$_5C_3$

ME 02.74/740: Bio-inspired Robotics

HW 05: Optimal Control of a Jumping Leg

Due: Sunday October 16th, 11:59 PM

Please read this assignment carefully, submitting all code and responses online as a single PDF. Provide text copies of your MATLAB code in a separate zip file.

1 Computational Optimal Control of a Jumping Leg

Sample code for this problem, available on Canvas, includes simulation of the jumping leg that we have discussed in class. A graphical description of the model and its parameters are given below in Figure 1. The model has one control input τ, and has two degrees of freedom y and θ. A force F_y is modeled at the foot during contact.

We can use the provided simulation to help us optimize the performance of the jumping leg. In this problem, we'll find an open loop torque trajectory (torque as a function of time) that maximizes the height of the leg's center of mass at the apex of its jump. We'll use a single shooting approach.

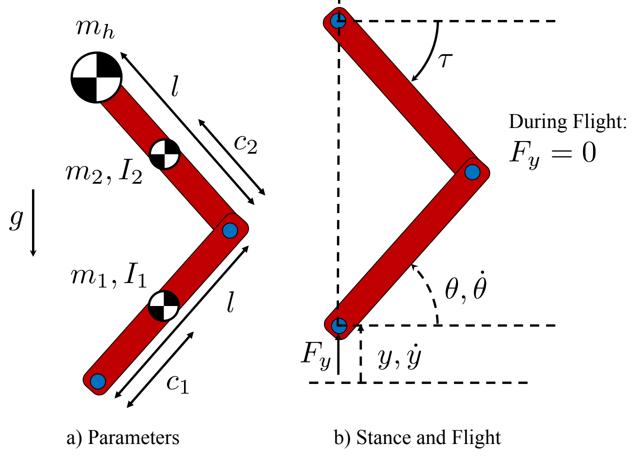


Figure 1: Jumping leg used in this homework assignment. Simulation code available on Canvas.

1.1 Problem Description

The single shooting optimization problem can be expressed mathematically as

$$\max_{x} y_{cm}(x, z_0, p), \tag{1}$$

where y_{cm} is a function that returns the maximum height of the center of mass of the jumping leg during a simulation. z_0 is a specified initial state, p includes the specified model parameters, and decision variables x are given as

$$\longrightarrow x = [t_f, t_{f,ctrl}, T_0, T_1, T_2, \ldots, T_N], \tag{2}$$

where t_f is the simulation time, $t_{f,ctrl}$ is the temporal duration of the control trajectory, a Bezier curve function of time parameterized by torques at equally spaced time points $T_0 = T(t = 0), T_1, T_2, \ldots, T_N = T(t = t_{f,ctrl})$. The optimization should take place subject to simple bounds on the decision variables

x= [tf ctrl.tf ctrl.T]

- $0.4\text{s} \le t_f \le 1\text{s}$ (i.e. the simulation final time must lie within a range),
- $0.1\text{s} \le t_{f,ctrl} \le 1\text{s}$ (i.e. the control duration must lie within a range), and
- $-2\text{Nm} \le T_i \le 2\text{Nm}$ (i.e. the control torque values must lie within a range),

and the nonlinear constraints $c_{eq}(x, z_0, p) = 0$ and $c_{ineq}(x, z_0, p) \le 0$ (*not greater than, as I mistakenly said in lecture!*), where

- $0 =$ $c_{eq} = t_{f,ctrl} - t_{takeoff}$ (i.e. $t_{f,ctrl}$ must correspond with takeoff),
- $c_{ineq,1} = -\min_t \theta(t), t \in [t_0, t_f]$ ≤ 0 (i.e. the leg must not fall below the ground), and
- $c_{ineq,2} = \max_t \theta(t) - \frac{\pi}{2}, t \in [t_0, t_f]$ ≤ 0 (i.e. the leg must not hyperextend).

