



ユーザー登録 (/signup?)  
 録 redirect\_to=%2FRyosukeH%2Fitems%2Fb947efa9dbcefa396baa)

ログイン (/login?)  
 redirect\_to=%2FRyosukeH%2Fitems%2Fb947efa9dbcefa396baa)



@RyosukeH (/RyosukeH) 2017年06月16日に投稿  
 (/RyosukeH)



# より良い制御を目指して～モデル予測制御(実装編)～

C++(/tags/C++) Control(/tags/Control) Udacity(/tags/Udacity)

👍 10



本記事は前回のより良い制御を目指して～モデル予測制御の導入～  
 (http://qiita.com/RyosukeHonda/items/bf95c58e352774ed8de7)の続編となっています。今回はC++でモデル予測制御の実装を行おうと思います。

また、本記事の内容はUdacityの自動運転エンジニアのTerm2  
 (http://qiita.com/RyosukeHonda/items/4853e9a403fc760c25ea)の最終課題で扱われているものです。

## 前回のおさらい

PID制御では、コーナー直後の自動車の挙動が振動的になり、上手く制御できていないことがわかりました。人間が自動車を操縦する際には、あらかじめ、コーナーを確認し、減速しコーナーに突入するかと思います。このような動作を「モデル予測制御を用いて実現しよう」というものでした。

モデル予測制御は逐次、最適制御問題を解くことにより将来の予測値を求め、その予測値を用いて制御する方法です。最適制御問題を解くためにコスト関数を設定し、コスト関数を最小化(最大化)する必要があります。前回の記事では次のようにコスト関数を設定していました。

$$\int_0^T (a \times cte_t^2 + b \times e\psi_t^2 + c \times (v_t - v_{ref})^2 + d \times (\delta_t - \delta_{t-1})^2 + e \times \delta_t^2 + f \times (a_t - a_{t-1})^2) dt$$

ここで $a \sim g$ は任意の定数となっており、パラメータ調整を行う必要があります。また自動車制御の入力はハンドル量( $\delta$ )とアクセル量( $a$ )となっています。この入力制限としては、ハンドル角は $-25^\circ$ から $+25^\circ$ まで、アクセル量は $-1$ から $+1$ までとしています( $-1$ はフルブレーキ、 $+1$ はフルアクセルを表しています。)

また、以下の式も成り立っています。

$$\begin{aligned} x_{t+1} &= x_t + v_t \cos \psi_t dt \\ y_{t+1} &= y_t + v_t \sin \psi_t dt \\ v_{t+1} &= v_t + a_t dt \\ \psi_{t+1} &= \psi_t + \frac{v_t}{L_f} \delta_t dt \\ cte_{t+1} &= cte_t + v_t \sin e\psi_t dt \\ e\psi_{t+1} &= e\psi_t + \frac{v_t}{L_f} \delta_t dt \end{aligned}$$

## 実装

以上の制限のもとで、コスト関数が最小になる入力求めていくのですが、IPOPT(Interior Point OPTimizer)というライブラリーを用いればコスト関数を最小化するようなハンドル角とアクセル量を求めることができます。例題とそのソースコードがここ ([https://www.coin-or.org/CppAD/Doc/ipopt\\_solve\\_get\\_started.cpp.htm](https://www.coin-or.org/CppAD/Doc/ipopt_solve_get_started.cpp.htm))にありますので、それにのらい、実装しようと思います。

例題の問題設定は以下のようになっています。

最小化するべき関数

$$\text{minimize} : x_1 x_4 (x_1 + x_2 + x_3) + x_3$$

制約条件

$$\begin{aligned} \text{subject to} : \quad & x_1 x_2 x_3 x_4 \geq 25 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40 \\ & 1 \leq x_1, x_2, x_3, x_4 \leq 5 \end{aligned}$$

