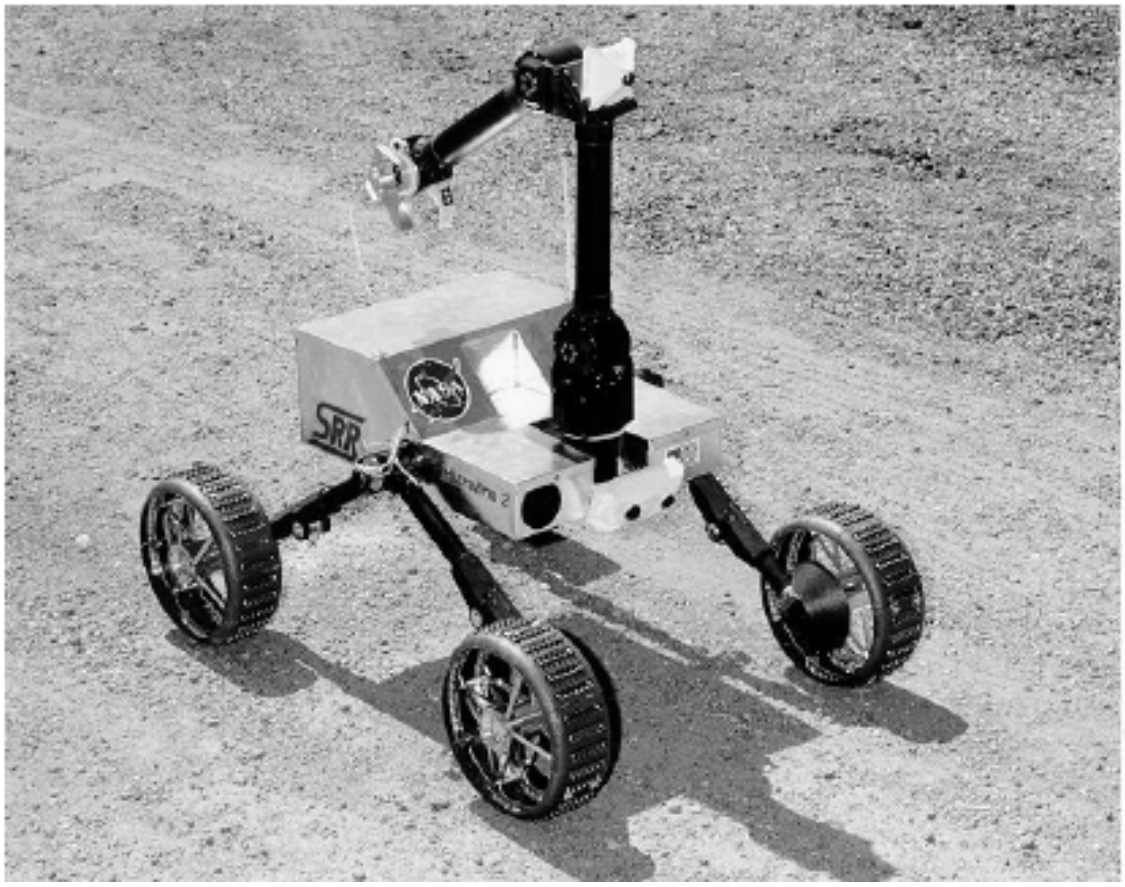

Figure 2.2: The *Sample-Return Rover* [3]

Discussion

Prerequisite Knowledge

Ackermann Steering Model

Modeling vehicles using the Ackermann steering model is a common practice, including for the *Curiosity Mars Rover* (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 3.1 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.

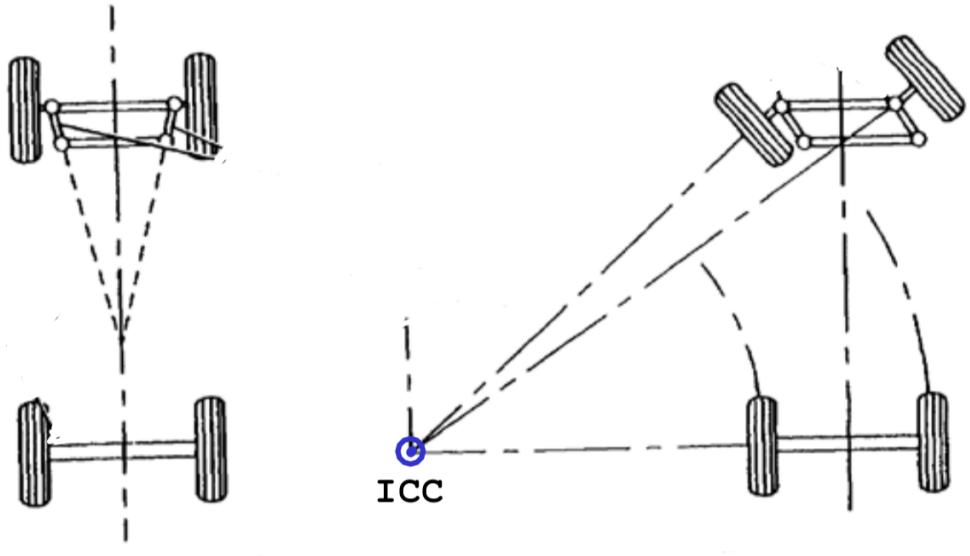

Figure 3.1: A Vehicle with Ackermann Steering Going Straight (Left) and in a Turning Position (Right)

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 *Traction Control Algorithm* (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 θ. 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 θ, these rotation matrices are defined in Equations 3.3, 3.4, 3.5. These 3 rotations are also called the roll(ϕ), pitch (θ), and yaw(ψ) matrices. The order of rotation is through yaw (x,(ψ)),pitch (y,(θ)), and roll (z,(ϕ).

