1 What to Model

The goal is to create a dynamics model of a wheeled robot that can later be used for model predictive control. Specifically, given a state history estimate $[\mathbf{X}] = [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_t]$ and a control input history $[\mathbf{U}] = [\boldsymbol{u}_0, \ldots, \boldsymbol{u}_t, \boldsymbol{u}_{t+1}]$, the goal is to predict the state derivative $\dot{\boldsymbol{x}}$ given $[\mathbf{X}]$ and $[\mathbf{U}]$.

The desired model can be written as

$$\dot{\boldsymbol{x}} = f([\mathbf{X}], [\mathbf{U}]) \tag{1}$$

If we assume that the system is a Markov process (depends only on the current state and input), then the model can be written as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \tag{2}$$

In this case, $\boldsymbol{x}$ is a list of the robot's position, orientation, linear velocity, angular velocity, and the states of any moving sub-assemblies, actuators, and sensors on the robot. $\boldsymbol{u}$ is a list of inputs to the robot's actuators, including the motors which apply torque to the wheels.

The physics to be captured in the model includes rigid-body dynamics ($\boldsymbol{F} = m\boldsymbol{a}$), kinetic friction, and rolling friction. If the robot is not sufficiently rigid, robot elastics ($\boldsymbol{F} = [\mathbf{K}]\boldsymbol{x}$) may also be modeled. If the robot moves at sufficiently high speeds, aerodynamic drag ($D = \frac{1}{2}qSC_D$) may also be modeled.

Machine learning is useful because the plant model can be extremely complex, especially if nonlinearities such as friction, drag, and noise are taken into account. These effects, which are difficult to capture with analytical models, can be fit to using data-driven methods.

Because the model will be used for model-predictive control, it must be fast so that it can be executed repeatedly in real-time.

2 Data

Experimental and simulated data can be augmented by creating duplicates that have been rotated in azimuth, since the robot's response should be invariant to the direction it is facing (unless it has a magnetometer).

The coordinate system will be the body coordinates of the robot aligned with its inertial principal axes, so that complex interactions between degrees of freedom are minimized. If the model elastics are to be modeled, the elastic states should be in generalized model coordinates, again, to minimize interactions between states. To cirvumvent the singularities present in Euler representations of orientation, quaternions should be used for encoding orientation.

Ideally, experimental data would be used since it captures all discrepancies (friction, drag, etc.). Realistically, a combination of experimental (high-confidence) and cheaper simulated data (low-confidence) would be fused to train the model. Compared to experimental data, simulation data would be low-cost to generate, since it can be generated repeatedly with only a computer. However, it would fail to accurately capture friction, drag, structural dynamic damping, actuator free-play, and other real-world effects.

Because of this, poorly curated simulations could simulate scenarios which could not exist in real life (with extreme speeds, extreme bending, over-driven actuators, etc.) Thus, all training data must be carefully generated to capture the operating envelope of the robot; no more, no less.If

the training data exceeds the operating envelope of the robot, the model may capture unrealistic behaviour. If the training data does not cover the operating envelope of the robot, the model will be extrapolating at the boundaries of the envelope with unpredictable results, limiting the generizability of the model.

3 Architecture

The neural ODE architecture would best fit this modeling problem. The physical system being modeled is a continuous-time system with a state which evolves continuously (without discontinuities) over time. This system's dynamics are governed by ODEs, and thus can be captured by a neural ODE. Just by selecting the neural ODE architecture, these characteristics of the physical system are captured in the model. Additional physics can be captured by adding terms manually to the neural ODE. For example, if the neural ODE is given by $\boldsymbol{f}_\theta$:

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{f}_\theta(\boldsymbol{x}(t), t) \tag{3}$$

and if the first three states are cartesian position and the second three states are cartesian velocity, then the model can be modified to include an extra term to bring the derivative relationship outside of the neural ODE, simplifying the modeling problem for the optimization:

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \boldsymbol{x} + \boldsymbol{f}_\theta(\boldsymbol{x}(t), t) \tag{4}$$

The neural ODE is more generalizable compared to methods which are constrained to a fixed time step size, since it is formulated in continuous time and can be trained and evaluated on data with any intervals between points.

4 Loss Function and Optimization Method

Optimization will be used to tune the parameters of the neural ODE model to minimize the mean squared error between the predicted state derivative to the "truth" state derivative for each point in each data series. This loss function is canonical for regression tasks. Due to the complexity and multidisciplinary interactions of the physical features being modeled, this loss function is likely non-convex and potentially messy.

The minimization will be performed using stochastic gradient descent with gradients computed using reverse-mode automatic differentiation. This optimization algorithm is efficient for training neural networks on large datasets.

Physics could be enforced in the optimization by constraining it to only generate models for which the energy of the system is guaranteed to be non-increasing. However, formally formulating such a constraint would be challenging. If a large training dataset is available, this constraint could be approximated by only generating models for which the energy of the system is non-increasing over all training data points.