Contenido

[Descripción de scripts 1](#_Toc125526293)

[calcCRC\_CCIT.m 1](#_Toc125526294)

[conectarPMU.m 1](#_Toc125526295)

[CRC\_16\_CCITT.m 2](#_Toc125526296)

[decodeCFG2.m 2](#_Toc125526297)

[decodeDataframe.m 2](#_Toc125526298)

[getCRC.m 2](#_Toc125526299)

[guardararchivo.m 2](#_Toc125526300)

[PDC.mlapp 2](#_Toc125526301)

[readDFPMU.m 3](#_Toc125526302)

[sendCMD.m 3](#_Toc125526303)

[sockettest.m 3](#_Toc125526304)

[sqlitetest.m 3](#_Toc125526305)

[testip.m 3](#_Toc125526306)

[test\_scripts.m 3](#_Toc125526307)

# Descripción de scripts

Los scripts están separados para que sean como un tipo de funciones (similar a los módulos de python) y poder reutilizar el código. La app de Matlab manda llamar estos scripts.

## calcCRC\_CCIT.m

El estándar C37.118 utiliza la verificación de redundancia cíclica (CRC) en la transmisión y recepción de dataframes para verificar y corregir errores. Existen diferentes implementaciones del código crc ([ver Types of CRCs](https://quickbirdstudios.com/blog/validate-data-with-crc/#:~:text=CRC%20is%20an%20error%20detection,stored)%20along%20with%20that%20data.&text=The%20check%20value%20is%20called,additional%20information%20to%20the%20message.)) por eso hay diferentes scripts con diferentes códigos CRC. Solo uno de ellos funcionó con la implementación del protocolo C37.118 que estaba usando.

## conectarPMU.m

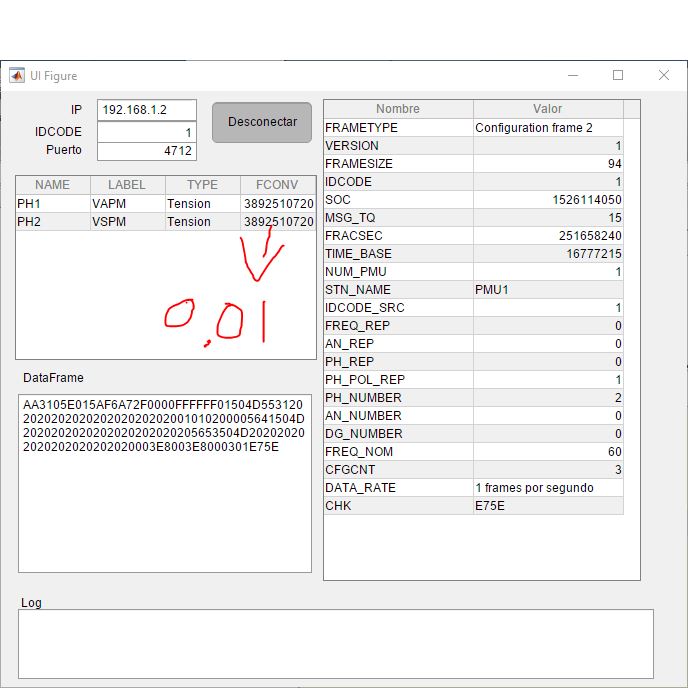
El script manda un command frame “Send CFG2 file” a la PMU. Este configuration frame 2 es el que le dice a Matlab como esta configurada la PMU (e.g. nombre de la PMU, frecuencia nominal, numero de frames por segundo, etc.). Se tiene como argumentos de la función la IP de la PMU, el puerto de la PMU y el IDCODE de la PMU. Regresa el configuration frame 2 (sin decodificar) y el objeto TCP que contiene el objeto [socket](https://es.wikipedia.org/wiki/Socket_de_Internet) que se creó.

Ilustración 1. Ejemplo de configuration frame 2.