これを本問題設定に置き換えると次のようになります。

最小化するべき関数

$$\text{minimize} : (2000 \times cte_t^2 + 1800 \times e\psi_t^2 + 1 \times (v_t - v_{ref})^2 + 3 \times (\delta_t - \delta_{t-1})^2 + 5 \times \delta_t^2 -$$

$a \sim g$ は試行錯誤により決定しました。CTEと目標軌道との角度差に大きな重みをおいています。

制約条件

$$\begin{aligned} \text{subject to} : \quad & x_{t+1} = x_t + v_t \cos \psi_t dt \\ & y_{t+1} = y_t + v_t \sin \psi_t dt \\ & v_{t+1} = v_t + a_t dt \\ & \psi_{t+1} = \psi_t + \frac{v_t}{L_f} \delta_t dt \\ & cte_{t+1} = cte_t + v_t \sin e\psi_t dt \\ & e\psi_{t+1} = e\psi_t + \frac{v_t}{L_f} \delta_t dt \end{aligned}$$

今回は0.1秒ごとに10ステップ先、つまり0.1秒先から1.0秒先まで0.1秒間隔で予測するものとした。以下のソースコード中のCppADは自動的に微分を求めるライブラリーとなっています。

```

#include "MPC.h"
#include <cppad/cppad.hpp>
#include <cppad/ipopt/solve.hpp>
#include "Eigen-3.3/Eigen/Core"

using CppAD::AD;

// Set the timestep length and duration
//Predict until 10 time step ahead
size_t N = 10;
// Set dt to 0.1 since the latency is 0.1s(we have to predict at least 0.1s ahead)
double dt = 0.1;

const double Lf = 2.67;
//基準のCTE, 角度差を0で初期化
double ref_cte = 0;
double ref_epsilon = 0;
//目標スピードを100.0MPHに設定
double ref_v = 100.0;

/*
0~N-1まではxの値
N~2N-1まではyの値
2N~3N-1まではpsiの値
3N~4N-1まではvの値
4N~5N-1まではcte値
5N~6N-1まではepsilonの値
6N~7N-1まではdeltaの値
7N~8N-1まではaの値を保存
*/
size_t x_start = 0;
size_t y_start = x_start + N;
size_t psi_start = y_start + N;
size_t v_start = psi_start + N;
size_t cte_start = v_start + N;
size_t epsilon_start = cte_start + N;
size_t delta_start = epsilon_start + N;
size_t a_start = delta_start + N - 1;

class FG_eval {
public:
    // Fitted polynomial coefficients (目標軌道のフィッティング)
    Eigen::VectorXd coeffs;
    FG_eval(Eigen::VectorXd coeffs) { this->coeffs = coeffs; }

    typedef CPPAD_TESTVECTOR(AD<double>) ADvector;
    void operator()(ADvector& fg, const ADvector& vars) {
        // Implement MPC
        // fg a vector of constraints, x is a vector of constraints.
        // NOTE: You'll probably go back and forth between this function and
        // the Solver function below.

        //コスト関数を設定
        fg[0] = 0;

        for (int i = 0; i < N - 1; i++) {
            fg[0] += 2000 * CppAD::pow(vars[cte_start + i] - ref_cte, 2);
            fg[0] += 1800 * CppAD::pow(vars[epsilon_start + i] - ref_epsilon, 2);
            fg[0] += CppAD::pow(vars[v_start + i] - ref_v, 2);
        }

        for (int i = 0; i < N - 1; i++) {
            fg[0] += 3 * CppAD::pow(vars[delta_start + i], 2);
            fg[0] += 5 * CppAD::pow(vars[a_start + i], 2);
        }
    }
}

