Autonomous Control Systems
Notes and Problem Packet

LACC 2019 Summer Program:

Instructor: Nat Snyder

Day 2:

Day 2:

Agenda

Part 1: Examples of building the "open-loop" control problem (state-space)

Part 2: Position and velocity responses of the system to "unit"-step inputs

Part 3: Intro to feedback control; sensors and algorithms used for real world control problems

Part 4: Course Project: Designing a ① "way-point" (position) controller and a ② "cruise" (velocity) controller for an autonomous car using python.

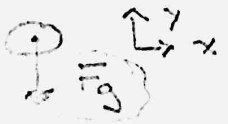
Part 5: (Extra) Derivation of Least Squares and applications to the Stock Market

Part 1/5

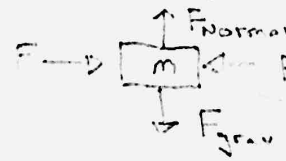
Revisiting FBD's and forming the "Open-Loop" System Dynamics

Key ideas for building your system:

- So far we have seen forces act on objects, such as gravity on a ball



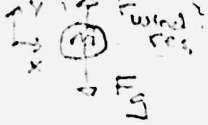
- We also have seen friction (kinematic) being modelled in the FBD:



$F_{fric} = \mu \cdot F_N$

$\mu \in [0, 1]$ (usually)

- We saw that "drag" (wind resistance) can be modelled as well:

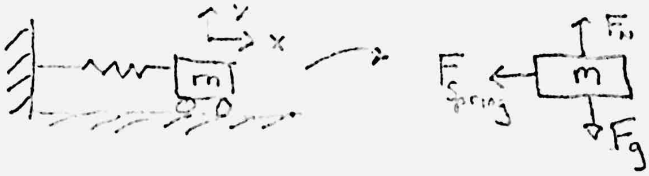


We described wind resistance as proportional to speed:

$F_{wind} = \beta \cdot V(t)$

β: some number; $V(t)$: speed of mass

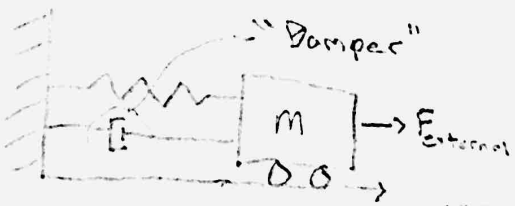
- We can also model things like springs:



$F_{spring} = K \cdot x(t)$

$K \rightarrow$ "Stiffness"
units (N/m)

- We model friction in systems sometimes like we do with wind resistance → we call this energy loss "Damping"

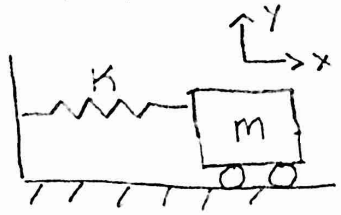


$F_{damp} = c \cdot V(t)$

c: damping constant; V(t): speed

Part 1/5: • With knowledge (and practice) identifying these forces, we can set-up the "system-model" for Control.

Example #1:



• A spring with stiffness K is attached to a cart with mass M. The floor has little to no friction. (No Damping)

a) Draw FBD of mass

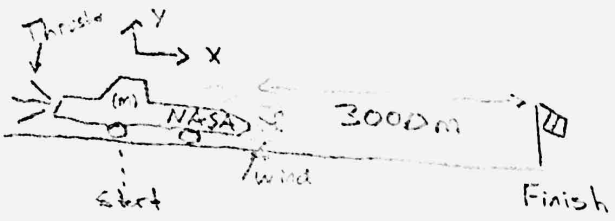
b) Put equation for a_{net_x} into "Linear-System" form ("state-space")

$$\dot{x} = Ax + Bu$$

c) If the cart starts at $x = .2m$ (spring is stretched), will the object come to a stop eventually?

