MoonRanger 4-Wheeled Kinematics Model

Ben Kolligs

August 14, 2021

1 Introduction

In order to estimate MoonRanger's pose and generate motion from drive arcs we can use a kinematic model. The model takes wheel velocities and determines the rover's body frame velocity, and vice versa. This document outlines a kinematic model for MoonRanger that is 4-wheeled, 3D, and offers a method for estimation of slip based off of terrain parameters. The main reference for the method presented here is the work of Neal Seegmiller in his PhD thesis, [1]. Note that throughout this document, we use the terms actuation kinematics to describe solving for joint rates given body velocity, and navigation kinematics to describe solving for body velocity given joint rates.

2 Development of Method

2.1 Transformation based kinematics

The first step of formulating the kinematics is to define a transform table defining the actuated joints of MoonRanger.

i	**Frame**	**Parent**	**Type**	**Actuated?**	x	y	z
1	body	world					
2	FL	body	RY	Y	d	w	$-h$
3	FR	body	RY	Y	d	$-w$	$-h$
4	BL	body	RY	Y	$-d$	w	$-h$
5	BR	body	RY	Y	$-d$	$-w$	$-h$

Table 1: MoonRanger's transform table, where d is the x distance from the body center, w is the y distance from the body center, and z is z distance from body center. Note that RY means the joint is revolute along the y-axis.

Using this description of MoonRanger, we can then calculate the homogeneous transformations between each frame and it's parent where the rotation matrix is a function of the current value of the joint, θ_i.

$$T_i^{p(i)} = \begin{bmatrix} R_i^{p(i)}(\theta_i) & {}^{p(i)}r_i^{p(i)} \\ 0 & 1 \end{bmatrix}, \tag{1}$$

where $p(i)$ is the parent of frame i. We also require the transforms from wheel joint to the wheel contact point. For this formulation, we assume that the rover wheels only contact one point on the ground at a time, and that this contact point is always a distance below the wheel hub in the local z-direction equal to the radius of the wheel. This is a simplification from that presented in [1], though it is possible to estimate the wheel contact point using a mesh of the ground or the current pitch of the rover in 3-space.

Once we have the frames of the wheels and wheel contact points, we can visualize the rover frames as in figure 1. Using these transformations, we can obtain the actuation and navigation kinematics and use the transforms from wheels to the body of the rover and derive a wheel Jacobian A, which describes the change in wheel joint movements based off of body frame velocities. The wheel Jacobian is used to act on

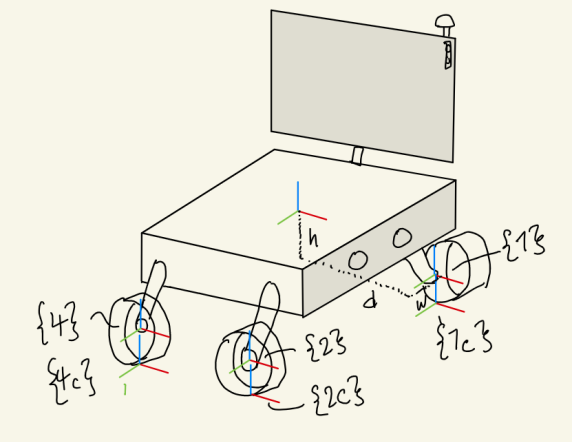

Figure 1: Frames of the wheel joints, and contact points.

the rover state vector q:

$$q = \begin{bmatrix} \psi_x \\ \psi_y \\ \psi_z \\ x \\ y \\ z \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} \tag{2}$$

The first six entries are the pose, and the last four are the joint values.

