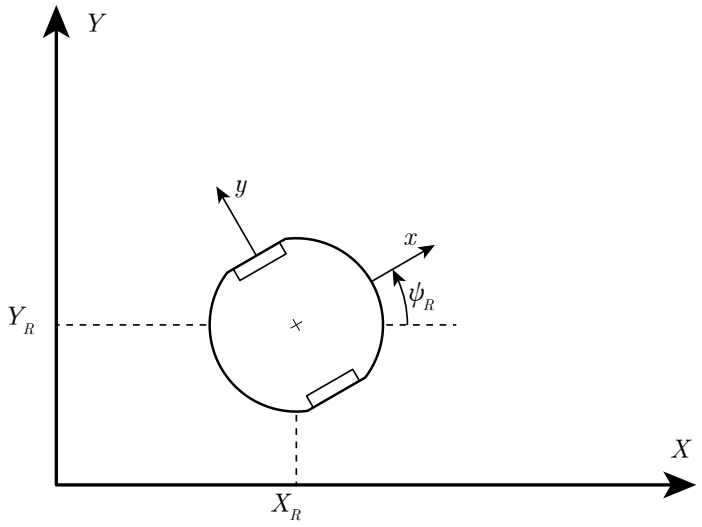
Homework 0x03 - System Modeling and Simulation

The coding portion of this homework assignment is to be completed individually; however, it does relate to your term project so you will be allowed to collaborate with your teammates on the modeling portion of the assignment. Students unfamiliar with dynamic modeling, such as those outside of the ME department, are encouraged to work with their ME classmates on the modeling aspect of the assignment. Your lecture and lab instructors may also be able to provide some resources as well.

The assignment will be due on Monday, November 27th before 11:10 AM after Thanksgiving break.



Background

For this assignment, you will work with an interactive notebook, called Jupyter, to write Python code. Jupyter allows you to combine code, images, equations, and the output from your code, including plots, all in an interactive web interface. It is the ideal tool for this type of homework assignment because it will allow you to present your analysis and results in the same document where your code lives.

The goal for this assignment is to develop a simplified model of the mechanical system that you are designing for your term project. That is, you will come up with a simplified differential drive model for your Romi robot and then you will simulate this model using a solver of your own implementation.

You will additionally get some practice working with tools like NumPy, which allow you to perform more sophisticated mathematical operations including tools from linear algebra.

Modeling

The math and techniques needed to model a differential drive robot are no more complex than what is covered in ME 212, Dynamics, and ME 322, System Dynamics. Your instructor `may` provide additional support in lecture or through supplemental notes regarding modeling techniques and equation generation, but that will depend on scheduling and other lecture topics that may benefit your lab progress; do not wait to begin the modeling for this assignment.

In general, models of mechanical systems come in two varieties: kinematic models, and full dynamic models. Kinematic models do not involve forces, masses, or Newton's laws of motion and are therefore much simpler, but lower fidelity. As a result, these models do not account for things like wheel slip or motor torque. To account for these effects, a full dynamic model is required. For the sake of this assignment it is *strongly advised* to work with a kinematic model. A kinematic model will require the assumption that the wheels of the robot do not slip on the ground which should be largely correct unless the robot is driving very quickly or bumps into obstacles.

The following steps will help guide you through development of your kinematic model.

1. Start by drawing a detailed, but simplified, schematic of your system with all important geometry clearly defined. Things like the wheel radius and track width will be necessary, but you should also consider defining a "point of interest" on your robot located some distance away from the origin at the center point between the two wheels. That way your simulation can help you determine information at the location where your sensors are mounted even if they're not located at the origin.
2. Consider the kinematics associated with the robot and determine equations or expressions for the following parameters:
   - The angular velocity of the robot with respect to its environment.
   - The velocity of the robot origin in a coordinate system attached to the robot.
   - The velocity of the robot origin in a coordinate system attached to the environment.
   - The velocity of the point of interest in a coordinate system attached to the environment.

   You may also want to work out the geometry of the system as well. In other words, you will want to be able to determine the location of the point of interest in a coordinate system attached to the environment from the robots orientation and the location of its origin. As you develop these equations consider what information you will have access to, such as the angular velocity of the two wheels. Consider these known parameters as the system inputs.
3. Formulate a set of differential equations that incorporate all the kinematic relationships you've determined. That is, choose a set of state variables associated with the robot at an arbitrary location and orientation, and develop a first-order ODE for each state variable. Compose a set of state equations from these first-order ODEs to represent the system dynamics in the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ where the components of $\boldsymbol{x}$ are your state variables and the components of $\boldsymbol{u}$ are your system inputs.
4. Formulate a set of output equations that relate the state variables to desired output parameters that you want to plot or study. That is, come up with a set of desired outputs from your simulation, some of which may be state variables themselves, and work out the relationship between each output and your state variables and inputs in the form $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$.

The Assignment

Create an interactive Python notebook within Jupyter or Jupyter Lab.