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

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

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

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 3.2 shows these frames from the left side of the robot. A top view is also provided as Figure 3.3.

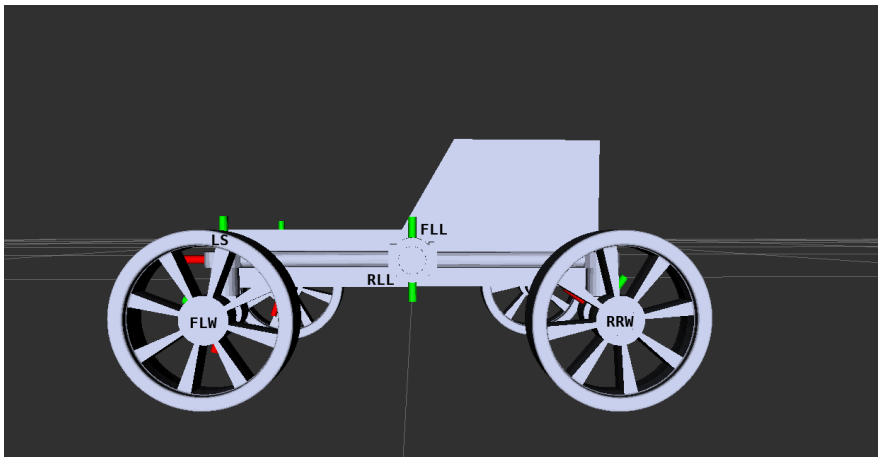

Figure 3.2: Side View of the Robot Model and its Coordinate Frames

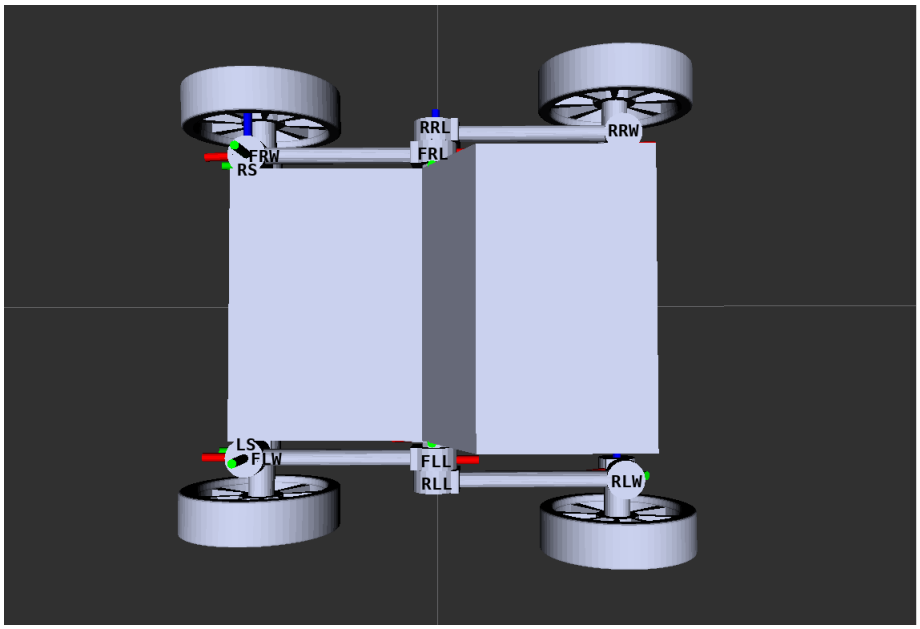

Figure 3.3: Side View of the Robot Model and its Coordinate Frames

A summary of the acronyms for the coordinate frames used in Figures 3.2 and 3.3 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 3.4,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 ψ, k, or δ.

