

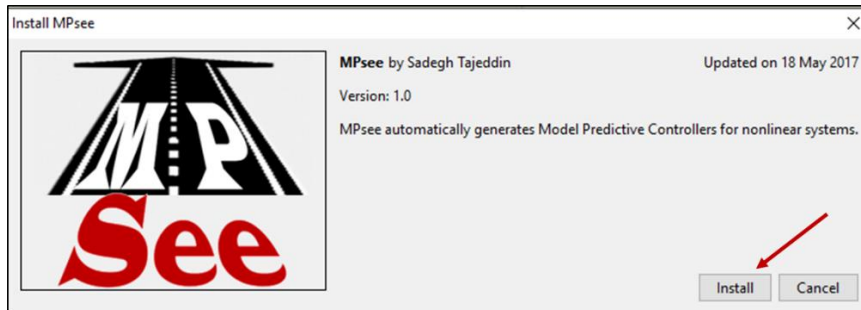
MPsee is a MATLAB-based mathematical tool that collects all the necessary information about the optimal control problem, such as dynamics of the system, objective function, constraints, etc., plus the solver options set by the user and then automatically generates an NMPC controller for simulation and implementation purposes.

Basically, MPsee consists of two separate set of calculations: offline and online; meaning that the mathematical process of solving an optimal control problem, explained in previous section, is categorized into two parts. The first part including forming the Hamiltonian and calculation of the derivatives are performed, symbolically offline, with the help of MATLAB symbolic toolbox and the resulted functions and expressions are outputted as MATLAB standard function codes. Then, these functions are repeatedly called during the online calculations within the controller's function. This procedure helps significantly reduce the online computational load by preprocessing the problem offline.

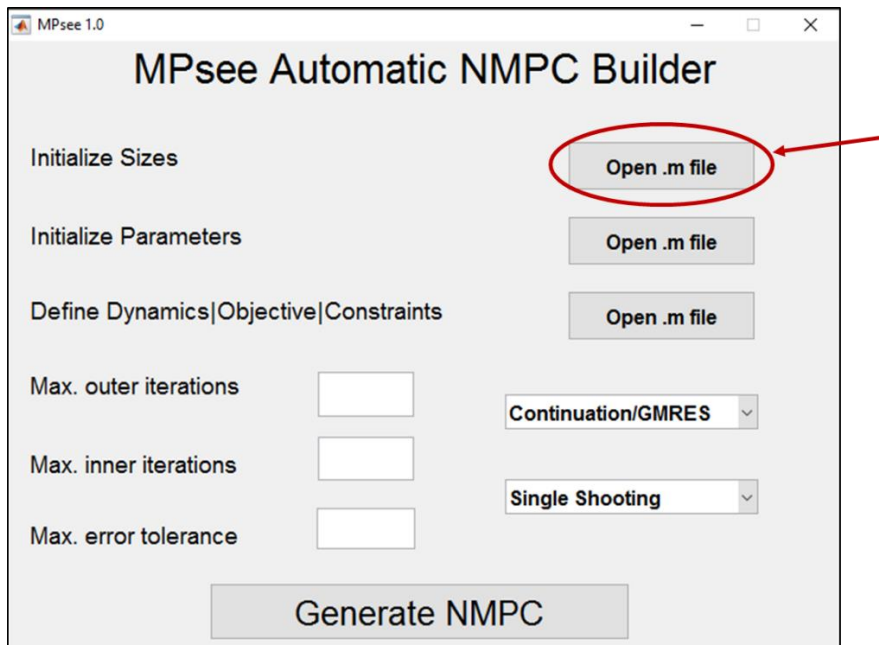
Using MPsee, a Graphical User Interface (GUI), receives the optimal control problem definition from the user. In the different predefined fields of the GUI, the user can enter the expressions for the control-oriented model, objective function, equality or inequality constraints and terminal cost. Moreover, the user can define his/her own User-Defined Parameters (UDP) to be used inside the expressions. Additionally, depending on the control problem, the control-oriented model or the objective function might have one or more Time-Varying Parameters (TVP) which the user can also define them in the GUI and later provide the relevant signals in the simulations. For example, in order to solve an optimal tracking problem, one can define a reference signal as a TVP in the problem definition section and then provide the reference signal in the SIMULINK file. Please note that TVPs can be defined as frozen or also dynamically as an array over the prediction horizon.

