eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología
Euskadi 2006

Gradu Amaierako Lana / Trabajo Fin de Grado

Ingenieritza Elektronikoko Gradua / Grado en Ingeniería Electrónica

# Desarrollo de un sistema de control remoto utilizando EPICS

Memoria

Egilea/Autor:

Alain Porto

Zuzendaria/Director/a:

Josu Jugo

# Índice

## *Resumen*

En el siguiente documento se describe el proceso que se ha llevado a cabo para realizar el trabajo de fin de grado “Desarrollo de un sistema de control remoto utilizando EPICS” del grado de Ingeniería Electrónica en colaboración con el Centro de Láseres Pulsados de Salamanca, CLPU [1] durante el curso 2014-2015. El trabajo se centra en el estudio de un Freeware de sistemas distribuidos llamado EPICS, para la monitorización de un motor de paso y el uso de una cámara.

Este proyecto consta de tres partes. Un primer capítulo está dedicado a introducir los conceptos básicos de EPICS que serán necesarios para el desarrollo del proyecto en su totalidad. Tales como la estructura de un sistema estándar EPICS, los conceptos de Process Variable, IOC, database, device support y driver support. También se hablará de las herramientas complementarias utilizadas para llevar a cabo el trabajo, como Python, C, C++ o AsynDriver.

Una segunda parte dónde se describe en detalle el proceso llevado a cabo para crear las distintas aplicaciones para el motor y la cámara:

Para el caso del motor, se desarrollará un sistema EPICS que permita controlar la posición del motor. También se utilizara el lenguaje de programación Python para éste propósito. Por último se desarrollará una interfaz gráfica con el programa CSS.

La parte de la cámara, se centrará en crear un control para la toma de una foto de manera digital, con la ayuda de EPICS. En este caso también se dispondrá de Python, y se desarrollará una interfaz gráfica con CSS.

En una tercera, se describe el proceso de como se han hecho los distintos entornos gráficos para cada una de las aplicaciones.

# 1 Introducción

En la actualidad, cada vez es más cotidiano encontrarse con sistemas científicos de gran envergadura dedicados a la investigación y la satisfacción de la curiosidad humana. Estamos hablando de centros como el CERN, el gran telescopio sudafricano, SALT o el Argonne National Laboratory ANL.

Estos centros son lugares dotados de los medios necesarios para realizar investigaciones, experimentos, prácticas y trabajos de carácter científico, tecnológico o técnico. Con lo que están equipado con instrumentos de medida o equipos con los que se realizan experimentos, investigaciones o prácticas diversas. Como es de esperar, para llevar a cabo todas éstas investigaciones, es necesario una gran cantidad de equipos que deben ser controlados, de modo que necesitan de sistemas que administren los aparatos. Este arsenal tecnológico necesita de tecnología de sistemas distribuidos para poder ser controlado. Existen varios estándares para este tipo de sistemas, como son COBRA, SIMATIC o Industrial IT system 800x A [2].

Uno de los estándares más extendidos es el conjunto de herramientas software bautizado como EPICS.

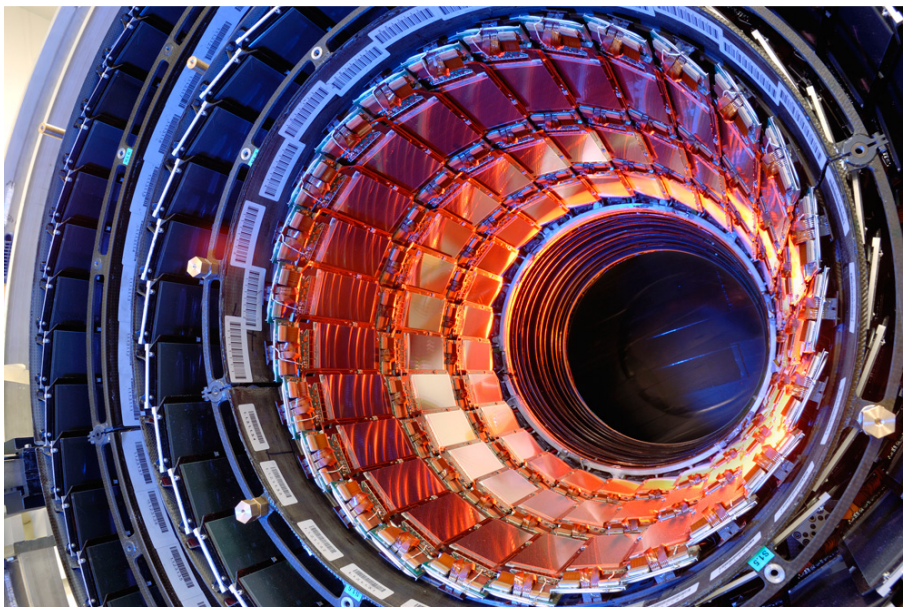


Figura 1: CMS de LHC

Éste sistema está siendo utilizado en diversos centros tanto para adquisición de datos, como para control supervisor, control en lazo cerrado y optimización. Los colaboradores actuales que utilizan este sistema para su centro ANL, SNS, LBL[3]. Varias universidades utilizan también EPICS como sistema estándar para control de procesos [4].

## 1.1 ¿Qué es EPICS?

EPICS es un conjunto de herramientas de software y aplicaciones [5] que proporcionan una estructura de software para sistemas distribuidos de gran capacidad, como pueden ser aceleradores de partículas, telescopios o experimentos de gran tamaño.

En este tipo de sistemas se ven envueltos decenas o centenas de ordenadores, que deben conectarse entre sí para compartir información sobre el control del proceso, que se llevará a cabo de manera remota, ya sea en una sala central o por internet. Para solventar este problema EPICS utiliza la el modelo cliente/servidor, en concreto TCP/IP .

La naturaleza de los sistemas distribuidos es poder controlar procesos, para lo que se necesitan variables que almacenen datos sobre el estado de los mismos, en EPICS este tipo de variables se conocen como 'process variable' o PV. EPICS se basa en el modelo cliente/servidor, es decir, una máquina requiere un servicio (cliente) y otra se lo presta (servidor). Intuitivamente podemos barruntar que el cliente en este caso solicitará un PV (process variable) y quien se la facilitará es el servidor [6]. El protocolo de comunicación que se utiliza para acceder del cliente al servidor se llama 'Chanel Access' o CA, de modo que para ser consecuentes al cliente se le conoce como 'Channel Access Client' o CAC y al servidor 'Channel Access Server' o CAS [7].

## 1.2 Estructura EPICS

Un sistema estándar de EPICS está formado por varios CAC (channel access client) y CAS (channel access server) y cada uno de estos a su vez contendrá PVs (process variables). Los clientes podrían ser programas que se están ejecutando en un ordenador, que monitorizan el proceso y acceden a las variables. Por otro lado, los servidores serían maquinas capaces de medir una magnitud y hacerla accesible desde la red.

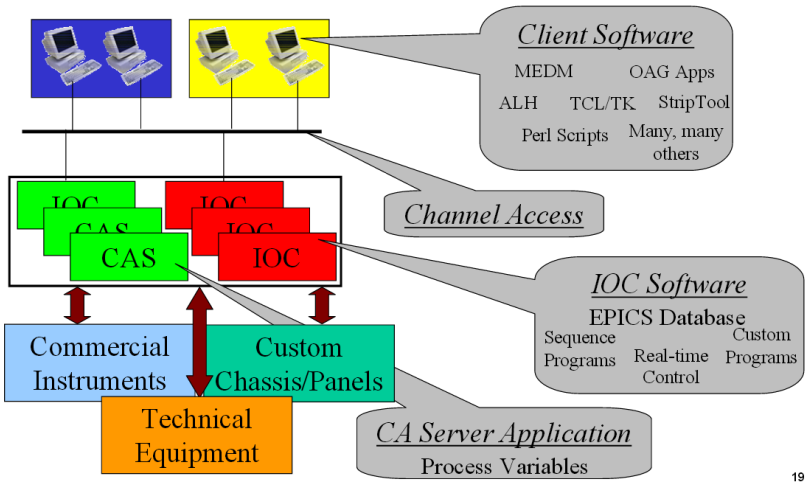



Figura 1 Ejemplo de sistema típico EPICS

### 1.2.1 Process Variable, PV

Es un dato asociado con un proceso o una variable física (estado, posición, velocidad...). Para identificarlo necesitamos asociarle un nombre único (lo que se llama id), para que el acceso sea univoco. Ejemplos de id's y valores de una PV serían:

1. Articulation1:Motor1.speed 0.5 rad/s
2. Articulation1.status 'Rotating'
3. room34:Camera4.pixelArray {1000101,001010,110101}

Donde la primera parte es el id, y la parte en azul es el valor.

Se pueden asignar atributos que cambien dependiendo del valor de la PV, como las unidades de la medida, alarmas que adviertan sobre un valor mínimo y máximo en la variable, o limites que las variables no pueden superar.

## 1.3 Métodos principales de CA (Channel Access)

Se ha mencionado que el principal objetivo de EPICS es obtener información de un proceso. Para ello se necesitan métodos para obtener información de los servidores, y cambiar ésta según el criterio del usuario. EL CA pone a disposición del usuario tres métodos para llevar a cavo esta tarea, 'caput' (Channel Access put), 'caget' (CA get) y 'camonitor' (CA monitor) [8]. Intuitivamente se puede observar que 'caget' servirá para obtener una variable de un servidor, es decir para obtener la PV que esta almacenada en un CAS. El método 'caput' es el contrario, permite que un cliente modifique en una variable del servidor, o dicho de otra manera, permite que un CAC modifique una PV de un CAS. Por último está 'camonitor', que devuelve la PV cada vez que esta cambia.

En la siguiente imagen se ilustran cada uno de los métodos que utiliza CA:

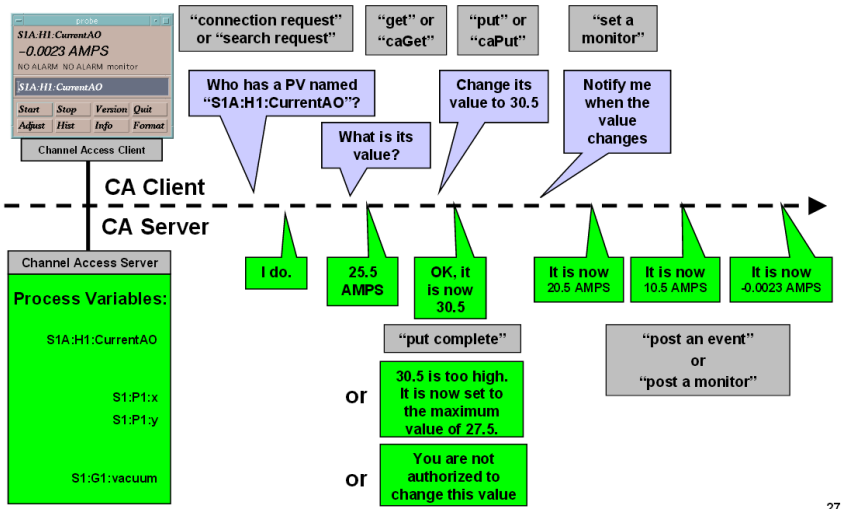



Figura 2 Ilustración del uso de los métodos de CA

Cada una de estas variables a las que se accede, deben estar controladas por un sistema que las gestione, en EPICS a este concepto se le conoce como Input Output Controller, IOC.

## 1.4 IOC (Input Output Controller)

Se trata del sistema que gestiona las PVs que son necesarias para la monitorización o control de cierto proceso. Este tipo de sistema se puede desglosar en las siguiente partes.