To solve this problem numerically, you will write MATLAB functions that evaluate the objective function $-y_{cm}(x, z_0, p)$ (note that we minimize the negative of a function to maximize the original function) and the constraint functions for any given x, z_0, and p. Then you will provide handles to these functions and a guess x_0 (which need not minimize the objective function or satisfy the constraints) to `fmincon()`, which will evaluate the functions at the guess and iteratively refine the guess until either

- it determines that it cannot solve the problem, or
- it finds a solution x^* for which the objective function is minimized and the constraints are satisfied.

1.2 Provided Code

Begin by familiarizing yourself with the provided code, opening `run_simulation.m` as a starting point. Supporting code in `hybrid_simulation.m` simulates the leg through stance and flight. Jump performance is shaped through an input `ctrl`, a structure with the fields

- `ctrl.tf`, the duration over which the control trajectory is defined, and
- `ctrl.T`, an array of torque values at equally spaced time points from 0 to `ctrl.tf`.

The function `control_laws` ? receives this structure as an input. During the stance phase, `control_laws` returns the instantaneous torque corresponding with the present time assuming that `ctrl` specifies a Bezier control trajectory, which you need to implement yourself. During the flight phase, `control_laws` returns an instantaneous torque to regulate the angle of the leg (according to a simple PD-controller).

implement

The `run_simulation` file includes sample code that you will complete to set up and solve the trajectory optimization problem. Prior to any of your changes to the code, however, `run_simulation` will simulate a suboptimal initial guess trajectory.

X: 0.9510 0.6987 0.0449

1.9974 1.9989

1.3 Tasks

1. Run `run_simulation.m` without modification; familiarize yourself with what it does and how it works.

2. Complete `objective.m` to evaluate $y_{cm}(x, z_0, p)$. This will require running `hybrid_simulation` with appropriate inputs.

3. Complete `constraints.m` to evaluate $c_{eq}(x, z_0, p)$ and $c_{ineq}(x, z_0, p)$. This, too, will require running `hybrid_simulation` with appropriate inputs.

4. Complete `BezierCurve.m` to generate control input trajectory. You can verify your implementation by using the provided `BezierCurve_test.m` code.

5. Complete `run_simulation.m` to set up the optimization problem, solve it, and visualize the solution. You can start with $t_f = 0.5$, $t_{f,ctrl} = 0.35$, and $T_i = 1.0$ for $i = [0, 1, 2]$ as your guess for the optimization variables. x= [tf ctrl.tf ctrl.T]

   Note that you should expect the optimization to take around 30 – 100 iterations to converge for the problems in this PSet. After working, collect all data, plots, etc. as needed for turn in (described at the end of the document).

6. Modify your objective and constraint functions in order to minimize the time t_f under an additional constraint, that the apex center of mass height $y_{cm}(x, z_0, p) = 0.4$. Collect all data, plots, etc. as needed for turn in (described at the end of the document).

7. Modify your code in order to minimize $\Sigma_{t_0}^{t_f} \tau^2$ under the same additional constraint the apex center of mass height $y_{cm} = 0.4$. Note that $\Sigma_{t_0}^{t_f} \tau^2$ is a common metric used to approximate the electrical energy consumed by an electromagnetic actuator (within a constant factor). Collect all data, plots, etc. as needed for turn in (described at the end of the document).

Turn In

For each task 1.3.4 - 1.3.6, include in your PDF submission:

- A plot of the trajectory of the center of mass.
- A plot of the trajectory of the torque input.
- The numeric values for the optimal decisions variables x^*.
- The numeric objective function value at the optimum.

Copy printed vals

Turn in a single copy of your code that is capable to perform tasks 1.3.4-1.3.6. You can selectively comment out parts of the code as needed for each task.

