# Ccontrol

## Introduction

This is an introduction how to use CControl. This is a library made in C only with static memory. That means that CControl does not allocates memory from malloc.

The library has the main focus on advanced control techniques.

- Autotuning Predictive Control(APC) – Self tuning controller with prediction
- Generalized Predictive Control(GPC) – Adaptive tuning controller with prediction
- Model Predictive Control(MPC) – Predictive controller with constraints
- Linear Quadratic Integral Control(LQI) – Infinite prediction

These control techniques are resting on minimizing a quadratic cost function.

$$J = x^T Q\, x + u^T Ru$$

But that is only theory. This library brings the theory to the reality. Every controller have it’s own specialty and feature. Here is an table that tells you what controller you want to have.

| Control problem\Controller | APC | GPC | MPC | LQI |
|---|---|---|---|---|
| Stochastic system that changes over time | | X | | |
| Systems with limits | | | X | |
| Simple slow systems that not changes | X | | | |
| Multivariable system | | | | X |

## Requirements

To use this library, you need to be familiar with C-language, an microcontroller and understanding in control theory. Not so much control theory. But knowing some state space modeling is very useful here.

# Control techniques

I'm going to go thru all the math that are inside CControl.

## Recursive Least Squares

Estimating a transfer function on this form. Actually CControl does not creating a transfer function for you. But you can use this formula above to create a state space model.

$$A(q)y(t)=B(q)u(t)+C(q)\varepsilon(t)$$

Where $A(q)$ is the denominator polynomial and $B(q)$ is the numerator polynomial and $C(q)$ is the noise numerator polynomial. They are going to be placed like this:

$$y(t)=\frac{B(q)}{A(q)}u(t)+\frac{C(q)}{A(q)}\varepsilon(t)$$

We estimate this by using C-function

```
void rls(float* theta, float u, float y, int* count)
```

Where `float* theta` is a 1D-vector of

$$\theta=(a_1\ldots a_n,b_1\ldots b_m,c_1\ldots c_e)$$

and u and y is input receptive output. Count is a variable that initialize with 0 and its counting to 2, then stop. The size of all polynomial are the same. You conifiure them with this command in Configuration.h header file in CControl.

```
#define POLY_LENGTH 6 // Length of polynomials A(q), B(q) and C(q) in Recursive least squares
```

Notice that the total length is `3*POLY_LENGTH`
The rest of the Recursive Least Square looks like this below. You don't need to know this math, only need to know how you can use rls function.

$$\phi^T(t-1)=(-y(t-1)\ldots-y(t-n)\ldots u(t-1)\ldots u(t-m)\ldots\varepsilon(t-1)\ldots\varepsilon(t-e))$$
$$\varepsilon(t)=y(t)-\phi^T(t-1)\hat{\theta}(t-1)$$
$$\hat{\theta}(t)=\hat{\theta}(t-1)+P(t)\phi(t-1)\varepsilon(t)$$
$$P(t)=\frac{1}{\lambda}\left(P(t-1)-\frac{P(t-1)\phi(t-1)\phi^T(t-1)P(t-1)}{\lambda+\phi^T(t-1)P(t-1)\phi(t-1)}\right)$$

You can tune in lambda in the Configuration.h header.

```
#define q 1000 // Initial diagonal values for P matrix for system identification. 1000 is a good number
#define LAMBDA 1 // Forgetting factor for system identification. Lambda Should not be less than zero
```

**Hint**: If you want to know more. Please by the book "Adaptive Control" by Karl Johan Åström and Björn Wittenmark. It's a old book, but you cannot find any better book about adaptive control. It's a very practical book that have the industry as the audience.

## Kalman filter

CControl have a kalman filter too. It's very easy to use. Just call this C-function.

```
void kalman(float* A, float* B, float* C, float* K, float* u, float* x, float* y)
```

Where A, B, C are system matrices and K are kalman gain matrix, u is past input vector, x are the past state vector we are standing on and y is the past output vector. The kalman filter will compute this formula for you. In other words, this kalman filter will update the state vector.

$$x = Ax - KCx + Bu + Ky$$

## Linear Quadratic Integral Control

To create a LQI controller. Begin to create a state space model like this. This is called augmented state space model and this have integral action included.

$$\begin{bmatrix} x(k+1) \\ x_i(k+1) \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 \\ -C & 1 \end{bmatrix}}_{\mathring{A}} \begin{bmatrix} x(k) \\ x_i(k) \end{bmatrix} + \underbrace{\begin{bmatrix} B \\ -D \end{bmatrix}}_{\mathring{B}} u(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r(k)$$

$$y(k) = \underbrace{\begin{bmatrix} C & 0 \end{bmatrix}}_{\mathring{C}} \begin{bmatrix} x(k) \\ x_i(k) \end{bmatrix} + Du(k)$$

Then you run the .m file Linear_Quadratic_Integral_Control. With the argument of the model.