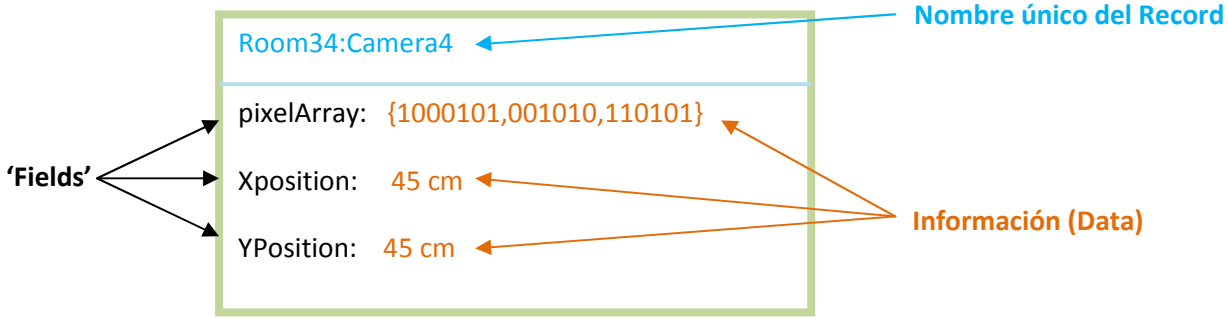
### 1.4.1 iocCore (Input output Controller Core)

El iocCore es un conjunto de funciones de EPICS que permiten definir las PV y crear algoritmos para control a tiempo real. Se puede decir que el iocCore está compuesto por los siguientes elementos:

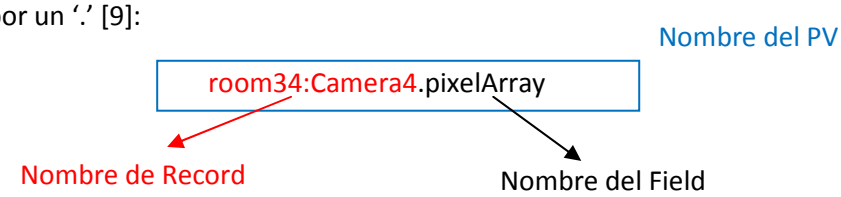
- Records
- Databases
- Driver Support
- Device Support

*Records*

Es el subconjunto de rutinas del iocCore que hacen posible definir PVs. Es un objeto con nombre único, con ciertas propiedades (que llamaremos 'fields') que contienen información (Data). Un ejemplo sería:



La referenciación de una PV se crea a partir de la concatenación del nombre del record y el field separado por un '.' [9]:




Hay muchos tipos de record, dependiendo de la naturaleza de las variables del proceso a medir, como son [10]:

- Ai-(Analog Input) Record dedicado a señales analógicas
- Bo- (binary Output) Record específico para salidas binarias
- Cacl-(Calculation) Para hacer cálculos con las PVs de otros records

*Databases*

Es dónde se definen los record, es decir, es donde se decide que atributos se le asignan a cada uno y que funcionalidad tiene.

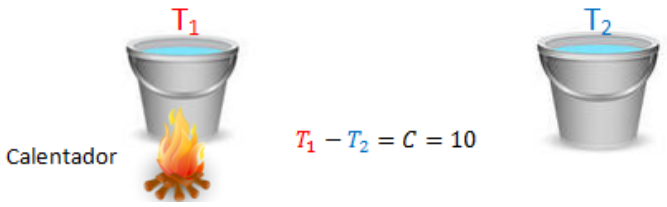
*Driver Support y Device Support*

Las PVs representan una variable física o un proceso, para medirlas se suelen utilizar dispositivos, y estos dispositivos deben entenderse con EPICS. Para que esto suceda necesitamos desarrollar drivers que hagan que los datos recogidos por los dispositivos se conviertan en PVs. Para esta tarea están el Driver Support y Device Suport, que se utilizan dentro de los records, ya que es en estos dónde se definen las PVs.

Una vez introducidos estos conceptos, se puede definir un IOC con más detalle. Se trata de una máquina que ejecuta una instancia de iocCore en su interior, y es accesible desde el CA. Estrictamente, se llama IOC cuando la máquina donde se implementa está dedicada a hacer control en tiempo real. Cuando la máquina es de uso 'no dedicado' (un ordenador de propósito general) se llama 'softIOC'.

### 1.4.2 Ejemplo de IOC

En esta sección se muestra un ejemplo de IOC para intentar clarificar al máximo los conceptos. Como ejemplo, se necesita un sistema que tiene que mantener una diferencia de temperatura constante C=10 entre dos cubos de agua, y se dispone de un calentador para uno de los cubos:



Para implementar este sistema en un IOC, se piensa en las PVs que se necesitan. Son necesarias PVs para cada una de las temperaturas de los cubos, otra PV para la diferencia de temperaturas C y otra que active el calentador. Ahora en los records, dado que las temperaturas son analógicas lo ideal sería utilizar un ai-Record para cada una. Para obtener la diferencia de temperaturas, lo mejor sería un record tipo CALC y por último, para activar el calentador un record de tipo BO ya que es de tipo binario, o calentamos o no. El IOC quedaría como sigue:

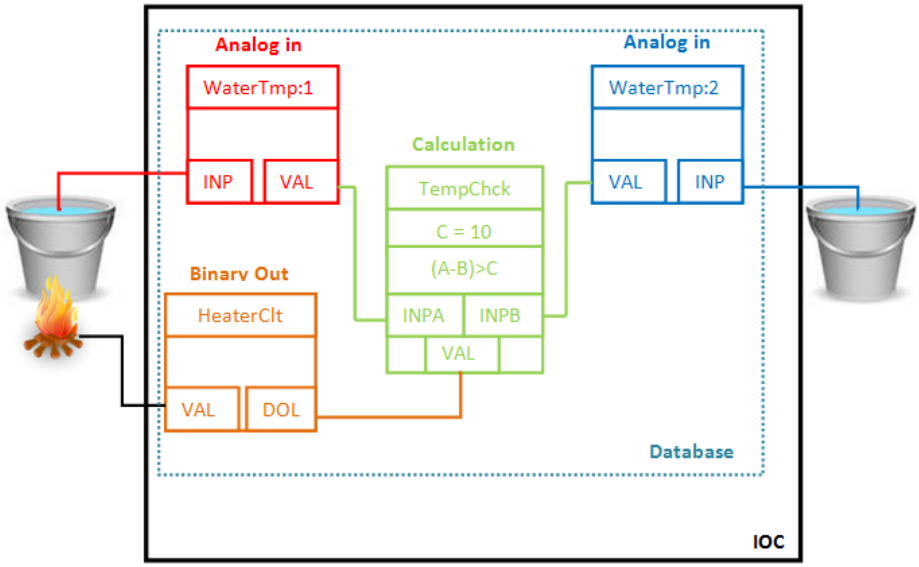



Figura 3: IOC completo

Como se puede observar, se mide la temperatura $T_1$ y $T_2$ que se almacena en los fields VAL de los records WaterTmp:1 y WaterTmp:2 . Después estos valores se pasan a un record CALC, que lleva a cabo la operación (A-B)>C, donde A es la entrada que llega a INPA y B la entrada que llega a INPB ($T_1$ y $T_2$ respectivamente), el resultado se almacena en el field VAL del record CALCulation. Por último este se pasa al DOL (un field del record BinaryOut que nos indica de donde queremos obtener la señal de entrada), que dependiendo del VAL del Calculation, provocara que el calentador se encienda o se apague.

El database, sería el conjunto de records utilizados en el sistema, y el IOC, aunque no se aprecie en este ejemplo, incluiría también los dispositivos utilizados para comunicarse con el hardware (como tarjetas de adquisición, ADC, CAD…).

Ahora que se han hecho una introducción a EPICS, cabe mencionar que en algunos desarrollos se han utilizado otros softwares complementarios. De modo que facilitase la comunicación con los dispositivos físicos. En concreto se ha utilizado el complemento software AsynDriver que se explica en la siguiente sección.

## 1.5 AsynDriver

Se conoce como AsynDriver [11] al conjunto de rutinas software que facilitan la comunicación de EPICS con los dispositivos físicos. El problema de comunicación de EPICS es que para cada hardware es necesario definir un driver. Para cada tipo de dispositivo (GPIB, Serial, Ascii...) es necesario crear driver supports y devices supports:

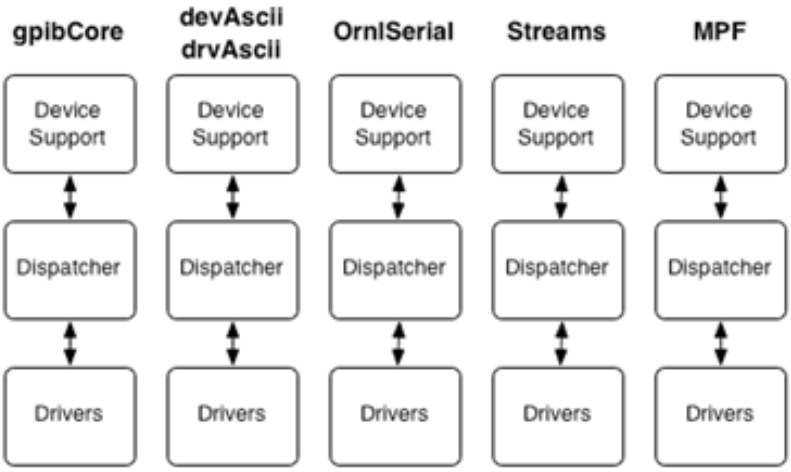



Figura 1: Problema de EPICS

Los desarrolladores de AsynDriver crearon esta herramienta de modo que todos los drivers se comunicasen directamente con un único software. De esta manera el software para los drivers sería independiente del hardware:

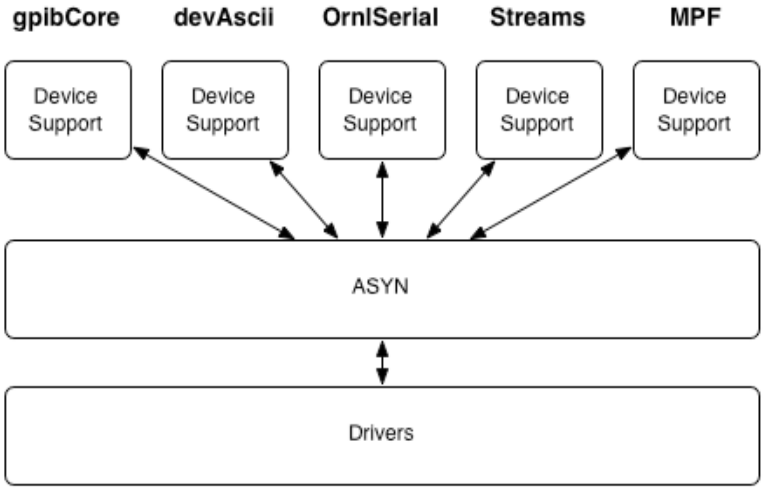



Figura 4: Ventaja de AsynDriver

AsynDriver cambia el modo de acceso que tienen el software al dispositivo. En vez de comunicarse con el driver, se utiliza lo que se llama "frameWork" (asynInt32, asynFloat64 ...), el software se comunica con este, y asynDriver hace lo pertinente para comunicarse con el dispositivo. De modo que hay que configurar el software y los frameWorks de manera que apunten correctamente a lo que se desea.

Si ésta utilidad no trajese el driver que se busca, tiene una alternativa para facilitar la creación de drivers, una clase de C++ llamada AsynPortDriver.

En este trabajo se utilizarán tanto GPUB-serial como el AsynPortDriver.

## 1.6 Control System Studio CSS

Control System Studio o CSS es un conjunto de utilidades desarrolladas en una plataforma basada en Eclipse, diseñado para plataformas de control de gran extensión, como son los aceleradores de partículas [12].

