

# 適応MPCコントローラの実装

このサンプルでは、適応MPCを設計するブロック“Adaptive MPC Controller”を用いた設計の例を示す。  
また、設計後のCコード生成、SIL、PILの例も合わせて紹介する。

## 初期化

```
clc; Simulink.sdi.clear; Simulink.sdi.clearPreferences; Simulink.sdi.close;
proj = currentProject;
model_name = 'Vehicle_system_Adaptive_MPC';
ada_controller_name = 'Adaptive_MPC_Controller';
ts = set_TimeStep(Simulink.data.dictionary.open('sim_data_vehicle.slidd'));
```

## プラントモデルを定式化

4輪の前輪操舵車両を考える。等価二輪モデルで近似する（図1）。

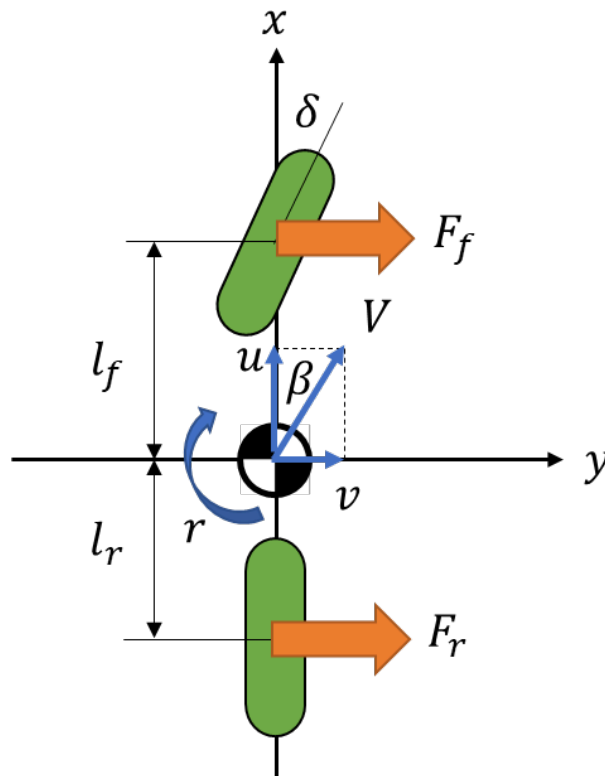


図1. 等価二輪モデル

ここで、等価二輪モデルの数式を元にプラントモデリングとMPCの定式化を行っていく。このモデルにおいては、以下の式eq\_1, eq\_2が成り立つ。

```
% 変数定義
m = sym('m','real'); u = sym('u','real'); v = sym('v','real');
r = sym('r','real'); F_f = sym('F_f','real'); F_r = sym('F_r','real');
I = sym('I','real'); l_f = sym('l_f','real'); l_r = sym('l_r','real');
v_dot = sym('v_dot','real'); r_dot = sym('r_dot','real');
```

```
V = sym('V','real'); beta = sym('beta','real'); beta_dot = sym('beta_dot','real');
% 方程式
eq_1 = m * (v_dot + u * r) == F_f + F_r
```

$$eq\_1 = m (\dot{v} + r u) = F_f + F_r$$

$$eq\_2 = I * r\_dot == l\_f * F\_f - l\_r * F\_r$$

$$eq\_2 = I \dot{r} = F_f l_f - F_r l_r$$

ここで、 $v$ は横速度、 $r$ は車のヨーレート（z軸方向の回転角速度）である。 $\dot{v}$ と $\dot{r}$ はそれぞれ $v$ と $r$ の時間微分を表す。 $F_f$ と $F_r$ は前後のタイヤから発生する横向きの力を表す。

車体の横滑り角 $\beta$ が小さい範囲の運動を考えると、次のような関係が成り立つ。

$$u \cong V, v = V \sin \beta \cong V \beta$$

速度を一定と仮定すると、 $\dot{v}$ は以下のようになる。

$$\dot{v} = V \dot{\beta}$$

これらの関係を用いると、eq\_1は、以下のように書き直すことができる。

```
% 代入
eq_1 = subs(eq_1, [u, v_dot], [V, V * beta_dot])
```

$$eq\_1 = m (V \dot{\beta} + V r) = F_f + F_r$$

前後のタイヤの横滑り角を $\beta_f$ 、 $\beta_r$ とすると、横力は

$$F_f = -2K_f \beta_f, F_r = -2K_r \beta_r$$

となる。

また、 $\beta_f$ と $\beta_r$ は近似的に次のように表すことができる。

$$\beta_f = \beta + \frac{l_f}{V} r - \delta, \beta_r = \beta - \frac{l_r}{V} r$$