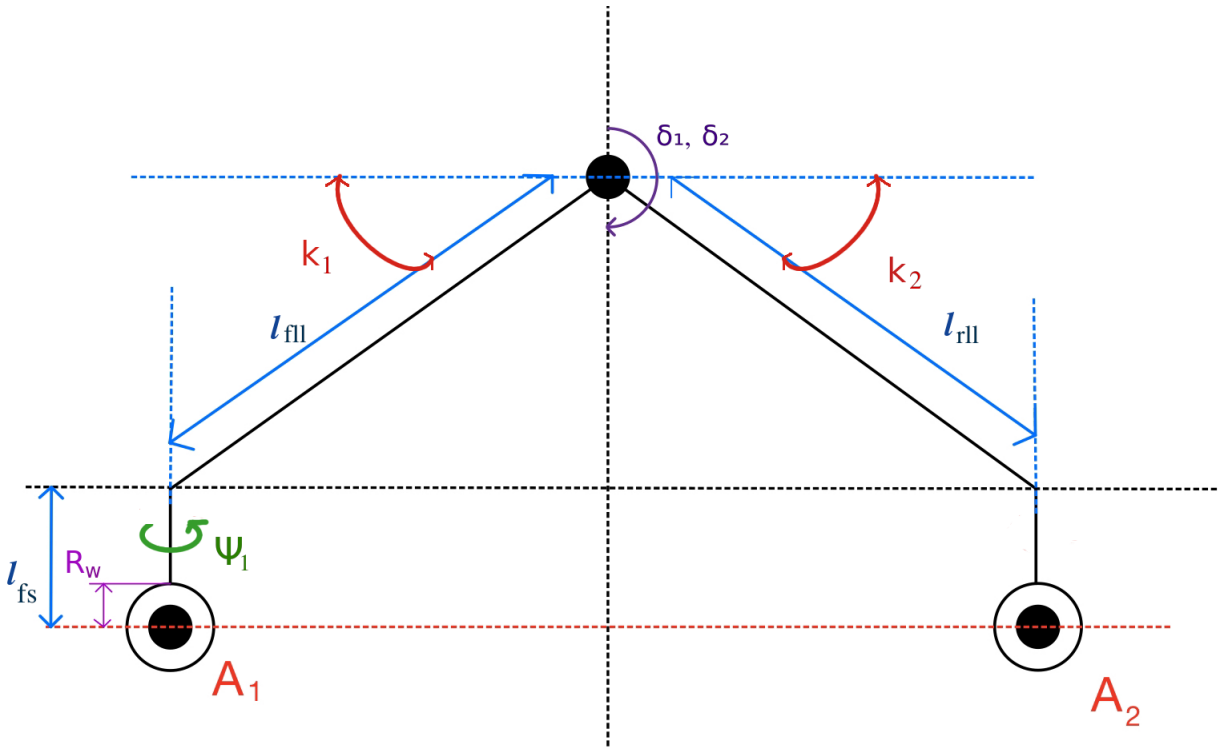

Figure 3.4: Side Diagram of the Rover Model on Flat Ground

Symbol	Description
A1	Wheel 1 - Front Left
A2	Wheel 2 - Rear Left
R_w	Wheel Radius
ψ_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
δ_1	Front Left Leg Pivot Angle
δ_2	Rear Left Leg Pivot Angle

Table 3.2: Description of Symbol Nomenclature

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 ve-

hicle'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 3.1.2. 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]:

$$ {}^{G}V_{Q} = {}^{G}_{H}\vec{\omega} \times \left({}^{G}_{H}R \cdot {}^{H}Q \right), \text{ when } {}^{H}V_{Q} = 0_{3x1} \tag{3.7} $$

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]:

$$ {}^{G}V_{Q} = \left({}^{G}_{H}R \cdot {}^{H}V_{Q} \right) + \left({}^{G}_{H}\vec{\omega} \times \left({}^{G}_{H}R \cdot {}^{H}Q \right) \right) \tag{3.8} $$

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 3.8 will fully define that solution.

Overview

The following subsections explains the *Traction Control Algorithm* (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 *Curiosity Mars Rover* (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
$\dot{\vec{\theta}}$	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
$\dot{\vec{\delta}}$	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
$\dot{\vec{\theta}}$	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 $\dot{\vec{\delta}}$. 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 α of a joint which rotated over a small length of time Δt, the velocity of the joint at its final position can be estimated as follows:

$$\dot{\alpha} \approx \dot{\alpha}_{avg} = \frac{\alpha_f - \alpha_i}{\Delta t} \tag{3.9}$$

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 3.5, 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.

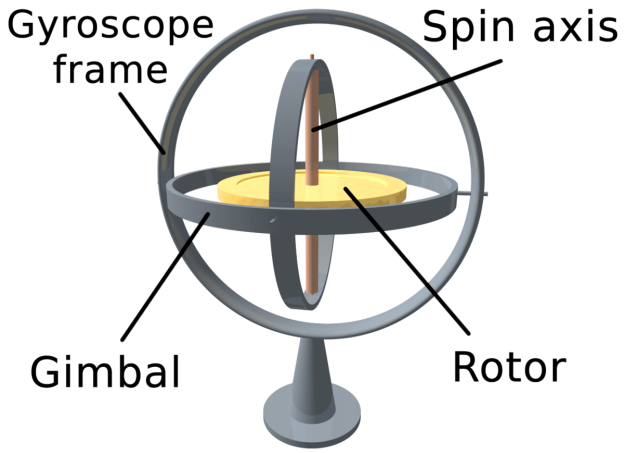

Figure 3.5: A 3D Rendering of a Gyroscope

Calculations

Contact Angle Frame

As mentioned in Section 3.1.1, basic Ackermann Steering is typically used to model a vehicle assumed to be traversing a planar surface, or at least a terrain whose inconsistencies are negligible. The *Traction Control Algorithm* (TCA) acknowledges that is not the case here by defining this contact angle frame relative to each wheel's frame, an illustration of which can be seen in Figure 3.6.

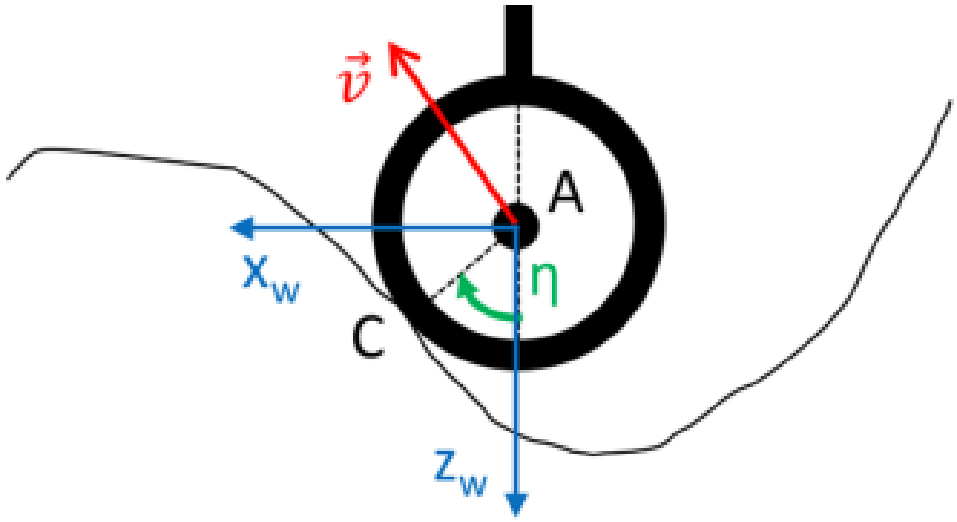

Figure 3.6: The Contact Angle η Defines the Contact Wheel Frame for Some Wheel A [4]