Hay dos maneras de ver CSS, una como usuario final, y otra como un desarrollador y/o administrador del sistema. Como usuario final, CSS sería capaz de mostrarnos la interfaz gráfica desarrollada por algún administrador con solo ejecutar la aplicación. Como desarrollador, permite crear entornos gráficos con controles estándares (botones, indicadores, graficadoras...) que monitorizan PVs del software EPICS.

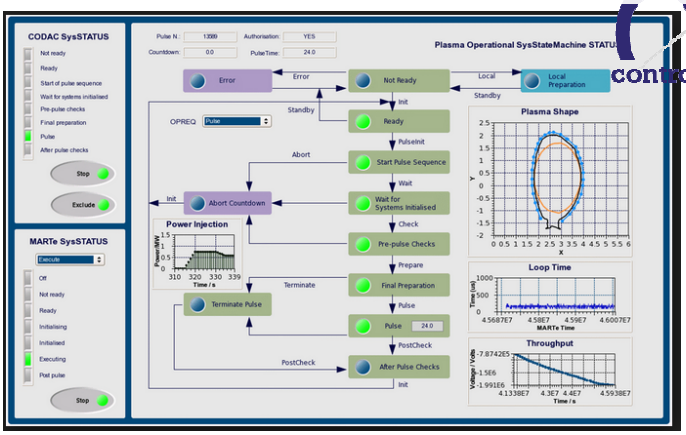



Figura 5: Ejemplo de Interfaz gráfica con CSS

# 2 Objetivos

En este trabajo se pretenden alcanzar los siguientes objetivos:

1. **Desarrollar un IOC para el control del motor Zaber T-HLA28-S**.
   Se pretende desarrollar un sistema EPICS, capaz de comunicarse con el motor a través del protocolo de comunicación que el fabricante especifique. El sistema deberá traducir los datos introducidos por el usuario en comandos comprensibles por el motor.
2. **Crear un interfaz gráfico para el control del motor Zaber**.
   Se va a tratar de crear un entorno gráfico tal que las funcionalidades del anterior objetivo sean fácilmente llevados a cabo por un usuario final.
3. **Desarrollar un IOC para el control de la cámara EO-5012c**.
   Se creará un sistema que sea capaz de tomar una foto y que la introduzca en EPICS.
4. **Desarrollar un entorno gráfico para el control de la cámara**.,
   Se desarrollara una interfaz gráfica sencilla, con el fin de que un usuario final pueda utilizar este sistema sin conocimientos sobre la materia.
5. **Implementar todos estos casos en un IOCSoft**
   Se pretende desarrollar todos los sistemas tanto el del motor y el de la camara en un ordenador de propósito general.

# 3 Desarrollo de un primer IOC

En este capítulo se pretende simular un sistema de segundo orden utilizando EPICS para la compresión más profunda de cómo funciona EPICS. Para ello se hará uso de un softIOC que se creará en ordenador cualquiera con un sistema operativo basado en Linux, es este caso Ubuntu. En los siguientes pasos se supone que EPICS está ya instalado en el ordenador, en caso contrario consulte el Apéndice.

## 3.1 Creación de la aplicación

Para empezar se comienza una sesión en una terminal, y a continuación se accede al directorio donde se ha instalado EPICS ,que habremos definido una macro para éste que llamaremos EPICS_BASE (véase apéndice). Se crea una nueva carpeta mediante el comando 'mkdir':

```
EPICS_BASE$ mkdir sistema
```

Entramos en la carpeta:

```
EPICS_BASE$ cd sistema
```

EPICS trae consigo programas que ahorran tener que escribir todo el soporte para una aplicación. Todos esto se encuentran en la subcarpeta '/bin' dentro del directorio donde está instalado EPICS. Se utilizará el programa makeBaseApp.pr para generar la primera aplicación:

```
EPICS_BASE/sistema$ $EPICS_BASE/bin/makeBaseApp.pl -t ioc sistema
```

Con este comando se crearán las carpetas y archivos necesarios para definir el IOC. Para que éste pueda ser ejecutado necesita otra subcarpeta llamada '/iocBoot', para ello utilizando el mismo programa, se añade la opción '-i':

```
EPICS_BASE/sistema$ $EPICS_BASE/bin/makeBaseApp.pl -i -t ioc sistema
```

## 3.2 Sistema de segundo Orden:

Ahora se pretende definir un sistema de segundo orden en EPICS en tiempo discreto. Un sistema de segundo orden en tiempo discreto, viene definido por su ecuación en diferencias :

$$x(n) = Ay(n-2) + By(n-1) + Cy(n)$$

Dónde y es la señal de salida, x la de entrada y A,B y C constantes. Lo que nos indica esta ecuación es que la salida depende de la magnitud de la señal en instantes anteriores, es decir, se puede decir que el sistema tiene 'memoria' de lo que ha ocurrido anteriormente con la señal y(n).

## 3.3 Implementación en EPICS

Para implementarlo en EPICS primero se piensa en un problema en bloques, es decir, se hace un boceto de que es lo que debería hacer la aplicación. De modo que en este caso el problema se separa en dos parte. Una se encargará de generar una señal tipo, y la otra de simular un sistema de segundo orden.

### 3.3.1 Implementación de la Señal de entrada

En este caso se creará un tren de pulsos del cual es posible seleccionar el periodo. Como la señal se repite periódicamente, se define la señal en un periodo y se repite constantemente. Para ello se crea primero un eje temporal que dure exactamente un periodo de la señal, y para definir el pulso, la mitad del periodo la señal estará en alta, y la otra mitad en baja, si se repite esto, se obtendrá un tren de pulsos:

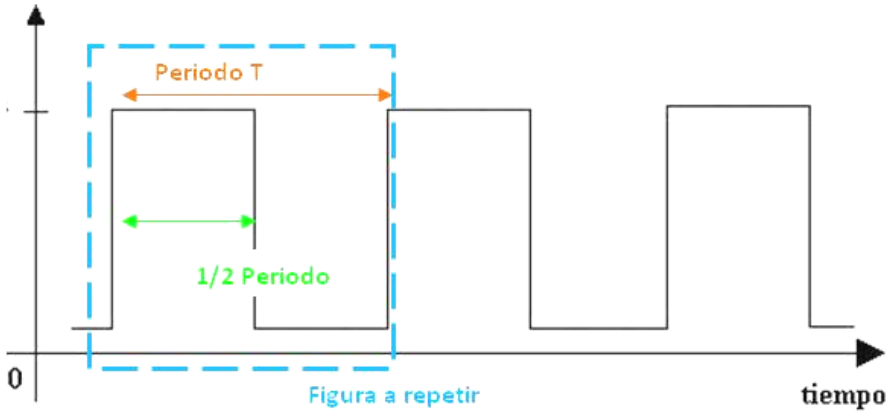



Figura 5: Tren de pulsos

De modo que las condiciones para crear el periodo son fáciles, si t (que va desde 0 hasta el periodo T) es menor que T/2 la señal será 1, de cualquier otra manera será 0. Para crear la señal de t lo que vamos hacer es ir sumando uno hasta que se llegue al periodo, veamos si se aclara el concepto con un diagrama de flujo, acompañado con el bloque en EPICS:

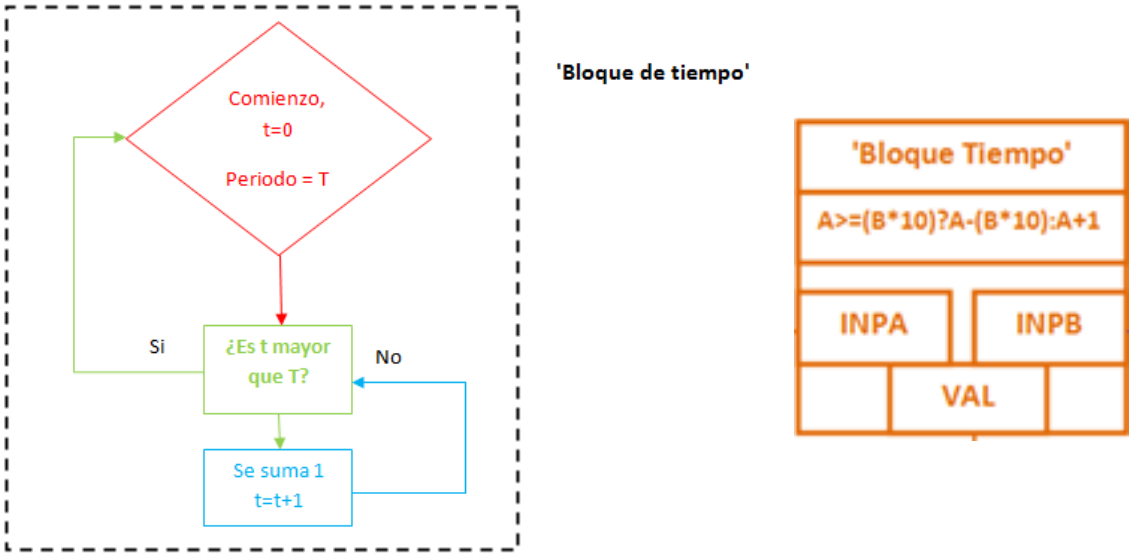



Figura 6:Diagrama de flujo y el Equivalente en EPICS

De modo que el valor de t en cada momento será el 'tiempo'. Ahora si imponemos condiciones sobre t, podremos crear la señal pulso, como vemos en el siguiente diagrama:

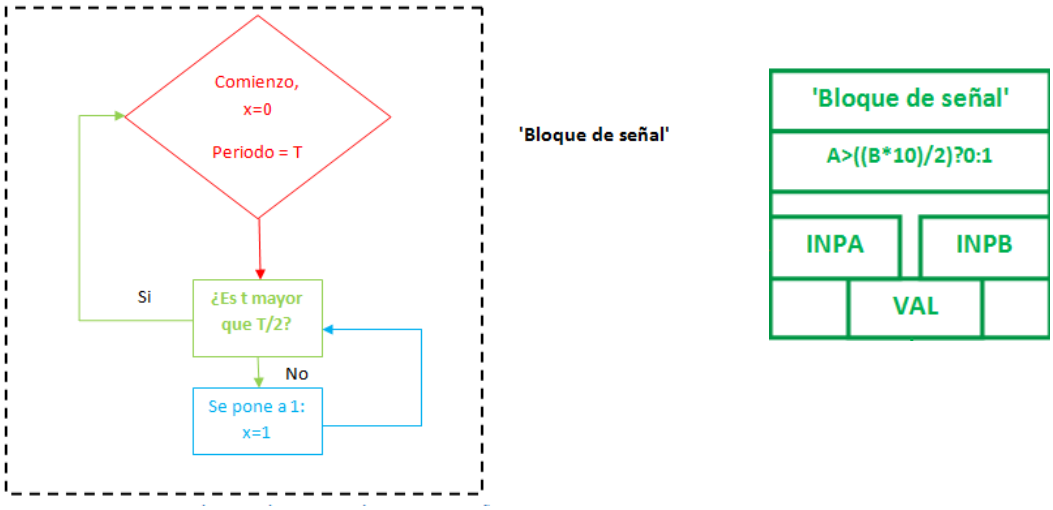



Figura 7:Diagrama de flujo y su equivalente en EPICS

Ahora para traducir esto a lenguaje EPICS, primero se piensa en el tipo de records que se van a utilizar, debido a que todo es cálculo numérico, se utilizarán records del tipo CALC:

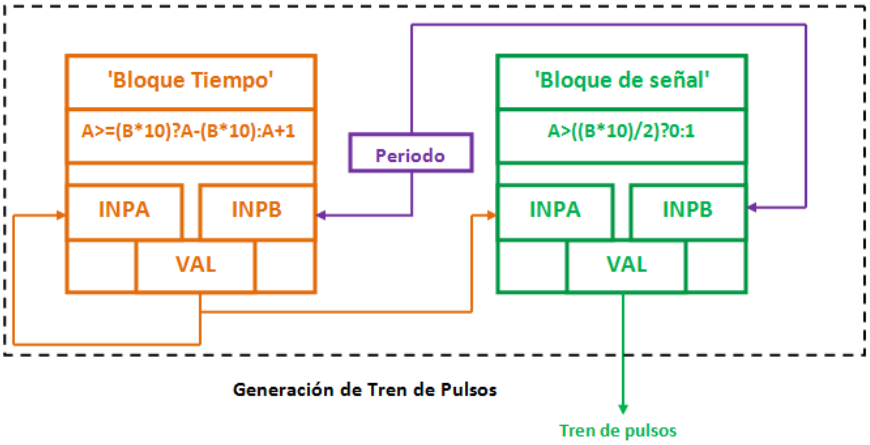



Figura 8: Diagrama de bloques del sistema que genera la señal

Dónde las expresiones que aparecen son las que utiliza el record CALC [3], que indica que valor va a tener el field VAL. Funciona de la siguiente manera:

$$\text{(A+B)<(C+D)?E:F+L+10} \begin{cases} El\ resultado\ es\ E\ si\ (A+B) < (C+D) \\ El\ resultado\ es\ F+L+10\ si\ (A+B) \geq (C+D) \end{cases}$$

Ahora se pasa a traducir el diagrama de bloques a lenguaje EPICS. Como se ha mencionado cada uno de los bloques de antes será un record CALC:

Bloque de tiempo:

```
record(CALC,tiempo){
      field(SCAN,".1 second")
      field(INPA,"tiempo.VAL")
      field(INPB,"periodo.VAL")
      field(FLNK,"signal")
      field(CALC,"A>=(B*10)?A-(B*10):A+1")
}
```

Donde los field SCAN, significa cada cuanto se procesa el record (en nuestro caso cada 0.1 segundos), INPA INPB cuáles serán las entradas, el field CALC, que dice cual será el cálculo a computar y FLINK (fordward link) que indica que si éste record ha sido procesado debe

procesarse el record que se indique en el valor de FLINK (en nuestro caso cada vez que se procese el record 'tiempo' se procesará 'signal' también).

- Bloque de señal

```
record(CALC,signal){
      field(INPA,"tiempo.VAL")
      field(INPB,"periodo.VAL")
      field(CALC,"A>((B*10)/2)?0:1")
}
```

De esta manera se tiene en el field VAL del record 'signal' el tren de pulsos que se deseaba.

- Record del Periodo

```
record(CALC,periodo){
      field(PINI,"YES")
      field(CALC,"3")
}
```

Este record es una caja que mantiene constante el valor del periodo, para cambiarlo se accede mediante el CA, con la subrutina 'caput'. De esta manera se cambia el periodo de la señal. El field PINI es para que el record se procese nada más se inicie la aplicación, de esta manera tendrá un periodo por defecto de 3s.

Todo esto se escribirá en un archivo .txt (archivo de texto plano), que se tendrá que se creará en el directorio /EPICS_BASE/sistema/db/, ya que esto se trata de un database. Como se ha visto en el ejemplo de IOC que se planteaba en el apartado de la introducción, estamos creando los records que componen el IOC, en nuestro caso un SoftIOC.

### 3.3.2 Implementación del sistema de 2º orden

Una vez se tiene la señal de entrada definida, se pasa a implementar el sistema. La idea es crear records que mantengan el valor de la señal para un n dado (n instante de tiempo discreto), luego los valores de estos se sumen de acuerdo con la ecuación en diferencias que se ha presentado antes. En este caso las constantes A,B y C se representaran como el periodo en el anterior caso, es decir, se guardarán en un record que mantendrá constante el valor a lo largo del tiempo, con unos valores predeterminados que pueden cambiarse al gusto del usuario.

El diagrama de bloques sería el siguiente:

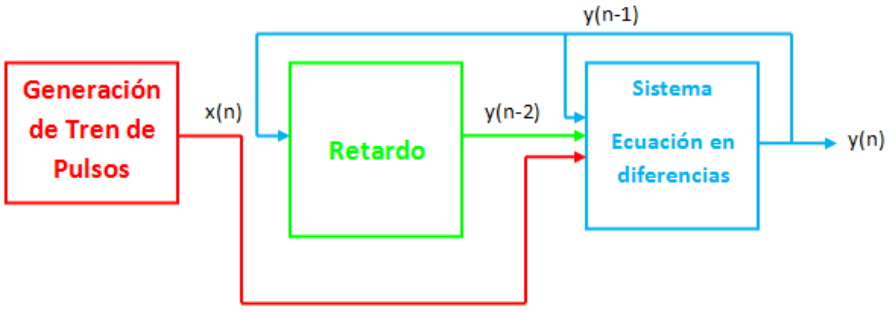



Figura 9: Diagrama de bloques del sistema completo

Como se puede observar, se crea un record que simplemente copie y retenga la señal (lo que se ha llamado como retardo) y otro al que le llega el retardo y la señal original, que computa la ecuación en diferencias, simulando un sistema de 2º orden. La ecuación en diferencias para obtener y(n) sería:

$$x(n) = Ay(n-2) + By(n-1) + Cy(n) \Rightarrow$$

$$\mathrm{y(n)} = \frac{1}{\mathrm{C}}\mathrm{x(n)} - \frac{\mathrm{B}}{\mathrm{C}}\mathrm{y(n-1)} - \frac{\mathrm{A}}{\mathrm{C}}\mathrm{y(n-2)}$$

Así se obtiene y(n) en función de la señal de entrada y los valores anteriores de y(n). Como en el anterior caso se implementarán todos estos bloques con records de tipo CALC, de modo que en lenguaje EPICS quedaría como:

- Sistema

```
record(CALC,sistema){
      field(INPA, "signal.VAL")
      field(INPB,"sistema.VAL")
      field(INPC,"retardo.VAL")
      field(INPD,"coeficiente1.VAL")
      field(INPE,"coeficiente2.VAL")
      field(INPF,"coeficiente3.VAL")
      field(CALC,"A*(1/D)-(E/D)*B-(F/D)*C")
}
```

Donde INPD, INPE,INPF son los coeficientes de la ecuación en diferencias, y CALC es la ecuación en diferencias que antes hemos mostrado.

- Retardo

```
record(CALC,retardo){
      field(INPA,"sistema.VAL")
      field(CALC,"A")
      field(FLNK,"sistema")
}
```

- Coeficientes

```
record(CALC,coeficienteN){
      field(PINI,"YES")
      field(CALC,"1")
}
```

Este record se repite para todos los coeficientes, lo único en lo que cambia es el nombre del record.

Todo lo que se ha escrito se añade al archivo .txt creado antes en el directorio /EPICS_BASE/sistema/db.

## 3.4 Ejecución de la aplicación 'sistemaApp'

Una vez se ha definido la database con todo lo anteriormente escrito, es hora de ejecutar la aplicación para ver si ésta funciona correctamente. Para ello, en primer lugar se debe decir a EPICS que utilice la database que se ha definido , es decir, se accede al directorio iocBoot para modificar lo siguiente:

Se entra en la carpeta donde hemos creado la aplicación:

```
cd $EPICS_BASE/sistema
```

Se entra en el directorio iocBoot/iocsistema, en todas las aplicaciónes que se creen aparecerá el nombre de la aplicación añadida a ioc.

```
cd iocBoot/iocsistema
```

Ahora se edita el archivo st.cmd, que es el archivo que toma como referencia EPICS para iniciar la aplicación:

```
vi st.cmd
```

Dentro de este fichero se escribe siguiente:

```
dbLoadDatabase("../../dbd/sistema.dbd",0,0)

sistema_registerRecordDeviceDriver(pdbbase)

dbLoadRecords("../../db/sistema.txt")

iocInit()
```

Lo que hace este script es lo siguiente, con la función dbLoadDatabase() se cargan las definiciones de los databases que se van a utilizar, en este caso será el que crea por defecto EPICS, sistema.dbd. Lo siguiente que aparece es el registro de los mismos. dbLoadRecords() carga los records que se han definido para el softIOC, es decir lo que se ha escrito en todos los apartados anteriores. Como argumento se pasa la ubicación del archivo de texto donde están definidos los records, en nuestro caso '$EPICS_BASE/sistema/db/sistema.txt ' o bien con direcciones relativas, '../../db/sistema.txt'. Por último aparece la línea iocInit() que indica que se inicie el IOC.

Ahora que está definido el st.cmd, se inicia el IOC, de modo que se ejecuta lo siguiente:

```
../../sistema/bin/linux-x86/sistema st.cmd
```

Pasa lo siguiente. La aplicación "sistema" está en los ejecutables binarios en el directorio '/bin', y se le pasa como argumento el contenido del archivo st.cmd, este lo ejecuta paso a paso. Si todo está correctamente se obtendría lo que sigue:

```
#!../../bin/linux-x86/sistema2

## You may have to change sistema2 to something else

## everywhere it appears in this file

#< envPaths

## Register all support components

dbLoadDatabase("../../dbd/sistema.dbd",0,0)

sistema_registerRecordDeviceDriver(pdbbase)
```

```
## Load record instances

dbLoadRecords("../../db/sistema2.txt")

iocInit()

Starting iocInit

############################################################################
############

## EPICS R3.15.0.2 $Date: Tue 2014-10-07 01:31:33 -0500$

## EPICS Base built Oct  9 2014

############################################################################
############

iocRun: All initialization complete

## Start any sequence programs

#seq sncsistema2,"user=alain"

epics>
```

Donde se observa que se ha ejecutado todo lo contenido en el archivo sistema.txt. Para ver si se han cargado todos los records podemos utilizar el comando del iocsh dbl (database list), que muestra todos los records cargados:

```
epics> dbl

coeficiente1

coeficiente2

coeficiente3

periodo

retardo

signal

sistema

tiempo
```

## 3.5 Interfaz gráfica con CSS

Ahora nos interesa ver la evolución de la señal, para ello nos vamos a valer de Control System Studio (CSS) que es capaz de monitorizar a tiempo real PVs. Lo hace con la utilidad Data Browser. De lo cual obtenemos lo siguiente:

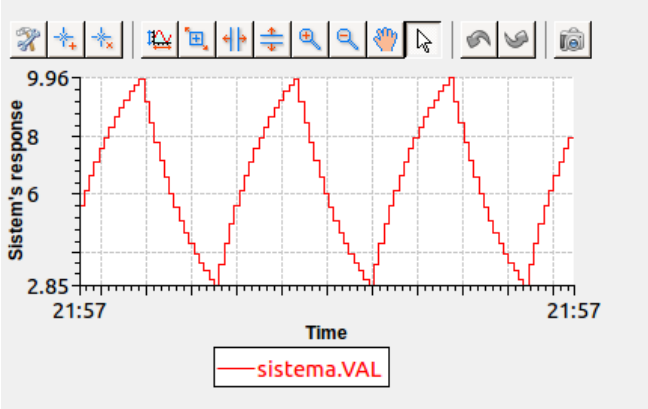



Figura 10:Respuesta Del sistema

Donde la imagen representa la respuesta del sistema ante una entrada de tren de pulsos con un periodo de 3s:

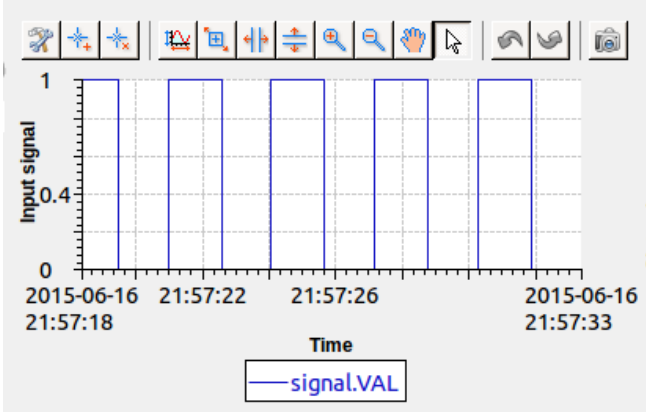



Figura 11: Señal de entrada del sistema

Ahora se crea un OPI (Operator Interface), que lo hemos hecho con la utilidad de CSS BOY (Best OPI YET). El panel de control que obtenemos es el siguiente:

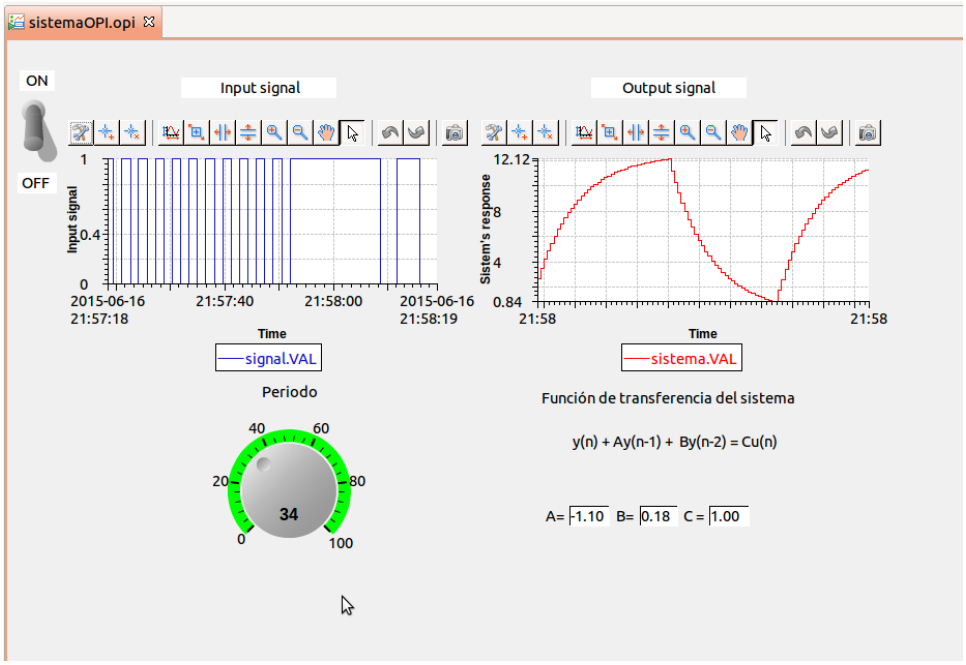



Figura 12: OPI del IOC implementado

Lo que se ha hecho, es crear controles tanto como para los coeficientes que defines el sistema (coeficientes de las ecuaciones en diferencias) y para el periodo de la señal de entrada. Para los coeficientes se ha elegido un control tipo texto, ya que de esta manera el usuario puede introducir de manera precisa los números que regirán el sistema. Por otra parte el control del periodo se ha llevado a cabo con un 'Knob', con precisión unidad. También se han creado 'ploters' para poder observar tanto la señal de entrada como la evolución de la respuesta del sistema. Por último se ha añadido al OPI un 'switch' que se encarga de ejecutar los comandos necesarios para poner en marcha el sistema.

# 4 Desarrollo de un IOC para control de motor Zaber T-HLA28-S

En esta sección se habla de la programación utilizada para controlar el motor Zaber T-HLA28-s facilitado por el CPLU de Salamanca. El lenguaje de programación utilizado será Python, no obstante se explicará lo que cada función hace de modo que sea independiente del lenguaje de programación. Más adelante se redacta la integración de el control creado en EPICS.

## 4.1 El motor Zaber modelo T-HLA28-S

El motor T-HLA28-S es un actuador lineal controlado por ordenador con una resolución de 0.1µm y un recorrido de 28mm. Requiere de una alimentación de 15V y se conecta mediante USB, utilizando el protocolo RS-232 a cualquier ordenador. Tiene la ventaja de poder conectarse en cadena con motores del mismo tipo [12].

Figura 13: Motor zaber T-HLA28-s

La configuración del protocolo RS-232 que utiliza es: 9600 bauds, No Hand Shaking, 8 data bits (byte), No parity, One Stop Bit [13]. El motor Zaber entiende una serie de comandos, que sirven para controlarlo (mover el motor a cierta posición, devolver la posición, mover el motor a una velocidad constante...). Todas estas instrucciones se envían en un grupo de 6 bytes con la siguiente estructura [14]:

- Byte 1: El número de motor al que queremos mandar la instrucción, para cuando se encuentren en cadena (0 todos, y de manera ascendente desde el más cercano al más lejano)
- Byte 2: El número de comando. El motor traduce el número que le enviemos a una instrucción [15]
- Byte 3,4,5,6: Data empezando por el byte menos significativo. La información que necesita el comando, como la posición cuando se ejecuta la instrucción de moverse a cierto punto.

**Ejemplo:** Imaginemos que queremos que el motor número 1 se desplace hasta la posición 231072. Sabiendo que el número de comando para desplazarse a cierta posición es el 20, y que

231072 son tres bytes (3,134,160), es decir, que al poner tres bytes contiguos crean el número 100000, 3→ 0000 0011, 134→ 1000 0110, 160→1010 0000 que al unirlos:

$$0000\ 0011\ 1000\ 0110\ 1010\ 0000_2 = 231072_{10}$$

De modo que los bytes que enviaríamos serían:

- Byte1 : 1 (0000 0001 en binario)
- byte 2: 20 (0001 0100)
- byte 3: 160 (1010 0000)
- byte 4: 134 (1000 0110)
- byte 5: 1 (0000 0001)
- byte 6: 0 (0000 0000)

*Todas las instrucciones que entiende el motor Zaber y su significado están la referencia [15] de la bibliografía.

## 4.2 Programación del control de motor en Python

Ahora que se conoce el funcionamiento, se desarrollará en Python un programa que sea capaz de enviar instrucciones al motor Zaber de la manera antes descrita. La idea es que un usuario cualquiera sabiendo que número de comando, el número de dispositivo y que datos se le quiere pasar, el progama sea capaz de traducir estos datos a un array de bytes que enviará al motor. El único inconveniente en esta conversión se presenta en los bytes de data, ya que en los demás la conversión es inmediata (ya que es un número de 0-255 y se puede escribir tal cual).

Para abordar el problema se troceará de manera que tengamos problemas más pequeños, primero se creará una función que pase los datos a binario, y luego uno que convierta estos números binarios en bytes.

### 4.2.1 Función de conversión datos-binario

En Python existe una función que lleva a cabo este proceso, bin(). Con el inconveniente de que devuelve el numero con un '0b' pegada delante:

bin(5)= '0b101'

De modo que se crea un programa que elimine esta b, y se quede con el resto:

```
def dataToBinary(data):
        binaryString = bin(data)
        newBinaryString = ""
        for i in binaryString:
                if(i == 'b'):
                        continue
                else:
                        newBinaryString += i
        return str(int(newBinaryString))
```

Figura 14:Código de la función

En esencia el programa recorre la string que devuelve bin() y cuando se ve que el carácter coincide con 'b', lo desecha.

### 4.2.2 Función conversión binario-bytes

Esta función tiene que tener en cuenta que los bytes que se van a transmitir siempre tienen que ser 4, de modo que por defecto la conversión dará como resultado un array de bytes con sólo ceros. El programa lo que hace es dividir el número binario que recibe en grupos de 8, y los pone en orden inverso, ya que como se ha visto antes, el byte menos significativo se envía el primero:

```
def binaryToByte(binary):
        list = [0,0,0,0]
        length = len(binary)
        counter = 0
        byteNumber = 0
        number = ""
        while(length > 0):
                if(counter == 8):
                        list[byteNumber] = number
                        counter = 0
                        byteNumber = byteNumber + 1
                        number = ""
                elif(byteNumber == 4):
                        break
                number =  binary[length-1] + number
                length = length - 1
                counter += 1
        list[byteNumber] = number
        return list
```

Figura 15:Código Python de la función

Una vez se tienen estas dos funciones, se crean otras que unen estas dos de forma que lo que se envíe sea justamente lo que el motor entiende. Para enviar los datos por el USB se utiliza la librería PySerial [6] para Python.

```
def byteListToCommandList(byteList):
        list = []
        for i in byteList:
                list.append(int(str(i),2))
        return list

def commandStructure(byte1=0,byte2=1,byte3=0,byte4=0,byte5=0,byte6=0):
        return chr(byte1)+chr(byte2)+chr(byte3)+chr(byte4)+chr(byte5)+chr(byte6)

def zaberCommand(deviceNumber,commandNumber,data):
        byteList = byteListToCommandList(binaryToByte(dataToBinary(data)))
        return commandStructure(deviceNumber,commandNumber,byteList[0],byteList[
```

Figura 16: Código para las funciones que unen las anteriores

### 4.2.3 programa principal

El programa principal consta de un loop infinito en el que se imprime un mensaje por pantalla pidiéndole al usuario que introduzca los datos necesarios para ejecutar una instrucción:

```
#programa principal
deviceNumber = 1
mySerial = serial.Serial("/dev/ttyUSB0")
while(True):
        deviceNumber = int(raw_input("Introduzca el número de dispositivo: "))
        commandNumber = int(raw_input("Introduzca el número de Comando: "))
        data = int(raw_input("Introduzca posicion absoluta: "))
        if(data == -1):
                homeCommand()
                mySerial.close()
                break
        else:
                zaberCommand(deviceNumber,commandNumber,data)
```

Figura 17: Código del programa principal

## 4.3 Conexión del motor a EPICS mediante AsynDriver

Una vez se ha desarrollado un programa que es capaz de traducir datos al protocolo de comunicación del motor Zaber, se utilizará éste para integrarlo en un sistema EPICS. Se va a utilizar la facilidad AsynDdriver para ello. Dado que el motor se comunica por el puerto USB con el protocolo RS-232, se elige la utilidad devGpibSerial de AsynDriver.

En la versión que se ha descargado de AsynDriver vienen varios ejemplos de las aplicaciones que se pueden hacer con esta utilidad. Entre ellos se encuentra uno, 'testGpibSerial' que se trata justamente de lo que se necesita.

Se va a modificar esta utilidad con el fin de acomodarlo a la aplicación. Si se observa en la carpeta /dev, al conectar el motor, aparece un nuevo archivo llamado 'ttyUSB0' que significa que un nuevo dispositivo de USB se ha conectado (el motor Zaber).

En primer lugar se accede al database de la aplicación ejemplo 'testGpibSerial'. En este archivo encontraremos un record asyn (llamado asynRecord), que es lo que representa el dispositivo. Se reconfigurará para que funcione con el motor T-HLA28-S. Para ello se tiene que tener en cuenta que la comunicación que utiliza es byte a byte, o sea binario. El asynRecord tiene varios fields para configurar. Uno de ellos es el OFMT (output format), de modo que se modifica éste para que sea binario:

```
field(OFMT,"Binary")
```

También se modifica el IFMT (input format) en binario, ya que el motor responde en binario.

```
field(IFMT,"Binary")
```

A continuación, se cambia la el archivo st.cmd del ejemplo testGpibSerial, ya que por defecto vine descrito para conectarse a un dispositivo 'ttyS0', cambiamos la línea de código donde se encuentra esta conexión por el nombre del motor Zaber:

```
drvAsynSerialPortConfigure("L0","/dev/ttyUSB0",0,0,0)
```

**Nombre del dispositivo Zaber según el sistema**

Ahora falta modificar los parámetros del protocolo RS-232, es decir la paridad, el Baud-Rate, el hand-shaking, el EOL y el stop bit. El EOL (end of line) se ha determinado experimentalmente que es '/n' los demás venían especificados por el dispositivo. Con la función asynSetOption() podemos configurar estos parámetros:

```
asynSetOption("L0", -1, "baud", "9600")
```

Para las demás configuraciones se cambia "baud" por el parámetro que queramos determinar ("parity","EOL"...).