Example #2:

A Nasa supercar uses a rocket thruster and sleek vehicle body to achieve extremely high speeds. The team wants a state space system of equations so it can optimally control the rocket thrusters. Assume wind resistance coefficient of $B=.1$



mass = 40 kg

$\Sigma = 0.15$

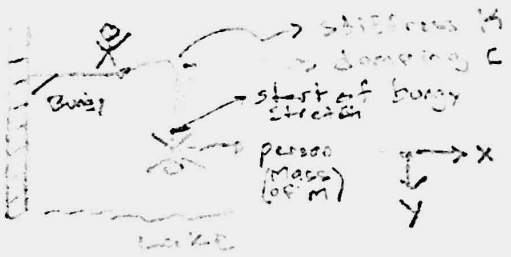
a) Draw FBD of car:

b) Put equations into state space form: $\dot{x} = Ax + Bu$

c) Will this supercar slow down to rest (eventually)? Does this make sense with respect to #1c's result? (assuming thrusters are cut-off)

Part 1/5

Example #3:



A bungy jump company would like to track the motion of a jumper until he or she comes to rest. They'd also like this model so they can predict if another bungy chord should be used (with less or more stiffness).

- Assume the following:
  - → Mass of person: m
  - → stiffness of bungy: k
  - → damping in bungy: c

a) Draw FBD of person

b) Put equations into state-space form $\dot{x} = Ax + Bu$

c) Sketch what you predict the vertical motion of the bungy-jumper would look like.

Part 2/5

Using python to find the system's response to control commands

Key ideas for dynamics with python:

- We now have a handle on drawing FBD's and writing the governing dynamics $\dot{x} = Ax + Bu$

- Now we can see how the system responds to initial conditions and control commands ("u") using python.

Steps to build state-space system in Python:

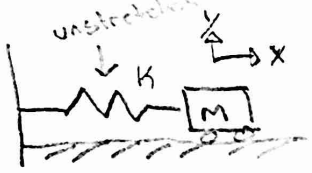
1) import scipy.signal as sig
2) import numpy as np
3) import matplotlib.pyplot as plt
4) create numpy matrices for A, B
5) create diagonal matrix of 1s the same dimensions as A. Call it C
6) If $A \in \mathbb{R}^{n \times n}$ & $B \in \mathbb{R}^{n \times p}$, create matrix of zeros of size nxp. Call it D
7) create state space object
   ss1 = sig.StateSpace(A, B, C, D)

- Once we have built our state-space object, "ss1" we can apply actuation commands or find how the system responds to an initial position or velocity.

11

Initial condition example:

Example # 4: Consider the scenario in example #1, the cart-spring with no friction from the floor. Suppose the cart was moved 0.5 meters to the right of its unstretched position and the cart is let go with zero initial velocity.



"Initial conditions" here are:

initial position (x_0) = .5 m from origin

initial velocity (V_0) = 0 m/s

mass = 5 kg

Spring stiffness = 10 N/m

* use function lsim() in scipy.signal

a) use state-space equations from Example #1, create a python state space object.

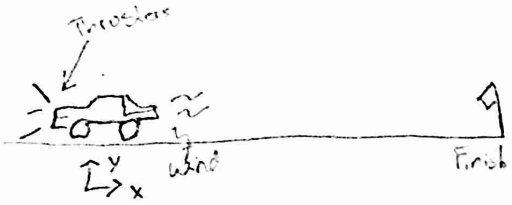
b) * Plot the response for $x_0 = .5m$ & $V_0 = 0$ m/s (using matplotlib)

c) Redo for $x_0 = .2$ m and $V_0 = 5$ m/s (plots)

System response to control commands & IC's of zero

$X_0 = 0, V_0 = 0$

Example #5: Reconsider example #2's setup. We will set the thrusters to exert 100 N of force.



$m = 40$

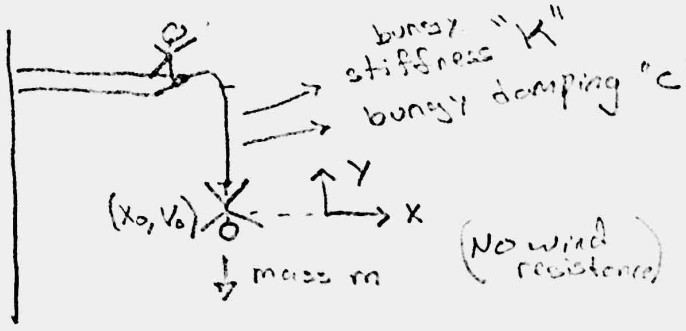
a) Find state space system, make scipy state-space object.

b) Plot "response" → (pos, vel) when the driver has the gas-pedal pushed to 50 N Force (input "u") by using the scipy "step" function. Assume initial position is 0m and velocity is 0 m/s.
→ Plot for 10 seconds.

c) What would the response look like if he had pushed pedal to 100N? (Plot again).
(Hint: Increasing the input: (1 → 100) increases output by the same gain).

