Catch the Falling Ball

Catch the Falling Ball

0.0/25.0 points (graded)

For this problem, we will make use of Drake to make an Acrobot "catch" a falling ball. In this case, catching means matching the position of the end of the Acrobot with the position of the ball. Imagine that the Acrobot has a big net attached to the end, if you'd like.

The state of the combined Acrobot-ball system is $x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$ where

$$q = \begin{bmatrix} \theta_1 \\ \theta_2 \\ x_{ball} \\ z_{ball} \\ \phi_{ball} \end{bmatrix}$$

With θ_1 and θ_2 the usual angles of the acrobot, and the other three coordinates relating to the ball position and orientation. Note, for the entire problem, since we are only concerned with the position of the ball, you may safely ignore its orientation ϕ_{ball}.

We have provided stub code to introduce you to the way Drake constructs trajectory optimization problems, available as a .zip here. Please unzip the stub code into the **examples/Acrobot** directory. The code is set to use the Direct Collocation method that you explored earlier in this problem set. The initial optimization problem to solve is to minimize torque squared (see the *running_cost_fun* in the code), while matching the final positions described above. The Acrobot and ball are given some initial state, which will bring the ballistic trajectory of the ball close enough to catch. You will note that the duration of the trajectory is restricted to be $1.5 \leq T \leq 3$.

NOTICE: when submitting responses to this question, **x_grade** is a fairly large matrix. Make sure that when you copy and paste it into the boxes below, that it pasted properly. At the end of the sample code, it prints out **x_grade** in a format that should help facilitate this copy-paste. Also see Matlab's *clipboard* function.

Part 1

The first task is to get *pset05_catch.m* working. You will find a few places in the file that indicate "YOUR CODE HERE." In particular, you need to finish the constraint on the final state and choose the initial state and control vectors for the optimization program. For this part, you do not yet need to finish the function *final_state_obj*.

A successful implementation will find a locally optimal trajectory, where the Acrobot catches the ball! Execute the script

```
run_pset5_catch_parts_1_and_2.m
```

And submit **x_grade** below. Make sure that the trajectory optimization returned a feasible solution (that there was no error)! You can paste the output of

```
mat2str(x_grade)
```

in the box, or you can use this matlab command:

```
clipboard('copy', x_grade)
```

to copy the value of x_grade to your clipboard, and then paste that result directly into the box.

```
1
```

Press ESC then TAB or click outside of the code editor to exit

Unanswered

```
% intialization can play a significant role in convergence of nonlinear programs,
% as you hopefully found out by exploration in this problem.
%
% one choice is to use a random iniitalization for u, and to use a constant x
trajectory, such as
x_init_vec = repmat(x0,1,N);
u_init_vec = randn(1,N);

% the "catching" constraint can be formulated as
q = x(1:5);
qd = x(6:10);
kinsol = obj.doKinematics(q);

hand_body = 3;% body index, so p.body(3) is the lower link
pos_on_hand_body = [0;-2.1]; % position of the "hand" on the lower link, 2.1m is the
length

[hand_pos,dHand_pos] = obj.forwardKin(kinsol,hand_body,pos_on_hand_body);

ball_pos = [q(3);q(4)];
dBall_pos = [zeros(2,2) eye(2) zeros(2,1)];

% note the 2x5 zero matrix at the end of df, because f is independent of qdot
f = ball_pos - hand_pos;
df = [dBall_pos - dHand_pos zeros(2,5)];
```

Part 2

Now, we would like to add the additional cost that awards catching the ball at a higher point in its trajectory. There are a number of different ways to do this, and the precise implementation is left to you. See the stub function *final_state_obj* for instructions on how to create this cost within Drake. Ensure that the cost is significant enough that it affects the outcome, since it is being balanced with the original torque squared cost. Again, execute the script

```
run_pset5_catch_parts_1_and_2.m
```

And submit **x_grade** below. Your solution must catch the ball at least slightly higher than the nominal, 3 second motion.

```

```

Press ESC then TAB or click outside of the code editor to exit

Unanswered

```
% There were many ways to reward an early catch, and the grader was only looking for
the time of the
% catch to occur 10ms earlier than it did in Part 1. Some submissions to this
problem failed to
% satisfy the catch criteria at all, and one possibility for this is that the
submission resulted
% from a failure of the solver. Poorly conditioned objective functions and
constraints, as well as bad
% initial seeds can cause this sort of behavior, and it's important to check any
errors/warnings
% Here's one possible objective function, which explicitly rewards the final height
of the hand.
q = x(1:5);
qd = x(6:10);
kinsol = obj.doKinematics(q);

hand_body = 3;% body index, so p.body(3) is the lower link
pos_on_hand_body = [0;-2.1]; % position of the "hand" on the lower link, 2.1m is the
length

[hand_pos,dHand_pos] = obj.forwardKin(kinsol,hand_body,pos_on_hand_body);

f = -100*hand_pos(2);
df = -100*[0 dHand_pos(2,:) zeros(1,5)];
```

Part 3

For the final part of this problem, implement a time-varying LQR controller. In particular, we would like to choose a cost that penalizes the distance from the ball to the hand at the point where the Acrobot should be catching the ball. Since the trajectory was designed with this as a constraint, it should not be too hard to generate an appropriate Q_f. In *run_pset5_part3*, set the LQR cost terms. The script will use the time-varying controller to simulate a couple of times with perturbed initial conditions. Verify that your controller finishes close to the ball (within 0.07 m), and submit **x_grade** below.

HINT: Try finding a Q_f such that $x(T)^T Q_f x\,(T)$ approximates the distance-squared between the hand and the ball! This is a little tricky, simply guessing Q_f is unlikely to work.

```

```

Press ESC then TAB or click outside of the code editor to exit

Unanswered

```
% The key here was to think carefully about Qf, rather than choosing something
simple like the
% identity matrix To directly penalize the final distance from the hand to the ball,
consider
% the function f = (distance ball to hand)^2. If xf is the final state from the
nominal trajectory,
% then f(xf) = 0 and df/dx(xf) = 0.
%
% Then, if H is the hessian of f, then (x-xf)^T H (x - xf) is the quadratic
approximation of f around
% the nominal.  Since H isn't positive definite, we might want to add some
regularization, although it
% is not strictly necessary. One solution is to take Qf = K*H + I, code shown below
for K=100.

% get final state
xf = xtraj.eval(3);
qf = xf(1:5);

% same code as in the constraint function
kinsol = p.doKinematics(qf);

hand_body = 3;% body index, so p.body(3) is the lower link
pos_on_hand_body = [0;-2.1]; % position of the "hand" on the lower link, 2.1m is the
length

[hand_pos,dHand_pos] = p.forwardKin(kinsol,hand_body,pos_on_hand_body);

ball_pos = [qf(3);qf(4)];
dBall_pos = [zeros(2,2) eye(2) zeros(2,1)];

f = ball_pos - hand_pos;
df = [dBall_pos - dHand_pos zeros(2,5)];

% The Hessian is actually df'*df!
Q = eye(10);
R = 1;
Qf = Q + df'*df*100;
```

Submit You have used 0 of 3 attempts

Answers are displayed within the problem