Mobile Robots 2017

Homework #2

WMR Kinematics and Dynamics

Note to myself - This file has conditional text formatting to hide the answers.

Generally, I'd prefer if you submit all homeworks entirely in electronic form. PDF files are best. See the web page for how to get PDF tools from Adobe for free.

You should do this and all homeworks alone without any undue help from your fellow students.

Grades in this course are based mostly on homeworks. Therefore, don't make the mistake of ignoring any questions asked. Answer each one explicitly.

Submit your results in files named {WMRKinematics}.YourAndrewId.java etc., and Hw5.YourAndrewId.pdf (or doc or whatever). You can also submit paper for the external documentation.

1. Wheel Constraints and Generalized Bicycle [38 Points]

Recall that Wheeled Mobile Robot (WMR) constraints express a disallowed direction for each wheel. This section will explore this idea in more detail. Recall the basic geometry of a wheel constraint:

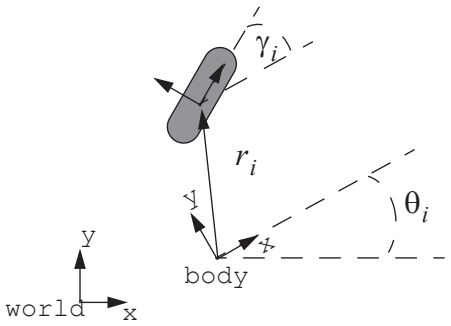

Figure 1. Wheel Model. Velocities in the direction orthogonal to the wheel direction are disallowed

The basic constraint equation is written in Pfaffian form in world coordinates as:

$$\left[(\hat{y}_i^w)_x \; (\hat{y}_i^w)_y \; (\hat{x}_i^w \cdot \underline{r}_i)\right]\left[{}^w v_x \; {}^w v_y \; \omega\right]^T = 0$$

where the subscript i indicates wheel i. Or, in terms of the steer angles and body frame yaw:

$$\left[-s\theta\gamma_i \; c\theta\gamma_i \; (\underline{r}_i \cdot \hat{x}_i)\right]\left[{}^w v_x \; {}^w v_y \; \omega\right]^T = 0$$

a) [2 points] Show how all three elements, when written out, of the above constraint formula (not just the first matrix) are elements of vector dot products.

b) [2 points] Since they are dot products, they can be expressed in any coordinate system we like. Rewrite the second form of the constraint in body coordinates.

c) [2 points] Any vehicle moving around an Instantaneous Center of Rotation (ICR) can be represented in terms of a generalized bicycle as shown:

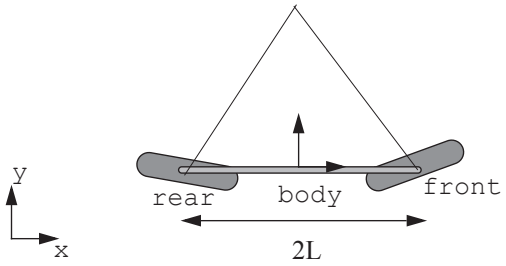

Figure 2. Generalized Bicycle. The two wheels must be steered appropriately to meet at the ICR. The ICR can be anywhere so the steer angles are not equal in general.

Write the two independent disallowed direction constraints for the generalized bicycle in body coordinates as a single matrix equation. That's both constraints in one matrix equation with two rows.

In this and all future questions, assume the wheels are positioned arbitrarily on the vehicle at a position (r_x, r_y) which is different for each wheel. Write r_1 for the quantity $\underline{r}_1 \cdot \hat{x}_1$ etc.

d) [4 points] There are three degrees of freedom $\begin{bmatrix} v_x & v_y & \omega \end{bmatrix}^T$ for the generalized bicycle but only two constraints - leaving one dof in the constraint nullspace.

i) [2 points] Solve the two constraint equations from the last question for the linear velocities assuming that ω is known. Its not necessary to invert the matrix explicitly. Just write it with a -1 exponent.

ii) [2 points] Solve for v_y and ω assuming v_x is known. Its not necessary to invert the matrix explicitly. Just write it with a -1 exponent.

e) [8 points] The previous results can be used to generate initial conditions that are consistent with the constraints. Suppose it is desired to start the vehicle with an initial curvature and initial linear velocity oriented precisely along the x axis by setting $v_y = 0$. Since there are no holonomic constraints, all elements of the initial state can be chosen arbitrarily, so we can set:

$$\underline{q}_0 = \begin{bmatrix} x_0 & y_0 & \theta_0 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$

However, if the intention is to start the vehicle with a horizontal initial velocity, this choice will not produce that result. Indeed, the constraints are violated by this setup. The velocity vector of the body frame is not necessarily oriented along the x axis of the body frame so setting θ_0 to zero does not place the velocity on the x axis. Even for a bicycle it is oriented at an angle to the forward axis as will be derived below.

i) [2 points] Use your solution to the first part of question d) but this time write the constraints in the world frame and use the equation for the inverse of a 2X2 matrix to explicitly invert the matrix.