Example #6:

(System response to input (u) commands and initial conditions)



$m = 80\ kg$

$K = 400\ N/m$

$c = .3\ (Ns/m)$

- Lets revisit the bungy jumper example. We'd like to plot the position and velocity responses when the bungy rope is taunt, just before the rope starts to stretch. Assume the jumper is at $x_0 = 0m$ with $v_0 = -20\ m/s$, when the experiment starts, at time $t = 0$.

a) Sketch FBD, write state space system, put system into scipy state-space object.

b) Plot the bungee-jumper's position + velocity responses for $t = [0, 30s]$ (using (matplot + scipy))

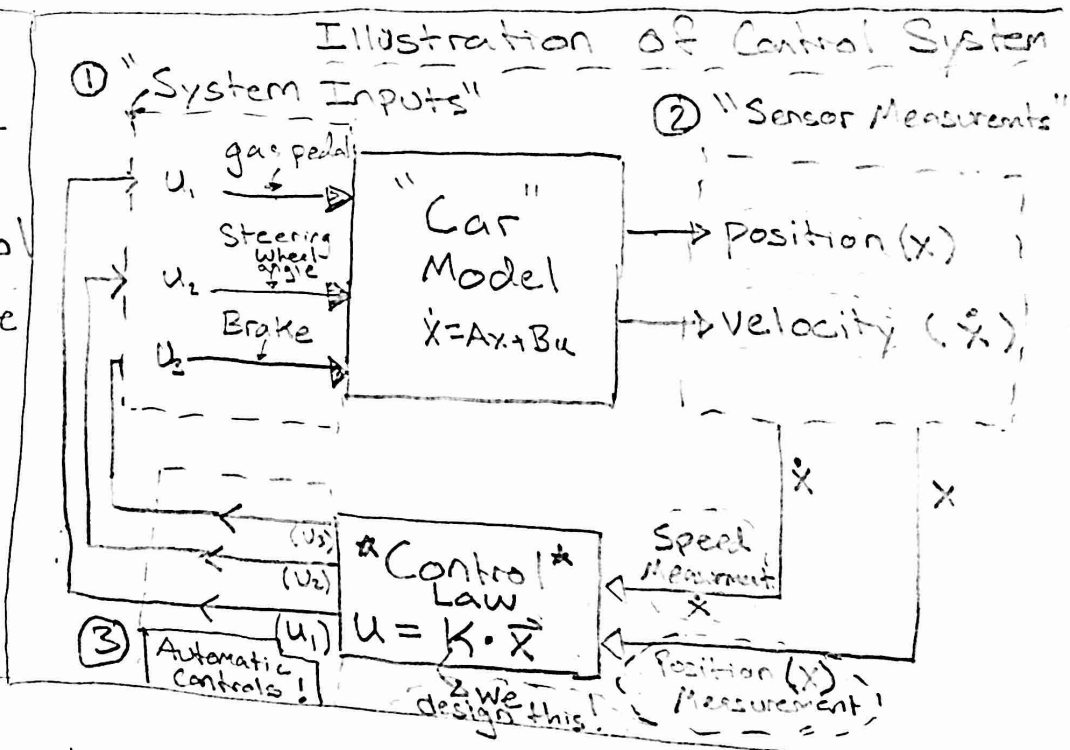
(Hint: since "gravity" is a system "input" "u" → on the "B" term, Use step function, since gravity is always acting on jumper.)

Part 3/5!

Intro to Automatic Optimal Control

- In control theory, we rely on sensor instrumentation (Camera, INU, GPS, Radar, Lidar) to tell us what speed objects are moving at or where (position) the object is.

- We can use sensors to build control laws to move cars, robots, rockets, etc automatically



- There are two types of optimal control:

  Regulating → Keeping object at rest at origin.

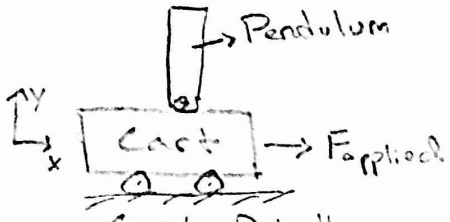
  Tracking → Having an object move with certain speed or acceleration.
  (our goal)

1) Regulation Case:

The regulation case: Goal is to keep object at rest at the origin.