Hint: Please use Matavecontrol in following steps:

1. Use sys = ss(delay, A, B, C, D) function to create a state space modeling
2. use sysd = c2d(sys, sampleTime) to turn the model to a discrete modeling
3. Now you can create a state space model like above.

When you got the model and you have run the .m file Linear_Quadratic_Integral_Control.m. Open then the Linear_Quadratic_Integral_Control example in Documents folder and paste all the matrices in and set your reference vector etc.

LQI using the following formulas:

1. $u(k) = \frac{L_i}{(1-q_i)} r - (Lx(k) - L_i x_i(k))$ - Control
2. $x_i(k+1) = x_i(k) + r(k) - y_m(k)$ - Integrate
3. $x(k+1) = (A - KC)x(k) + Bu(k) + Ky_m(k)$ - Estimate

You call LQI from this function

```
void lqi(float* y, float* u, float qi, float* r, float* L, float* Li, float* x, float* xi, int anti_windup)
```

**Hint**: Look at the Linear_Quadratic_Integral_Controller example in Documents folder

# Generalized Predictive Control

Generalized Predictive Control is like an adaptive Model Predictive Control without constraints.

This is how you enable prediction by using a state space model.

$$\begin{bmatrix} y(2) \\ y(3) \\ y(4) \\ \vdots \\ y(n) \end{bmatrix} = \underbrace{\begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{n-1} \end{bmatrix}}_{\Phi} x(k) + \underbrace{\begin{bmatrix} CB & 0 & 0 & 0 & 0 & 0 \\ CAB & CB & 0 & 0 & 0 & 0 \\ CA^2B & CAB & CB & 0 & 0 & 0 \\ CA^3B & CA^2B & CAB & CB & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{n-2}B & CA^{n-3}B & CA^{n-4}B & CA^{n-5}B & CA^{n-6}B & CA^{n-j}B \end{bmatrix}}_{\Gamma} U(k)$$

All GPC does is to compute the best input values like this:

$$U = (\Gamma^T \Gamma + \alpha I)^{-1} \Gamma^T (R - \Phi x_0)$$

All you need to do is to call this function

```
void gpc(float* A, float* B, float* C, float* x, float* u, float* r)
```

And it will give you the first element of U → `float* u`

Also you tune in the horizon and prediction like this:

```
#define HORIZON 50 // How long we want to look in the future
#define ALPHA 0.1 // This will prevent dead-beat control and gives more smooth input values
#define INTEGRATION TRUE // Enable integral action inside model - Recommended
```

I recommend integral action TRUE because it gives you a better reference following of y to r.

# Autotuning Predictive Control

This is exactly the same as GPC, but the difference is that APC computes a control law from

$$U = (\Gamma^T \Gamma + \alpha I)^{-1} \Gamma^T (R - \Phi x_0)$$

That will looks like this:

$$K_r = (\Gamma^T \Gamma + \alpha I)^{-1} \Gamma^T R$$
$$L = (\Gamma^T \Gamma + \alpha I)^{-1} \Gamma^T \Phi x_0$$

Where Kr is the reference gain for better tracking and the L is the control law. After you have found your control law and reference gain, then you gain use the lqi function and kalman funtion to control and estimate the states.

**Hint**: If you have integration enabled, then the last element of L is your integral law. It's only a number. I recommend to always have integration. It's just better.

## Model Predictive Control

This is the most expensive controller to use. Here you need to have a microcontroller that can handle the datatype double.

Model predictive control trying to minimize the quadratic objective cost function:

$$J = \frac{1}{2} U^T \left( \Gamma^T \alpha I \Gamma \right) U + \left( \Gamma^T \alpha I \Phi x_0 - \Gamma^T Q r \right)^T U$$

With subject to:

$$lb_U \leq U \leq ub_U$$
$$lb_Y \leq \Gamma x \leq ub_Y$$

That means MPC controls the input and output. Example: If you say that if a hydraulic actuator have the piston range 0-2 and input is 0.1-4 liter/minute. Then U cannot be lower than 0.1 and not greater than 4 and the Y cannot be lower than 0 and greater than 2.

CControl have not a quadratic function solver included. So I have been using qpOASES_e that computes the best inputs. It works great and I have also compared it with GNU Octave's QP-solver function qp. There is a working example about MPC too. Just look at the Documents folder.

**Hint**: Look at the thesis I have posted too in this folder. It explains the math.