Lecture 6: Vehicle Modeling

Professor Katie Driggs-Campbell
February 25, 2021

ECE484: Principles of Safe Autonomy
Administrivia

• Late policy has been posted → four “grace” days

• Oral exam update
  ▪ Monday 4/12 to Tuesday 5/4
  ▪ 15-minute time slots on Mondays 5-6pm, Tuesdays 5-6pm, Wednesdays 11am-12pm, or by appointment (if needed)
  ▪ Questions and rubric released by 4/5
  ▪ Decreasing bonus points, e.g.: (9-session_num)/10*bonus_points ← tentative
Today’s Plan

• What’s a model?
• Planning and Control Motivation
  ▪ Open-loop control
• Vehicle Models
  ▪ How to design your model
  ▪ Dubin’s Car
  ▪ Advanced Models: bicycle, tire dynamics
Environment & Agent Models

Sensors

Perception

Decision-Making

Trajectory Planning

Low-level Control

Compute Platform

Simulation & Validation
Typical planning and control modules

- **Global navigation and planner**
  - Find paths from source to destination with static obstacles
  - Algorithms: Graph search, Dijkstra, Sampling-based planning
  - Time scale: Minutes
  - Look ahead: Destination
  - Output: reference center line, semantic commands

- **Local planner**
  - Dynamically feasible trajectory generation
  - Dynamic planning w.r.t. obstacles
  - Time scales: 10 Hz
  - Look ahead: Seconds
  - Output: Waypoints, high-level actions, directions / velocities

- **Controller**
  - Waypoint follower using steering, throttle
  - Algorithms: PID control, MPC, Lyapunov-based controller
  - Lateral/longitudinal control
  - Time scale: 100 Hz
  - Look ahead: current state
  - Output: low-level control actions
What is control?

- A means of regulating or limiting something
- Algorithms (or process) for manipulating a system to achieve to desired value
Open-Loop Example

- **Temperature Sensor**
- **Controller**
- **Heater**

- **Check temperature every 30 minutes**
- **If temperature $\leq 70$ then heat on for $T$ minutes**
- **If temperature $\geq 75$ then heat off for $T$ minutes**

Temperature: 70
Closed-Loop Example

Goals
if \( v > V_d \), want \( + \) input
if \( v < V_d \), want \( - \) input
if \( v \approx V_d \), want a small or no input

Look at errors instead!

\( e = V_d - v \)
Lecture 2: Modeling the scenario

• What is a model of a system?

• A *mathematical model* describes how a system behaves.
  - What are the key parameters and states?
  - How are the parameters selected by nature?
  - What are the initial conditions of the state?
  - How do the state change over time?
  - What parts of the model are available for observation/analysis?

• Models include the implicit and explicit assumptions (biases) we are making about the system
Breakout Room Discussion

1. Pick an AV scenario and try to think of the desired behaviors, the requirements of the controller, and the actions / inputs to the system.

2. How would you define your model for your scenario?  
   Hint: think of the model and simple controller in MP0

3. Other than control, what are some other use cases of models?
1. Pick an AV scenario and try to think of the desired behaviors, the requirements of the controller, and the actions / inputs to the system

2. How would you define your model for your scenario?

3. What are some other use cases of models?

---

Drones
- input: fans, propellers
- improved control design

Cars
- ACC: maintain distance
- regulate error

Drive-thru Bot
- many tasks!
- Following a curve
- steering + ACC

MPO
- improved control design
Coordinate Systems and Configurations

Car pose = \[
\begin{bmatrix}
  x \\
  y \\
  \theta
\end{bmatrix}
\]

End effector pose = \[
\begin{bmatrix}
  x \\
  y \\
  z \\
  \theta_x \\
  \theta_y \\
  \theta_z
\end{bmatrix}
\]
Dynamical Systems Model

Describe behavior in terms of instantaneous laws:
\[
\frac{dx(t)}{dt} = \dot{x}(t) = f(x(t), u(t))
\]

where \( t \in \mathbb{R}, x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m \), and \( f: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n \) gives the dynamics / transition function.
Mobile Robotics: Differential Drive

Instantaneous Center of Curvature:

\[
\begin{bmatrix}
I_{cc_x} \\
I_{cc_y}
\end{bmatrix} = \begin{bmatrix}
x - R \sin \theta \\
y - R \cos \theta
\end{bmatrix}
\]

\[ v_r = \omega (R + \frac{l}{2}) \]
\[ v_e = \omega (R - \frac{l}{2}) \]
\[ R = \frac{l}{2} \frac{(v_r + v_e)}{(v_r - v_e)} \]

\[ \omega = \frac{v_r - v_e}{l} \]
Simple vehicle model: Dubin’s car

• Key assumptions
  ▪ Front and rear wheel in the plane in a stationary coordinate system
  ▪ Steering input, front wheel steering angle $\delta$
  ▪ No slip: wheels move only in the direction of the plane they reside in

• Zeroing out the accelerations perpendicular to the plane in which the wheels reside, we can derive simple equations

Dubin’s Model

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= \frac{v}{l} \tan \delta
\end{align*}
\]
Many more advanced models...

