UNIVERSIDAD POLITÉCNICA DE VALENCIA

TRABAJO FINAL DE MÁSTER

MÁSTER EN AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

INTEGRACIÓN DE INFORMACIÓN DE VISIÓN EN SERVIDOR OPC-UA

Autor:

David Castillo Sáez

Dirección:

Francisco Blanes Noguera

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN **1**

1.1 MOTIVACIÓN DEL TRABAJO 1
1.2 OBJETIVOS 2

HARDWARE UTILIZADO **3**

2.1 ODROID XUA4 3
2.2 CÁMARA UEYE XS 6

SOFTWARE Y TECNOLOGÍAS IMPLEMENTADAS **9**

3.1 INTRODUCCIÓN A LA TECNOLOGÍA OPC 9
3.2 CLASIC OPC 10
3.3 EVOLUCIÓN A OPC-UA 14
3.4 CONFIGURACIÓN DE LA CONEXIÓN 17

DESARROLLO E IMPLEMENTACIÓN **19**

4.1 PLANTEAMIENTO DEL PROBLEMA 19
4.2 ADQUISICIÓN DE IMÁGENES 21
4.3 ALGORITMO DE RECONOCIMIENTO DE IMÁGENES 27
4.4 IMPLEMENTACIÓN DEL SERVIDOR OPCUA 35
4.5 IMPLEMENTACIÓN DEL CLIENTE OPCUA 45

RESULTADOS **49**

CONCLUSIONES **53**

ANEXO INSTALACIÓN ..... ¡ERROR! MARCADOR NO DEFINIDO.

INSTALACIÓN DEL ENTORNO (LINUX EMPOTRADO Y LIBRERÍAS) ..... **¡Error! Marcador no definido.**
INSTALACIÓN DE OPENCV ..... **¡Error! Marcador no definido.**
INSTALACIÓN DE UEYE XS 2 ..... **¡Error! Marcador no definido.**
INSTALACIÓN DE OPCUA ..... **¡Error! Marcador no definido.**
CÓDIGOS ..... **¡Error! Marcador no definido.**

BIBLIOGRAFÍA **55**

ÍNDICE DE FIGURAS

Figura 2. 1 OdroidXU4 3
Figura 2. 2 Conjunto de componentes en la OdroidXU4 4

Figura 3. 1 Módulo de acceso a datos (DA) 10
Figura 3. 2 Objetos creados por el cliente OPC 11
Figura 3. 3 Subscripción del cliente OPC al servidor 11
Figura 3. 4 Interconexión de elementos que forman el servidor OPC 12
Figura 3. 5 Niveles de implementación del OPC 13
Figura 3. 6 Características principales del OPCUA 14
Figura 3. 7 Capas que forman parte del OPCUA 15
Figura 3. 8 Distintas clases de nodos en un servidor OPCUA 16
Figura 3. 9 Contexto de suscripción a cambios y eventos 17

Figura 4. 1 Superficie de trabajo 22
Figura 4. 2 Código Camera.h 22
Figura 4. 3 Código Camera.cpp 23
Figura 4. 4 Clase que implementa la cámara UEYE XS 24
Figura 4. 5 Función que inicializa la cámara 25
Figura 4. 6 Ejemplo de imagen con fondo blanco 26
Figura 4. 7 Ejemplo de imagen con fondo azul oscuro 26
Figura 4. 8 Definición de variables y métodos de la clase 27
Figura 4. 9 Imagen obtenida con la cámara 28
Figura 4. 10 Imagen transformada a escala de grises 29
Figura 4. 11 Implementación del algoritmo de detección 29
Figura 4. 12 Algoritmo de Canny para detección de bordes 30
Figura 4. 13 Detección del número de monedas 30
Figura 4. 14 Bucle para cálculo de centros y radios 32
Figura 4. 15 Contornos dibujados de las monedas detectadas 32
Figura 4. 16 Resultado del algoritmo implementado 32
Figura 4. 17 Discriminación de monedas por radio 33
Figura 4. 18 Dibujo del contorno de círculos 34
Figura 4. 19 Código main_server.cpp 35
Figura 4. 20 Función de implementación del hilo 36
Figura 4. 21 Valores monetarios y constantes 37
Figura 4. 22 Variables públicas de la clase servidor 37