To work with this generator:

1. Install the MPsee toolbox



2. Go to your arbitrary MATLAB directory.
3. Open MPsee user interface by typing "MPsee" in the Command Window.
4. Initialize the vector sizes. To do so, open *go\_sizes.m* file and enter the sizes of your state and input vectors. **Save changes to the .m file**



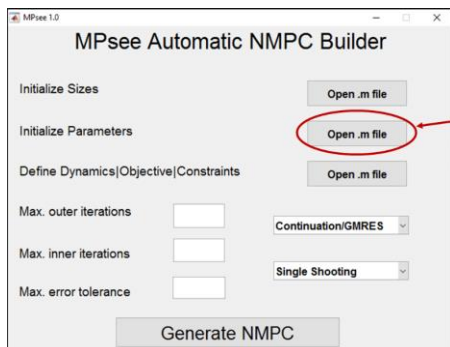
```

%% =====
%% ===== Define Sizes Here =====
%% =====
dimx=1;      %% number of states
dimu=1;      %% number of inputs
dimec=1;     %% number of equality constraints
dimic=1;     %% number of inequality constraints
N=10;        %% length of horizon

```

NOTE: only make changes where there is a red square. Do not change the variable names! Do not add comments to the code!

5. Define the parameters. **Save the .m file**. Parameters can be defined in three different ways:



```

%% =====
%% ===== Define Parameters Here =====
%% =====
% Fixed Parameters
g=9.81;
m=1720;
f1=0.01;
f2=0.00012;
CFD=0.3598;
Umin=-5;
Umax=5;
w1=1;
w2=0.1;
w3=0;
% Time Varying Parameters
syms tau dt % Required (Do Not Change!)
syms Vf Theta
Vmax=Vf+10/3.6;
Vmin=5;
%-----Frozen Time-Varying Parameters-----
TVP_f=[Vf dt]; % "dt" MUST be the last Frozen IVP
%-----Dynamic Time-Varying Parameters-----
TVP=[Theta];
% Exterior Penalty Costs Weights
R_value=[1;1;1;1];

```

- a. Fixed Parameters, that are set offline and will not change during the simulation. You can add as many as fixed parameters as needed.

```
%% _____ Fixed Parameters _____
a=1;
b=2;
Umin=-5;
Umax=5;
w1=1;
w2=2;
```

NOTE: Delete the dummy variables in the red box and add your fixed parameters. Do not add comments to the code!

- b. Dynamic Time-varying Parameters (Dynamic TVP) that change during the simulation and NMPC considers their changes inside the prediction horizon.
- c. Frozen Time-varying Parameters (frozen TVP) that change during the simulation and NMPC considers them constant inside the prediction horizon though updates their value every sampling time.


```
%% _____ Time Varying Parameters _____
syms tau dt          %% Required (Do Not Change!)
syms Xf d
Xmax=Xf+10;
Xmin=Xf-10;
```

NOTE: Delete the dummy variables in the red box and add your symbolic variables. You can also define other symbolic complex variables (as shown in the blue box) in terms of the primary ones. Do not add comments to the code!

```
TVP_f=[Xf dt]; %% "dt" MUST be the last Frozen TVP
%% -----Dynamic Time-Varying Parameters-----
TVP=[d];
```

NOTE: You must list your frozen TVPs and dynamic TVPs. Only make changes to the code where there is a red box as shown above. Do not add comments to the code!

```
%% Exterior Penalty Costs Weights
R_value=[1;1;1;1];
```



NOTE: You must set the weighting factors for the penalty costs of any inequality constraints that you may have. For example, if you have 4 inequality constraints, you must define 4 weighting factors inside the “*R\_value*” vector. These values must be only numerical.

NOTE 1: Any Time-varying Parameter must be defined as MATLAB symbolic variable. By default, MPsee has one frozen TVP, *dt*, that is the timestep size inside the prediction horizon; and one dynamic TVP, *tau*, that is the stepping number inside the prediction horizon. Do not change these two parameters in the settings.