$\delta$ は操舵角である。

eq\_1、eq\_2に上記の式を代入すると、以下のようになる。

```
% 変数定義
K_f = sym('K_f','real'); K_r = sym('K_r','real'); delta = sym('delta','real');
beta_f = sym('beta_f','real'); beta_r = sym('beta_r','real');
% 代入
eq_vec = subs([eq_1, eq_2], [F_f, F_r], [-2 * K_f * beta_f, -2 * K_r * beta_r]);
eq_vec = subs(eq_vec, [beta_f, beta_r], ...
    [beta + (l_f / V) * r - delta, beta - (l_r / V) * r]);
eq_1 = eq_vec(1)
```

$$eq\_1 =$$

$$m(V\dot{\beta} + Vr) = -2K_f\left(\beta - \delta + \frac{l_f r}{V}\right) - 2K_r\left(\beta - \frac{l_r r}{V}\right)$$

```
eq_2 = eq_vec(2)
```

```
eq_2 =
```

$$I\dot{r} = 2K_r l_r \left(\beta - \frac{l_r r}{V}\right) - 2K_f l_f \left(\beta - \delta + \frac{l_f r}{V}\right)$$

$\dot{\beta}$ と $\dot{r}$ を求める計算式を導出する。

```
sol_vec = solve([eq_1, eq_2], [beta_dot, r_dot]);
beta_dot = sol_vec.beta_dot
```

```
beta_dot =
```

$$-\frac{2K_f V \beta + 2K_r V \beta - 2K_f V \delta + 2K_f l_f r - 2K_r l_r r + V^2 m r}{V^2 m}$$

```
r_dot = sol_vec.r_dot
```

```
r_dot =
```

$$-\frac{2(K_f l_f^2 r + K_r l_r^2 r + K_f V \beta l_f - K_r V \beta l_r - K_f V \delta l_f)}{I V}$$

$\beta$ と $r$ は、 $\dot{\beta}$ と $\dot{r}$ を時間積分することで得られる。

## 状態空間表現

車両の運動を状態空間表現に置き換える。

[入力U]

操舵角 $\delta$ 、車両加速度 $a$

```
normal_input_names = {'delta', 'a'};
% 変数定義
a = sym('a', 'real');
U = [delta; a];
```

[状態x]

車両の速さ $V$ 、横滑り角 $\beta$ 、ヨーレート $r$ 、ヨー角 $\theta$ 、X方向位置 $p_x$ 、Y方向位置 $p_y$

```
state_names = {'px', 'py', 'theta', 'r', 'beta', 'V'};
% 変数定義
theta = sym('theta', 'real'); px = sym('px', 'real'); py = sym('py', 'real');
X = [px; py; theta; r; beta; V];
```

[出力y]

車両の速さ $V$ 、ヨーレート $r$ 、ヨー角 $\theta$ 、X方向位置 $p_x$ 、Y方向位置 $p_y$

```
output_names = {'px', 'py', 'theta', 'r', 'V'};
Y = [px; py; theta; r; V];
```

連続非線形の状態方程式 $\dot{X} = f(X, U)$ の $f$ を作る。

$$\dot{x} = V \sin \theta$$

$$\dot{y} = V \cos \theta$$

$$\dot{\theta} = r$$

$$\dot{r} = \text{sol\_vec.r\_dot}$$

$$\dot{\beta} = \text{sol\_vec.beta\_dot}$$

$$\dot{V} = a$$

よって、

```
f = [ ...
      V * cos(theta);
      V * sin(theta);
      r;
      sol_vec.r_dot;
      sol_vec.beta_dot;
      a;
      ]
```

$$f = \begin{pmatrix} V \cos(\theta) \\ V \sin(\theta) \\ r \\ -\frac{2(K_f l_f^2 r + K_r l_r^2 r + K_f V \beta l_f - K_r V \beta l_r - K_f V \delta l_f)}{I V} \\ -\frac{2 K_f V \beta + 2 K_r V \beta - 2 K_f V \delta + 2 K_f l_f r - 2 K_r l_r r + V^2 m r}{V^2 m} \\ a \end{pmatrix}$$