Call the determinant “det” to reduce clutter.

ii) [6 points] Set the vertical component of the velocity to zero and solve for the yaw angle which is consistent with this condition.

f) [2 points] Provide a geometric interpretation for the set of all points (x,y) for which the third component of the Pfaffian weight vector (which is $(\underline{r}_i \cdot \hat{x}_i)$) for a given wheel is constant.

g) [4 points] Find an explicit formula for the set of all of the points (x,y) in the last question.

h) [14 points] The last result allows us to compute an equivalent generalized bicycle at any position for any steering configuration of any vehicle.

i) [6 points] Use the last result and the left pseudoinverse and show how to find the ICR corresponding to any number >=2 of steering constraints.

ii) [8 points] Write the equation that determines the steering angle given both the position (r_x, r_y) of a wheel anywhere in the body frame and the ICR position in the body frame.

2. Implementing a 4 Wheel Steering System [20 points]

Read the lecture slides on the solution for the kinematics of a wheeled mobile robot with 4 steered and offset wheels.

The file applicationHW3VehicleDynamics.java is provided in source form. The designation HW3 does not mean it is the third assignment, its just the third one I implemented. It contains the main driver routines including world creation, graphics etc. It configures a GUI and a drawing canvas using the Simple Mobile Robot Development Environment (“smrde”) to produce the figures needed in the exercises using Java code. You do not need to understand the code very well. Your job will be to fill in a few small holes in the code.

Most of your work will be done in the package applications.homeworks.unlocked.hw3 in the file WMRKinematics.java. This file is a copy of a working solution with small pieces removed. You should be able to run the program and select the VehicleDynamics homework. Once you have this displayed be sure that steerCtrl check box is unchecked and if not click it and press Reset. A reset applies all the values in the GUI fields to the program.

Then select Part 1 and Go and you should see a robot car driving in a circle without steering the wheels correctly.

2.1 4 Wheel Steer Without Wheel Offsets [15 points]

Read the javaDocs for `smrde.math.Matrix and smrde.math.Pose2D` thoroughly. Edit the file `WMRKinematics.java` to complete the following functions ...

```
WMRKinematics.worldToBodyVelocity();
WMRKinematics.setSteerAxisJacobian();
```

`WMRKinematics.bodyToSteerAxisVelocity();`

`WMRKinematics.steerAxisToBodyVelocity();`

`WMRKinematics.bodyToWorldVelocity();`

`WMRKinematics.computeDesiredWheelAngles();`

Most of these are called by the function `driveCar()` in ApplicationHW3VehicleDynamics.java. Now when you run the system with the `steerCtrl` box checked (and do a reset to read the GUi state into the application), you should see a car that is steering correctly for a circle.

Why all that work to compute 4 fixed angles you ask? Prepare to be amazed. The variable `OmegaGain` controls the angular velocity of the robot independently from the translatioonal. When it is 1.0 the car rotates once per circle. Set it to 0.0 (and press reset) and run and see what happens. The values 0.5 and 1.5 and 2.0 are also interesting.

Write out the math that describes your solution in an external documentation file. You can copy and paste from the slides. I just want see that you could find the right math and use it. Take a screen capture near the top of the circle for the case where `OmegaGain` = 0.0 and include it in your external documentation as well. There is a button for screen capture in png format. You can press stop to stop the graphics at the right place before doing a screen capture.

2.2 4 Wheel Steer with Wheel Offsets [5 points]

So far, the solution has cheated by assuming that the wheels were at the steer axes and the wheel velocities were wrong even though the angles were right. There is no way to tell this from the graphics. Now for the exact solution. Complete the remaining incomplete functions ...

`WMRKinematics.setWheelJacobian();`
`WMRKinematics.steerAxisToWheelVelocity();`

`WMRKinematics.wheelToSteerAxisVelocity();`

Turn on the `offsetSteering` checkbox in addition tot he `steerCtrl` checkbox and run it again.

Do a brief writeup of the math similar to part i) but no need for a screen capture this time. I will run your code to make sure it works.

3. Implementing Lagrangian Dynamics[24 points]

In this section, you will implement wheel, rigid, and rotary constraints to make a bicycle and a car drive on screen. Five different test cases are provided. In increasing order of complexity, these are

- Part 2a) a wheel that turns based on constant applied torque.
- Part 2b) a bicycle that drives in a circle
- Part 2c) a bicycle that steers
- Part 2d) a car that drives in a circle
- Part 2e) a car that steers

As you develop and debug your the solution, the bike or car may slow down, wiggle somewhat, and explode as the constraint loops go unstable. Try setting `defaultDeltaTime` to something

like 0.3 to force this behavior. Who knew dynamics was so much fun?