NOTE 2: dynamic and frozen TVPs are defined for MPsee as *TVP* vector and *TVP\_f* vector, respectively. It is important to note that later in the Simulink environment, user must provide signals for dynamic and frozen TVPs in the same order that he/she defines them in *TVP* and *TVP\_f* vectors.

NOTE3: *dt* must always be the last frozen TVP defined in the *TVP\_f* vector.

6. Define the system’s dynamics (control\_oriented model), objective function and constraints. Open the relative file *go\_problem.m* . Functions (*fxu*, *Gxu*, *Cxu*) must be defined as vectors and functions (*Lk*, *Phi*) must be defined as scalar in terms of “*Uk*” the input vector, “*Xk*” the states vector and any parameter defined earlier. **Save the .m file**

$$\begin{aligned}
 \dot{X}k &= fxu(Xk, Uk, TVP, TVP_f) \\
 Gxu(Xk, Uk, TVP, TVP_f) &= 0 \\
 Cxu(Xk, Uk, TVP, TVP_f) &\leq 0 \\
 J &= Phi(Xk) + \int Lk(Xk, Uk, TVP, TVP_f)
 \end{aligned}$$

MPsee 1.0

## MPsee Automatic NMPC Builder

Initialize Sizes

Initialize Parameters

Define Dynamics|Objective|Constraints

Max. outer iterations  Continuation/GMRES

Max. inner iterations  Single Shooting

Max. error tolerance

```

%% =====
%% ##### Define Optimal Control Here #####
%% =====
% Uk is the vector of inputs
% Xk is the vector of states
% _____ Vector Field Definition
% _____ in terms of Uk, Xk
fxu=[Uk(1)-(CFD/m*Xk(1)^2+g*sin(Theta)+g*(f1+f2*Xk(1))*cos(Theta))];
% _____ Constraints Definition
% _____ in terms of Uk, Xk
Gxu=0; % Equality Constraints
Cxu=[Xk(1)-Vmax % Inequality Constraints
     Vmin-Xk(1)
     Uk(1)-Umax
     Umin-Uk(1)];
% _____ Objective Function Definition
% _____ in terms of Uk, Xk
Lk=0.5*w1*(Xk(1)-Vf)^2+0.5*w2*(Uk(1))^2; % Tranjectory Cost (Integral Terms)
Lk=Lk+R'*Cxu.^2; % Required (Do Not Change!)
Phi=0.5*w3*(Xk(1)-Vf)^2; % Terminal Cost
    
```

```

%% #####
%% ##### Define Optimal Control Here #####
%% #####
%% Uk is the vector of inputs
%% Xk is the vector of states
%% _____ Vector Field Definition _____
%% _____ in terms of Uk, Xk _____
fxu=[sin(Xk(2))
     U(1)^2];

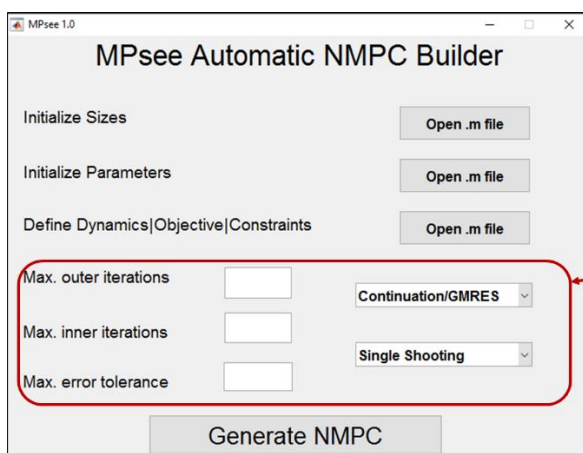
%% _____ Constraints Definition _____
%% _____ in terms of Uk, Xk _____
Gxu=[0]; %% Equality Constraints
Cxu=[Xk(1)-Vmax
     Vmin-Xk(1)
     Uk(1)-Umax
     Umin-Uk(1)]; %% Inequality Constraints

%% _____ Objective Function Definition _____
%% _____ in terms of Uk, Xk _____
Lk=0.5*w1*(Xk(1)-Xf)^2+0.5*w2*(Uk(1))^2; %% Trajectory Cost (Integral Terms)
Lk=Lk+R'*Cxu.^2; %% Required (Do Not Change!)
Phi=0.5*w3*(Xk(1)-Xf)^2; %% Terminal Cost