Una vez se tienen modificados el archivo st.cmd, puede iniciarse la aplicación. Hay que tener en cuenta que el acceso a los dispositivos que se encuentran en la carpeta /dev, es privilegiado, es decir, necesitamos permisos de administrador para acceder a ellos, de modo que tenemos que ejecutar la aplicación EPICS como sudo (Superuser Do):

```
sudo ../../bin/$EPICS_HOST_ARCH/testGpibSerial st.cmd
```

## 4.4 Integración de Python en EPICS

Dado que se había desarrollado un programa en Python que controlaba el motor, se integrará éste en EPICS para aprovechar lo que ya tenemos hecho. Python dispone de una librería para utilizar éste como un CAC llamada PyEpics, de modo que puede acceder a las PVs que haya en la red, y aplicar lenguaje Python para modificarlas. La librería se puede descargar de la página web del desarrollador [16].

Para instalar la librería, se descarga el archivo .tar más reciente, se extrae y se ejecuta la siguiente línea de comando en una terminal dentro de la carpeta pyEpics:

```
python setup.py install
```

Ahora que se ha instalado la librería se debe modificar el programa Python anterior para que éste pueda ejecutarse desde EPICS.

La idea es la siguiente. Se implementan en la aplicación nuevos records que almacenen los datos necesarios para enviar una instrucción al motor (numero de comando, datos...), estos datos los recogerá el programa Python y se los enviará al motor mediante el asynRecord que se ha definido en la aplicación.

Para introducir los nuevos records se entra al database de la aplicación testGpibSerial y se añaden los records 'commandNumber', 'deviceNumber' y 'data', todos ellos serán records de tipo CALC, se escriben de la forma siguiente:

```
    record(CALC,"commandNumber"){
        field(PINI,"YES")
        field(CALC,"20")
    }
```

Los demás records serán idénticos.

Ahora se cambia el archivo Python de manera que reciba estos datos, se los asigne a variables , y después aplique la función zaberCommand que los transforma en una ristra de bytes comprensible por el motor. De modo que se añaden las siguientes líneas al programa:

```
import epics

[...]

commandNumber = epics.caget("commandNumber")
deviceNumber = epics.caget("devicenumber")
data = epics.caget("data")

sendBytes = zaberCommand(deviceNumber,commandNumber,data)

epics.caput("asynRecord",sendBytes)
```

Figura 18: Código del programa modificado

Dónde '[...]' significa que se incluye en ese espacio el código completo del programa anterior.

## 4.5 Interfaz gráfico con CSS

Por último para que la interacción con EPICS sea más cómoda se desarrolla un interfaz gráfico utilizando CSS. El interfaz contará con un botón que al oprimirlo permita enviar la información al motor, ciertos espacios para poder escribir el número de comando, numero de dispositivo y el dato, y contará también con una barra indicadora de la posición en la que se encuentra el motor, asimismo se creara una barra para introducir los datos de forma visual.

La idea es que cada uno de los widgets este asociado a una PV, y cuando se oprima el botón, el programa Python se ejecute y envíe la información al motor. El interfaz queda así:

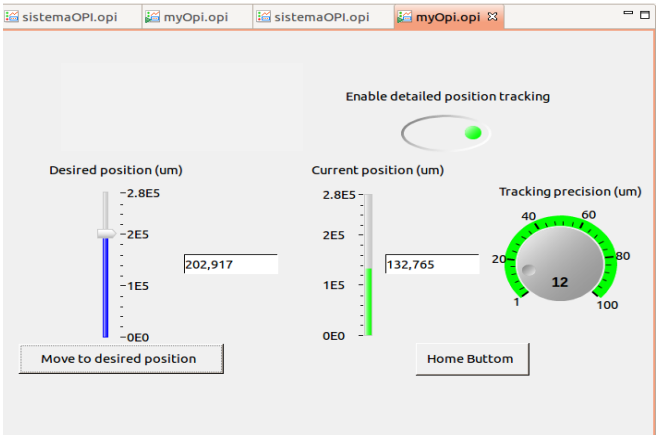



Figura 19: OPI del IOC del motor

En este OPI se ha optado por poner controles tipo 'bar', que no son más que barras que pueden ser desplazadas y indican el nivel de cierta variable. En nuestro caso, hemos utilizado este control para mover el motor, el usuario movería la barra de color azul hasta la posición deseada. Alternativamente, se ha añadido un control de tipo texto para cuando la precisión requerida en alguna aplicación sea mayor. Una vez la posición está fijada, se debe clicar el botón que dice 'Move to desired Position', al hacerlo CSS ejecutara los programas pertinentes para facilitar los datos que necesita EPICS para enviar el comando pertinente al motor. También se ha añadido un botón 'Homme Buttom' que al oprimirlo el motor vuelve a 'home' la posición 0.

Para monitorizar el motor, se ha utilizado un indicador de tipo 'bar', este indica la posición real del motor en cada momento siempre y cuando el botón booleano 'enable position tracking' este activado, de cualquier otro modo, indicaría la posición del motor una vez se haya ejecutado la acción. Se acompaña con un indicador de texto para una mayor precisión en la medida.

Por último cabe destacar el control tipo 'Knob' llamado 'tracking precision'. Hemos dicho que cuando el botón 'enable position tracking' este activado, la barra indicadora mostrará la posición real del motor en todo momento. Dado que para conseguir la posición del motor es necesario enviar comandos, de modo que las comunicaciones son mucho más lentas debido a la congestión de la línea de comunicación, se ha optado por añadir el control 'tracking precision' que permite elegir la precisión con la cual queremos que la posición se actualice en la pantalla. De esta manera, dejamos que el usuario decida, dependiendo de la situación que se encuentre, el compromiso entre velocidad de comunicación (y consecuentemente de desplazamiento) y precisión de la medida.

En la figura 19 se observa que el 'tracking position' esta activado, y el momento de la toma de la foto coincide con el momento en el que se ha enviado la información al motor (se ha oprimido el botón de 'Move to desired position') y el motor se está moviendo con la precisión establecida en el 'Knob', lo que se muestra en el indicador de color verde.

# 5 Desarrollo de un IOC para control de Cámara EO-5012C

Este ensayo trata del desarrollo del control necesario para la cámara de Edmun Optics EO-5012C facilitado por el CPLU de salamanca. En esta ocasión utilizaremos nuevamente el lenguaje de programación Python, también haremos uso de C ya que debemos implementar nuevas funciones en los archivos que trae por defecto AsynDriver. Lo que se pretende hacer es una aplicación en EPICS que sea capaz de tomar una foto y mandársela a un CAC cualquiera.

## 5.1 Cámara EO-5012C

La EO-5012C es una cámara de alta definición fabricada por Edmund Optics en Alemania. Cuenta con una resolución en píxeles de 2560x1920 (HxV) con una profundidad de color de 8 bits y un sensor tipo CMOS [17]. Se conecta al ordenador mediante USB, y es controlado mediante drivers facilitados por el fabricante.

Figura 20: Cámara EO-5012C

## 5.2 Instalación del driver de la cámara

Para que la cámara interactúe correctamente con el ordenador es necesario la instalación de los drivers. Los drivers se descargan del sitio web del desarrollador [18], se extrae el archivo, y como sudo se ejecuta lo siguiente:

```
sudo sh ./ueyesdk-setup*.sh
```

Una vez instalado, se ejecuta como un servicio, por lo que ejecutamos:

```
sudo /etc/init.d/ueyeethdrc start
```

Ahora se observa como el LED de la parte posterior de la cámara pasa de color rojo a verde:

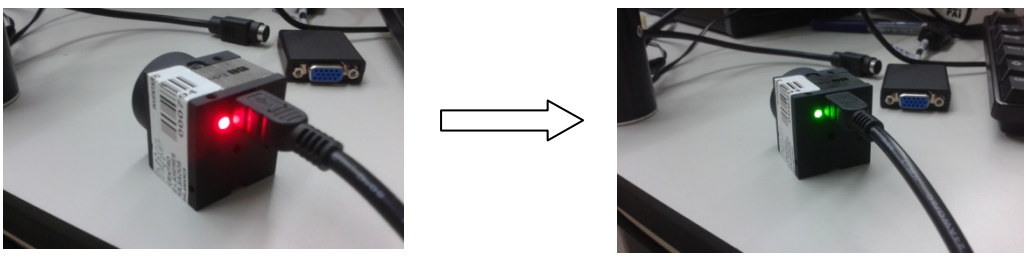


Figura 2: Cambio de color del LED

## 5.3 Conexión de la cámara a EPICS mediante AsynDriver

En esta ocasión no se dispone de ninguna utilidad asynDriver que solucione todos los problemas. Sin embargo está la opción asynPortDriver que es una clase de C++ en la utilidad AsynDriver que permite escribir drivers de forma más sencilla [19]. De modo que por todo lo anteriormente expuesto, elegimos asynPortDriver como software para el driver.

En los ejemplos de asynDriver, los desarrolladores crearon un driver para una especie de osciloscopio virtual. Aprovecharemos este ejemplo para no tener que redactar todo desde cero, se implementarán las funciones en este ejemplo y se integrarán a nuestro problema.

En primer lugar, se localizar las funciones (rutinas y subrutinas) del driver de la camara EO-5012C. Se trata de un archivo con las pseudo-definiciones de las funciones que utiliza el driver, lo que se conoce como header file. En nuestro caso se trata del archivo uEye.h y se encuentra en el mismo directorio donde teníamos instalado el driver de la cámara.

Una vez localizado el header, se empieza a modificar los archivos de la aplicación 'testAsynPortDriver'. Para ello se entra en carpeta /src, donde están los archivos fuente. y se presta especial atención en 'testAsynPortDriver.cpp'. En este se definen las funciones que se van a ejecutar cada vez que se llame a uno de los distintos tipos de 'dispositivos asyn' (asynInt32,asynFloat64,asynInt32Array ...). En nuestro caso se va a crear un record que se llame 'takePicture' que al ponerlo a uno, saque una foto. Como solo se va a utilizar 1 (para sacar la foto) y 0 (para estado de espera), lo mejor es que se utilice asynInt32 como dispositivo. Ahora tenemos que acudir a las librerías de la cámara EO-5012c para saber que funciones se deben ejecutar para poder tomar una foto. El algoritmo es el siguiente [20]:

1. Inicializar la cámara con is_InitCamera()
2. Seleccionar el formato con la función is_Format()
3. Guardar espacio en memoria con is_AllocImageMem()
4. Indicar dónde se sitúa la memoria almacenada, is_SetImageMem()
5. Obtener la foto, is_FreezeVideo()
6. Liberar el espacio de la cámara utilizado, is_FreeImageMem()

Para llevar a cabo estos pasos, se crea una función dentro de la utilidad de asynDriver asynPortDriver, lo que se conoce como método, ya que es una función dentro de una clase. Al método lo llamaremos como el record, takePicture(). No tendrá argumentos ya que no son necesarios para el sencillo ejemplo desarrollado. Lo que se hará dentro de ella, será cerciorarse de que esta función ha sido la que se ha ejecutado, luego llevar a cabo los pasos 1,2,3,4 y 5 del algoritmo, más adelante copiar la memoria de la foto, que está guardada en la cámara en un array definido en un record de EPICS. De esta manera, se obtendrá un array de datos accesible desde cualquier punto de la red, con lo que se estará 'compartiendo' la imagen que se ha tomado con los usuarios de la red.