This example will give you a taste for how you can write a general kinematic or dynamic simulator for any kind of vehicle. Unlike the MATLAB example in the course notes, this example is a multibody example, so there will be pose coordinates for each wheel and for the body in the state vector. There will be 9 degrees of freedom for a bicycle before constraints are applied and 15 for a car. All of the bodies will move as a unit because you will write constraints that remove degrees of freedom and thereby force them to do so.

Be sure you understand the difference between the multibody and unibody bicycle examples in the notes before you proceed. Also, the equations for rigid and rotary joint constraints in the plane were provided in the numerical methods section of the notes.

3.1 How the Code Works

One of the most important classes is `WMRRigidBodyDynamicsInterface.` It sets up an association between the rigid bodies that comprise the test vehicle and where their associated state variables are placed in the system state vector. It also associates constraints with indices in the constraint Jacobian. In this way, any number of bodies and constraints can be supplied. The number of bodies and constraints must be fixed at initialization time so the process is to "register" them with the interface until you have described your body, and then tell the system when you are done. Once you are done, the layout of all the matrices becomes known and you can start simulating.

A SMRDE object that can acts like a rigid body implements the `Forceable` interface. The `WMRRigidBodyDynamicsInterface` class creates a `WMRDynamicsBody` from each such (`Forceable`) body. Such objects provide an interface between the matrices and vectors needed to implement a Lagrangian dynamics model and the details of SMRDE's internal world model and display list entities. In particular, the initial inertial state of `WMRDynamicsBodies` is read from SMRDE and the motion generated by the dynamics engine is written to SMRDE. The net effect of all that is that the graphics will update automatically when you change the numbers in the state vector using constraints and/or integration. Inside SMRDE, the inertial state of objects is not maintained because the main world model is a tree of poses. However, inertial state it can be computed for any body and it can be set for any body by accessing and/or changing the appropriate relative states in the tree. The `WMRRigidBodyDynamicsInterface` class also provides the method `computeMotion` which is the point of entry into the dynamics integration process.

The `WMRRigidBodyDynamicsInterface.computeMotion` method calls a method of the `WMRRLagrangeDynamics` class (also called `computeMotion`). This will then call a method of `SecondOrderDEIntegrator` called `integrateDynamics` which will in turn call `DEIntegrator.integrateDynamics` which will use Runge Kutta to do the integration. At the time this code was written, the need for constraint trim was not well understood so there is no constraint trim and the system is not as stable as it could be. However, there are Baumgarte stabilization loops and these are good enough for this purpose.

3.2 Wheel Constraint [6 points]

Edit the file WMRWheelConstraint.java. Complete the functions `Jacobian()` and `Fd()`. Once these are correct, the wheel should drive in a circle. Write up the math you used with a brief explanation. You should probably write it first before you change the code so that you have a plan.

3.3 Rigid Constraint [6 points]

Edit the file WMRRigidConstraint.java. Complete the functions `Jacobian()` and `Fd()`. Once these are correct, the bicycle should drive in a circle. Write up the math you used with a brief explanation. You should probably write it first before you change the code so that you have a plan.

3.4 Rotary Constraint [6 points]

Edit the file WMRRotaryConstraint.java. Complete the functions `Jacobian()` and `Fd()`. Once these are correct, the bicycle should drive in a sinusoidal steering configuration. Write up the math you used with a brief explanation. You should probably write it first before you change the code so that you have a plan.

3.5 Consistent Steering [6 points]

Edit the file WMRSteeringController.java and complete the functions `consistentLeftSteerAngle and consistentRghtSteerAngle.` These compute the steer angles of the car front wheels which are consistent with a central wheel between the two - in others words, which is consistent with a bicycle of the same wheelbase at the same place. You can use your solution to problem 1)h)ii) or something similar. Once these are correct, the car should drive in either configuration. Write up the math you used with a brief explanation. You should probably write it first before you change the code so that you have a plan.

Note that when the bicycle steer angle is a small negative number, the expression for the radius of curvature:

$$R = \frac{L}{\tan\alpha}$$

produces a negative radius of curvature because the curvature is negative (to the right). If this expression were to be solved for the steer angle:

$$\alpha = \operatorname{atan2}(L, R)$$

The computed angle comes out as PI plus the small negative angle you want. This and related issues can be solved by using the sgn() (signum function) of the steer angle in strategic places. You will need to do this to tweak your formulas for the other two steer angles.