XXXXXXXXXXXXXXXXXX ADD FIGURE REF HERE XXXXXXXXXXXXXXXXXX
Figure 3.6 uses the vector $\vec{v}$ to denote the linear velocity of the wheel A, which is calculated using the concept of velocity transformation introduced in Section 3.1.4. The geometry of the *Curiosity Mars Rover* (CMR) is symmetrical across the body's x-axis (front to back), but asymmetrical across its width (side to side), as can be seen in Figure **??**. This implies that while the theory of velocity transformation applies to both, the specific calculations to accomplish this for one wheel is slightly different than some others. For example, the calculation of a wheel linked to the rocker joint can be seen in Equation 3.11, while the calculation for one of the wheels linked to one of the bogie pivot joints can be seen in Equation 3.13 [4]. Essentially, the velocity is transferred an additional time because there is another joint intermediate to the bogie wheels that cannot be ignored.

$$ {}^{bd}V_D = \begin{bmatrix} \dot{x}_0 \\ 0 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \begin{bmatrix} x_{od} \\ 0 \\ z_{od} \end{bmatrix} \right) \tag{3.10} $$

$$ {}^{bd}V_{A_1} = {}^{bd}V_D - \begin{bmatrix} \omega_x \\ \omega_y + \dot{\beta} \\ \omega_z \end{bmatrix} \times \left({}^{bd}_{rk1}R \cdot \begin{bmatrix} -l_{fd}cos(\kappa_1) \\ -Y_{of} \\ -l_{fd}sin(\kappa_1) \end{bmatrix} \right) \tag{3.11} $$

$$ {}^{bd}V_{B_1} = {}^{bd}V_D - \begin{bmatrix} \omega_x \\ \omega_y + \dot{\beta} \\ \omega_z \end{bmatrix} \times \left({}^{bd}_{rk1}R \cdot \begin{bmatrix} l_{db}cos(\kappa_2) \\ 0 \\ -l_{fd}sin(\kappa_2) \end{bmatrix} \right) \tag{3.12} $$

$$ {}^{bd}V_{A_3} = {}^{bd}V_{B_1} - \begin{bmatrix} \omega_x \\ \omega_y + \dot{\beta} + \dot{\rho}_1 \\ \omega_z \end{bmatrix} \times \left({}^{bd}_{rk1}R \cdot {}^{rk1}_{bg1}R \cdot \begin{bmatrix} -l_{bm}cos(\kappa_3) \\ Y_{om} \\ -l_{bm}sin(\kappa_3) \end{bmatrix} \right) \tag{3.13} $$

See [4] for a more detailed explanation as to the meaning of each variable in Equations 3.10-3.13. With an understanding of those variables, it can be seen that these variables follow the general form of Equation 3.8.

For those calculations to be performed, the linear velocity of the rover's body must be estimated. One of the existing requirements for the CMR is that at least one of its wheels must be operating at its peak speed at all times (other than when it is stationary). This requirement along with the assumption that the rover is moving in the rover body's x-axis (forward or backward) means that its linear velocity can be assumed to be completely in its x-axis. This is where the initial linear velocity value $[\dot{x}_0 \;\; 0 \;\; 0]^T$ comes from in Equation 3.10.

Once the linear velocities of the six wheels resolved to the rover's body frame, ${}^{bd}V_{A_{1-6}}$, are calculated, the contact angle η_i can be solved for the i^{th} wheel. The i^{th} contact angle frame is defined as coincident with the i^{th} wheel frame (their origins are the same) with a rotation of η_i about the same axis that wheel rotates about, Y_{w_i}. It can be seen in Figure 3.6 that the linear velocity of the wheel is perpendicular to this contact angle frame, so the calculation for finding η_i is straightforward:

$$\eta_i = -\arctan\left(\frac{{}^{w_i}V_{A_i}^z}{{}^{w_i}V_{A_i}^x}\right) \tag{3.14}$$

Solving for Commanded Wheel Rates

In the general case, the objective is to find the wheel rates given some desired velocity of the rover. When calculating the contact angle frames for each wheel, it was proven that the linear velocity of the rover can be resolved in the wheel frames, while accounting for velocities induced by angular velocities from joint motions. That calculation was performed for the current estimated linear velocity of the vehicle, but now the same routine can be done to calculate the linear velocities of the wheels that would achieve this objective linear velocity of the rover body. In other words, Equations 3.10-3.13 (and those required for the other wheels, left out in the interest of brevity) can be reused other than the fact that the initial velocity in Equation 3.10 will no longer be $[\dot{x}_0 \;\; 0 \;\; 0]^T$ but this objective velocity.

By itself, calculating this linear velocity of the wheels is not beneficial, but it can be built onto to calculate the angular velocity of the wheel required to generate its desired linear velocity, thereby solving the problem. To do this, the linear velocity of the wheel with respect to the rover's body frame must be transformed to the linear velocity of the contact angle frame, as seen in Equation 3.15.

$$\begin{aligned} {}^{\eta_i}V_{A_i} &= {}^{\eta_i}_{bd}R \cdot {}^{bd}V_{A_i} \\ &= {}^{\eta_i}_{w_i}R \cdot {}^{w_i}_{bd}R \cdot {}^{bd}V_{A_i} \end{aligned} \tag{3.15}$$