The reason we need to know the wheel contact point frames is that we are going to enforce a "wheel contact velocity" v_c constraint on the vehicle. This contact velocity constraint determines whether the wheel is slipping or not, and will be proportional to the rover speed, and the terrain parameters of the ground. The wheel contact constraint must satisfy the following equation involving the wheel Jacobian A_i.

$$v_c = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{3}$$

$$A_i \dot{q} = v_c \tag{4}$$

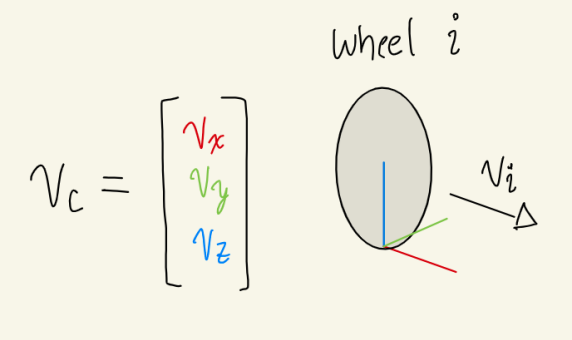

Figure 2: The contact frame of the wheel used to estimate slip.

The wheel contact constraint models slip as an adjustment of the rover pose in the world frame, which is partially why we use a state vector that includes both the pose and wheel velocities on top of each other. For example if slip is 0 then all the v_x, v_y entries in the constraint vector are 0, meaning no deviation occurs in these directions in the overall system. This is shown in figure 2.

The Jacobian A_i will be of size 3×10, due to the wheel contact constraint we enforce in equation 4. For wheel i the formulation of the wheel Jacobian A_i is as follows:

$$\Theta_i \in \mathbb{R}^{3\times 4} \tag{5}$$

$$\Theta_i[:, i] = R_i^w[:, a(i)] \times (r_c^w - r_i^w) \tag{6}$$

$$A_i = \begin{bmatrix} [r_c^w - r_1^w]_\times^T R_1^w & R_1^w & \Theta_i \end{bmatrix} \tag{7}$$

Where Θ_i is the joint Jacobian of wheel i, $a(i)$ is the axis of rotation of the joint i, R_b^a is the rotation matrix from frame b to frame a, and r_b^a is the position vector in x, y, z from frame b to frame a; for example, R_1^w is the transform of the body frame relative to the world frame. Note that $\Theta_i[:, i]$ means that we fill the i^{th} column of Θ with the calculation $R_i^w[:, a(i)] \times (r_c^w - r_i^w)$. We can form the full system Jacobian A by stacking the four wheel Jacobians horizontally. Lastly we perform a transform on the Jacobian from world to contact coordinates, $A = (R_c^w)^T A$ in order to ensure that the Jacobian produces the relevant output from actuation or navigation kinematics in the rover frame.

Now to actually compute the actuation or navigation kinematics, we use our constraint from earlier, and apply the pseudo-inverse of the Jacobian, like so:

$$\dot{q(unknown)} = A[:, unknown]^\dagger (v_c - A[:, known]\dot{q}(known)) \tag{8}$$

Here *unknown* and *known* describe the parts of the state vector q that we are trying to solve for. We can then slice up the Jacobian in order to solve for these unknowns. For example, when solving either kinematic direction (actuation or navigation) we solve one of the following equations:

$$\dot{\theta} = A_u^{\dagger}(v_c - A_k v_b^w) \tag{9}$$

$$v_b^w = A_u^{\dagger}(v_c - A_k \dot{\theta}) \tag{10}$$

where $\dot{\theta}$ is the joint rate, v_b^w is the desired body velocities, A_u, A_k are the sliced up Jacobians corresponding to the pose and joint values, and v_c is the contact velocity constraint from equation 4.

3 Implementation

The approach detailed in the previous section and in [1] is meant to function as a general approach to solving 3D kinematics for wheeled mobile robots given and vehicle geometry with both fixed and free link joints. Seegmiller's formulation is essentially a 6 step process consisting of these steps:

1. Use the current joint positions to construct the list of transforms to each joint i and each contact frame c
2. Use the list of transforms to construct the current Jacobian matrix A
3. Solve either for the actuation kinematics (joint velocities) or navigation kinematics (body velocity)
4. Use the calculated joint or body velocity to inform state update of rover
5. Assemble the state vector and propagate vehicle state through time
6. Repeat from step 1

However when we apply this to MoonRanger, there is key information that we know will stay constant throughout the mission, which invites an oppurtunity to simplify the original problem. This includes various dimensions of Moonranger, and the fact that Moonranger is a four wheeled skid steer vehicle. In addition, we made the simplifying assumption earlier that the wheel contact frame is fixed at the wheel radius directly beneath the wheel hub, and does not change throughout the mission. This assumption sacrifices accuracy in order to gain computational ease. It is also

not necessary for the kinematic model to maintain a 3D state of the rover, because this is performed by the state estimator module. This means that we can potentially skip most of the steps of the previous list and only proceed with steps 3 and 4.