La función takePicture queda como sigue:

```
void testAsynPortDriver::takePicture()

{

          int* pid = (int *)malloc(sizeof(int));

          char** ppcImgMem = (char **)malloc(sizeof(char));

          UINT formatId = 13;

          UINT* pParam = &formatId;

          HIDS hCam = 0;

          is_SetErrorReport (hCam, IS_ENABLE_ERR_REP);

          int respuesta = is_InitCamera(&hCam,NULL);

          if(respuesta != IS_SUCCESS){

                printf("Ha habido un fallo al iniciar la
camara, error %d\n",respuesta);

          }

          is_ImageFormat(hCam,IMGFRMT_CMD_SET_FORMAT,pParam,4);

          is_AllocImageMem (hCam, width, height, bitspixel,
ppcImgMem,  pid);

          is_SetImageMem (hCam, *ppcImgMem, *pid);

          is_FreezeVideo(hCam,IS_WAIT);

          is_SetExternalTrigger(hCam, IS_SET_TRIGGER_OFF);


          memcpy(pData_, *ppcImgMem, 640*480*3);


          callParamCallbacks();

          doCallbacksFloat64Array(pData_,640*480*3,  P_Waveform,
0);

          IMAGE_FILE_PARAMS ImageFileParams;

          ImageFileParams.pwchFileName = NULL;
```

```
            ImageFileParams.pnImageID = NULL;
            ImageFileParams.ppcImageMem = NULL;
            ImageFileParams.nQuality = 0;

      ImageFileParams.pwchFileName=L"/home/alain/Desktop/foto.bmp
";
            ImageFileParams.nFileType = IS_IMG_BMP;

      is_ImageFile(hCam,IS_IMAGE_FILE_CMD_SAVE,(void*)&ImageFileP
arams,sizeof(ImageFileParams));
            setIntegerParam(P_TakePictureNow,0);
            is_FreeImageMem(hCam,*ppcImgMem,*pid);
            is_ExitCamera (hCam);
}
```

Como se puede observar hemos utilizado la función memcpy() para copiar la memoria de la cámara en un array. Para meter los datos en el array hemos utilizado la función de asyn 'doCallbacksFloat64Array()' pasándole como argumento un puntero que indica la dirección de memoria en la cámara, el número de elementos (la resolución de la imagen) y el array de asynFloat64 al que queremos que se vuelquen los datos.

Una vez hecho esto, ahora se tiene que hacer que el método se ejecute cuando el valor del record 'takePicture' cambie. Para ello se modifica la parte del fichero .cpp donde se encuentra el método 'writeInt32' que es donde se dirige asynDriver cuando se ha modificado un record que tiene un dispositivo asociado del tipo asynInt32. EL código de éste método es:

```
asynStatus  testAsynPortDriver::writeInt32(asynUser  *pasynUser,
epicsInt32 value)
{
    int function = pasynUser->reason;
    asynStatus status = asynSuccess;
    const char *paramName;
```

```
    const char* functionName = "writeInt32";

    printf("\nLa funcion P_TakePictureNow es %d, y la funcion
que se ha ejecutado, %d\n",P_TakePictureNow,function);


    /* Set the parameter in the parameter library. */

    status = (asynStatus) setIntegerParam(function, value);


    /* Fetch the parameter string name for possible use in
debugging */

    getParamName(function, &paramName);


    if (function == P_Run) {

        /* If run was set then wake up the simulation task */

        if (value) epicsEventSignal(eventId_);

    }

    else if (function == P_VertGainSelect) {

        setVertGain();

    }

    else if (function == P_VoltsPerDivSelect) {

        setVoltsPerDiv();


    }

    else if (function == P_TimePerDivSelect) {

        setTimePerDiv();

    }else if(function == P_TakePictureNow){

           takePicture();

    }
```

Como se observa se ha añadido una sentencia condicional, en la que sí 'function' es iguala al apalabra clave 'P_TakePictureNow', se ejecutara el bloque dentro de la sentencia. 'Function' es una constante que hace referencia a lo que un cliente asyn ha escrito.

## 5.4 Uso de Python para Trascribir los datos en EPICS a una Imagen

Por último, ahora que se ha conseguido 'compartir' los datos mediante EPICS, en un array de datos del tipo epicsFloat64, se debe crear un programa que sea capaz de transformar los éstos datos en una imagen.

El modo de toma de imagen que se había escogido tenía un fin. Este modo era VGA con 8 bits de profundidad de imagen, lo que se traduce en que cada uno de los pixeles utiliza tres tipos de colores (Rojo, Verde y Azul, RGB en inglés) y asigna a cada uno de ellos ocho bits, o un byte. Es decir, si queremos representar un color cualquiera, éste se puede descomponer como suma de los colores primarios, rojo verde y azul. De modo que si ajustamos la intensidad de cada color y los sumamos podemos obtener el color de partida. A esto se le conoce como síntesis aditiva [21].

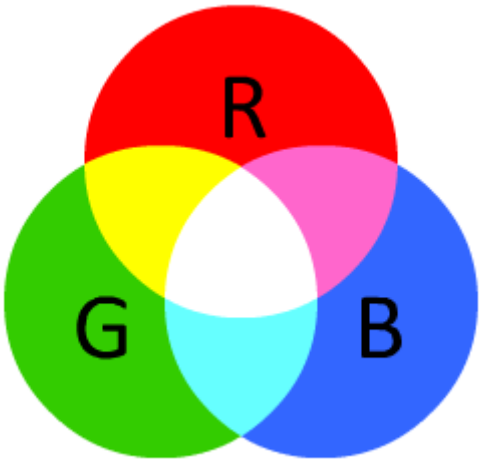



Figura 21: Síntesis aditiva RGB

La intensidad viene determinada por el byte asignado a cada color, de modo que disponemos de 255 niveles de intensidad por cada color (lo que se conoce como 8 bits de profundidad), ya que asignamos un byte o 8 bits a cada uno de ellos.

Ahora que se sabe lo que es el modo VGA con 8 bits de profundidad, se debe relacionar esto con el tipo de datos epicsFloat64. Este tipo de dato significa que se utilizan 64 bits para representar un número de la siguiente manera:

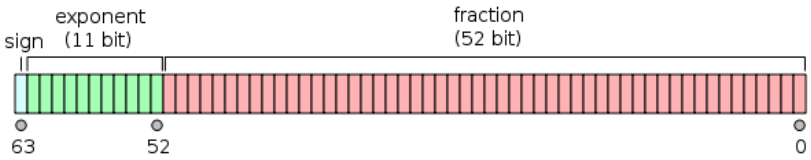



Figura 22: Representación de un número en epicsFloat64

Dónde el número se expresa como:

$$N = (-1)^{sign}\left(1 + \sum_{i=1}^{52} b_{52-i}2^{-i}\right)2^{exp-1023}$$

Dónde b es el peso asociado a cada bit de fracción, exp es el exponente en código exceso a 1023 (es decir que el 0 se representa con 1023, el 1 como 1022 , el -1023 como 0 y así sucesivamente), y sign representa el bit 63 [22].

Como se observa, para representar este número se utilizan 64 bits, o bien 8 bytes. De modo que si se pone atención, se tiene que un dato de este tipo es equivalente a dos pixeles y medio en 'formato imagen', ya que cada pixel dispone de tres colores y le asigna un byte a cada uno, de modo que tendríamos tres bytes por pixel.

Para llevar a cabo esta conversión con Python, se ha buscado una librería que permita trabajar con imágenes, PIL [23]. Para instalarlo se procede como siempre, se descarga el archivo .tar, se descomprime y se ejecuta el 'setup.py install' como sudo. La librería viene con una clase llamada Image que permite crear una imagen. La idea para este programa es obtener el array de datos epicsFloat64 de EPICS con la librería pyEpics, convertir cada uno de los epicsFloat64 a tres bytes y luego introducir estos tres bytes en un pixel de la imagen que vamos a construir.

Para transformar los epicsFloat64 en bytes, se va a utilizar una clase de Python llamada struct, que dispone de un método llamado pack, que es capaz de transformar un número de cualquier formato en su representación en bytes:

```
>>> struct.pack('i',12)
b'\x0c\x00\x00\x00'
```

Ahora, lo que se hace es ir convirtiendo estos bytes a números en hexadecimal, éstos a su vez en caracteres que se irán concatenando para crear una string de caracteres hexadecimales. De esta manera se pueden 'desconvertir' cogiendo grupos de seis en seis y asignando a cada color de cada pixel dos dígitos hexadecimales (que equivalen a un byte).

Los datos que recoge el array definido en EPICS en formato epicsFloat64, tienen la siguiente forma:

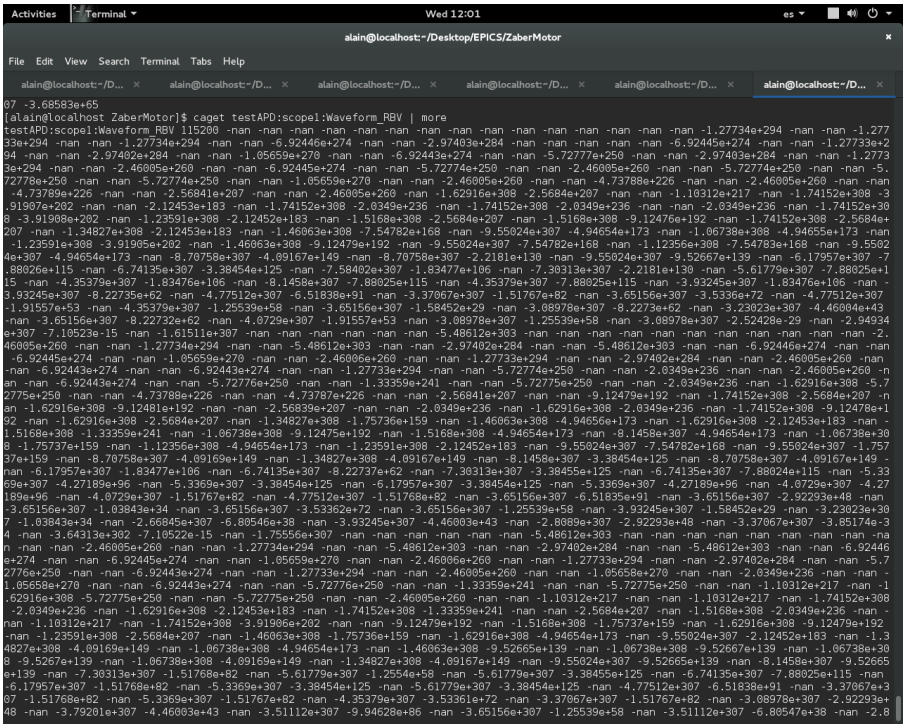



Figura 23: Array de 'Raw data' en formato epicsFloat64

Hay que tener en cuenta que cada uno de estos datos representa los colores de "dos pixeles y medio" por lo que es normal que salgan números poco comunes y exorbitados. También debe quedar claro que estos números representan colores y no valores numéricos.

Todos estos datos equivalen a esta imagen:

Figura 24: Imagen equivalente

La imagen es de tonos rosados, debido a que la cámara carece de objetivo, por lo que recibe luz de todas direcciones y no es capaz de focalizar la imagen a una sola región del entorno. En cualquier caso, la conversión de datos es siempre la misma, ya que sólo depende de los datos que facilite la cámara, no del proceso.

