Mini ACADO Course, University of Freiburg

11th of May 2015, by Rien Quirynen

Swing-up of an inverted pendulum (PART 1)

NOTE: In case you did not do this yet, go through the ACADO installation instructions here: `www.acadotoolkit.org`. For this course, we decided to use the master instead of the stable branch so make sure to use the following command to download the code:

`git clone https://github.com/acado/acado.git -b master ACADOmaster`.

The website and our discussion forum are also good starting points for additional information and support in the future.

The guiding example that will be used throughout this course is the classical system of a pendulum, mounted on top of a cart as illustrated in Figure 1. It forms an ideal tutorial example for NMPC since it is simple and intuitive but it can also exhibit rather fast dynamics and nonlinear behavior. The position of the cart will be denoted by p and

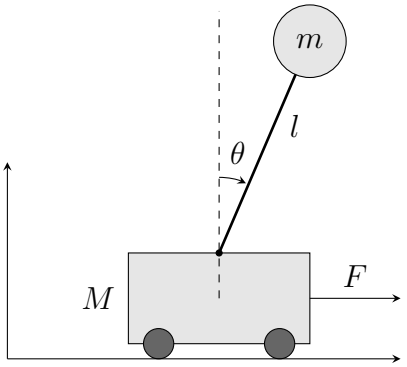

Figure 1: Schematic illustrating the inverted pendulum on top of a cart.

the pendulum configuration described by the angle θ, using the convention that $\theta = \pi$ rad corresponds to the pendulum hanging down. The system dynamics are described by the following implicit ODE system

$$\begin{aligned} (M+m)\ddot{p} - ml(\ddot{\theta}\cos(\theta) - \sin(\theta)\dot{\theta}^2) - F = 0, \\ l\ddot{\theta} - \ddot{p}\cos(\theta) - g\sin(\theta) = 0. \end{aligned} \tag{1}$$

By solving for the differential state derivatives $\ddot{p}$ and $\ddot{\theta}$, one can obtain the following explicit ODE formulation which is mathematically equivalent

$$\begin{aligned} \ddot{p} &= \frac{-ml\sin(\theta)\dot{\theta}^2 + mg\cos(\theta)\sin(\theta) + F}{M + m - m(\cos(\theta))^2}, \\ \ddot{\theta} &= \frac{-ml\cos(\theta)\sin(\theta)\dot{\theta}^2 + F\cos(\theta) + (M+m)g\sin(\theta)}{l(M + m - m(\cos(\theta))^2)}. \end{aligned} \tag{2}$$

1. The task is to make the pendulum perform a full swing-up starting from the stable position described by $p = 0\,\text{m}$ and $\theta = \pi\,\text{rad}$ while the reference point is unstable and corresponds to $0\,\text{m}$, $0\,\text{rad}$. Let us assume that the control input as well as the position of the cart are both bounded, respectively by $-20 \leq F \leq 20$ and $-2 \leq p \leq 2$. The OCP formulation in continuous time is the following

$$\begin{aligned}
\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad & \int_0^T \left\| \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} - y(t) \right\|_W^2 \mathrm{d}t \;+\; \|x(T) - y_N\|_{W_N}^2 && \text{(3a)} \\
\text{subject to} \quad & x(0) \;=\; \bar{x}_0, && \text{(3b)} \\
& \dot{x}(t) \;=\; f(x(t), u(t)), \qquad && \forall t \in [0, T], \quad \text{(3c)} \\
& -2 \;\leq\; p(t) \;\leq\; 2, \qquad && \forall t \in [0, T], \quad \text{(3d)} \\
& -20 \;\leq\; F(t) \;\leq\; 20, \qquad && \forall t \in [0, T], \quad \text{(3e)}
\end{aligned}$$

where $y(t)$ denotes the reference trajectory, the states are defined as $x = [p, \theta, \dot{p}, \dot{\theta}]^\top$ and the horizon length $T = 2\text{s}$. The used NMPC sampling time is equal to $T_s = 0.05\text{s}$, to be able to deal with the fast nonlinear dynamics in the system. A multiple shooting discretization will be used with $N = 40$ intervals over the control horizon.

You can download the template code for exercise 1 from our event website and go through it carefully to make sure you understand each part of it. You will realise that the following sections in the code are missing to obtain a working simulation:

(a) Define the explicit system of ODE equations (2), describing our system's dynamics. For this, we need to define a function `ode` like this:

```
ode = [  dot(x) == f(x,u)  ];
```

where `dot` is an ACADO defined operator which returns the time derivative and f denotes the expressions in the right-hand side of our model.