As mentioned before in Section 3.3.1, some of the frames from the body of the rover to each wheel are asymmetric relative to one another, so ${}^{w_i}_{bd}R$ depends on the geometry of the links between the body and the wheels. These are fully defined in [4]. On the other hand, because the contact angle for the i^{th} wheel η_i is always a rotation about Y_{w_i}, each transformation from the wheel frame to the contact angle frame can be given by the rotation matrix in Equation 3.16, and the velocity of the wheel A can be resolved in its respective contact angle frame.

$${}^{\eta_i}_{w_i}R = \begin{bmatrix} cos(\eta_i) & 0 & sin(\eta_i) \\ 0 & 1 & 0 \\ -sin(\eta_i) & 0 & cos(\eta_i) \end{bmatrix} \tag{3.16}$$

Now to visualize how a linear velocity of a wheel with respect to its contact angle relates to its angular velocity, consider the relationship between the arc length of a circle s, its radius r, and the rotation about the center of it that creates the arc α, seen in Equation 3.17. Taking the derivative of both sides of the equation with respect to time results in Equation 3.18. If the arc length of a circle is thought of instead as the arc *displacement* as a function of time, then $\dot{s}$ can be thought of as the scalar arc velocity, where the velocity of the point on the arc at a point in time is tangential to the position on the arc with a magnitude equal to the radius of the circle times the derivative of the angle, which is now an angular velocity.

$$s = r\alpha \tag{3.17}$$

$$\dot{s} = r\dot{\alpha} \tag{3.18}$$

Recall the definition of the contact angle frame from Figure 3.6 and how the contact angle is calculated in Equation 3.14. This frame is defined as having its $+x$ direction pointing in the same direction as the linear velocity of the wheel, which can be seen by rotating the wheel frame about the Y_w axis by η and finding that the X_w axis has aligned with the linear velocity vector. This relationship is the same as the one in Equation 3.18, where the arc velocity is the x component of the linear velocity of the wheel resolved in the contact angle frame and the radius of the circle is the radius of the wheel, but the parallel to the arc velocity has more complexity to it. The existing angular velocity experienced in the contact angle frame of a wheel (that is, without any rotational velocity of the wheel) has contributions due to the angular velocity of the vehicle in its own body's frame and angular motion in joints between the body and the wheel. The joints that are responsible for steering the wheels to point a direction do not rotate while the rover is performing a move, so there is no contribution to angular velocity from them. Only the rocker and possibly a bogie pivot joint would contribute, depending on which wheel this concerns.

Because the angular velocity experienced by the wheel may be nonzero, then the contribution that it has about the wheel's y-axis (its axis of rotation) must be accounted for as well, as seen in Equations 3.19 and 3.20. This may mean a higher commanded wheel rate is required to meet the objective linear velocity, or it could mean less.

$$\zeta_i = {}^{\eta_i}_{bd}R \cdot \begin{bmatrix} \omega_x \\ \omega_y + \dot{\lambda} \\ \omega_z \end{bmatrix} \tag{3.19}$$

$${}^{\eta_i}V^x_{A_i} = R_w \left(\dot{\theta}_i + \zeta^y_i\right) \tag{3.20}$$

Where $\dot{\lambda}$ is the angular velocity induced onto the i^th wheel. The actual values for these can be found in [4].

Maximizing Commanded Wheel Rates

As mentioned in Section 3.3.1, one of the existing design considerations that the team behind the TCA solution had to account for was that at least one wheel on the rover must be at its peak velocity while moving. The solution is to rearrange Equation 3.20 to solve for $\dot{\theta}_i$. As stated when deriving this equation, the linear velocity of the wheel acts only in the x-axis of the contact wheel frame. Therefore, it can be assumed that the y and z components of the velocity can be treated as zero:

$$^{\eta_i}V^{y}_{A_i} = 0 \tag{3.21}$$

$$^{\eta_i}V^{z}_{A_i} = 0 \tag{3.22}$$

This formulation results in multiple pages of equations, see [4] for them in full.

Simulation

Development Technologies

A 3D rendering of the *Sample-Return Rover* (SRR) was developed in SolidWorks, which was then exported to be compatible with Gazebo, a powerful robot 3D simulation software. ROS Kinetic was used to communicate data across different components of the simulation, and some open source contributions from the ROS community were used to enable functionality such as joint controllers, hardware simulation, etc.

In actuality, the model produced does not exactly represent the SRR as seen in Figure 2.2. It proved difficult to model the wheels as the SRR did, so instead the model seen in Figure 3.7 was created in its place with the alteration to each leg like the one seen in Figure 3.8.