出力方程式 $Y = h(X)$ の $h$ は以下のようになる。

```
h = [X(1); X(2); X(3); X(4); X(6)]
```

h =

$$\begin{pmatrix} p_x \\ p_y \\ \theta \\ r \\ V \end{pmatrix}$$

得られた計算式を**MATLAB**コード生成する。

ただし、そのままコード生成すると、割り算の箇所はゼロ割を回避できていない。そこで、生成されたコードを上書きして自動でゼロ割回避を挿入する関数を用意した。

以下のようにその関数を適用する。適用後の関数は(元々のファイル名) + '\_azd' という名前となる。

```
file_path = [char(proj.RootFolder), separator, 'gen_script', separator, 'calc_nonlinear_f.m'];
matlabFunction(f, 'File', file_path);
insert_zero_divide_avoidance(file_path);
file_path = [char(proj.RootFolder), separator, 'gen_script', separator, 'calc_nonlinear_h.m'];
matlabFunction(h, 'File', file_path);
insert_zero_divide_avoidance(file_path);
```

## MPCのための定式化

**Adaptive MPC**は、非線形モデルを逐次線形化してモデル予測制御を行っている。そこで、**f**と**h**を線形化したA、B、C行列が必要である。

**Ac = jacobian(f, X)**

**Ac =**

$$\begin{pmatrix} 0 & 0 & -V \sin(\theta) & 0 & 0 \\ 0 & 0 & V \cos(\theta) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -\frac{2(K_f l_f^2 + K_r l_r^2)}{I V} & -\frac{2(K_f V l_f - K_r V l_r)}{I V} & \frac{2(K_f l_f^2 r + K_r l_r^2 r + K_j)}{I V} \\ 0 & 0 & 0 & -\frac{m V^2 + 2 K_f l_f - 2 K_r l_r}{V^2 m} & -\frac{2 K_f V + 2 K_r V}{V^2 m} & \frac{2(2 K_f V \beta + 2 K_r V \beta - 2 K_f V)}{V} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

**Bc = jacobian(f, U)**

**Bc =**

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{2 K_f l_f}{I} & 0 \\ \frac{2 K_f}{V m} & 0 \\ 0 & 1 \end{pmatrix}$$

```
Cc = jacobian(h, X)
```

```
Cc =  


$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```

同じようにコード生成とゼロ割回避を挿入する。

```
file_path = [char(proj.RootFolder), separator, 'gen_script', separator, 'calc_Ac.m'];  
matlabFunction(Ac, 'File', file_path);  
insert_zero_divide_avoidance(file_path);  
file_path = [char(proj.RootFolder), separator, 'gen_script', separator, 'calc_Bc.m'];  
matlabFunction(Bc, 'File', file_path);  
insert_zero_divide_avoidance(file_path);  
file_path = [char(proj.RootFolder), separator, 'gen_script', separator, 'calc_Cc.m'];  
matlabFunction(Cc, 'File', file_path);  
insert_zero_divide_avoidance(file_path);
```

初期の状態空間モデルを数値化する。

```
x0 = [0; 0; 0; 0; 0; 1];  
u0 = zeros(size(U, 1), 1);  
UdD0 = zeros(size(U, 1), 1);  
  
uNum_MPC = size(u0, 1);  
xNum_MPC = size(x0, 1);  
yNum_MPC = size(Y, 1);  
  
% パラメータ  
m_val = 2000;  
l_f_val = 1.4;  
l_r_val = 1.6;  
I_val = 4000;  
K_f_val = 12e3;  
K_r_val = 11e3;  
div_min = 1e-3;  
  
[Ad,Bd,Cd,Dd,Ud,Yd,Xd,DXd] = calc_Discrete_SS_for_Adaptive( ...  
    I_val, K_f_val, K_r_val, l_f_val, l_r_val, m_val, ts, x0, u0, div_min);  
dsys = ss(Ad, Bd, Cd, Dd, ts);  
dsys.InputName = normal_input_names;  
dsys.StateName = state_names;  
dsys.OutputName = output_names;  
dsys
```

```
dsys =
```

```
A =  

      px      py      theta      r      beta      V  
px      1      0      0      0      0      0.02
```

py	0	1	0.02	0.0001696	4.203e-07	0
theta	0	0	1	0.01562	5.816e-05	0
r	0	0	0	0.5964	0.004909	0
beta	0	0	0	-0.002455	0.6313	0
V	0	0	0	0	0	1