(b) Define the stage cost and terminal cost as follows

```
ocp.minimizeLSQ( W, h );
ocp.minimizeLSQEndTerm( WN, hN );
```

where the residual functions h and h_N and the weighting matrices W and W_N are needed to define the discrete objective $\sum_{k=0}^{N-1} \|h_k - y_k\|_{W_k} + \|h_N - y_N\|_{W_N}$. Note that the reference trajectory y will be defined after code generation.

(c) Define the constraints in the OCP, using for example the syntax:

```
ocp.subjectTo( xmin <= x <= xmax );
```

which defines a simple box constraint for a state called x on each stage.

(d) Define the remaining parameters to perform a closed-loop NMPC simulation:

- the initial downward position of the pendulum `X0` and the unstable upward position `Xref`, which forms the reference point to be tracked.
- choose a suitable initial state (`input.x`, $[(N+1)\times n_x]$) and control (`input.u`, $[N \times n_u]$) trajectory to be passed to the RTI solver. A good choice here can be important for the convergence of our scheme as we are solving a nonlinear OCP problem.

- we also need to define the reference trajectory, which remains constant in our case. You can use the matlab function `repmat` to define:
  `input.y = [repmat(Xref,N,1) repmat(Uref,N,1)];`
  NOTE: the dimension of `input.yN` must correspond to the residual function h_N in the terminal cost, which is typically different from the stage cost since the last node defines no corresponding control variables.
  `input.yN = Xref.';`

Do you manage to get a closed-loop NMPC simulation running? If yes, does the pendulum reach the reference point and how would you describe the performance?

NOTE: the flag `EXPORT` at the beginning of the script should be set to zero if one does not want to rerun the code generation and compilation process.

2. Tuning: the performance of the NMPC scheme will strongly depend on the various parameters in the OCP formulation such as the number of shooting intervals and the length of the horizon, the number of integration steps and the weighting matrices defining the stage cost. Try to look for values, especially of the weighting matrices, resulting in a satisfactory behavior of the NMPC controller. To avoid the need to go through the code generation and compilation process each single time we change our formulation, one can define e.g. the weighting matrices as

   ```
   W = acado.BMatrix(eye(length(h)));  WN = acado.BMatrix(eye(length(hN)));
   ocp.minimizeLSQ( W, h );            ocp.minimizeLSQEndTerm( WN, hN );
   ```

   after which they can be defined and passed to the generated solver as

   ```
   input.W = diag([ ... ]);            input.WN = diag([ ... ]);
   ```

   The `BMatrix` keyword stands for *Boolean Matrix* and defines the sparsity of the weighting matrix here.

3. RTI: remember that ACADO generates a custom RTI algorithm to solve each OCP instance, without iterating the SQP scheme each time until convergence (see literature for more information on the Real-Time Iteration scheme itself). As a result, the sampling time of the NMPC scheme needs to be small enough to let the solver converge over time and result in a stabilizing controller. Also the choice for the initialization and shifting procedure is typically very important. Let ACADO shift the optimization trajectories after each call to the solver by setting

   ```
   input.shifting.strategy = 1;
   ```

   which means that the last state will be reused at the end of the horizon. You can also use different ways of initializing the state trajectory

   `input.x = repmat(X0,N+1,1);` (initialize in the initial state)

   `input.x = repmat(Xref,N+1,1);` (initialize in the reference)

   Try to get an idea of how these choices affect the convergence of the scheme and therefore the control performance.

4. Modeling: let us try to use the implicit ODE formulation from (1) instead:

```
f = [ dot(p) - v == 0; ...
      ...
```

Note that we also need to choose one of the implicit integrators for both the simulation and for the OCP solver:

```
mpc.set( 'INTEGRATOR_TYPE', 'INT_IRK_GL2' );  (GL2, GL4, GL6, GL8)
```

or

```
mpc.set( 'INTEGRATOR_TYPE', 'INT_IRK_RIIA1' );  (RIIA1, RIIA3, RIIA5)
```

5. Stability: one often defines a terminal cost and/or a terminal constraint for stability reasons of the NMPC scheme. A simple example would be to impose the last state in the horizon to be equal to our reference point. Remember that you can specifically define constraints on the last stage as follows:

```
ocp.subjectTo('AT_END', ...);
```

Note that you need a sufficiently long control horizon for the resulting OCP formulation to be feasible.