2. objective.m

hybrid_simulation (z0, ctrl, p, tspan)

- z0 → initial state
- ctrl → Control struct (ctrl.tf, ctrl.T)
  - ctrl.tf → duration for control trajectory
  - ctrl.T → array of torques
- p → params for simulation
- tspan → [t0, tf]

↳ Outputs: tout, zout, uout, indices

- zout → state
- uout → control traj
- indices → indicies in tout when each phase ended

iphase_list = lists out phases to later check phase changes for indicies, values are 1, 2, 3

jump (2) max height (3)

→ objective (x, z0, p)

- x → array of decision variables

Output: f = scalar value of func

3.

fmincon(x) = f

so z0, p provided using anonymous func (anon func e.g. ode45())

fmincon(func, x0, A, b)

minimizes func, $A * x \le b$

	Problem.x0 = x0
f(x) to minimize	Problem.objective = f
$Ax \le B$	P. Aineq = A P.Bineq = B
$Cx = D$	P. Ceq = C P.deq = D
$l \le x \le u$	P.lb = l P.ub = u
$h(x) \le 0$	P.nonlcon = h or g
$g(x) = 0$	
	P.solver = 'fmincon'
	x = fmincon(problem)

z = [y, θ, ẏ, θ̇]

x = [tf ctrl.tf ctrl.T]

z = θ, θ̇

$0 \le t \le tf$

0 =
- $c_{eq} = t_{f,ctrl} - t_{takeoff}$ (i.e. $t_{f,ctrl}$ must co
- $c_{ineq,1} = -\min_t \theta(t), t \in [t_0, t_f]$ ≤ 0 (i.e. the l
- $c_{ineq,2} = \max_t \theta(t) - \frac{\pi}{2}, t \in [t_0, t_f]$ ≤ 0 (i.e. th

```
4. for i = 1:n                                   n=length(ctrl_t)
       Ctrl_input(i)= BezierCurve (x(3:end), Ctrl_t(i)/Ctrl.tf)
   end
```

Ctrl.T

% done (time)?

torques @ equally spaced time points

Ctrl_t = linspace(0, ctrl.tf, 50)

Ctrl_pt = [0, 1.5, 0, 3, 2]

t=0   dub

t = 0, 0.01, 0.02 ...

COM = 4,500

div = # of segments of curve = n+1?

n = # of points

```
for u=0:$\frac{1}{div}$:1
for i=1:n
    UB(i) = factorial(n-1) / (factorial(i-1) * factorial(n-i)) * (u^(i-1)) * ((1-u)^(n-i));
end
end
```

n = len(ctrl_t)   linspace(0, ctrl_tf, 50)

```
for i=1:n
    ctrl_input(i) = Bez(Ctrl.T, $\frac{Ctrl\_t(i)}{Ctrl.tf}$)
end
```

$\overbrace{0, 0.01, 0.02 \ldots tf}^{50}$

$t = \frac{0}{50}, \frac{1}{50}, \frac{2}{50}, \frac{3}{50} \ldots 1$

$T = [T_0, T_1 \ldots T_N]$

$$\frac{(n-1)!}{(i-1)!\,(n-i)!} \cdot u^{i-1} \cdot (1-u)^{n-i}$$

$u = 0 : \frac{1}{div} : 1 = 0, \frac{1}{div}, \frac{2}{div}, \ldots \frac{div}{div}$

$i = 1:n = 1, 2, 3 \ldots n$

n = # of points

$$(1-t)^n P_0 + \binom{n}{1}(1-t)^{n-1} t P_1 + \cdots + \binom{n}{n-1}(1-t) t^{n-1} P_{n-1} + t^n P_n$$

