Digital Signal Processing project

This project is designed to allow users to implement any continuous transfer function (Laplace format) on the MCU. It converts the continuous function to the discrete one using one of the approximation algorithms (ZoH, Tustin, (etc.)), generates the recurring equation, and executes it. We can use this project either to generate signals, simulate them, and simulate digital filters.

Discretize continuous transfer function:

Discretizing continuous functions is expressing the transfer function in the Z domain. Which means expressing the output at a discrete instant (which is multiple of the sampling period), with the input at the same and some previous instants, and some previous outputs.

This discrete function is at the origin of generating the recurring equation that will be implement on the MCU.

Discretizing the continuous transfer function uses different discretizing transformations, such as Zoh, Tustin, (etc.). It consists of replacing the term "p" of Laplace transformation by a specific term depending on the used discretizing transformation.

Let's discover how to discretize using Tustin transformation!

Let $G(p) = \frac{1}{1+p^2}$

G represents the transfer function of a sine.

According to Tustin, $p = \frac{2}{T_s}.\frac{1-z^{-1}}{1+z^{-1}}$, where T_s is the sampling period. Then we obtain, $G(z) = \frac{1+2z^{-1}+z^{-2}}{\left(1+\frac{4}{T_s^2}\right)+\left(1-\frac{8}{T_s^2}\right)z^{-1}+\left(1+\frac{4}{T_s^2}\right)z^{-2}}$.

To facilitate the implementation of the recurring equation, it is advised to have the constant term of the denominator equal to 1. The we have to divide all the equation by $\left(1+\frac{4}{T_s^2}\right)$.

We finally obtain, $G(z) = \frac{\frac{1}{\left(1+\frac{4}{T_s^2}\right)}+\frac{2}{\left(1+\frac{4}{T_s^2}\right)}z^{-1}+\frac{1}{\left(1+\frac{4}{T_s^2}\right)}z^{-2}}{1+\frac{\left(1-\frac{8}{T_s^2}\right)}{\left(1+\frac{4}{T_s^2}\right)}z^{-1}+z^{-2}}$ (1)

Now after having an idea on the discretization, it is time to generate the recurring equation!

Recurring equation:

By obtaining the discretized function (1) , and knowing that $G(z) = \frac{Y(z)}{U(z)}$, we can easily write:

$$Y(z) = \frac{1}{\left(1+\frac{4}{T_s^2}\right)}U(z) + \frac{2}{\left(1+\frac{4}{T_s^2}\right)}z^{-1}U(z) + \frac{1}{\left(1+\frac{4}{T_s^2}\right)}z^{-2}U(z) - \frac{\left(1-\frac{8}{T_s^2}\right)}{\left(1+\frac{4}{T_s^2}\right)}z^{-1}Y(z) - z^{-2}Y(z) \quad (2)$$

Assuming that S(z) = S(k), where S is a discretized signal, z the Z domain variable and $k \in \mathbb{Z}$ is the current instant. S(k) represents the signal value at the instant k.

Also, assuming that $S(z-i) = S(z)z^{-i},\ i \in \mathbb{Z}$, we finally obtain:

$$Y(k) = \frac{1}{\left(1+\frac{4}{T_s^2}\right)}U(k) + \frac{2}{\left(1+\frac{4}{T_s^2}\right)}U(k-1) + \frac{1}{\left(1+\frac{4}{T_s^2}\right)}U(k-2) - \frac{\left(1-\frac{8}{T_s^2}\right)}{\left(1+\frac{4}{T_s^2}\right)}Y(k-1) - Y(k-2) \quad (3)$$

This recurrent equation presents the basics of the implementation on the MCU.

Implementation of the discretization on the MCU:

To deal with the variable replacement during the discretization phase we were invited to define new structures and implement some algebraic calculation.

Polynomial structure

Each polynomial is characterized by its degree and its coefficients.

```
typedef struct
{
        uint8_t degree;
        float coef[DSP_MAX_POLYNOM_DEGREE+1];
}DSP_Polynom_tst;
```

Functions to treat polynomial

```
DSP_Return_ten DSP_PolynomInit(DSP_Polynom_tst* Polynom_st);

DSP_Return_ten DSP_PolynomCreate(char* PolynomChar , DSP_Polynom_tst* Polynom_st);
DSP_Return_ten DSP_SumPolynom(DSP_Polynom_tst Polynom1_st, DSP_Polynom_tst Polynom2_st, DSP_Polynom_tst*
PolynomSum_st);
DSP_Return_ten DSP_MulPolynom(DSP_Polynom_tst Polynom1_st, DSP_Polynom_tst Polynom2_st, DSP_Polynom_tst*
PolynomMul_st);
DSP_Return_ten DSP_MulScalarPolynom(float Scalar, DSP_Polynom_tst Polynom1_st, DSP_Polynom_tst* PolynomMul_st);
DSP_Return_ten DSP_DevideScalarPolynom(float Scalar, DSP_Polynom_tst Polynom1_st, DSP_Polynom_tst* PolynomMul_st);
DSP_Return_ten DSP_PowPolynom(DSP_Polynom_tst Polynom_st, uint8_t pow , DSP_Polynom_tst* PolynomMul_st);
DSP_Return_ten DSP_CpyPolynom(DSP_Polynom_tst* Polynom1_st , DSP_Polynom_tst Polynom2_st);
```

Those functions are defined in ***DSP_Polynom.c***.

Transfer function structure

Each transfer function is presents a numerator, denominator, one input and one output.

```
typedef struct
{
        DSP_Polynom_tst num;
        DSP_Polynom_tst denom;
        uint8_t k;
        float Y[DSP_MAX_POLYNOM_DEGREE];
        float U[DSP_MAX_POLYNOM_DEGREE];
}DSP_TF_tst;
```

K represents the variable that allows to deal with the discrete instants. It represents the current instant.

Discretization function:

```
DSP_Return_ten DSP_C2D(DSP_TF_tst ContTF, DSP_TF_tst* DiscTF, DSP_ZAlgorithm_ten Algo);
```

In the implemented code, after `/* Numerator */`, we fund the code the calculates the numerator of the discretized function.

After `/* Denominator*/`, we fund the code the calculates the numerator of the discretized function.

After `/* Finalize the calculation depending on the degree of the Numerator and the Denominator */`, we find the finalization of either the numerator or the denominator, depending on respectively their degree.

Finally, me make the constant term of the denominator equal to 1.

Initialize the signal generation:

```
void DSP_vGenerateSignal(DSP_TF_tst* tf);
```

It determines the current instant index k in the structure DSP_TF_tst.

Recurring equation routine:

```
float DSP_fRecurringEquationRoutine(DSP_TF_tst* tf);
```

This routine allows the calculation of the output basing on the recurring equation and then save the previous Y and U values that will be needed to calculate the next value in the next instant.

PS: It is important to mention that this routine must be executed each Ts (Sampling period). We can manage it through implementing a timer or through implement the code in a task and use the osDelay API.