The main benefit of simplifying this in this manner is that is drastically reduces the amount of online computation required to perform the four wheeled kinematics, nor do we need to concern ourselves with detailing this information in the Jacobian. Therefore we can simplify the problem like so:

1. Determine the symbolic form of the Jacobian required to calculate kinematics
2. Use this Jacobian to calculate the form of the pseudo-inverses, again simplifying online computation
3. Solve either actuation or navigation kinematics and output result

First, to find the symbolic form of the Jacobian. Because the algorithm produces a Jacobian in the contact frame coordinates as opposed to the world coordinates, the values are stable throughout the journey of the rover. Knowing this, we can produce the Jacobian like so:

$$A_{moonranger} = \begin{bmatrix} 0 & -h-r & -w & 1 & 0 & 0 & -r & 0 & 0 & 0 \\ h+r & 0 & l & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ w & -l & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -h-r & w & 1 & 0 & 0 & 0 & -r & 0 & 0 \\ h+r & 0 & l & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -w & -l & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -h-r & -w & 1 & 0 & 0 & 0 & 0 & -r & 0 \\ h+r & 0 & -l & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ w & l & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -h-r & w & 1 & 0 & 0 & 0 & 0 & 0 & -r \\ h+r & 0 & -l & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -w & l & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0] \end{bmatrix} \tag{11}$$

After the Jacobian is determined, we can cache the pseudo inverse of the velocity and joint rate Jacobians required for the computation in equations 9 and 10. This means that the kinematics in either direction becomes as simple as a single matrix multiplication.

Notice that the Jacobian here doesn't include any sort of trigonometry even though it's construction uses several different rotation matrices. This is a direct result of Moonranger being skid-steered. If the joints were able to swivel, or the

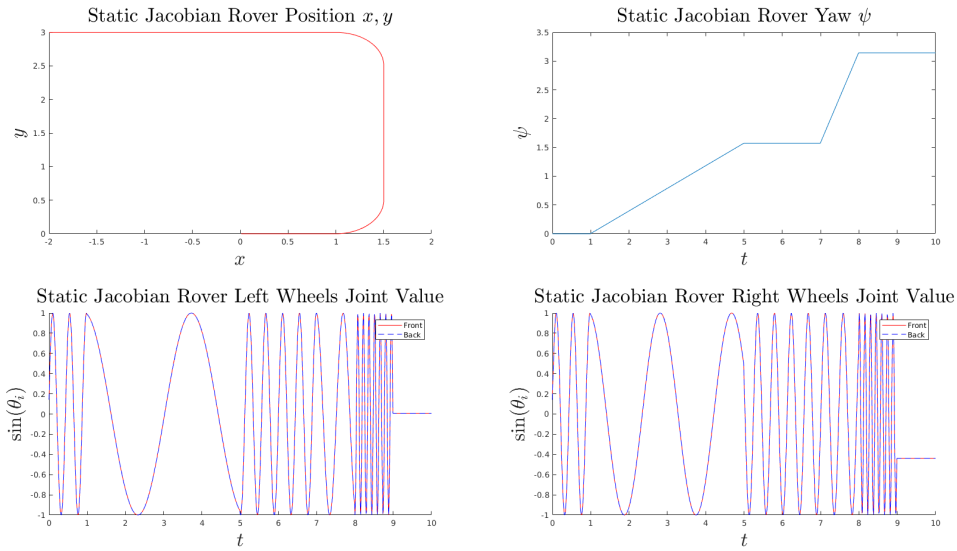

Figure 3: Results of the kinematic model being told to perform a couple drive arcs. Notice how the model is able to produce different speeds for individual wheels.

joint frames arranged at certain offsets from the body frame, this Jacobian would be more complicated, though similarly static. Results of using this Jacobian to calculate velocity kinematics and then updating the vehicle's state through time are shown in figure 3, which was generated in Matlab.