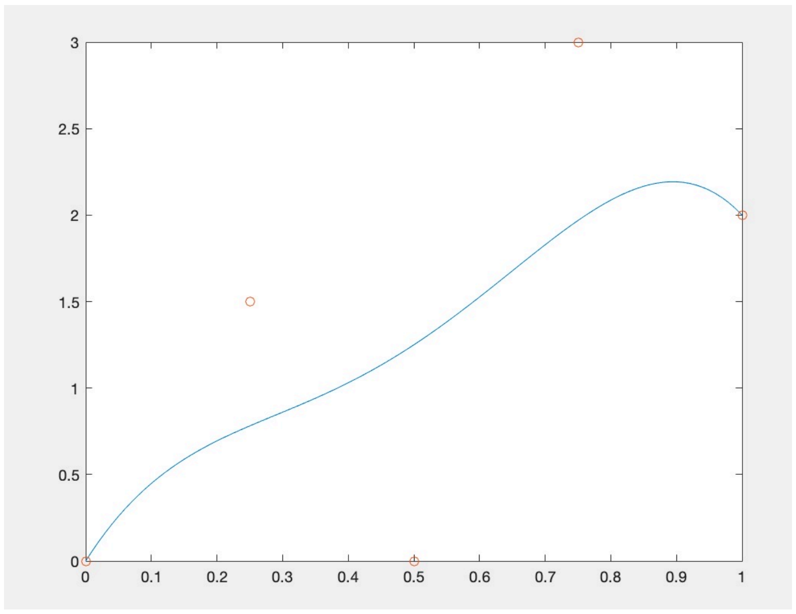
$$nCi = \frac{n!}{i!(n-i)!}$$

$$\frac{nCi \cdot i}{n} \cdot u^{i-1} \cdot (1-u)^{n-i}$$

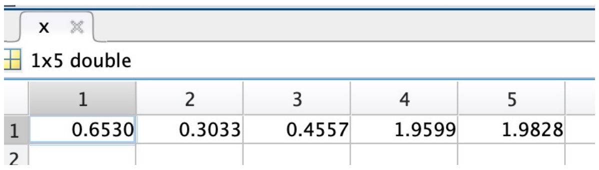
$$u(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i$$