```

```

// Minimize the value gap between sequential actuations.
for (int i = 0; i < N - 2; i++) {
    fg[0] += 100 * CppAD::pow(vars[delta_start + i + 1] - vars[delta_start + i], 2);
    fg[0] += 10 * CppAD::pow(vars[a_start + i + 1] - vars[a_start + i], 2);
}

//Setup constraints
fg[1 + x_start] = vars[x_start];
fg[1 + y_start] = vars[y_start];
fg[1 + psi_start] = vars[psi_start];
fg[1 + v_start] = vars[v_start];
fg[1 + cte_start] = vars[cte_start];
fg[1 + epsi_start] = vars[epsi_start];

for (int i = 0; i < N-1; i++){
    // The state at time t+1.
    AD<double> x1 = vars[x_start + i + 1];
    AD<double> y1 = vars[y_start + i + 1];
    AD<double> psi1 = vars[psi_start + i + 1];
    AD<double> v1 = vars[v_start + i + 1];
    AD<double> cte1 = vars[cte_start + i + 1];
    AD<double> epsi1 = vars[epsi_start + i + 1];

    // The state at time t.
    AD<double> x0 = vars[x_start + i];
    AD<double> y0 = vars[y_start + i];
    AD<double> psi0 = vars[psi_start + i];
    AD<double> v0 = vars[v_start + i];
    AD<double> cte0 = vars[cte_start + i];
    AD<double> epsi0 = vars[epsi_start + i];

    // Only consider the actuation at time t.
    AD<double> delta0 = vars[delta_start + i];
    AD<double> a0 = vars[a_start + i];

    AD<double> f0 = coeffs[0] + coeffs[1] * x0 + coeffs[2] * x0 * x0 + coeffs[3] * x0 *
    AD<double> psides0 = CppAD::atan(coeffs[1] + 2.0 * coeffs[2] * x0 + 3.0 * coeffs[3]

    //制約条件
    fg[2 + x_start + i] = x1 - (x0 + v0 * CppAD::cos(psi0) * dt);
    fg[2 + y_start + i] = y1 - (y0 + v0 * CppAD::sin(psi0) * dt);

    fg[2 + psi_start + i] = psi1 - (psi0 - v0 * delta0 / Lf * dt);
    fg[2 + v_start + i] = v1 - (v0 + a0 * dt);
    fg[2 + cte_start + i] = cte1 - ((f0 - y0) + (v0 * CppAD::sin(epsi0) * dt));
    fg[2 + epsi_start + i] = epsi1 - ((psi0 - psides0) - v0 * delta0 / Lf * dt);

}

}
};

//
// MPC class definition implementation.
//
MPC::MPC() {}
MPC::~MPC() {}

//モデル予測制御を解く
vector<double> MPC::Solve(Eigen::VectorXd state, Eigen::VectorXd coeffs) {
    bool ok = true;
    size_t i;
    typedef CPPAD_TESTVECTOR(double) Dvector;

    double x = state[0];