[Kinematic] Bicycle Model

[Dynamic] Tire Models


Dynamical system models

**Dubin’s car model**

\[
\begin{align*}
\dot{v} &= a & \text{Speed} \\
\frac{ds_x}{dt} &= v \cos(\psi) & \text{Horizontal position} \\
\frac{ds_y}{dt} &= v \sin(\psi) & \text{Vertical position} \\
\frac{d\delta}{dt} &= v\delta & \text{Steering angle} \\
\frac{d\psi}{dt} &= \frac{v}{l} \tan(\delta) & \text{Heading angle}
\end{align*}
\]

**Physical plant**

\[
\begin{align*}
\frac{dx}{dt} &= f(x, u) & \text{System dynamics} \\
x[t + 1] &= f(x[t], u[t]) \\
x &= [v, s_x, s_y, \delta, \psi] & \text{State variables} \\
u &= [a, v\delta] & \text{Control inputs}
\end{align*}
\]

Nonlinear dynamics

Generally, nonlinear ODEs do not have closed form solutions!
Nonlinear **hybrid** dynamics

**Physical plant**

\[ \frac{dx}{dt} = f(x, u) \]

System dynamics

\[ x[t + 1] = f(x[t], u[t]) \]

State variables

\[ x = [v, s_x, s_y, \delta, \psi] \]

Control inputs

\[ u = [\alpha, v_\delta] \]

**Decision and control software**

- speed up
- merge left
- close to front car
- cruise
- merge right
- speeding
- slow down
Typical system models

Hybrid system model

Physical plant
\[
\frac{dx}{dt} = f_{mode}(x, u)
\]

Decision and control software

- **speed up**: \[ \frac{dx}{dt} = f_{up}(x, u) \]
- **merge left**: \[ \frac{dx}{dt} = f_{left}(x, u) \]
- **cruise**: \[ \frac{dx}{dt} = f_{cruise}(x, u) \]
- **merge right**: \[ \frac{dx}{dt} = f_{right}(x, u) \]
- **slow down**: \[ \frac{dx}{dt} = f_{down}(x, u) \]

Nonlinear *hybrid* dynamics

Interaction between computation and physics can lead to unexpected behaviors
Summary

• Dynamical systems models allow us to reason about low-level behaviors of systems and determine what is and is not feasible
  ▪ Typically required to design controllers!
• Discussed a few types of models from simple to complex
• Next time: Look at simple PID control design for trajectory following
Extra Slides
An aside: Coordinate transformations
Rotation matrix

The following matrix rotates a vector \([x, y]\) counter-clockwise by an angle of \(\theta\)

\[
R(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\]

That is:

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}
\]

Derivation
\[
x' = r\cos(\beta + \theta) = r(\cos \beta \cos \theta - \sin \theta \sin \beta)
\]
\[
= r\cos \beta \cos \theta - r\sin \theta \sin \beta
\]
\[
= x \cos \theta - y \sin \theta
\]
Path following control

- The path followed by a robot can be represented by a trajectory or path parameterized by time → from a higher-level planner
- Defines the desired instantaneous pose $p(t)$

$$p(t) = [x(t), y(t), \theta(t)]$$
Path following control

• Desired instantaneous pose $p(t)$
• How to define error between actual pose $p_B(t)$ and desired pose $p(t)$ in the form of $y_d(t) - y(t)$

$p(t) = [x(t), y(t), \theta(t)]$

$p_B(t) = [x_B(t), y_B(t), \theta_B(t)]$
Path following control

The error vector measured vehicle coordinates

\[ e(t) = [\delta_s(t), \delta_n(t), \delta_\theta(t), \delta_v(t)] \]

\([\delta_s, \delta_n]\) define the coordinate errors in the vehicle’s reference frame: along track error and cross track error

- **Along track error**: distance ahead or behind the target in the instantaneous direction of motion.
  \[ \delta_s = \cos(\theta_B(t)) (x(t) - x_B(t)) + \sin(\theta_B(t)) (y(t) - y_B(t)) \]

- **Cross track error**: portion of the position error orthogonal to the intended direction of motion
  \[ \delta_n = -\sin(\theta_B(t)) (x(t) - x_B(t)) + \cos(\theta_B(t)) (y(t) - y_B(t)) \]

- **Heading error**: difference between desired and actual orientation and direction
  \[ \delta_\theta = \theta(t) - \theta_B(t) \]
  \[ \delta_v = v(t) - v_B(t) \]

→ Each of these errors match the form \( y_d(t) - y(t) \)

\[ p(t) = [x(t), y(t), \theta(t), v(t)] \]

\[ p_B(t) = [x_B(t), y_B(t), \theta_B(t), v_B(t)] \]
Open-loop waypoint following

• We can write an open-loop controller for a robot that is naturally controlled via angular velocity, such as a differential-drive robot:

\[ u_{\omega,OL}(t) = \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \frac{\dot{x}(t)^2 + \dot{y}(t)^2}{\sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}} \\ \dot{\theta}(t) \end{bmatrix} \]

• We can write an open-loop controller for a robot with car-like steering:

\[ u_{\kappa,OL}(t) = \begin{bmatrix} v(t) \\ \kappa(t) \end{bmatrix} = \begin{bmatrix} \frac{\dot{x}(t)^2 + \dot{y}(t)^2}{\sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}} \\ \dot{\theta}(t) \end{bmatrix} \]