```

B =
      delta      a
px      0      0.0002
py 9.917e-06      0
theta 0.001429      0
r      0.1319      0
beta   0.1921      0
V      0      0.02

```

```

C =
      px      py      theta      r      beta      V
px      1      0      0      0      0      0
py      0      1      0      0      0      0
theta   0      0      1      0      0      0
r      0      0      0      1      0      0
V      0      0      0      0      0      1

```

```

D =
      delta      a
px      0      0
py      0      0
theta   0      0
r      0      0
V      0      0

```

サンプル時間: 0.02 seconds  
 離散時間状態空間モデル。

Adaptive MPCを用いたMPC制御器を構成する。

```
mpcObj = mpc(dsys)
```

```

-->"mpc" オブジェクトの "PredictionHorizon" プロパティが空です。PredictionHorizon = 10 を試用します。
-->"mpc" オブジェクトの "ControlHorizon" プロパティが空です。2 であると仮定します。
-->"mpc" オブジェクトの "Weights.ManipulatedVariables" プロパティが空です。既定の 0.00000 を仮定します。
-->"mpc" オブジェクトの "Weights.ManipulatedVariablesRate" プロパティが空です。既定の 0.10000 を仮定します。
-->"mpc" オブジェクトの "Weights.OutputVariables" プロパティが空です。既定の 1.00000 を仮定します。
    for output(s) y1 y2 and zero weight for output(s) y3 y4 y5

```

MPC object (created on 06-Jun-2020 09:42:40):

```

-----
Sampling time:      0.02 (seconds)
Prediction Horizon: 10
Control Horizon:    2

```

Plant Model:

```

-----
2 manipulated variable(s) -->| 6 states |
                                |         | --> 5 measured output(s)
0 measured disturbance(s) -->| 2 inputs  |
                                |         | --> 0 unmeasured output(s)
0 unmeasured disturbance(s) -->| 5 outputs |
-----

```

Disturbance and Noise Models:

```

Output disturbance model: default (type "getoutdist(mpcObj)" for details)
Measurement noise model: default (unity gain after scaling)

```

Weights:

```

ManipulatedVariables: [0 0]
ManipulatedVariablesRate: [0.1000 0.1000]
OutputVariables: [1 1 0 0 0]
ECR: 100000
State Estimation: Default Kalman Filter (type "getEstimator(mpcObj)" for details)

Unconstrained

```

```

% 予測ホライズン、制御ホライズンの設定
mpcObj.PredictionHorizon = 16;
mpcObj.ControlHorizon = 1;

% ノミナル状態を更新
mpcObj.Model.Nominal = struct('U',[Ud; UdD0], 'Y',Yd, 'X',Xd, 'DX',DXd);

% 制約
% 操舵角は30deg以内であること
delta_limit = 30;
mpcObj.ManipulatedVariables(1).Max = delta_limit * pi / 180;
mpcObj.ManipulatedVariables(1).Min = -delta_limit * pi / 180;
% 加速度は2m/s^2以内であること
acc_limit = 2;
mpcObj.ManipulatedVariables(2).Max = acc_limit;
mpcObj.ManipulatedVariables(2).Min = -acc_limit;

% チューニング
% out_dist_model = ss(zeros(yNum_MPC), zeros(yNum_MPC), zeros(yNum_MPC), ...
%                     diag([0; 0; 0; 0; 0]), ts);
% setoutdist(mpcObj, 'model', out_dist_model);
% noise_model = ss(zeros(yNum_MPC), zeros(yNum_MPC), zeros(yNum_MPC), ...
%                  diag([2.5; 2.5; 1; 1; 0]), ts);
% mpcObj.Model.Noise = noise_model;

% 最適化の重みを設定
mpcObj.Weights.OutputVariables = [1, 1, 0, 0, 1];
mpcObj.Weights.ManipulatedVariables = [0.1, 0.1];
mpcObj.Weights.ManipulatedVariablesRate = [0.0, 0.0];

```

設計の妥当性確認