```

```
double y = state[1];
double psi = state[2];
double v = state[3];
double cte = state[4];
double epsi = state[5];

size_t n_vars = 6 * N + 2 * (N-1);
// Set the number of constraints
size_t n_constraints = 6 * N;

// Initial value of the independent variables.
// SHOULD BE 0 besides initial state.
Dvector vars(n_vars);
for (int i = 0; i < n_vars; i++) {
    vars[i] = 0.0;
}

vars[x_start] = x;
vars[y_start] = y;
vars[psi_start] = psi;
vars[v_start] = v;
vars[cte_start] = cte;
vars[epsi_start] = epsi;

Dvector vars_lowerbound(n_vars);
Dvector vars_upperbound(n_vars);
// Set lower and upper limits for variables.
for(int i = 0; i < delta_start; i++){
    vars_lowerbound[i] = -1.0e19;
    vars_upperbound[i] = 1.0e19;
}
//ハンドル角の制限
for(int i = delta_start; i < a_start; i++){
    vars_lowerbound[i] = -0.436332;
    vars_upperbound[i] = 0.436332;
}
//アクセル量の制限
for(int i = a_start; i < n_vars; i++){
    vars_lowerbound[i] = -1.0;
    vars_upperbound[i] = 1.0;
}

// Lower and upper limits for the constraints
// Should be 0 besides initial state.
Dvector constraints_lowerbound(n_constraints);
Dvector constraints_upperbound(n_constraints);
for (int i = 0; i < n_constraints; i++) {
    constraints_lowerbound[i] = 0;
    constraints_upperbound[i] = 0;
}

constraints_lowerbound[x_start] = x;
constraints_lowerbound[y_start] = y;
constraints_lowerbound[psi_start] = psi;
constraints_lowerbound[v_start] = v;
constraints_lowerbound[cte_start] = cte;
constraints_lowerbound[epsi_start] = epsi;

constraints_upperbound[x_start] = x;
constraints_upperbound[y_start] = y;
constraints_upperbound[psi_start] = psi;
constraints_upperbound[v_start] = v;
constraints_upperbound[cte_start] = cte;
constraints_upperbound[epsi_start] = epsi;
```

```
// object that computes objective and constraints
FG_eval fg_eval(coeffs);

std::string options;
options += "Integer print_level  0\n";

options += "Sparse  true          forward\n";
options += "Sparse  true          reverse\n";

options += "Numeric max_cpu_time      0.5\n";

// place to return solution
CppAD::ipopt::solve_result<Dvector> solution;

// solve the problem
CppAD::ipopt::solve<Dvector, FG_eval>(
    options, vars, vars_lowerbound, vars_upperbound, constraints_lowerbound,
    constraints_upperbound, fg_eval, solution);

// Check some of the solution values
ok &= solution.status == CppAD::ipopt::solve_result<Dvector>::success;

// Cost
auto cost = solution.obj_value;
std::cout << "Cost " << cost << std::endl;

vector<double> result;
//ハンドル量とアクセル量の最適解
result.push_back(solution.x[delta_start]);
result.push_back(solution.x[a_start]);
//10ステップ先までの予測軌道
for(int i = 0; i < N-1; i++){
    result.push_back(solution.x[x_start+i+1]);
    result.push_back(solution.x[y_start+i+1]);
}
return result;
}
```

以上がモデル予測制御の実装となります。

シミュレータを動かすためのコード等は[こちら](#)

([https://github.com/RyosukeHonda/Model\\_Predictive\\_Control](https://github.com/RyosukeHonda/Model_Predictive_Control))にあります。もしよければ参照してみてください。

## 結果

結果は以下のgifのようになりました。

緑線はモデル予測制御により予測された予測軌道, 黄線は基準軌道となっています。

PID制御を用いた場合よりも, 滑らかに制御することができており, 人間の自動車操縦に近づいたように思います。

また, 本シミュレーションでは, 出力予測(緑線)を行ってから, 実際に自動車の制御入力を行うまでに, 0.1秒の時間差(レイテンシー)を設けています。これは, 予測された値を時間差なく出力することは実用上不可能であるためです。

本シミュレーションではモデル予測制御を用いて0.1秒先から1.0秒先まで0.1秒ごとに予測をしているため, レイテンシーがあったとしても, 制御を行うことができています。

全体の動画はこちら (<https://www.youtube.com/watch?v=YElSbNqtxEw>)から見るができます。



(<https://www.youtube.com/watch?v=YElSbNqtxEw>)

## 感想

シミュレータ上で自動車の制御を行いました。PID制御で制御しようとすると、コーナー後に車両がぐらつくのを防ぐことが困難でした。また、入力の時間差にも対応することができませんでした。一方モデル予測制御を用いると、人間が操作する挙動に近い動きにはなりました。また、入力の時間差にも対応することができました。しかしながら、今回全く議論しなかったパラメータの量が大幅に増えたため、パラメータチューニングに時間がかかりました。(PIDでは3つ、今回のモデル予測制御では7つ)。本記事ではKinematicモデルを用いましたが、より詳細なモデルであるDynamicモデルを用いると、もっと多くのパラメータが必要となるため、さらに大変になります。「簡単に制御したいならPID制御、細かな制御もしたいのならモデル予測制御」のように、制御対象に求められる要件に応じて制御方法を変えるべきであると感じました。