$$u = u + nC_i\,(1-t)^{n-i} t^i P_i$$

1.3.4

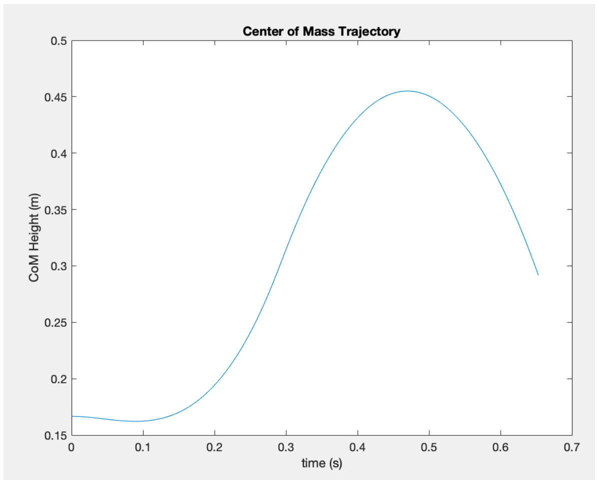


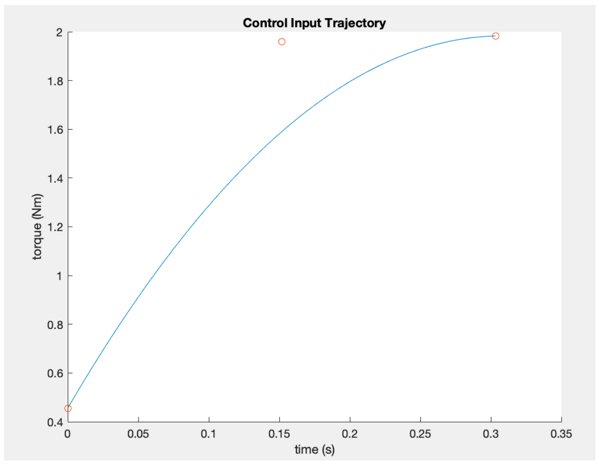
1.3.5


x

1x5 double

	1	2	3	4	5
1	0.6530	0.3033	0.4557	1.9599	1.9828
2					





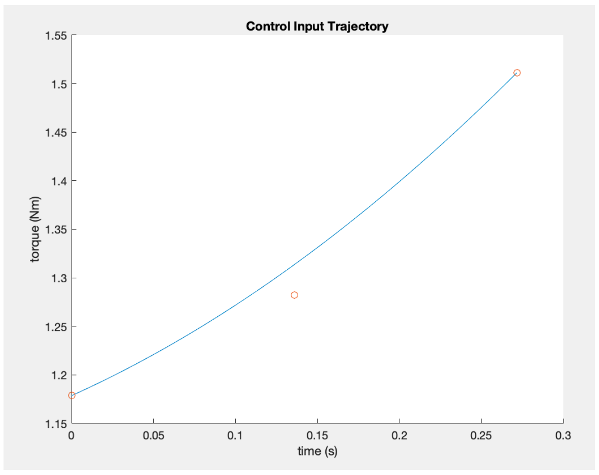
1.3.6

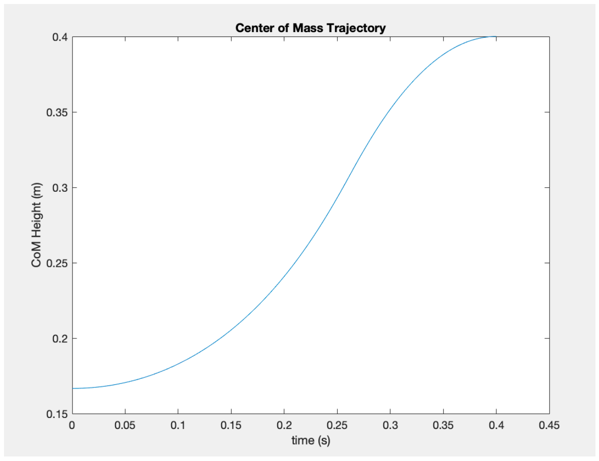
```
x =

    0.4000    0.2717    1.1786    1.2821    1.5112


fval =

    0.4000
```





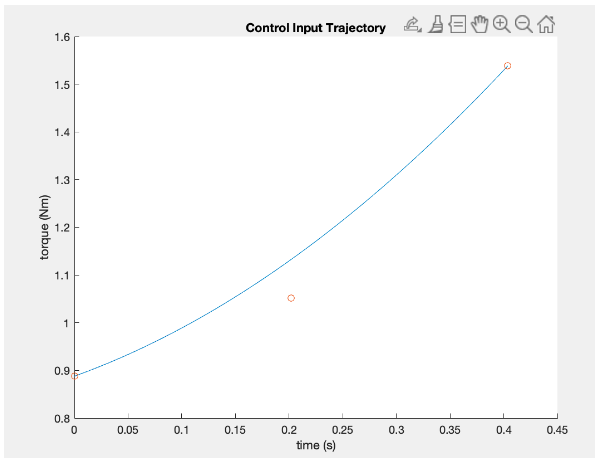
1.3.7

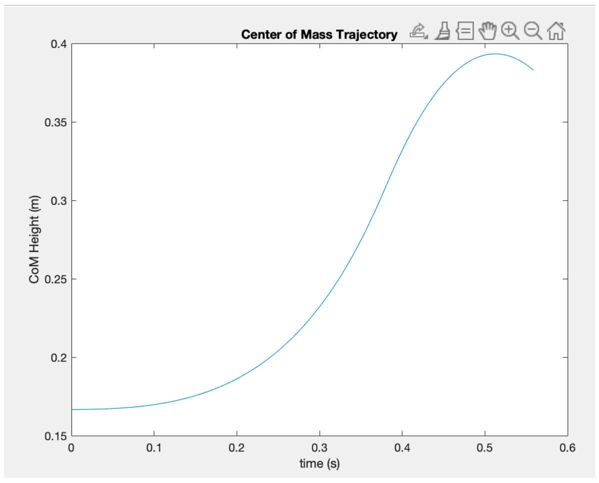
```
x =

    0.5590    0.4037    0.8879    1.0519    1.5384


fval =

    4.2613
```


Control Input Trajectory
torque (Nm)
time (s)


Center of Mass Trajectory
CoM Height (m)
time (s)