```
% review(mpcObj)
```

シミュレーション

```

open_system(model_name);
set_param([model_name, '/MPC_Controller'], 'SimulationMode', 'Normal');
% set_param(modelName, 'SimulationCommand', 'update');
sim(model_name);

```

測定出力チャネル #1 に外乱が追加されていないと仮定します。  
測定出力チャネル #2 に外乱が追加されていないと仮定します。  
-->測定出力チャネル #5 に追加された出力外乱は、合成ホワイト ノイズであると仮定します。  
-->測定出力チャネル #3 に追加された出力外乱は、合成ホワイト ノイズであると仮定します。



-->測定出力チャネル #4 に追加された出力外乱は、合成ホワイト ノイズであると仮定します。  
-->"mpc" オブジェクトの "Model.Noise" プロパティが空です。それぞれの測定出力チャネルにホワイト ノイズを仮定します。

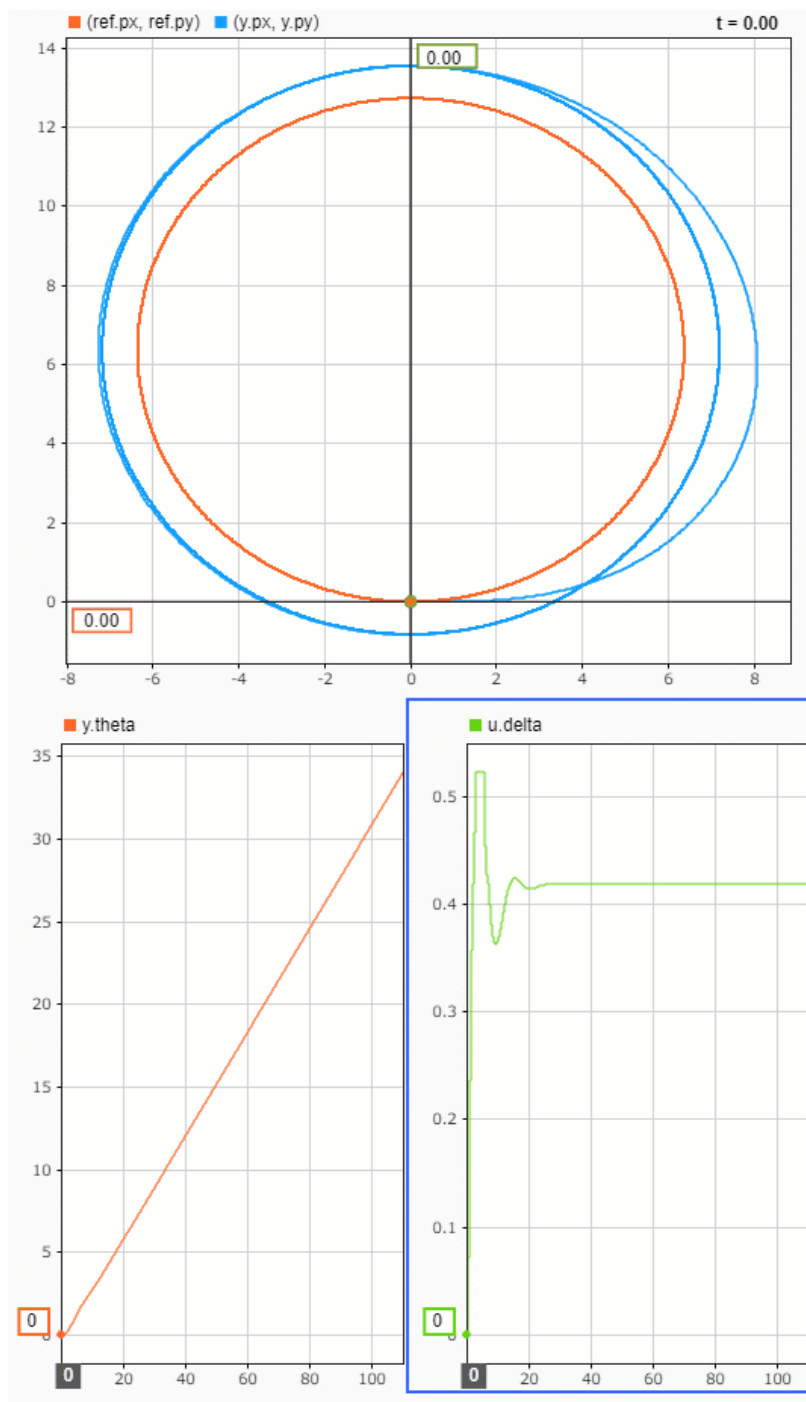
## 結果の表示

