Objectives

- Control function for float_t
- IIR Filter for int16_t

IIR Filter

Background

- A general form of the IIR filter

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdot + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdot + a_N z^{-N}}$$

- The first order IIR filter in the Z-domain

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

- the discrete time domain representation of the filter

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

[Example] 1st order LPF

$$\frac{Y(s)}{X(s)} = \frac{\omega_c}{s + \omega_c}$$

$$\dot{y}(t) + \omega_c y(t) = \omega_c x(t)$$

1. Backward Euler

$$\dot{y}(t) = \frac{y[k] - y[k-1]}{T_s}$$

$$\begin{aligned} &\frac{y[k] - y[k-1]}{T_s} + \omega_c y[k] = \omega_c x[k] \\ &(1 + T_s \omega_c) y[k] = T_s \omega_c x[k] + y[k-1] \\ &y[k] = \frac{T_s \omega_c}{1 + T_s \omega_c} x[k] + \frac{1}{1 + T_s \omega_c} y[k-1] \end{aligned}$$

2. Forward Euler

$$\dot{y}(t) = \frac{y[k+1] - y[k]}{T_s}$$

$$\begin{aligned} &\frac{y[k+1] - y[k]}{T_s} + \omega_c y[k] = \omega_c x[k] \\ &y[k+1] = T_s \omega_c x[k] + (1 - T_s \omega_c) y[k] \\ &y[k] = T_s \omega_c x[k-1] + (1 - T_s \omega_c) y[k-1] \end{aligned}$$

- if $\tau = 10[msec]$, $\omega_c = 100[rad/s]$, $f_c = \omega_c/(2\pi) = 16[Hz]$ and $T_s = 1[msec]$

$$b_0 = \frac{1m \times 100}{1 + 1m \times 100} = 0.091$$
$$b_1 = 0$$
$$a_1 = -\frac{1}{1 + 1m \times 100} = -0.909$$

IIR filter for int16 (optional)

`int16_t I16IirFilter1(int16_t i16In, i16Iir_t * pParam)`

- This function implements Direct Form I first order IIR filter.

```
typedef struct{
    int16_t i16B0;
    int16_t i16B1;
    int16_t i16A1;
    int16_t i16InBuffer[1];
    int32_t i32AccBuffer[1];
} i16Iir_t;
```

IIR filter for float_t

`float_t fltIirFilter1(float_t flt_in, fltIir_t * pParam)`

- This function implements Direct Form I first order IIR filter.

```
typedef struct{
    float_t flt_b0;     /*!< b0 coefficient of IIR1 filter float */
    float_t flt_b1;     /*!< b1 coefficient of IIR1 filter float */
    float_t flt_a1;     /*!< a1 coefficient of IIR1 filter float*/
    float_t flt_in_buffer[1]; /*!< input buffer of IIR1 filter */
    float_t flt_acc_buffer[1];/*!< internal accumulator buffer */
} fltIir_t;
```

Moving Average Filter

Basics

- Definition: summation form

$$y_n = \frac{1}{N}\sum_{i=0}^{N-1} x_{n-i}$$

  - To implement in programming language, this form requires too many storages and computations
- recursive form

$$y_n = y_{n-1} + \frac{1}{N}(x_n - x_{n-N+1})$$

  - This form requires less computation than the summation form, but it also requires many storages for x_n
- Simple form: Approximation form

$$N \times y_n = N \times y_{n-1} + x_n - x_{n-N+1}$$

$$\begin{aligned} Acc[n] &= Acc[n-1] + x[n] - x[n-N+1] \\ &\approx Acc[n-1] + x[n] - Acc[n-1]/N \\ y[n] &= Acc[n]/N \end{aligned}$$

- In each sample time

$$\begin{aligned} Acc &:= Acc + x \\ y &:= Acc/N \\ Acc &:= Acc - y \end{aligned}$$

Moving Average filter for float_t

`float_t fltMaFilter(float_t flt_in, fltMa_t * pParam);`

- This function implements a moving average recursive filter

```
typedef struct{
    float_t flt_acc;
    int16_t i16_length;
}fltMa_t;
```

PI Controller

PI Controller - Parallel form for float_t

`float_t fltPiCtrlP(float_t error, fltPiCtrlP_t * pParam);`

- This function implements PI Controller in parallel form

```
typedef struct{
    float_t     fltPGain;       /*!< K_p */
    float_t     fltIGain;       /*!< K_i * T_s / 2*/
    float_t     fltIGain_Pre;   /*!< K_i * T_s / 2*/
    float_t     fltUI_Pre;      /*!< u_i[k-1] */
    float_t     fltE_Pre;       /*!< e[k-1] */
    float_t     fltUpperLimit;
    float_t     fltLowerLimit;
    uint16_t    u16LimitFlag;   /*! Set if u[k] is out of range */
} fltPiCtrlP_t;
```