Examples:

#1)

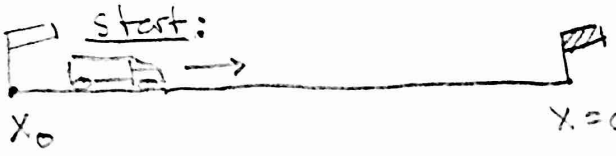


The inverted pendulum on a cart's task is to balance the up-right pendulum.

#2)

Given a dynamic model of a drone, Hover.

#3)



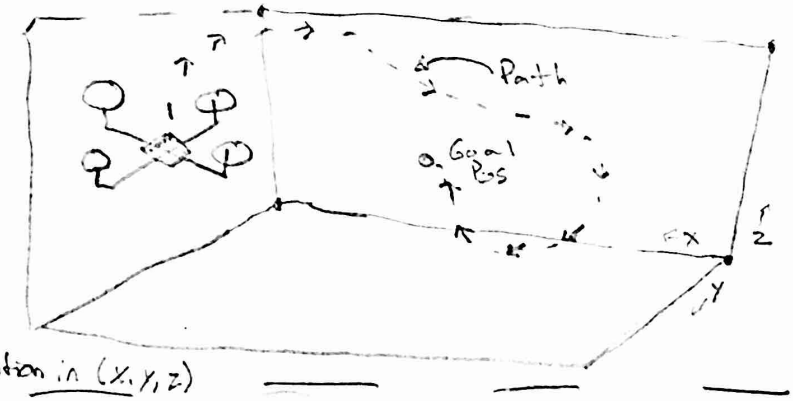
Correct for position error (drive car fast to land right on goal point)

(and many more!)

2) Tracking Cases

Examples:

#1)

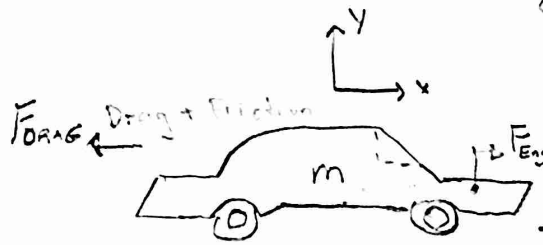


Have vehicles follow reference path to get to goal position in (X, Y, Z)

GOAL OF COURSE:

#2

Vehicle cruise Controller:



Control the cars speed based on a user's desired speed. (V_{Goal})

V_0

Want to speed up/down to V_{Goal}

(And many more!)

Sensor Instramentation

The world of control systems is made possible by sensors for "feedback control" of the system.

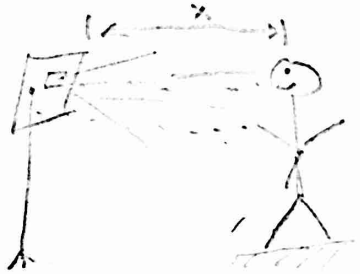
Lets look at some common types of sensors found in control systems.

Distance Sensors: (And Velocity)

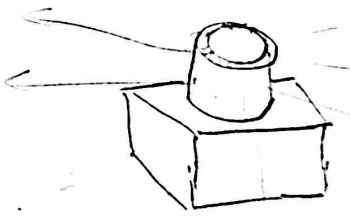
GPS: delivers x, y, z position in global coordinate

Infared distance sensor:

Radar Distance Sensor:



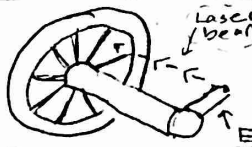
Lidar distance sensor:



- Senses in usually 270°-360° of the environment
- Returns range to objects

* Taking the change in distance between two readings, Δx, and dividing by elapsed time Δt, gives us approximations of velocity

Wheel Encoders:



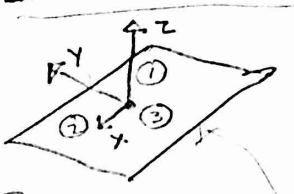
Encoder counts # of times laser beam is blocked by wheel spoke

Orientation Sensors: (And Angular Velocity)

Gyroscope: ω_z, ω_y, ω_x

Measures the Rate of rotation

Inertial Measurent Unit:



3 sensors x 3 axis = 9 measurments

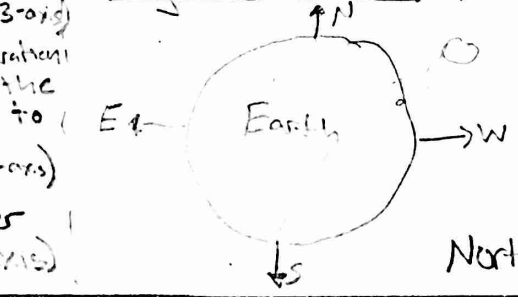
chip that has

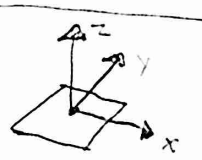
① Accelerometer: (3-axis)
→ Measures acceleration of the object the chip is attached to

② Gyroscope (3-axis)

③ Magnetometer (3-axis)

Magnetometer:

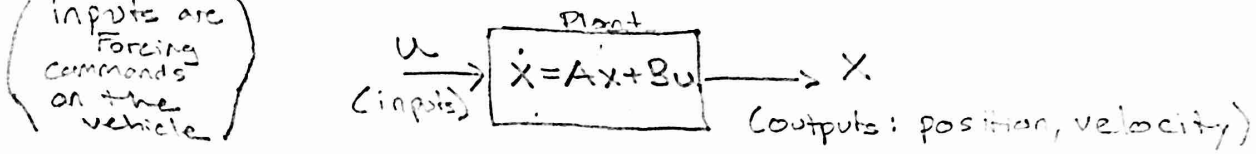




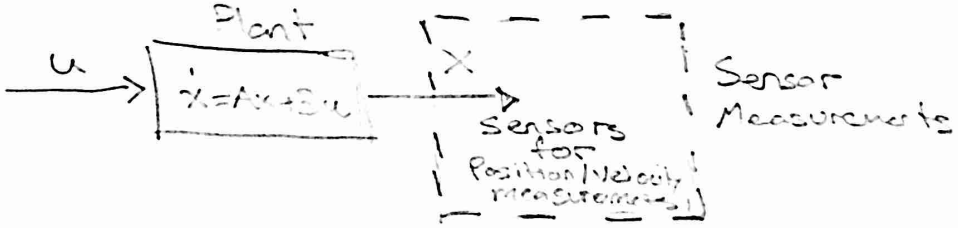
Gives your 3D orientation in terms of North-East-Down Coordinate

How to Draw A "Closed-Loop" Control Model

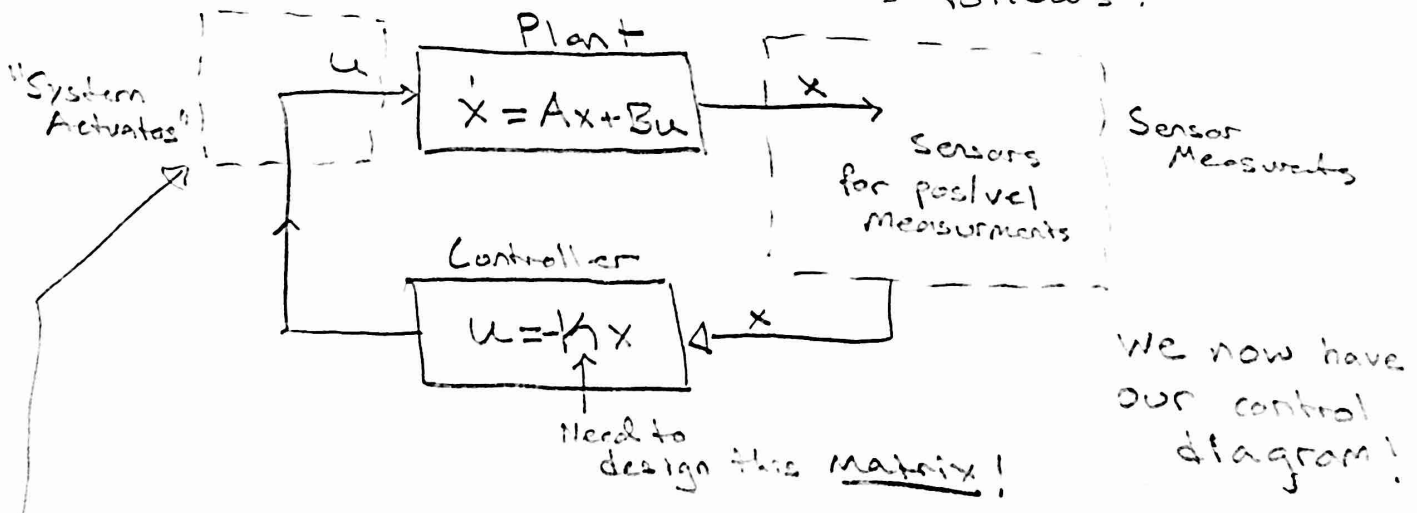
1) Regardless the "system", we refer to it as "The Plant" and draw it as follows:



2) We then can draw a dashed box around the system output arrows and label it "Sensor Measurements"



3) We now draw a box below the plant and label it "Controller" as follows:



4) Draw a small box around $\vec{u}$ (the input arrow) and label it: "System Actuators"

The Closed Loop Optimal Control Problem:

- The goal of our control task is to take our vehicles dynamics and create a "control policy" (or law) for regularization: (drive system to origin)
  or
  tracking: (drive system along trajectory or to reference the user specified.)
- Control engineers want to minimize the fuel consumption (energy) and time to reach the "origin" → (Goal point).
- We know now we write our open-loop system dynamics as $\dot{x} = Ax + Bu$
- Our cost equation we use is "mathematically built" so we can find the best set of control inputs {"$\vec{u}$"}'s to drive the vehicle optimally along a trajectory / set-point you choose!

"Cost Equation": Called the Linear Quadratic Regulator

$$J(x_0) = \int_0^{t_{final}} (x_t^T Q x_t + u_t^T R u_t)\, dt$$

(t_{final} → usually large; Σ = sum over all time till t_f)

Optimal control laws: $u^{opt} = -K^{opt} x$

→ Subject to $\dot{x} = Ax + Bu$ (As feedback)

→ And sensor's to give us state vector $\vec{x}$

$Q, R \rightarrow$ Control Parameters

$A \in \mathbb{R}^{n \times n}$ $B \in \mathbb{R}^{n \times p}$

$Q \in \mathbb{R}^{n \times n}$ → change values along diagonal
→ as values increase, control strengthens

We solve the whole LQR problem with 1 line of Python

$R \in \mathbb{R}^{p \times p}$ → change values along diagonal
→ As values increase, control weakens

- While the previous page looks tricky, after we write $\dot{x} = Ax + Bu$, we simply choose design parameters Q, R. These boost / weaken the control input after we receive measurements, from our sensor device.
  - ↳ for feedback control

- We solve (for K^*) using (scipy.signal as sig)

$$-, -, P = \text{sig.solve\_continuous\_are}(A, B, Q, R)$$

$$K^{opt} = -R.I * B.T * P \quad \text{w/ } K \in \mathbb{R}^{p \times n}$$

optimal control design matrix

We can't do better than this K-matrix, given our Q, R. → OPTIMAL

- "P" is the matrix which, amazingly, both:
  1) Tells us "cost-of-travel"
     solves → $J(x_0) = x_0^T P x$ ← AMAZING!
  2) Used for optimal control law:
     $u = -K^{opt} x \Rightarrow K^{opt} = -R^{-1} * B^T * P$

* Now, every time our sensors deliver us our state (pos, vel) $\vec{x}$, we use $\vec{u} = -K\vec{x}$ to propel system toward the goal point: (The origin)

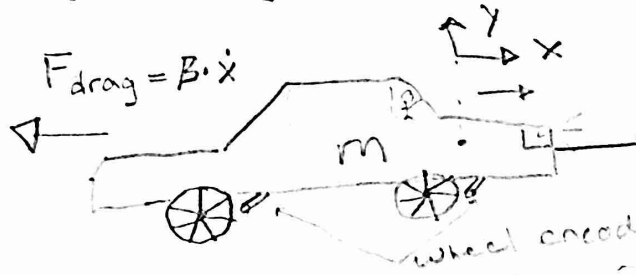
* This is how* commercial planes, space-craft, cars, and drones automatically control themselves

* (A common technique)

Part 4/5:

Course Project:

Consider the following model of a car, driving along a straight road:

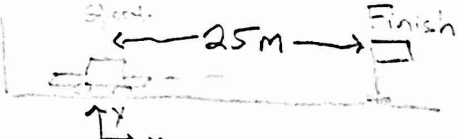


- The small car weighs $m = 125$ kg, with an effective drag (the loss from wheel friction + wind) of $B = 15.0$. The car engine model is said to be controlled by an engine throttle (gas pedal) which is modelled to be $u = \dot{F}_{engine}$ ($u = \dot{f}$).