```
plot_vehicle_result_in_SDI;
```

"Vehicle\_system\_Adaptive\_MPC/Reference"のSignal Editorブロックのアクティブなシナリオを"control\_y\_V"で動作確認する。このシナリオでは、制御対象のpyとVのステップ応答を確認するシナリオである。

線形MPCと同様にチューニングを行うことができる。98から104行目のコメントアウトされている部分を有効にして、応答が変わることを確認すること。

一方で、シナリオ"regular\_circle\_turn\_vehicle"は定常円旋回を行うシナリオである。以下のように、回転を続ける位置指令値に対して、一定距離を保ちながら追いつける動作をする。



この図はシミュレーションデータインスペクターのXYプロットを用いて作成することができる。

シナリオ"turn\_vehicle"は、直進と旋回を繰り返し行うシナリオである。各自で実行し、結果をXYプロットなどを用いて確認すること。

## コード生成

Embedded Coderコード生成結果を確認する。

```
return;
```

```
rtwbuild(ada_controller_name);
```

## SIL検証

SILモードでモデルとコードの等価性を調べる。

```
return;  
set_param([model_name, '/MPC_Controller'], 'SimulationMode', 'Normal');  
sim(model_name);  
set_param([model_name, '/MPC_Controller'], 'SimulationMode', 'Software-in-the-Loop (SIL)');  
sim(model_name);
```

結果を比較する。