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

$$U(s) = K_p E(s) + K_i \frac{E(s)}{s}$$

$$U(s) = U_p(s) + U_i(s)$$

$$where, U_i(s) = K_i \frac{E(s)}{s}$$

- In Backward Euler $s = \frac{1-z^{-1}}{T_s}$

$$\frac{u_i[k]-u_i[k-1]}{T_s}=K_ie[k]$$
$$u_i[k]=u_i[k-1]+K_iT_se[k]+0e[k-1]$$

- In Forward Euler $s=\frac{z^1-1}{T_s}$

$$\frac{u_i[k+1]-u_i[k]}{T_s}=K_ie[k]$$
$$u_i[k]=u_i[k-1]+0e[k]+K_iT_se[k-1]$$

- In Bilinear $s=\frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}}$

$$u_i[k]-u_i[k-1]=K_i\frac{T_s}{2}\times(e[k]+e[k-1])$$
$$u_i[k]=u_i[k-1]+\frac{K_iT_s}{2}e[k]+\frac{K_iT_s}{2}e[k-1]$$

- Summary

	Backward	Forward	Bilinear
PGain	K_p	K_p	K_p
IGain	K_iT_s	0	$K_iT_s/2$
IGain_Pre	0	K_iT_s	$K_iT_s/2$

PI Controller - Recurrent form for float_t

`float_t fltPiCtrlR(float_t error, fltPiCtrlR_t * pParam);`

- This function implements PI Controller in recurrent form

```
typedef struct{
    float_t     fltEGain;       /*!< Gain for e[k] */
    float_t     fltEGain_Pre;   /*!< Gain for e[k-1] */
    float_t     fltU_Pre;       /*!< u[k-1] */
    float_t     fltE_Pre;       /*!< e[k-1] */
    float_t     fltUpperLimit;
    float_t     fltLowerLimit;
    uint16_t    u16LimitFlag;   /*! Set if u[k] is out of range */
} fltPiCtrlR_t;
```

$$u(t)=K_pe(t)+K_i\int_0^t e(\tau)d\tau$$
$$U(s)=K_pE(s)+K_i\frac{E(s)}{s}$$
$$sU(s)=K_psE(s)+K_iE(s)$$

- In Backward Euler $s=\frac{1-z^{-1}}{T_s}$

$$u[k]-u[k-1]=K_pe[k]-K_pe[k-1]+K_iT_se[k]$$
$$u[k]=u[k-1]+(K_p+K_iT_s)e[k]+(-K_p)e[k-1]$$

- In Forward Euler $s=\frac{z^1-1}{T_s}$

$$u[k] - u[k-1] = K_p e[k] - K_p e[k-1] + K_i T_s e[k-1]$$
$$u[k] = u[k-1] + K_p e[k] + (-K_p + K_i T_s) e[k-1]$$

- In Bilinear $s = \frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}}$

$$(1-z^{-1})U[z] = K_p(1-z^{-1})E[z] + K_i \frac{T_s}{2}(1+z^{-1})E[z]$$
$$u[k] - u[k-1] = K_p e[k] - K_p e[k-1] + \frac{K_i T_s}{2}(e[k] + e[k-1])$$
$$u[k] = u[k-1] + (K_p + \frac{K_i T_s}{2}) e[k] + (-K_p + \frac{K_i T_s}{2}) e[k-1]$$

- Summary

	Backward	Forward	Bilinear
EGain	$K_p + K_i T_s$	K_p	$K_p + K_i T_s/2$
EGain_Pre	$-K_p$	$-K_p + K_i T_s$	$-K_p + K_i T_s/2$

Look-up table 1D

Basic

Example

x	-25	-15	-5	5	15	25	35	45
i (index)	0	1	2	3	4	5	6	7
y[i]	-1.0	0.0	1.0	2.0	3.0	4.0	5.0	6.0

- x should have same interval
- index should be in the range **[0, NumOfElements)**
  - if index is greater than (NumOfElements-1), then the value should be y[NumOfElements-1]
  - and vice versa
- x and i have the following relationships

$$x = i2x\_slope \times i + i2x\_offset$$
$$i = (x - i2x\_offset)/i2x\_slope$$

- Parameters
  - NumOfElements = 8
  - i2x_slope = 10
  - i2x_offset = -25
  - pTbl = y
- Calculation procedure
  1. find index, i

$$i = (int)(x - i2x\_offset)/i2x\_slope$$
$$x\_remainder = x - (i2x\_slope \times i + i2x\_offset)$$

2. calc y, according to i

$$\begin{aligned} y &= y[i] + \frac{y[i+1] - y[i]}{x[i+1] - x[i]} \times (x - x[i]) \\ &= y[i] + \frac{y[i+1] - y[i]}{i2x\_slope} \times x\_remainder \end{aligned}$$

1D Look-up function for float_t

`float_t fltLut1D(int16_t flt_in, fltLut1D_t * pParam);`

- This function implements Look-Up Table 1D

```
typedef struct{
    float_t     flt_i2x_slope;      /*!< slope */
    float_t     flt_i2x_offset;     /*!< offset */
    float_t*    flt_tbl_ptr;        /*!< pointer to table */
    int16_t     i16_num_of_elements;/*!< Number of elements */
    int16_t     i16_idx;            /*!< index */
    float_t     i16_remainder;      /*!< remainder */
} fltLut1D_t;
```