```

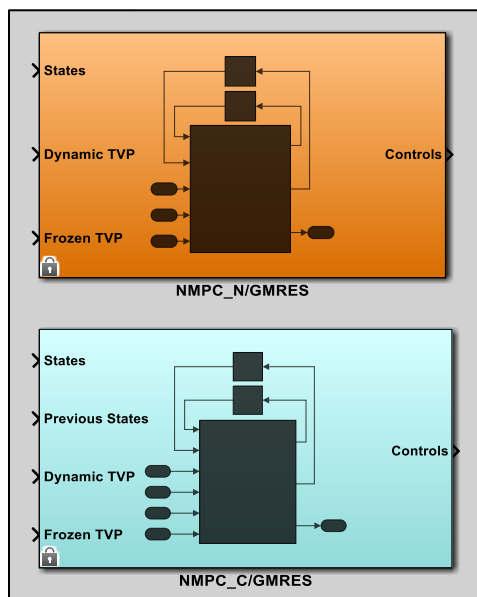
NOTE: You can only make changes inside the red boxes shown in the above picture. Use “Uk” as input vector, “Xk” as state vector and any of your fixed or time-varying parameters that you defined earlier to define “fxu”, “Gxu”, “Cxu” vectors and “Lk” and “Phi” scalar functions.

7. Finally, setup the real-time optimization method options and click *Generate NMPC*.



The image shows the 'MPsee Automatic NMPC Builder' window. It has three tabs: 'Initialize Sizes', 'Initialize Parameters', and 'Define Dynamics|Objective|Constraints'. The 'Define Dynamics|Objective|Constraints' tab is active. It contains three input fields for 'Max. outer iterations', 'Max. inner iterations', and 'Max. error tolerance'. To the right of these fields are two dropdown menus: 'Continuation/GMRES' and 'Single Shooting'. A red box highlights the area containing the iteration and tolerance fields and the dropdown menus. A red arrow points to the 'Continuation/GMRES' dropdown. At the bottom of the window is a 'Generate NMPC' button.

8. Use the relative NMPC Simulink block from the MPsee Simulink Library, *NMPCLib.slx*, for simulation. User must provide the vector of states to the NMPC block as a feedback signal. Any TVP must also be provided to the NMPC block in the order that was defined in the generation phase. Please note that Dynamic TVP signal to the NMPC block is generally a “ $p$  by  $N$ ” matrix, where  $p$  is the number of TVPs and  $N$  is the prediction horizon size. In other words, this signal is a concatenation of  $p$  vectors with a length of  $N$ .



9. After a successful NMPC build, a message similar to the following appears on the Command Window indicating a successful build and the specifications of the generated NMPC:

---

```

NMPC was successfully built:
Continuation= no
Shooting Method= single
Error tolerance= 0.01
Max. inner iterations= 5
Max. outer iterations= 2

```

---



10. After NMPC generation, two variables are added to the MATLAB Workspace which are essential for simulation: "dU0" and "U0". "U0" refers to the initial guess for the vector of unknowns, basically the optimal solution to the problem. Convergence of NMPC solution in the beginning of the simulation highly depends on this initial guess. By default, "U0" is a zero vector. However, user is strongly advised to change this vector to an initial guess close to the optimal solution.

## References

- [1] Sadegh Tajeddin (2016). *Automatic Code Generation of Real-Time Nonlinear Model Predictive Control for Plug-in Hybrid Electric Vehicle Intelligent Cruise Controllers*. UWSpace. <http://hdl.handle.net/10012/10740>
- [2] Tajeddin, Sadegh, Mahyar Vajedi, and Nasser L. Azad. "A Newton/GMRES Approach to Predictive Ecological Adaptive Cruise Control of a Plug-in Hybrid Electric Vehicle in Car-following Scenarios." *IFAC-PapersOnLine* 49.21 (2016): 59-65.
- [3] KelleyC, T. "Iterative Methods for Linear and Nonlinear Equations." Raleigh N. C.: North Carolina State University (1995).
- [4] Ohtsuka, Toshiyuki. "A continuation/GMRES method for fast computation of nonlinear receding horizon control." *Automatica* 40.4 (2004): 563-574.