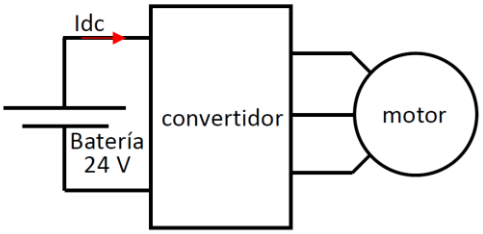
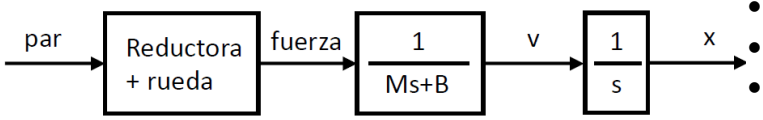
Trabajo Implementación sistemas de control diciembre 2022

Se va a utilizar el sistema de tracción de la figura para mover un vehículo eléctrico





- Diámetro rueda: 0.2 m
- Masa total:25 Kg
- Coef. rozamiento: 0.5 N/(m/s)
- Relación reductora: 1:20
- Tensión batería: 24 V

Se ha adquirido la señal de **par** durante el funcionamiento del sistema, disponible en el archivo txt adjunto (medida en N.m, con ruido en la medición del sensor). El periodo de muestreo es T=15ms. Se asume que no hay pérdidas (rendimiento del 100%).

Trabajo base (5 puntos)

1) Definir e implementar el procesamiento necesario en Matlab para leer los datos del fichero, obtener las variables que se indican a continuación. Representar dichas variables en figuras (entre paréntesis se indican las unidades para la representación)
    a. Fuerza de empuje de la rueda (N)
    b. Velocidad de giro del motor (rpm)
    c. Velocidad lineal del vehículo (km/h)
    d. Desplazamiento del vehículo (m)
    e. Potencia consumida por el motor (W)
    f. Corriente suministrada por la batería (A)

```
>> data_in=importdata('nombre_de_archivo_de_entrada.txt');
>> ...
```

2) Implementar en C++ con Qt-SDK (modo consola) dicho procesamiento, simulando el tiempo con un retardo de 5ms → 1seg de simulación por cada 200 datos del archivo), y generar archivo de salida en formato texto.

$$F(z)=\frac{Y(z)}{X(z)}=\frac{b_0+b_1.z^{-1}+...+b_m.z^{-m}}{1+a_1.z^{-1}+...+a_n.z^{-n}} \Rightarrow y_k=b_0.x_k+b_1.x_{k-1}+...+b_m.x_{k-m}-(a_1.y_{k-1}+...+a_n.y_{k-n}) \Rightarrow$$

- La tabla b debe tener m+1 coefs
- La tabla a debe tener n+1 coefs
- La señal x necesita m+1 valores (actual y anteriores)
- La señal y necesita n+1 valores (actual y anteriores)

Clases necesarias:

`MySignal` → para almacenar los valores actual y anteriores de las diferentes señales

`MyDiscreteSystem` → produce una señal de salida para una de entrada

Formato del programa **(a modo de ejemplo, no es necesario seguir escrupulosamente)**:

```
class MySignal
{
private:
   QVector<float> data;
   int maxDataLength;
...
public:
   bool Initialize(int i_maxLength);
   void Shift();
   void SetActualValue(float new_value);
   const QVector<float>& GetData();
   float GetActualValue();
} ;

class MyDiscreteSystem
{
private:
   QVector<float> b,a;
public:
   bool Initialize(const QString& pol_b,const QString& pol_a);
   float Calc1Step(const QVector<float>& x,const QVector<float>& y);
} ;


main()
{
    MySignal force_N,speedMotor_rpm,…; // Genéricamente Xj en lo que sigue
    MyDiscreteSystem torque2force,force2speed,…; // Genéricamente Fj en lo que sigue

    float wheel_diam,...;  // Todos en unidades del S.I.
    QString params_xml;

    Leer params_xml de archivo "params.xml"
    wheel_diam=XmlGetFloat(params_xml,"WheelDiam_mm")/1000.0;
    ...

    Inicializar sistemas discretos con sus parámetros:
    Fj.Initialize("[b0,b1,...,bm]","[a0,a1,…an]");  // Coefs personalizados para cada Fj

    Inicializar señales según el tamaño requerido:
        la vble Xj necesita el máx de length(Fj-1.a),length(Fj.b)
    Xj.Initialize(tam necesario para Xj);
    ...

    QFile input("input.txt"),output("output.txt");
    input.open(QIODevice::ReadOnly);
    output.open(QIODevice::WriteOnly);
    QString lineIn,lineOut;

    while (input.canReadLine())
    {
        Xj.Shift(); // ... todas las señales son un instante más antiguas ...
        lineIn=input.readLine();
        parse lineIn → X1.SetActualValue(valor leído de lineIn);
        ... por orden de cálculo ...
        Xj+1.SetActualValue(Fj.Calc1Step(Xj.GetData(),Xj+1.GetData()) );
        lineOut = crear línea de salida con los valores Xj.GetActualValue() seguido de \n;
        output.write(lineOut.toLatin1());
        QThread::mSleep(5);
    }
}
```

3) Implementar en Simulink el control de velocidad del sistema de la figura. El regulador de velocidad es PI, y su salida es el par motor (se asume que el control del motor tiene una respuesta instantánea). El control deberá de ser estable.

Mejoras (1 punto por cada una)

1) Obtener la FFT de la señal muestreada; identificando los armónicos del ruido
2) Diseñar e implementar en Matlab un filtro paso bajo para eliminar dicho ruido, eligiendo una de estas dos opciones (solo una!). Es necesario explicar brevemente el criterio elegido para seleccionar la frecuencia de corte del filtro, y comentar el resultado de aplicar el filtro.
    a. Filtro de primero orden (0.5 puntos)
    b. Filtro de Butterworth de segundo orden (1 punto)

Es necesario explicar brevemente el criterio elegido para seleccionar la frecuencia de corte del filtro, y comentar el resultado de aplicar el filtro.

3) Implementar dicho filtro en Matlab. Representar tanto en el tiempo como en frecuencia (FFT) las señales sin filtrar y las filtradas

4) Modificar programa C++ para que funcione en GUI con:
    - Selección de nombre de archivo de entrada, de salida, y de parámetros
    - Checkbox Run/Pause
    - Ejecución bajo QTimer

5) Sintonizar el regulador de velocidad del apartado 2) del trabajo base para que el sistema en cadena cerrada tenga un ancho de banda de 0.1 Hz.

ENTREGABLES

Comprimido en un archivo (Matlab.zip):

1. Script(s) de Matlab con el procesamiento
2. Documento pdf explicando los cálculos realizados y los resultados obtenidos (sólo ampliaciones)

Comprimido en un archivo (Cpp.zip):

3. Archivos .cpp , .h y .pro