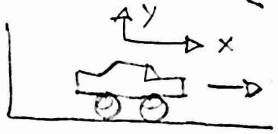
  Wheel encoders attached to the vehicle track the position and velocity of the car as the gas-pedal is pressed: $y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$ ← Wheel encoder model

- The goal is to build two controllers:

* (waypoint Navigator) ① K_1^{opt} → Used to move vehicle from position #1 to position #2 (setpoint pos #2)



* (cruise controller) ② K_2^{opt} → Used for vehicles automatic cruise controller (user sets desired vehicle speed)



- For K_{opt_1} (way-point navigator): Design an 'optimal' controller for the way-point navigator problem.

To do so:

1) Construct FBD of car.

2) Write in state-space form / in python.

3a) We want to "way-point" navigate (change position) so choose $Q = \begin{bmatrix} \alpha & 0 \\ 0 & 0 \end{bmatrix}$ $\rightarrow (2\times2)$ and $R = \beta$ $\rightarrow (1\times1)$, with $\{\alpha, \beta\}$ design parameters. (You should try a few combos)

3b) using $\{A, B, Q, R\}$, run

```
Sig.solve_continous_are(.)
```

- it will return a crutial matrix used in control loop

```
P = CARE(.)
```

- now, $K^{opt} = R.I * B.T * P$

4) Plot the position and velocity response of the closed-loop system: (with $U = -KX$)

Note: if $\dot{x} = Ax + Bu$, and $u = -Kx$, then $\dot{x} = \underbrace{(A - BK)}_{\text{"}A_{closed\text{-}loop}\text{"}} x$

Note: Initial conditions are $V_o = 0\,m/s$; $X_o = 25\,m$

5) Change design parameters Q and R so navigation is extremely fast and also extremely slow; plot.

• For K_{opt_2}: (cruise controller) Design an 'optimal' controller for the cruise-control system of the car.

To do so:

1) Construct FBD of car

2) write in state-space form:
Include acceleration ($\dot{v}$), $\dot{v}_{goal}$, $\dot{F}_{engine}$
(create 3x3 A-matrix and 3x1 B matrix)

---

3) We want our car to move at v_{goal}:

- Write the error dynamics as a 2D-system

$$e = v - v_{goal}$$
$$\dot{e} = \dot{v} - \dot{v}_{goal}$$
$$\ddot{e} = \ddot{v} - \ddot{v}_{goal}$$

"e" = velocity error, "$\dot{e}$" = rate of velocity error;

- wheel encoders track our velocity: so $y = [1\ 0]\begin{bmatrix} e \\ \dot{e} \end{bmatrix}$
- write error dynamics A_e, B_e matrices in python and select $Q = \begin{bmatrix} \alpha & 0 \\ 0 & 0 \end{bmatrix}$ and $R = \beta$ design parameters. (Try few combos)

---

4) solve optimization problem to find best "K_e" for control law $\dot{U}_e^* = -K_e^{opt} \cdot \vec{e}$ (using feed-back control)

→ $P = $ sig.solve_continous_are(A_c, B_c, Q, R)

$$K_e^{opt} = P_1 \cdot I \cdot B \cdot T \cdot P$$

$$\dot{U}_e = -K_e^{opt} \cdot \vec{e}$$

---

5) If $U_c = -K_e \vec{e} = -[K_{c_1}\ K_{c_2}]\begin{bmatrix} e \\ \dot{e} \end{bmatrix}$ → re-write as: $U_x = [K_1\ K_2\ K_3]\begin{bmatrix} v \\ v_{goal} \\ F_{engine} \end{bmatrix}$

6) Plot position and velocity responses in python using the closed loop system dynamics:

$$\dot{X} = (A - B \cdot K_x^{opt}) \cdot X$$

↳ Found from transforming error dynamics.

{ use initial conditions: $X_0 = 0$
$V_0 = \dot{X}_0 = 20\ m/s$
$V_{Goal} = \dot{y} = 30\ m/s$
Plot for $t = 15$ seconds. }