## 5.5 Interfaz gráfica con CSS

Ahora se crea una interfaz amigable para el usuario, una muy simple que disponga de un botón que al presionarse saque la foto y la cargue en nuestro ordenador. Queda lo siguiente:

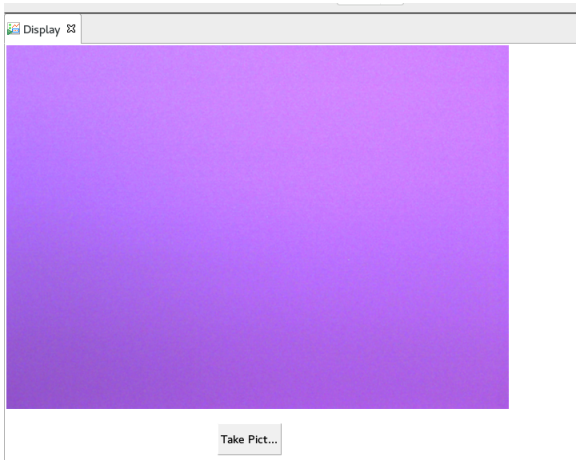



Figura 25: Interfaz gráfica

Como ya se ha mencionado, la interfaz dispone de un botón, al ser oprimido la imagen que se tome aparecerá en la pantalla (zona de color morado en la figura 25).

# 6 Conclusiones

En este proyecto hemos utilizado EPICS para llevar a cabo el control remoto tanto de un motor como de una cámara. Hemos cumplido los objetivos establecidos al crear un sistema capaz de gestionar la posición de un motor y la obtención mediante una cámara de una imagen.

Este proyecto sirve como una iniciativa para desarrollar mayores proyectos, en nuestro caso hemos creado un sistema suficientemente pequeño como para poder prescindir de EPICS. Sin embargo el uso de éste para este tipo de sistemas, es suficiente como para comprender a grosso modo la mayor parte de las funcionalidades que ofrece el conjunto de rutinas para sistemas distribuidos EPICS. No obstante, el uso de EPICS para este proyecto tiene un objetivo ya que se trata de una colaboración con el CLPU de Salamanca, y en un futuro se podría utilizar lo aprendido en este proyecto para implementarse en el centro. Además el escalado del sistema creado se torna muy sencillo.

Como se ha expuesto en el capitulo donde se describía el proceso llevado a cabo en el control de la cámara, se ha desarrollado un sencillo driver con la utilidad de asynDriver. De modo que se podría continuar con el desarrollo para añadir generalidad al driver, y de esta manera cualquier desarrollador de EPICS podría acceder a este driver y no tendría que rescribirlo nuevamente desde cero.

En este proyecto han surgido varios problemas que han sido solventados para poder llevar a cabo el trabajo. En primer problema, antes de trabajar con EPICS, ha sido tener que familiarizarse a administrar un sistema linux por la terminal mediante lenguaje bash.

En cuanto a la parte del motor, han surgido algunos problemas inesperados. En primer lugar se ha tenido que determinar experimentalmente el EOL del protocolo de comunicaciones del motor. El segundo problema ha sido al acceder desde un sistema linux al motor. En linux el acceso a un dispositivo es idéntico que al de un archivo. Al ejecutar la aplicación de EPICS que se conectaba al motor, no mostraba ningún tipo de error. Sin embargo al enviar comandos no el motor no respondía. El problema era que no se habían concedido los privilegios necesarios para el acceso a ese tipo de archivos.

En la parte de la cámara también han surgido problemas. En primer lugar, ha sido necesario familiarizarse con los lenguajes C y C++ para el desarrollo de los drivers de la utilidad AsynDriver. Después se ha tenido que determinar de manera experimental el orden que tenían los datos en el array de EPICS del tipo epicsFloat64.

Como ampliación del proyecto, se plantea la unión de los dos desarrollos que se han llevado a cabo en este proyecto, es decir, utilizar el sistema de control de la cámara que se ha desarrollado, y integrarlo con el sistema del motor. De esta manera se podría monitorizar la posición del motor de manera visual. De modo que si hubiese algún tipo de fallo, podría observarse gracias a la cámara.

# 7 Bibliografía

[1] CPLU,"CPLU home page" [en línea], Enlace: http://www.clpu.es/

[2] Alejandro S. León O., "Sistemas de Control Distribuido (DCS)" [en línea], Enlace: http://es.slideshare.net/alleonchile/sistemas-de-control-distribuido-dcs-7298975

[3] M. Knott D. Gurdt, "EPICS: A Control System Co-Development Success" [en línea], Enlace: http://www.aps.anl.gov/epics/EpicsDocumentation/EpicsGeneral/epics_success.html

[4] Bob Dalesio, "EPICS Overview" [en línea], Enlace: http://www.slac.stanford.edu/comp/unix/package/epics/training/documents/01_EPICS_Overview.pdf

[5] Argonne National Laboratory. "EPICS home page" [en línea]. Enlace: http://www.aps.anl.gov /epics/

[6] Martin R. Kraimer, Janet B. Anderson, Andrew N. Johnson, W. Eric Norum, Jeffrey O. Hill , Ralph Lange, Benjamin Franksen, Peter Denison. "EPICS Aplication Developer´s Guide". Release 3.14.12. 2015. Enlace: http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide.pdf

[7][8] Jeffrey O. Hill, "EPICS: Channel Access Reference Manual" [en línea], Enlace: http://www.aps.anl.gov/epics/base/R3-14/8-docs/CAref.html

[9] Argonne National Laboratory. "Getting started with EPICS" [en línea]. Enlace: http://www.aps.anl.gov/epics/docs/GSWE.php

[10] EPICS community. "EPICS Record Reference Manual" [en línea]. Enlace: https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14

[11] Mark Rives, Eric Norum, Marty Kraimer, "AsynDriver", 2015, Disponible en: http://www.aps.anl.gov/epics/modules/soft/asyn/R4-26/asynDriver.pdf

[12] DESY. "CSS home page" [en línea] Enlace: http://controlsystemstudio.org/

[13][14][15] Zaber company. "Zaber T-HLA28-S wiki" [en línea]. Enlace: http://www.zaber.com/wiki /Manuals/T-LA

[16] Epics comunnity. "PyEpics support" [en línea]. Enlace: http://cars9.uchicago.edu/software/python/pyepics3/

[17] Edmund Optics. "EO-5012C camera webpage" [en línea].Enlace: http://www.edmundoptics.com/cameras/ usb-cameras/ eo-usb-2-0-cmos-machine-vision-cameras/59368/

[18] Ids-imaging. "uEye Industrial cameras" [en línea]. Enlace: https://es.ids-imaging.com/download-ueye.html

[19] MarkRivers, "C++ base Class for AsynPortDriver" [en Línea], Enlace: http://www.aps.anl.gov/epics/modules/soft/asyn/R4-12/asynPortDriver.html

[20] Ids-imaging. "uEye Industrial camera Manual" [en línea]. Enlace: https://es.ids-imaging.com/download-ueye.html

[21] Wikipedia, "RGB Color model" [en línea], Enlace: https://en.wikipedia.org/wiki/RGB_color_model

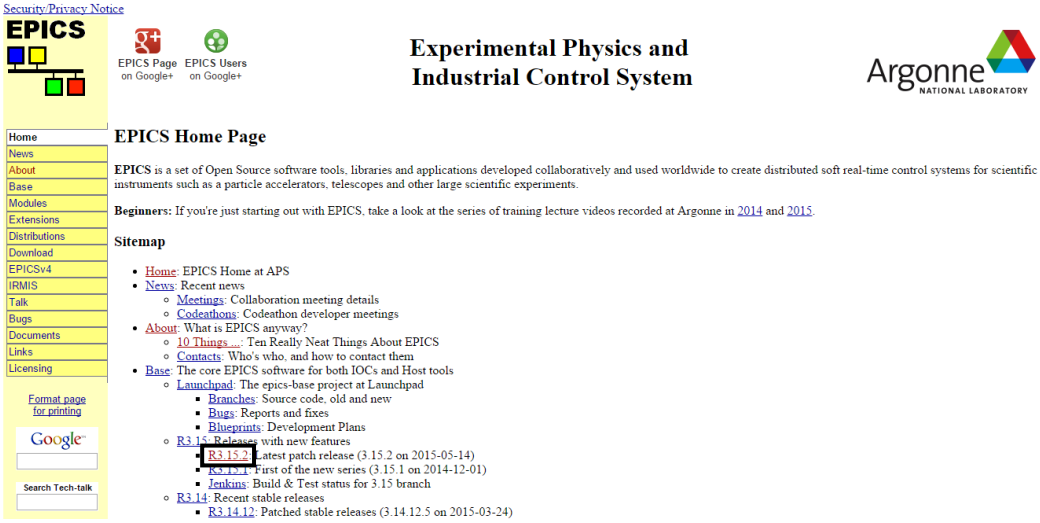
[22] Wikipedia, "Double Precission floating point format" [en línea], Enlace: https://en.wikipedia.org/wiki/Double-precision_floating-point_format

[23] Pyserial. "PySerial documentation" [en línea]. Enlace: http://pyserial.sourceforge.net/

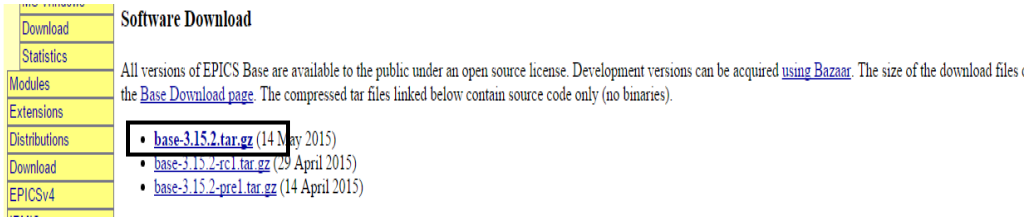
# 8 Apéndice: Instalación de EPICS en un entorno Linux

En este apéndice se muestran los pasos para instalar correctamente EPICS en un ordenador que ejecute una distribución Linux paso a paso.

En primer lugar debemos dirigirnos a la página oficial de EPICS http://www.aps.anl.gov/epics/, y nos pinchamos en apartado donde se publica la última versión de EPICS disponible:




Una vez aquí vamos a la sección de 'software downloads' y clickamos en la versión base más reciente:




Esperamos a que se descargue. Una vez la versión base está en nuestro equipo debemos crear una carpeta para que contenga EPICS, para ello abrimos una terminal y tecleamos lo que sigue:

```
mkdir EPICS
```

Y ahora entramos en la carpeta, y copiamos el archivo .tar en este directorio:

```
cd EPICS
```

```
cp /directorioBase/base-3.x.x.tar ./
```

Donde directorioBase es el directorio donde se encuentra el archivo .tar descargado y 3.x.x es la versión que hemos descargado. Ahora descomprimimos el fichero:

```
tar -xvf base-3.x.x.tar
```

Nos introducimos dentro del directorio que se ha creado y ejecutamos make.

```
cd base-3.x.x
```

```
make
```

Ahora que hemos instalado EPICS en nuestro sistema, sería conveniente introducir el directorio de los ejecutable binarios en los Paths de búsqueda de nuestro sistema, para ello, cambiaremos la variable de entorno PATH añadiéndole al final el directorio dónde se encuentran los ejecutables:

```
export PATH=PATH:/EPICS/bin/$EPICS_HOST_ARCH
```

Dónde EPICS es el directorio donde hemos instalado EPICS, y EPICS_HOST_ARCH es la variable de entorno que define EPICS para saber cuál es la arquitectura del sistema dónde se está ejecutando. Para que no tengamos que repetir el proceso de exportar la nueva variable PATH cada vez que abrimos una terminal, lo más correcto sería copiar la anterior línea de código en el archivo .bashrc que se encuentra en el directorio /home/$USER_NAME.