6. Modeling (linear subsystems): instead of a fully nonlinear system of differential equations, one might have some partially linear dynamics as well. The supported model formulation in ACADO has therefore the following three stage structure:

$$\dot{x}^{[1]} = A_1 x^{[1]} + B_1 u, \tag{4a}$$
$$0 = f_2(x^{[1]}, x^{[2]}, \dot{x}^{[1]}, \dot{x}^{[2]}, z, u), \tag{4b}$$
$$\dot{x}^{[3]} = A_3 x^{[3]} + f_3(x^{[1]}, x^{[2]}, \dot{x}^{[1]}, \dot{x}^{[2]}, z, u), \tag{4c}$$

which can similarly be defined in the code generation tool as follows:

```
sim.setLinearInput( A1, B1 );   or   ocp.setLinearInput( A1, B1 );
sim.setModel( f2 );             or   ocp.setModel( f2 );
sim.setLinearOutput( A3, f3 );  or   ocp.setLinearOutput( A3, f3 );
```

To illustrate this, let us for example control the rate of change dF of the force instead. You can do this first by extending the nonlinear dynamics with the extra equation `dot(F) - dF == 0`. Afterwards, try to use a linear input system from Equation (4a) and you might notice a difference in computation time already.

Note that only our new state F qualifies as a linear input state and needs to be declared first, since ACADO will use this order to figure out the model structure. Make sure you also update the least squares objective and the corresponding weights, given the new state and control input. The figure also needs to be updated with the new order of states, and you can call the visualize function as

```
visualize(time, state_sim(:,2:end), Xref, xmin, xmax);
```

7. Model mismatch: to make the closed-loop simulation more interesting, we could consider a mismatch between the predictive model and the actual system. Let us for example introduce a disturbance in the simulation model:

   ```
   dot(p) - v - d == 0; ...
   ```

   Here, we used an *OnlineData* variable `d` which can be defined after code generation:

   ```
   OnlineData d;
   ```

   So in the eventual simulation call, we add the disturbance as follows:

   ```
   sim_input.od = 0.2;
   ```

   ```
   states = simulate_system(sim_input);
   ```

   where `od` stands for *Online Data*. This model mismatch results in a steady-state tracking error for the position of the cart in the NMPC simulation. A simple way to prevent this is by adding a state which integrates this tracking error and including it with a certain weighting in the stage cost:

   ```
   dot(err) - (p - pref) == 0; ...
   ```

   Note that this state can again be implemented efficiently using a linear output system as defined earlier in Equation (4c). Make sure that you add the disturbance only to the simulator, while using the nominal model in the OCP formulation.

Swing-up of an inverted pendulum (PART 2)

This is mostly a *do-it-yourself* part which encourages you to use ACADO on a control problem you would find particularly interesting to solve, based on the tools seen so far.

Define your own Optimal Control Problem

It is maybe time to use ACADO code generation for your own (NMPC) problem to see how it performs in controlling for example

- a race car
- a tethered kite
- baby Betty plane
- ball attached to a carousel
- ...

which are only a few examples!

Below are some extra features you might be interested in, e.g. for your own optimal control project:

1. **EXTRA: The quadratic subproblem**

   For sufficiently large horizon lengths, it would become more efficient to directly solve the fully structured QP instead of using condensing combined with qpOASES.

   You can e.g. download the qpDUNES code from `https://github.com/jfrasch/qpDUNES/wiki` and copy it completely into the folder *part1/export_MPC/qpdunes*. Then, we can specify ACADO to use qpDUNES by setting the following options:

   ```
   mpc.set( 'SPARSE_QP_SOLUTION',          'SPARSE_SOLVER'  );
   mpc.set( 'QP_SOLVER',                   'QP_QPDUNES'     );
   ```

   If you want to hide all the extra printed information from qpDUNES, you can additionally set the option:

   ```
   mpc.set( 'PRINTLEVEL',                  0                );
   ```

   In case you have a license for the FORCES solver (information can be found on `https://www.embotech.com/FORCES-Pro`), you can also use that one:

   ```
   mpc.set( 'QP_SOLVER',                   'QP_FORCES'      );
   ```

   Another promising solver is HPMPC on which more information can be found on `https://github.com/giaf/hpmpc` and which can be used from ACADO, even though this interface is still under active development.

2. **EXTRA: ACADO Simulink interface**

   ACADO supports the automatic code generation of a Simulink interface and a corresponding `make` script by simply setting the option

   ```
   mpc.set( 'GENERATE_SIMULINK_INTERFACE', 'YES'          );
   ```

3. **EXTRA: NMPC-MHE closed-loop**

   If you are interested in closing your simulation loop with an observer instead of assuming full state information, you can export an OCP solver implementing a Moving Horizon Estimation (MHE) scheme. We can surely provide example code or help you with setting up this closed-loop simulation.

4. **EXTRA: Exact Hessian based, time-optimal swing-up**

   If you are interested in a time-optimal formulation of the inverted pendulum swing-up or in Exact Hessian based RTI in general, you can ask for help setting up this problem formulation or we can provide you with example code.