EN530.603 Applied Optimal Control
Homework #6

October 31, 2017

Due: November 7, 2017 (before class)

Professor: Marin Kobilarov

1. *Differential Dynamic Programing*: A general discrete optimal control code (ddp.zip) is provided along with two examples (2-dof robotic arm and a second-order wheeled vehicle model). You have two options: 1) *extend* one of the two provided models; or 2) implement a new model of your own choice in a similar manner as the two examples. In both cases you must include meaningful control bounds (by modifying ddp.m) and add environmental obstacles (recall HW4#3). Obstacles should be added as a penalty/repulsive potential term to the cost function. In your plots clearly show that the computed controls do not exceed the specified bounds.

   Details: Assume that we need to enforce a constraints of the form

   $$c_k(x_k, u_k) \leq 0, \text{ for all } k = 0, \ldots, N-1,$$

   where $c_k = (c_k^1, \ldots, c_k^m)$ are m constraint functions. This can be accomplished by adding penatly terms to the trajectory costs L_k, i.e. by using

   $$\bar{L}_k(x, u) = L_k(x, u) + \frac{\beta_k}{2} \|g_k(x, u)\|^2,$$

   in place of $L_k(x, u)$, where $g_k(x, u) = \max(c_k(x, u), 0)$ for some chosen coefficient $\beta_k > 0$ that controls the "softness" of the constraint (here max is applied independently to each component of the vector c_k). Then the Jacobian with respect to x is

   $$\nabla_x \bar{L}_k(x, u) = \nabla_x L_k(x, u) + \beta_k \partial_x g_k(x, u)^T g_k(x, u),$$

   and is similarly defined with respect to u. The Hessian is

   $$\nabla_x^2 \bar{L}_k(x, u) = \nabla_x^2 L_k(x, u) + \beta_k \partial_x g_k(x, u)^T \partial_x g_k(x, u) + \beta_k \sum_{i=1}^{m} g_k^i(x, u) \nabla_x^2 g_k^i(x, u)$$

   and is similarly defined with respect to u. In some cases (when either g_k is small or $\nabla_x^2 g_k^i(x, u)$ has small eigenvalues) the last term above can be ignored.

   See example `ddp_pnt_obst.m`

The rest of the homework will have you use the ACADO Toolkit: a framework for automatic control and dynamics optimization (`http://acado.github.io/index.html`). You will use ACADO to solve a few optimal control problems. Follow the instructions at `http://acado.github.io/matlab_overview.html` to install the ACADO Toolkit Matlab interface. We will refer to the root directory of your ACADO installation as `<ACADO_ROOT>`. Note: In Step 3 it is not necessary to download ACADO using Git. You can just download the .zip file if it is easier for you.

2. Optimal control with a car

   (a) ACADO allows users to symbolically specify dynamics, constraints, and cost functions. The framework can then use automatic differentiation to compute derivatives without the user specifying them directly. Using the provided template file `hw7_car.m`, symbolically define the dynamics for a car model given by

$$\begin{aligned} \dot{p_x} &= v\cos(\theta) \\ \dot{p_y} &= v\sin(\theta) \\ \dot{\theta} &= v\tan(\delta) \\ \dot{v} &= u_a \\ \dot{\delta} &= u_\delta \end{aligned}$$

   Each state of your system should be declared as a `DifferentialState` object and each control should be defined as a `Control` object. A symbolic differential equation object `f` is defined using the command:

```
f = acado.DifferentialEquation(0, t_f);
```

   where `t_f` is a variable storing a fixed final time. The dynamics of your system can be defined by adding symbolic differential equations to `f`. For example,

```
DifferentialState v;
Control u_a;
f.add(dot(v) == u_a);
```

   Use the examples in `<ACADO_ROOT>/interfaces/matlab/examples/ocp` for guidance.

   (b) In ACADO, the user must first define an optimal control problem:

```
ocp = acado.OCP(0.0, t_f, num_steps);
```

   where `num_steps` defines the number of of steps in the optimal control problem. The user then adds desired costs and constraints to the `ocp` object.
   Symbollically add a quadratic cost on the controls $u = (u_a, u_\delta)$. To do so, use the function

```
ocp.minimizeLSQ(u, 0);
```

   (c) Add path constraints to the optimal control problem so that

$$\begin{aligned} -5 \le v \le 5 \\ -5 \le u_a \le 5 \\ -\pi/4 \le \delta \le \pi/4 \\ -\pi/6 \le u_\delta \le \pi/6 \end{aligned}$$

   Path constraints can be added using the function `ocp.subjectTo`, e.g.

```
ocp.subjectTo(-5 <= v <= 5);
```

Don't forget to add the dynamics as a constraint using `ocp.subjectTo(f)`.

(d) Add boundary constraints to the optimal constrol problem so that

$$
\begin{aligned}
p_x(t_0) &= -10.0 \\
p_y(t_0) &= 1.0 \\
v(t_0) &= 0.0 \\
\theta(t_0) &= 0.0 \\
\delta(t_0) &= 0.0
\end{aligned}
$$

$$
\begin{aligned}
p_x(t_f) &= 0.0 \\
p_y(t_f) &= 0.0 \\
v(t_f) &= 0.0 \\
\theta(t_f) &= 0.0.
\end{aligned}
$$

Boundary constraints can be added using, e.g.

```
ocp.subjectTo('AT_START', p_x == -10.0);
ocp.subjectTo('AT_END', p_x == 0.0);
```

(e) Configure the settings of the optimal control algorithm. Get access to the algorithm settings object using `algo=acado.OptimizationAlgorithm(ocp)`. Then, using the `algo.set` function, set the KKT convergence tolerance to 10^{-8}. This determines how closely the conditions for optimality have to be satisfied before the optimization exits. Furthermore, set the discretization type to 'MULTIPLE_SHOOTING' and set the max number of optimization iterations to 500. Enter the command `help acado.OptimizationAlgorithm.set` for details.

(f) Using `t_f=10` and `num_steps=64`, run the optimal control algorithm and plot the car state and control trajectories.

3. Time optimal obstacle avoidance with a car

(a) Using your script for (2) as a starting point, add a symbolic time parameter to ACADO, e.g. `Parameter T`. Change your differential equation and optimal control problem variables to use the symbolic time `T` instead of a fixed final time `t_f`.

(b) Add a term to your cost function that minimizes the final time using the function `ocp.minimizeMayerTerm(T)` (a Mayer term is just a terminal cost). Modify your quadratic control cost to minimize only u_δ.

(c) Add a path constraint to avoid a circular obstacle with radius 1 placed at the position $(-7, 0)$. Constrain the final time `T` to be between 3 and 10 seconds.

(d) Without proper initialization, the optimization is unlikely to succeed. Using the control solution from problem 2, initialize the solution to this problem using `algo.initializeControls(u0)`, where `u0` is a matrix containing the control trajectory from problem 2 (which gets stored in the variable `out.CONTROLS` after running the optimization script).

(e) Run the optimal control algorithm and plot the state and control history. Plot the (p_x, p_y) trajectory along with the obstacle. Output the optimal final time.

Note: upload your code using the File upload link on the website; in addition submit a printout of your code (only the part that was written/modified by you) and generated plots.