Now that the Jacobian and its slices and pseudoinverse are known, the implementation can work as shown by the architecture in figure 4.

$\dot{\theta}$ - Wheel velocity
v_b^w - Body velocity
$A_u^\dagger$ - Unknown value jacobian
A_k - Known value jacobian
w_c - Wheel contact constraints

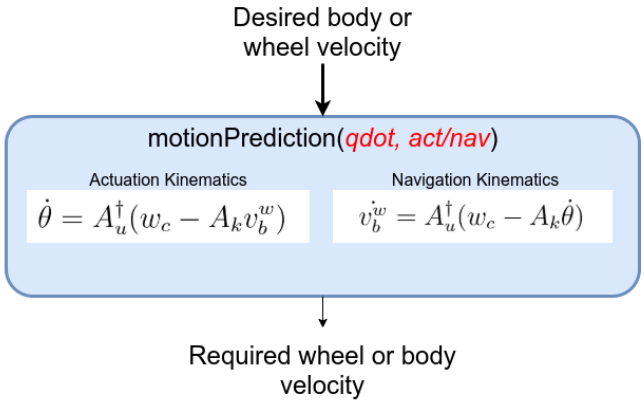

Figure 4: The kinematics code flow architecture.

A Proof of Jacobian Simplification

Recall from earlier that the formula for the Jacobian of wheel i is

$$\Theta_i \in \mathbb{R}^{3\times 4} \tag{12}$$

$$\Theta_i[:, i] = R_i^w[:, a(i)] \times (r_c^w - r_i^w) \tag{13}$$

$$A_i = \begin{bmatrix} [r_c^w - r_1^w]_\times^T R_1^w & R_1^w & \Theta_i \end{bmatrix} \tag{14}$$

Which is then assembled into the Jacobian for all four wheels A, and multiplied by the rotation matrix $A = (R_c^w)^T A$ which rotates the entire Jacobian into the contact frame of the rover. If we then apply this rotation to each individual part of the wheel Jacobian:

$$R_w^c \left([r_c^w - r_1^w]_\times^T R_1^w \right) = R_w^c (r_1^w - r_c^w) \times R_w^c R_1^w \tag{15}$$

$$= (R_w^c r_1^w - R_w^c r_c^w) \times R_1^c \tag{16}$$

$$= (r_1^c - r_c^c) \times R_1^c \tag{17}$$

$$= r_1^c \times R_1^c \tag{18}$$

We can see that this cross product is an expression of the contact to the base frame of wheel i. The middle matrix is simple: $R_w^c R_1^w = R_1^c$, another matrix referencing

contact frame from body frame. Lastly, the joint angle portion of the Jacobian:

$$R_w^c\Big(R_i^w[:,a(i)] \times (r_c^w - r_i^w)\Big) = R_w^c R_i^w[:,a(i)] \times (R_w^c r_c^w - R_w^c r_i^w) \tag{19}$$

$$= R_i^c[:,a(i)] \times (r_c^c - r_i^c) \tag{20}$$

So we now have expressions for the wheel Jacobian in the contact frame. The key assumption enabling this simplification is that the contact frame does not change with respect to the rover and therefore, the Jacobian can be treated as static. Without this assumption, this simplification does not hold, and the Jacobian would need to be recalculated in accordance with the current contact frame and joint angle. It is possible to model the contact frame as moving with the terrain, but that would require interfacing with the mapper to generate terrain contact points on the mesh, which would require more functionality be added to the vehicle controller and ultimately further complicate the system.

References

[1] Neal Seegmiller. *Dynamic Model Formulation and Calibration for Wheeled Mobile Robots*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, October 2014.