# -\*- coding: utf-8 -\*-

"""

Created on Mon Dec 6 12:54:40 2021

@author: Vadim

"""

# optimal\_bench.py - benchmarks for optimal control package

# RMM, 27 Feb 2020

#

# This benchmark tests the timing for the optimal control module

# (control.optimal) and is intended to be used for helping tune the

# performance of the functions used for optimization-base control.

import numpy as np

import math

import control as ct

import control.flatsys as flat

import control.optimal as opt

import matplotlib.pyplot as plt

import logging

import time

import os

#

# Vehicle steering dynamics

#

# The vehicle dynamics are given by a simple bicycle model. We take the state

# of the system as (x, y, theta) where (x, y) is the position of the vehicle

# in the plane and theta is the angle of the vehicle with respect to

# horizontal. The vehicle input is given by (v, phi) where v is the forward

# velocity of the vehicle and phi is the angle of the steering wheel. The

# model includes saturation of the vehicle steering angle.

#

# System state: x, y, theta

# System input: v, phi

# System output: x, y

# System parameters: wheelbase, maxsteer

#

def vehicle\_update(t, x, u, params):

# Get the parameters for the model

l = params.get('wheelbase', 3.) # vehicle wheelbase

phimax = params.get('maxsteer', 0.5) # max steering angle (rad)

# Saturate the steering input (use min/max instead of clip for speed)

phi = max(-phimax, min(u[1], phimax))

# Return the derivative of the state

return np.array([

math.cos(x[2]) \* u[0], # xdot = cos(theta) v

math.sin(x[2]) \* u[0], # ydot = sin(theta) v

(u[0] / l) \* math.tan(phi) # thdot = v/l tan(phi)

])

def vehicle\_output(t, x, u, params):

return x # return x, y, theta (full state)

vehicle = ct.NonlinearIOSystem(

vehicle\_update, vehicle\_output, states=3, name='vehicle',

inputs=('v', 'phi'), outputs=('x', 'y', 'theta'))

# Initial and final conditions

x0 = [0., -2., 0.]; u0 = [10., 0.]

xf = [120., 1.1, 0.]; uf = [10., 0.]

Tf = 10

# Define the time horizon (and spacing) for the optimization

horizon = np.linspace(0, Tf, 10, endpoint=True)

# Provide an intial guess (will be extended to entire horizon)

bend\_left = [10, 0.01] # slight left veer

# Set up the cost functions

Q = np.diag([5, 10, 1]) # keep lateral error low

R = np.diag([.1, 1]) # minimize applied inputs

quad\_cost = opt.quadratic\_cost(vehicle, Q, R, x0=xf, u0=uf)

#

res = opt.solve\_ocp(

vehicle, horizon, x0, quad\_cost,

initial\_guess=bend\_left, print\_summary=False,

# solve\_ivp\_kwargs={'atol': 1e-2, 'rtol': 1e-2},

minimize\_method='trust-constr',

minimize\_options={'finite\_diff\_rel\_step': 0.01},

)

#

u = res.inputs

t, y = ct.input\_output\_response(vehicle, horizon, u, x0)

#y = res.states

#t = res.time

print(t)

# Plot the results

plt.subplot(3, 1, 1)

plt.plot(y[0], y[1])

plt.plot(x0[0], x0[1], 'ro', xf[0], xf[1], 'ro')

plt.xlabel("x [m]")

plt.ylabel("y [m]")

plt.subplot(3, 1, 2)

plt.plot(t, u[0])

plt.axis([0, 10, 5.5, 22.5])

plt.plot([0, 10], [9, 9], 'k--', [0, 10], [11, 11], 'k--')

plt.xlabel("t [sec]")

plt.ylabel("u1 [m/s]")

plt.subplot(3, 1, 3)

plt.plot(t, u[1])

plt.axis([0, 10, -0.35, 0.35])

plt.plot([0, 10], [-0.1, -0.1], 'k--', [0, 10], [0.1, 0.1], 'k--')

plt.xlabel("t [sec]")

plt.ylabel("u2 [rad/s]")

plt.suptitle("Lane change manuever")

plt.tight\_layout()

plt.show()