## 参考文献

最適制御問題については以下の本を参考にしました、  
現代制御論 ([https://www.amazon.co.jp/gp/product/4339032123/ref=as\\_li\\_tl?ie=UTF8&camp=247&creative=1211&creativeASIN=4339032123&linkCode=as2&tag=ryo519-22&linkId=f30d36b042714fc9c0f5132ffcc26bab](https://www.amazon.co.jp/gp/product/4339032123/ref=as_li_tl?ie=UTF8&camp=247&creative=1211&creativeASIN=4339032123&linkCode=as2&tag=ryo519-22&linkId=f30d36b042714fc9c0f5132ffcc26bab))  
  
(<https://camo.qiitusercontent.com/c648d945170cea252fb8e9da7ee620fcab05e5ad/68747470733a2f2f69722d6a702e616d617a6f6e2d616473797374656d2e636f6d2f652f69723f743d72796f3531392d3232266c3d616d32266f3d3926613d34333339303332313233>)

📦 ストック

👍 いいね

10

(<https://qiita.com/RyosukeH/items/b947efa9dbcefa396baa/likers>)

Tweet

0

Tools

(<https://qiitadon.com/share?text=%F3%R2%RR%F3%R2%RA%FR%RQ%AF%F3%R1%R4%F5%RR%R6%F5%RF%A1%F3%R2%Q2%F7%QR%AF%F6%RC%R7%F3%R1%Q7%F3%R1%A6%F3%R0%QC%F3%R3%A2%F3>)

Like 0

7

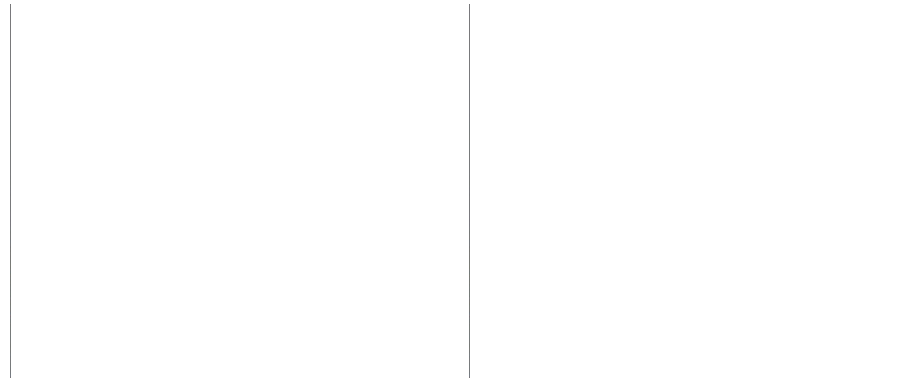


@RyosukeH (/RyosukeH)





自動運転, AIに興味があります.

(/RyosukeH)

+フォロー



関連記事 Recommended by (http://www.logly.co.jp/privacy.html)

-  より良い制御を目指して～モデル予測制御の導入～  
(https://qiita.com/RyosukeH/items/bf95c58e352774ed8de7) by RyosukeH
-  0からわかるPID制御(実装編) (https://qiita.com/RyosukeH/items/373f6451c4946b1e447e) by RyosukeH
-  0からわかるPID制御 (https://qiita.com/RyosukeH/items/9e5ce2ebdadd90e3db00) by RyosukeH
-  WSDM 2016 勉強会: Feedback Control of Real-...  
(https://qiita.com/komiya\_atsushi/items/f34fcfaa8215321b1fb9) by komiya\_atsushi

あなたもコメントしてみませんか :)

ユーザー登録(無料) (/signup?redirect\_to=%2FRyosukeH%2Fitems%2Fb947efa9dbcefa396baa%23comments)

すでにアカウントを持っている方はログイン (/login?redirect\_to=%2FRyosukeH%2Fitems%2Fb947efa9dbcefa396baa%23comments)