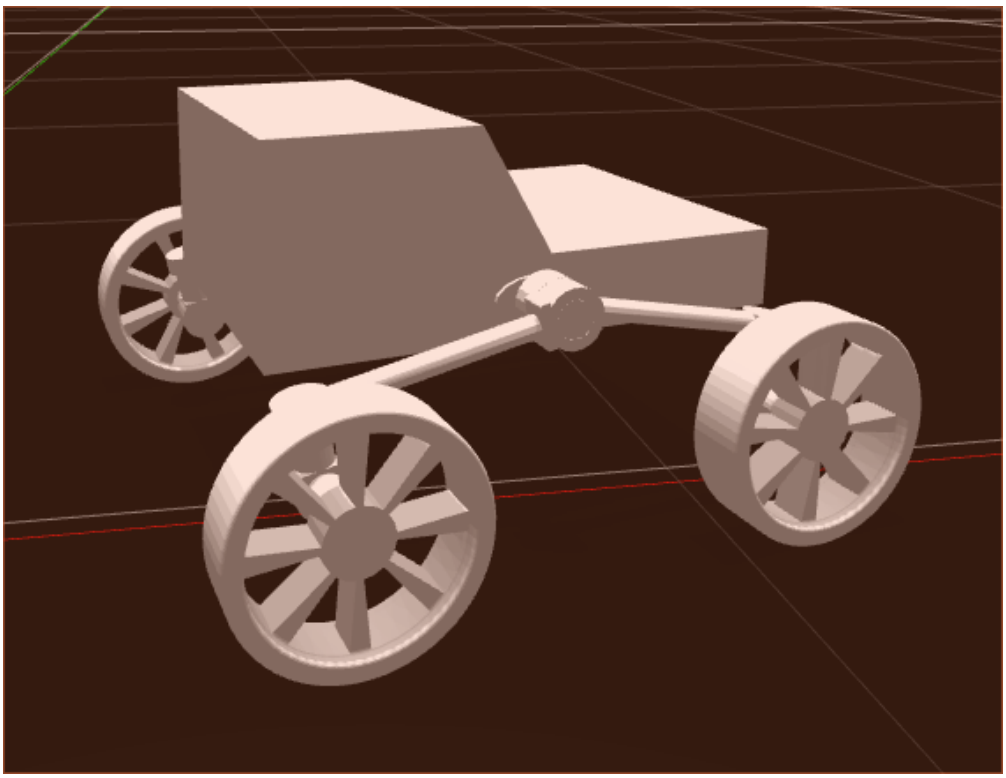

Figure 3.7: The 3D Rendering of the *Sample-Return Rover*

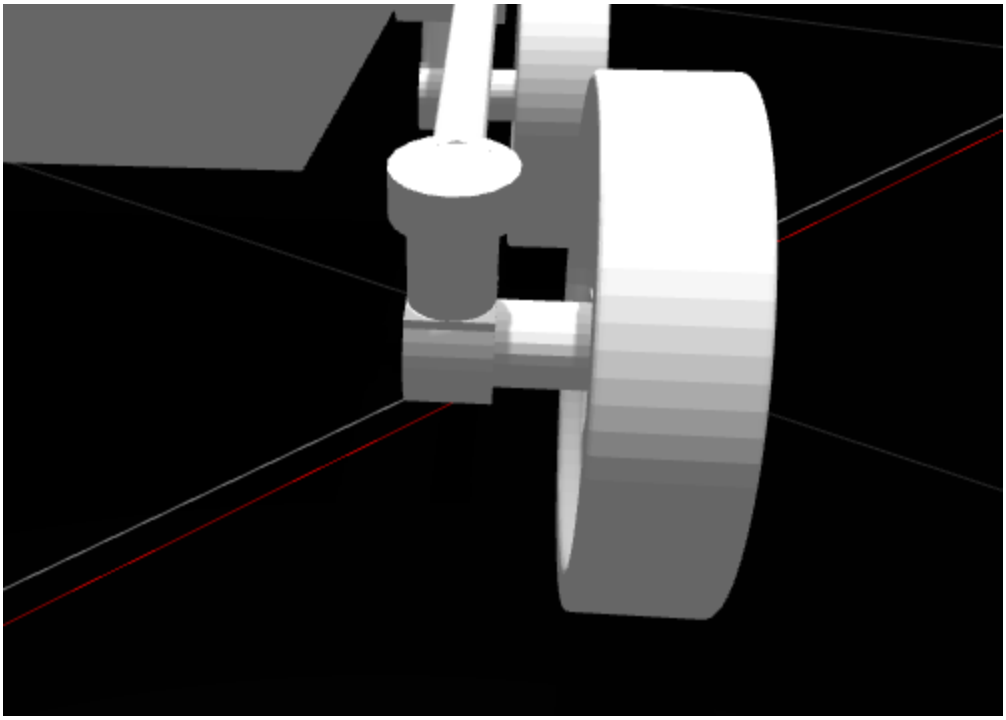

Figure 3.8: The Close-Up of the Kinematically Different Leg for the 3D Rendering of the *Sample-Return Rover*

The implementation of the *Traction Control Algorithm* (TCA) can be found in the *srr_kinematics* package. The files in this package containing the object that implements the algorithm are *include/srr_kinematics/TractionControlContainer.hpp* and *src/TractionControlContainer.cpp*, while the ROS node lives in *src/TractionController.cpp*.

This model has simulated versions of joint encoders and an *Inertial Measurement Unit* (IMU) from the Gazebo ROS Control and Gazebo ROS IMU libraries, respectively. With the first plugin, a joint state controller is able to regularly publish updates to the current joint positions so the traction controller node can update its control loop. Furthermore, this plugin is able to update current joint velocity. Important to note here is that this is a different approach to acquiring the joint velocities than the original TCA solution had implemented (a manually computed time derivative), but the result is intended to be the same. The second plugin allows a simulated IMU to be attached to a body at any location with any orientation. For the purposes of this simulation, it was attached to the rover's inertial frame, which reflects the original implementation.

The Gazebo ROS control plugin also allows for controllers on the individual joints. The traction control node owns ROS publishers that publish to the joint velocity controllers' topics, such that the calculated wheel rates $\vec{\dot{\theta}}$ are piped directly to the joint controllers.

Overall, not all functionality is able to be simulated for a couple of reasons. The first and foremost is that in the interest of time, the final step taken in Section 3.3.3 was not taken, as that implementation would have taken too much time, not to mention that it would have to be rederived due to the fact that this model is different than the one that those equations were originally meant for. This algorithm operating in a turn also has undefined behavior because the software behind

simulating proper Ackermann Steering and the controllers for it were not implemented as of yet. Finally, there are mechanical issues with the model at the moment where the legs resting on their pivots cause friction at the pivots themselves and at the wheels' rotors. In other words, it's sometimes difficult to drive the vehicle straight and with the chassis flat because the weight of the links grinds against other links.