```
compare_previous_run;
```

## PIL検証

「Linear\_MPC\_Design.mlx」と同様に、STM32 Nucleo F401REを用いたPIL検証を行う。手順については、「Linear\_MPC\_Design.mlx」を参照。





# Code Execution Profiling Report for Vehicle\_system\_Adaptive\_MPC/Adaptive\_MPC\_Controller

The code execution profiling report provides metrics based on data collected from a SIL or PIL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PIL test harness or inside the code generated for each component. See [Code Execution Profiling](#) for more information.

## 1. Summary

Total time	216548359738
Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	8.4e+07
Profiling data created	01-Apr-2020 09:29:31

## 2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
<a href="#">Adaptive MPC Controller initialize</a>	40321	40321	40321	40321	1	
<a href="#">Adaptive MPC Controller step [0.02 0]</a>	39959893	39365264	39959893	39365264	5501	 
<a href="#">Adaptive MPC Controller terminate</a>	1512	1512	1512	1512	1	

## 3. CPU Utilization

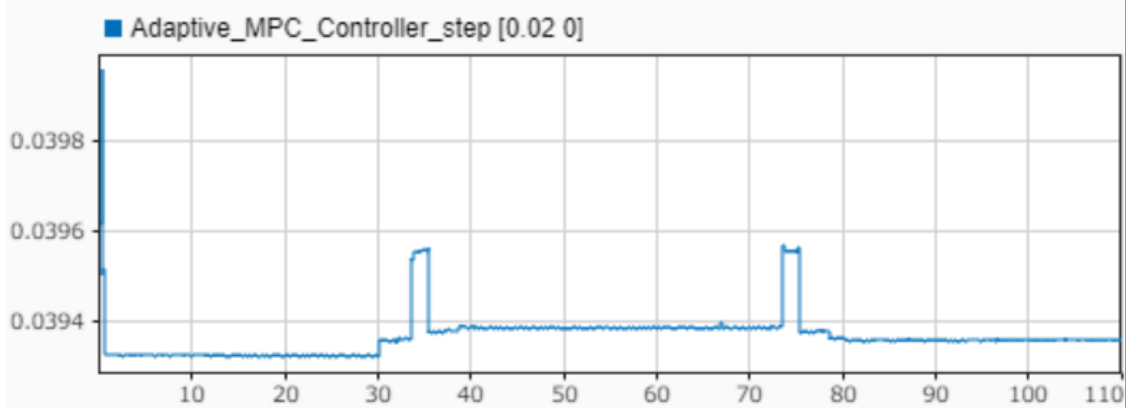
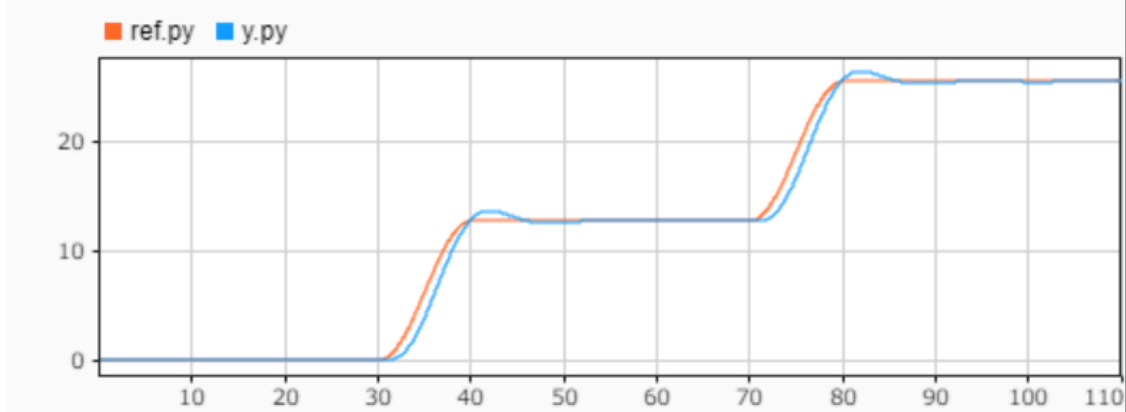
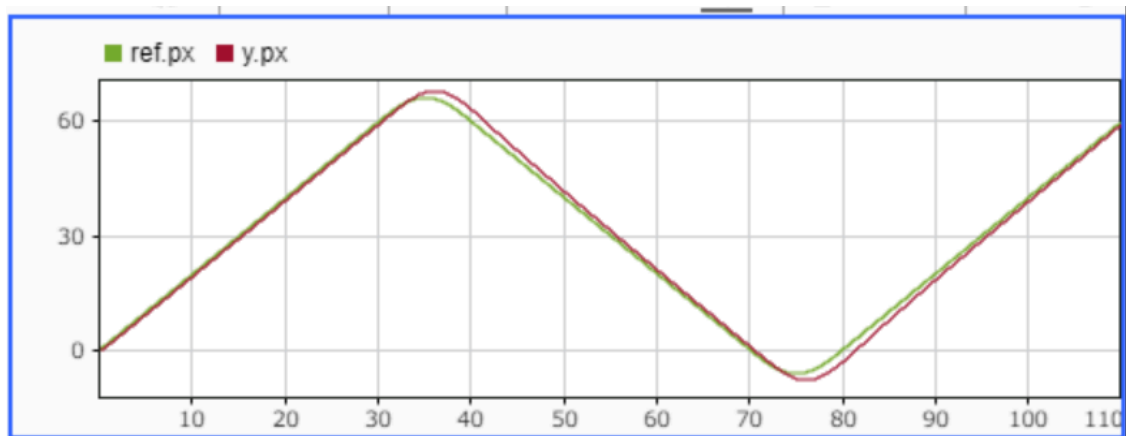
