1 Embedded Optimal Control with ACADO

Model Predictive Control (MPC) consists in repeatedly solving an Optimal Control Problem (OCP) on a receding horizon. In order to be able to meet the real-time requirements, autogenerated tailored solvers can be used. ACADO Toolkit is a software environment and algorithm collection for automatic control and dynamic optimization [1] that is able to export efficient tailored C code for applying nonlinear MPC to fast dynamic systems [3].

The exported code implements a direct method for optimal control (“first discretize then optimize”) based on Multiple Shooting [2]. The problem is discretized on a time grid and the nonlinear system dynamics are integrated separately on each interval using fixed-stepsize Runge Kutta integration methods [4]. The OCP is thus transcribed into a Nonlinear Programming Problem (NLP) which is solved by means of Sequential Quadratic Programming (SQP). The Real Time Iteration (RTI) scheme allows for significant speedups of computations. The latency between the moment at which the state is measured and the moment at which the control is applied to the system is also reduced to the solution of a QP (the same as for linear MPC!).

2 NMPC for an Overhead Crane

In this exercise, we will use ACADO code generation to control the overhead crane used in the previous exercises.

2.1 Simulation of Dynamic Systems

After installing and compiling the MATLAB interface of ACADO, run getting_started.m. This file exports an integrator tailored to the system and checks the precision of the simulation by comparing the result of the fixed-stepsize integrator to the one of the general-purpose integration routine ode45, which runs slower, but has accuracy guarantees. Try to increase the number of integration steps or use a different integrator and check how the accuracy is affected by the choice.

2.2 Nonlinear MPC

After getting familiar with the simulation of dynamic systems, let’s export an MPC controller and write a closed loop simulation.

1. Starting from the code you used in Exercise 1, define an optimal control problem. Hint Use the following syntax:
acadoSet('problemname', 'mpc');

Ts = ;
N = ;
n_steps = ; \% integrator steps
ocp = acado.OCP( 0.0, N*Ts, N ];
\% Export the weighting matrices, they will be defined at runtime
ExportVariable QQ(n_XD,n_XD) RR(n_U,n_U) QT(n_XD,n_XD)
ocp.minimizeLSQ( QQ,RR );
ocp.minimizeLSQEndTerm( QT );

ocp.subjectTo( ... <= uC <= ... );
ocp.setModel( f );

2. Define an MPC controller to be exported. **Hint** Use the following code:

mpc = acado.MPCexport( ocp );
mpc.set( 'HESSIAN_APPROXIMATION', 'GAUSS_NEWTON' );
mpc.set( 'DISCRETIZATION_TYPE', 'MULTIPLE_SHOOTING' );
mpc.set( 'INTEGRATOR_TYPE', 'INT_IRK_GL4' );
mpc.set( 'NUM_INTEGRATOR_STEPS', N*n_steps );
mpc.set( 'MEX_VERBOSE', 1 );

if EXPORT
    dirname = 'MPC_export';
    try
        rmdir(dirname,'s');
    catch exception
        warning(exception.message)
    end
    mkdir(dirname)
    copyfile('qpoases',[dirname,'/qpoases']) \% Copy qpOASES to the export folder
    mpc.exportCode( dirname );
end

3. Now you have exported a tailored OCP solver. Write a simulation loop that, starting from the initial state, computes the next control to be applied to the system and simulates the system to get the next state and iterate. Use the same weighting matrices, prediction horizon, initial state and reference that you used in the previous exercises. **Hint:** You can use the following syntax to call solve the MPC problem

[U_mpc,X_mpc,KKT_mpc] = MPCstep(X0,X,U,Xref,Uref,Q,R,S);

You can use ode45 for the simulation.

4. The terminal cost can be used to summarize the cost-to-go and have a better approximation of the infinite-horizon problem. For linear systems, the terminal cost weighting matrix is
obtained by computing the solution of the discrete-time algebraic Riccati equation (DARE). For nonlinear systems, the DARE solution obtained by using the linearization at the reference is a good approximation of the cost-to-go. **Hint** Use the following code to linearize the system at the reference and solve the DARE with the Matlab function `dare`.

```matlab
states = integrate(Xref, Uref);
A = states.sensX;
B = states.sensU;
```

**References**


