Murray Problem 3.12: Trajectory Optimization of a Thrust-Vectoring Aircraft

Source Filename: /main.py

Rico A. R. Picone

In this problem, we will compute optimal trajectories for the PVTOL system using two diferent methods: numerical optimization and exploiting (near) differential flatness. The numerical optimization method allows us to compute an optimal trajectory $(x_d(t), u_d(t))$ that minimizes a cost function J subject to the dynamics of the system. The numerical nature of this approach requires some care in selecting the parameters of the optimization problem, such as the initial guess, the cost function, and the constraints. The `control.optimal` module provides a function `solve_ocp()` that can be used to solve optimal control problems of this form.

Begin by importing the necessary libraries and modules as follows:

```
import numpy as np
import matplotlib.pyplot as plt
import control
import control.optimal as opt
import control.flatsys as fs
import time
from pvtol import pvtol, plot_results
```

The PVTOL system dynamics and utility functions are defined in the `pvtol` module, found here. We have loaded the dynamics as `pvtol`. Here is some basic information about the system:

```
print(f"Number of states: {pvtol.nstates}")
print(f"Number of inputs: {pvtol.ninputs}")
print(f"Number of outputs: {pvtol.noutputs}")
```

```
Number of states: 6
Number of inputs: 2
Number of outputs: 6
```

From `pvtol.py`, the state vector x and input vector u are defined as follows:

$$x = \begin{bmatrix} x & y & \theta & \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}^\top \quad u = \begin{bmatrix} F_1 & F_2 \end{bmatrix}^\top$$

Our first task is to determine the equilibrium state and input that corresponds to a hover at the origin:

```
xeq, ueq = control.find_eqpt(
    pvtol,  # System dynamics
    x0=np.zeros((pvtol.nstates)),  # Initial guess for equilibrium state
    u0=np.zeros((pvtol.ninputs)),  # Initial guess for equilibrium input
    y0=np.zeros((pvtol.noutputs)),  # Initial guess for equilibrium output
    iy=[0, 1],  # Indices of states that should be zero at equilibrium
)
print(f'Equilibrium state: {xeq}')
print(f'Equilibrium input: {ueq}')
```

```
Equilibrium state: [0. 0. 0. 0. 0. 0.]
Equilibrium input: [ 0.  39.2]
```

Set up the optimization problem parameters as follows:

```
x0 = xeq  # Initial state
u0 = ueq  # Initial input
xf = x0 + np.array([1, 0, 0, 0, 0, 0])  # Final state
Q = np.diag([1, 10, 10, 0, 0, 0])  # State cost matrix
R = np.diag([10, 1])  # Input cost matrix
cost = opt.quadratic_cost(pvtol, Q=Q, R=R, x0=xf, u0=u0)  # J
```

The cost function `cost` is constructed with `x0=xf`, the final state, because this is the desired state where the cost should be zero.

Set up the time horizon, time steps, and initial guess for the trajectory, a straight line from `x0` to `xf`, as follows:

```
tf = 5  # Time horizon
nt = 12  # Number of time steps
t = np.linspace(0, tf, nt)  # Time steps
x_init = np.vstack([
    np.linspace(x0[i], xf[i], nt) for i in range(pvtol.nstates)
])  # Straight line from x0 to xf
u_init = np.outer(u0, np.ones_like(t))  # u0 for all time steps
```

Solve the optimal control problem as follows:

```
solve_time_start = time.time()
opt1 = opt.solve_ocp(
    pvtol, t, x0, cost, log=True,
    initial_guess=(x_init, u_init),
)
solve_time_end = time.time()
print(f"Optimization time: {solve_time_end - solve_time_start:.2f} s")
```

```
Summary statistics:
* Cost function calls: 1562
```

```
* Cost function process time: 0.09845800000000304
* System simulations: 0
* Final cost: 1.53256102232247
Optimization time: 0.31 s
```

Extract the optimal trajectory and plot the results as follows:

```
print(f"Initial position: {opt1.states[0:2, 0]}")
print(f"Final position: {opt1.states[0:2, -1]}")
plot_results(opt1.time, opt1.states, opt1.inputs)
plt.show()
```

```
Initial position: [0. 0.]
Final position: [ 0.97210405 -0.00682428]
```