Example Plotting

The wheels of the robot are labeled 0-3, in the order of Front Left, Front Right, Back Left, and Back Right. In the code, the wheels are sometimes identified as the leg pivot that they are a child to.

As a sanity test, Figure 3.9 illustrates that when the vehicle is stationary and when the angles of the leg pivots are not changing, the contact angle stays the same. The noise seen here (and for the other plots) are due to small amounts of intentionally generated Gaussian noise in the simulated encoders and simulated IMU. Some of this is also probably due to some floating-point arithmetic error, though plot points that are below a certain threshold are ignored to avoid just this.

Figure 3.10 shows the contact angles of each wheel over time as the four leg pivot joints are raised and lowered (approximately) simultaneously. These results make sense because they become more positive or negative as the wheels approach being directly underneath the chassis (and the sign difference is only due to the difference in frames relative to the origin for front and back wheel pairs. An interesting phenomenon to note are the gradual declines seen at about time t = [40.5, 41.5] and [45.5, 46.5]. This implementation of the TCA uses a windowed average for the leg pivot joint positions just as the original implementation for the *Curiosity Mars Rover* (CMR) does, so these gradual declines are due to those averages being lowered as time passes. This is reinforced by the fact that the length of these windows both last about one second according to the graph, and the constant defined in the traction control ROS node as is specifies that the windowed average should contain 1.0 second's worth of samples.

Lastly, Figure 3.11 shows how the commanded wheel rates change as events similar to those seen in Figure 3.10 occur. A difference in which direction is considered positive rotation for this simulated vehicle causes the joint velocities appear to be negative when they are actually considered an "increase". Each downward spike occurs when the contact angles are increased, then the contact angles being lowered allows for the spike to be zeroed out.

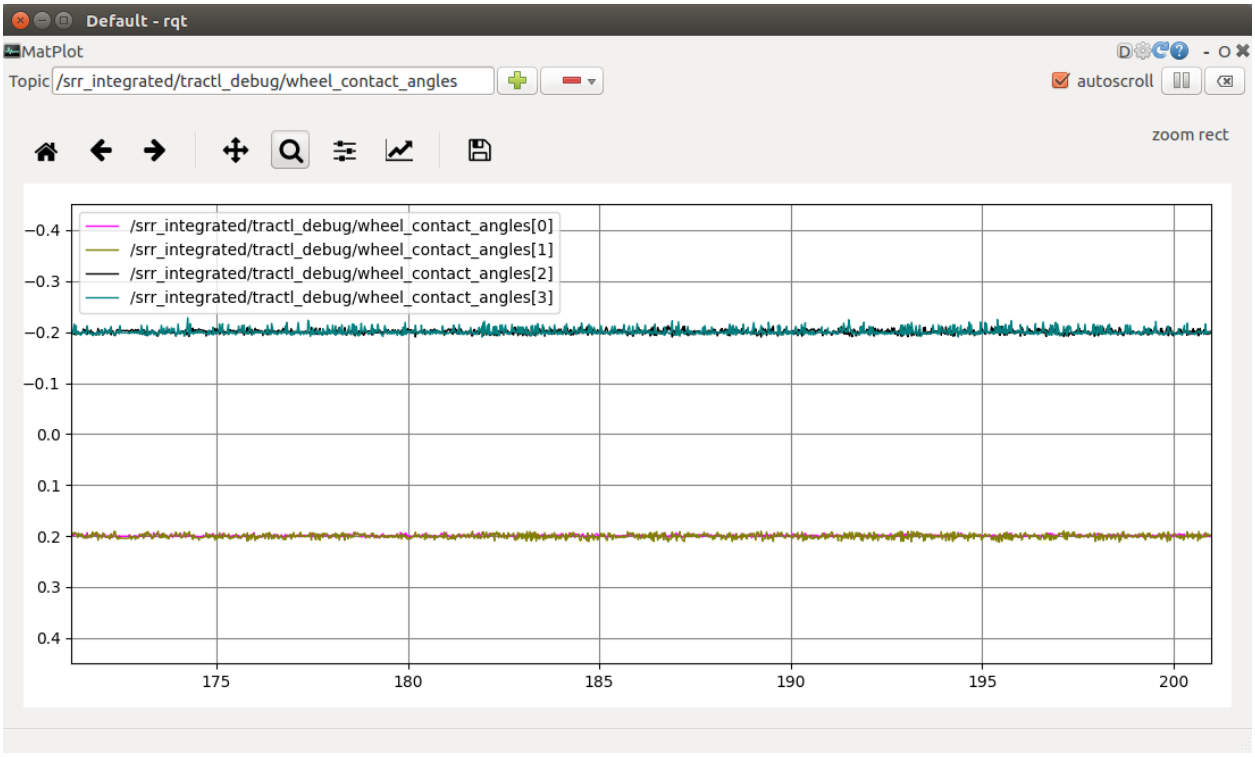

Figure 3.9: Static Contact Angles are Maintained While Vehicle is Stationary, at the Position Seen in Figure 3.7