1. If you do not have Jupyter Notebook or Jupyter Lab installed on your computer, please install one or the other before beginning the assignment.
2. Begin by familiarizing yourself with the provided Jupyter Notebook file that implements an extremely simple simulation of the DC motor model that was or will be covered in lecture. That is, run the program exactly as provided and read through the commented source code to understand how the different aspects of the program relate to one another.
3. The solver, as provided, implements a first-order forward Euler integration method. That is, the solver uses a first-order integrator to predict the future state of the system in terms of the present state of the system. As you may have learned in other courses, first-order integration methods are extremely low performing and will require a very large number of solution steps.

   A second solver declaration can be found in the provided file for a higher-order method called the Runge Kutta method. This solver is yet to be implemented, but you will complete the implementation to improve your simulation results. All of the equations are listed in the Jupyter Notebook, so you will just need to implement the code.
4. Once you're familiar with the the file as provided and have implemented the RK4 solver, you will make a copy of the example ODE function and customize the copy for your differential drive model. You will need to determine the approximate geometric properties for your system as required by the equations you found in the Modeling section above.

   As you modify the Jupyter document, replace or modify the existing mathematical derivation with your own derivation for your system dynamics. Follow a similar style to the example document, but you will be allowed to use hand-written analysis as long as it is completed neatly and scanned properly. Jupyter will allow you to directly embed images as needed.
5. The provided template includes an additional ODE function representing the closed-loop system dynamics. For the purpose of this assignment you can ignore this section of code, although it may be useful for you to experiment with different control techniques for the sake of your term project.
6. To complete your assignment, simulate the robot driving in a circle of a prescribed radius at a known velocity. That is, choose a velocity and a radius for the robot to follow and from those values determine the input parameters (wheel speeds) needed to achieve such a pattern in your simulation output. Then use these inputs to run your simulation.
7. Produce a set of plots showing your simulation results:
   (a) Plot the output variables you've selected, each against time, on a series of plots.
   (b) Plot the path traveled by the robot on a set of equally scaled X- and Y-axes to show that the simulation results do indeed draw a circle of the radius you selected.

Deliverables

Please submit on Canvas two files: your Jupyter notebook as a ".ipynb" file and your notebook exported as a ".html" file; the html output will allow the grader to see your output on Canvas while grading. Remember that although this assignment is meant to be collaborative, each student must complete their own code and submit unique files to Canvas.

- Include a moderately thorough dynamic analysis of your system design with very thorough annotation. This should include analysis of both axes of motion with the goal of developing differential equations of motion.

  Present the final differential equations as formatted equations clearly in the body of your Jupyter notebook in addition to the supporting analysis which can be done neatly by hand and scanned. In your analysis, determine if the two axes are coupled with one another and mention this clearly in your notebook as well.

  Equations can be entered using LaTeX markup. You may need to do a bit of reading to figure out the basics of equation formatting, or you can toil through the equations in the provided example and then modify them to suite your needs.

- Show a table of estimated geometric parameters used in your kinematic model. Do your best to estimate numeric values, however at this point its more important to know what parameters to include in the model than it is to know their precise value. You will have the opportunity to continuously modify your simulation parameters as the term comes to an end.

  It is easy to put tabular data into Jupyter because you are allowed to use Markdown cells in addition to the code cells within your notebook.

- Produce the plots listed at the end of the Assignment section above.

- Answer the discussion questions listed on the next page with thoughtfulness and attention to detail.

Discussion Questions

1. Explore the validity of the no-slip assumption and any other assumptions required for the kinematic model in favor of a full dynamic model. Consider things like how the motors may or may not easily integrate with the model, how the mass and friction properties of the system may influence the model, and any other effects that may not match the idealized nature of the kinematic model. Is the kinematic model good enough for our purposes?

2. One of the problems with models like this is that significant deviation or drift will occur between the simulation and the behavior of the real vehicle, especially when accumulated over a long simulation. The concept of dead reckoning is that we can predict the position of a system simply by integrating its velocity or acceleration over time. Naturally, small deviations, like those caused by minor wheel slip, add up over time. A small error in velocity, once integrated over time to produce position, can lead to wildly inaccurate estimations of position.

   How can we correct for this drift error? What other information would help us keep our simulated model tracking with reality? **Hint:** consider what kind of sensors may be added to the robot to measure the value of your selected state variables.

3. While this is not a controls course, you will need to implement some kind of controller on your robot to complete the term project. For this question, investigate ways in which you may incorporate feedback into your system to make the robot move in ways that will be advantageous for the term project. Address nuances such as:

   - What sensor inputs will be useful for feedback.
   - How you might split up a complicated path into segments such as straight lines, circular arcs, in-place turns, etc.

4. Is it possible to determine the orientation and location of the robot after it has followed a specific path simply by knowing the angle swept out by each wheel? If not, would it be possible to know the orientation and location with a time-history of the angle swept out by each wheel? If you know the robot's orientation and location, can you figure out the angle swept out by each wheel?

   Investigate what information is required to track the robots absolute position in space, or possible its position relative to particular maneuver, like a circular arc. What kind of data will you need to track in your firmware to make use of this information?