Task	Average CPU Utilization	Maximum CPU Utilization
<a href="#">Adaptive MPC Controller step [0.02 0]</a>	196.8%	199.8%
Overall CPU Utilization	196.8%	199.8%

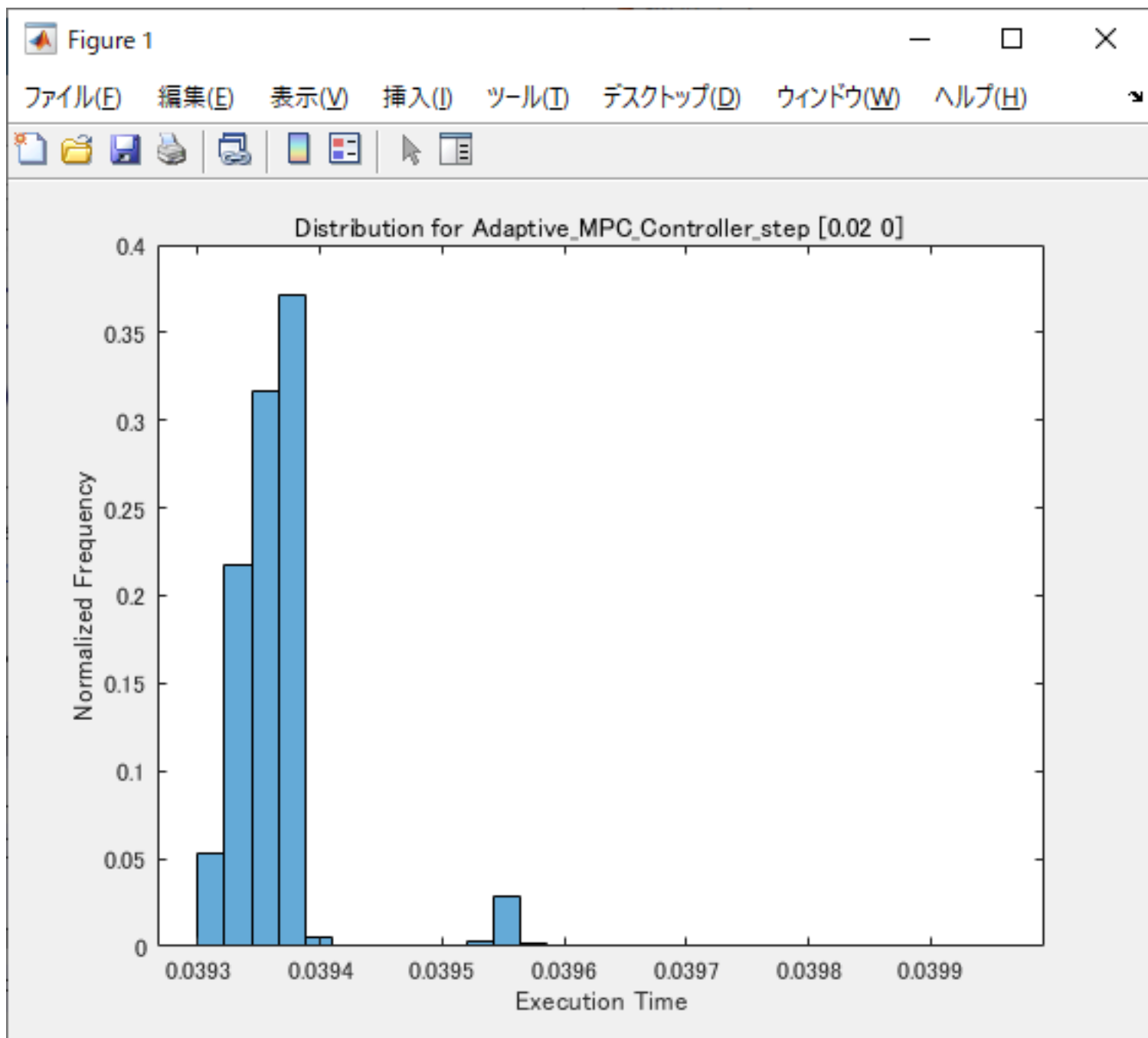
## 4. Definitions

**CPU Utilization:** Percentage of CPU time assigned to a task. Computed by dividing task execution time by sample time.

**Execution Time:** Time between start and end of code section.

**Self Time:** Execution time, excluding time in child sections.





1ステップ当たりの平均計算時間は39.96ms、CPU使用率は196.8%である。

モデルとコードの実行の比較結果は以下のようになった。

検査

比較

18 許容誤差内

0 許容範囲外

11 整列されていません

フィルターの比較

名前 (...)	...	最大...	結果
cost		0.00%	
qp_stat		0.00%	
est_state			
ref			✓ 5
u			✓ 2
x			✓ 6
y			✓ 5
px	0	0.00%	3.73e-06 ✓
py	0	0.00%	3.73e-06 ✓
theta	0	0.00%	1.44e-06 ✓
r	0	0.00%	9.82e-06 ✓
V	0	0.00%	1.61e-06 ✓

プロパティ		ベースライン	比較対象
名前	y.px (Run 1:...	y.px (Run 2:...	
説明			
ライン			
グローバル許容誤差	いいえ	いいえ	
絶対許容誤差	0	0	
相対許容誤差	0.00%	0.00%	
時間の許容誤差	0	0	
単位			
データ型	double	double	
サンプル時間	0.02	0.02	
実行	Run 2: Vehi...	Run 2: Vehi...	
整列	パス	パス	
モデル	Vehicle_sys...	Vehicle_sys...	
ブロック名	Bus Creator	Bus Creator	
ブロックパス	Vehicle_sys...	Vehicle_sys...	
端子	1	1	
次元	[1]	[1]	