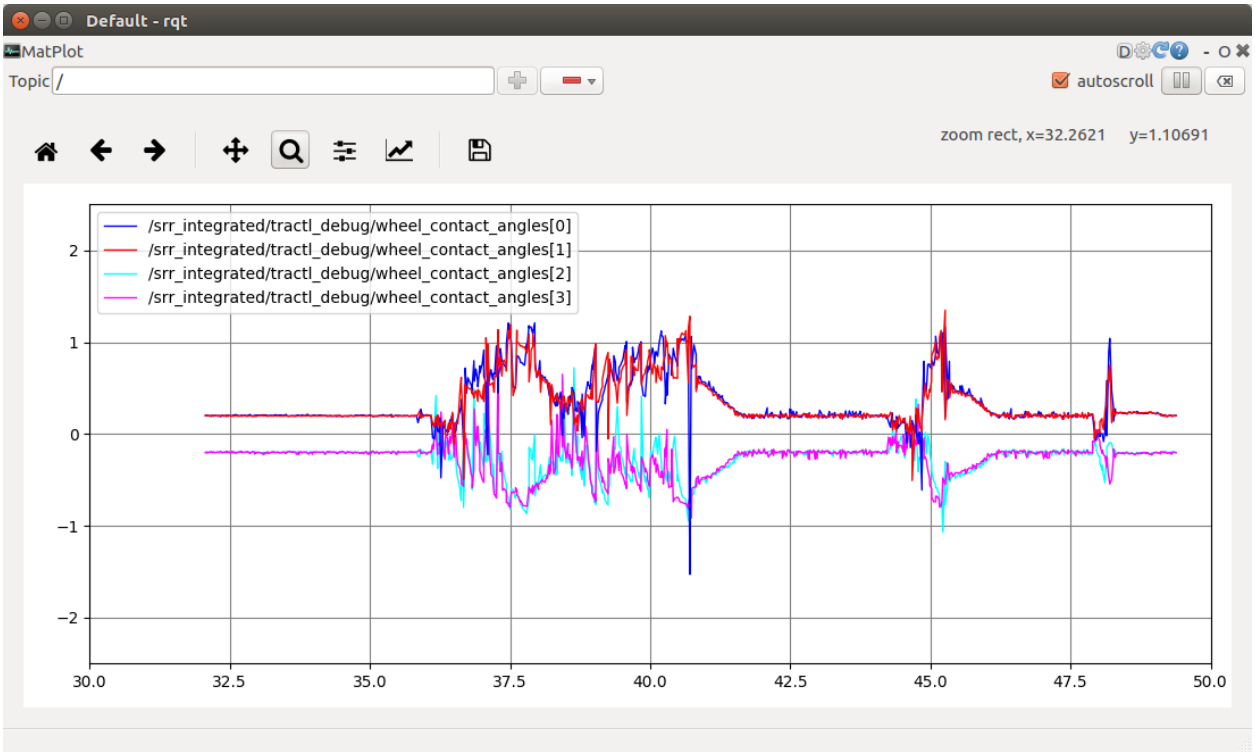

Figure 3.10: Contact Angles In/Decrease as Vehicle Raises Off of / Lowers Down to the Ground

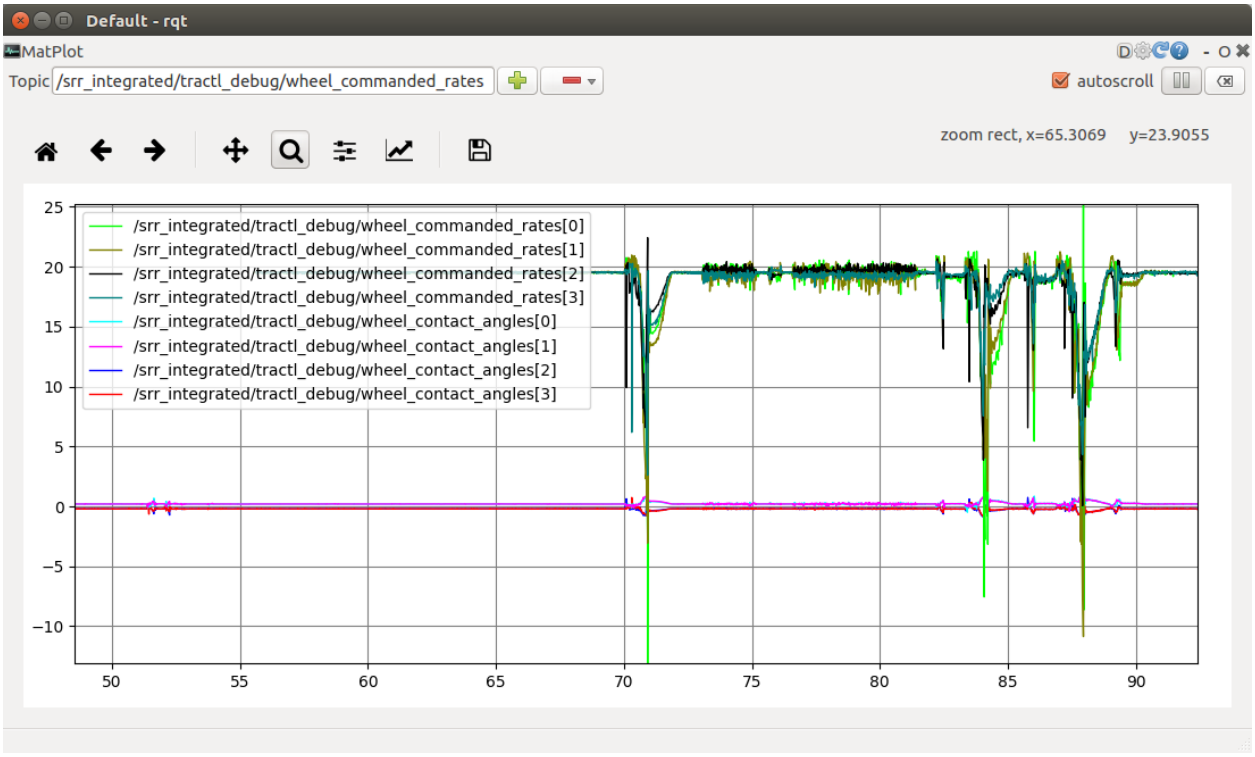

Figure 3.11: Commanded Wheel Rates Drop and Rise as the Contact Angle Changes

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`.