## CRC\_16\_CCITT.m

Otro script para calcular el CRC.

## decodeCFG2.m

Después de pedir el configuration frame 2 con conectarPMU.m hay que decodificarlo ya que los frames son una trama de datos del tipo Ilustración 2:

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Ilustración 2. Stream de caracteres.

Este script lo que hace es tomar el stream serie de caracteres recibidos de la PMU (solo CFG2, de ahí el nombre del script) y separarlo en bytes o strings para que sean legibles por un humano (ver tabla 9 del PDF del estándar).

## decodeDataframe.m

Un dataframe es el frame que contiene los datos de medición de la PMU. Estos datos vienen codificados (tabla 7,8 del estándar C37.118) por lo que hace falta decodificarlos y quitarles los bytes que no son necesarios, como el CRC. Este script hace eso.

NOTA: viene un comentario “falta el factor de conversión”, no me acuerdo si ya está corregido (creo que si). Este error se puede ver en la Ilustración 1, en lugar de ser 0.01 el factor de conversión es otro número.

## getCRC.m

Otro script para el CRC. Este es el que se terminó usando.

## guardararchivo.m

Parte de los script de prueba para guardar los datos leídos de la PMU a un archivo. En este momento no se necesita, no tengo implementado esas funciones, solo son tests.

## PDC.mlapp

Es el archivo de la app (interfaz gráfica de usuario – GUI). Este archivo se abre con el app designer de Matlab.

## readDFPMU.m

Este script es exclusivo de la app. Lee continuamente los dataframes, dependiendo del estado del botón conectar/desconectar. Aquí se calculan los ángulos con respecto a una fase de referencia. En esta parte tengo dudas si realmente así se calculan, pero creo que las pruebas que hice si me funcionaron.

## sendCMD.m

Manda un command frame a la PMU, ver tabla 12 y 13 del estándar C37.118.

## sockettest.m

Es un script de prueba, no es necesario para leer datos de la PMU. Creo que con este intentaba simular una PMU que manda datos.

## sqlitetest.m

Script de prueba para guardar datos en una base de datos sqlite. No es necesario.

## testip.m

Creo que fue el primer script que hice para leer datos de la PMU. Lo que hace es mandar un command frame a la PMU solicitando el configuration frame 2. Después de que el script recibe el configuration frame 2, lo decodifica y lo imprime en consola. Básicamente separe este script copiando y pegando las partes del código que luego se convirtieron en los scripts individuales.

## test\_scripts.m

Lo tengo en github. Este script sustituiría a la app, pero no lo he hecho. Una guía de como hacerlo es:

ip = "192.168.1.10";

puerto = 4712;

idcode = 1;

%se solicita el configuration frame 2

[tcpobj, dFrame] = conectarPMU(ip,puerto,idcode);

%se decodifica el configuration frame 2

[CFG2, cellPhasors] = decodeCFG2(dFrame);

%se manda del comando para que la PMU empiece a transmitir datos

cmd = 2;%turn on transmission of dataframes

sendCMD(cmd,idcode ,tcpobj)

%aquí esta lo no tan simple. hay que rellenar los datos obtenidos del

%configuration frame 2

%cellphasors contiene los datos de los fasores y cfg2 contiene los datos de

%configuracion de la pmu necesarios para decodificar el data frame

%la siguiente funcion debe de ir en un ciclo o algun codigo que se repita,

%ya que la pmu esta mandando datos indiscriminadamente (asincronamete)

%lo que hago en la app es leer continumamente los datos del socket y de ahi

%procedo a identificar los dataframes individualmente y decodificarlos.

[decodedDF, PHEst, ANEst, DIGBits] = decodeDataFrame(dFrame, repPH,repFREQ,repAN,...

repPHpol,PHNMR,ANNMR,DGNMR,timeBase,facConv);

%despues de decodificar los datos hay que visulizarlos, ya sea imprimiendo

%a consola o en una interfaz gráfica.