Figura 4. 23 Métodos públicos de la clase servidor 38
Figura 4. 24 Definición de constantes en la clase servidor 38
Figura 4. 25 Método para añadir tags al servidor 39
Figura 4. 26 Declaración de los contenedores de tags 39
Figura 4. 27 Método de asignación de tags 40
Figura 4. 28 Borrado de los contenedores 40
Figura 4. 29 Esquema básico de conexión de un LED 42
Figura 4. 30 Montaje del LED con transistor en un GPIO 42
Figura 4. 31 Tabla de pines de la OdroidXU4 43
Figura 4. 32 Activación de los GPIOs 44
Figura 4. 33 Código del main del cliente 45
Figura 4. 34 Bucle de ejecución del cliente 46
Figura 4. 35 Lectura de los tags del servidor por parte del cliente 46
Figura 4. 36 Reconexión del cliente 47

Figura 5. 1 Terminal de implementación del cliente OPC-UA 49
Figura 5. 2 Inicio del proceso del Servidor OPC-UA 50
Figura 5. 3 Adquisición y Reconocimiento de monedas 50
Figura 5. 4 Cancelación del cliente con la orden signal 51

Capítulo 1

INTRODUCCIÓN

1.1 MOTIVACIÓN DEL TRABAJO

Los sistemas de visión están cada vez más implantados en la sociedad actual en multitud de campos de trabajo: sistemas de vigilancia, sector de la robótica en funciones de pick and place, posicionamiento de objetos, contaje de productos o control de calidad de procesos industriales entre otros. Es en este último ámbito donde se encuentra enmarcado este trabajo fin de máster.

Los sistemas de fabricación de las empresas industriales son cada vez más exigentes con la calidad y acabado que muestras sus productos. Así mismo, hoy en día existen normas de estandarización en cuanto al material, tamaño y forma que deben tener los objetos que se fabrican, por lo que las empresas están implantando las tecnologías más punteras en el mercado para conseguir estos objetivos.

Por otra parte, tecnologías como el Big Data o el Internet de las cosas hacen que las empresas tengan toda la información relevante de cómo evoluciona el proceso de fabricación, tanto a nivel de etapas como a nivel de jerarquía. Hoy en día es fundamental conocer los valores de los dispositivos que se encuentran a nivel más bajo como sensores para poder garantizar un rendimiento máximo, así como poder detectar anomalías en el funcionamiento de equipos que conlleven a un fallo en un nivel superior del proceso.

Es por ello que este trabajo fin de máster tenga como finalidad abarcar este propósito, integrando un sistema de visión de información gobernado por un protocolo de comunicaciones utilizado actualmente a nivel industrial como OPCUA.

1.2 OBJETIVOS

El objetivo de este trabajo fin de máster es el desarrollo de un servidor OPC-UA con capacidad de tratamiento y gestión de datos vinculado a sistemas de inspección mediante visión artificial. El trabajo se encuentra dividido en varias etapas, constituyendo cada una de ellas una parte fundamental en la integración del proceso completo.

- Desarrollo de una plataforma en Linux empotrado basado en una OdroidXU4 como el elemento de hardware principal.
- Utilización de una cámara industrial ids UEYE XS como elemento de adquisición en el proceso de visión artificial. Este elemento se encarga de capturar las imágenes que serán tratadas posteriormente.
- Desarrollo de un algoritmo de reconocimiento basado en visión artificial utilizando el software OpenCV. Dicho algoritmo se encargará de realizar una clasificación de los objetos detectados.
- Desarrollo e implementación de un cliente y servidor OPC-UA con capacidad de gestionar el envío, recepción y la visualización de los datos. Tanto cliente como servidor se regirán por las características propias de este protocolo de comunicaciones.

Capítulo 2

HARDWARE UTILIZADO

En la realización del trabajo fin de máster se han utilizado varios elementos que componen el hardware del proyecto. Estos permiten la implementación de un servidor OPC-UA a través del código programado para cada uno de ellos. Los elementos hardware principales que componen dicho desarrollo son la ODROIDXUA y la cámara UEYE XS. A continuación, se describen cada uno de ellos.

2.1 ODROID XUA4

La placa Odroid XU4 es un sistema embebido muy versátil, capaz de utilizarse en tareas como decodificadores, propósito general web, prototipo de retoques de hardware, controlador para la domótica, estación de desarrollo software, etc. El sistema operativo más utilizado en este dispositivo es el Ubuntu, tal y como se ha hecho para este proyecto. Sin embargo, existen otros sistemas operativos compatibles con este dispositivo como Android, Fedora, Debian o OpenELEC.

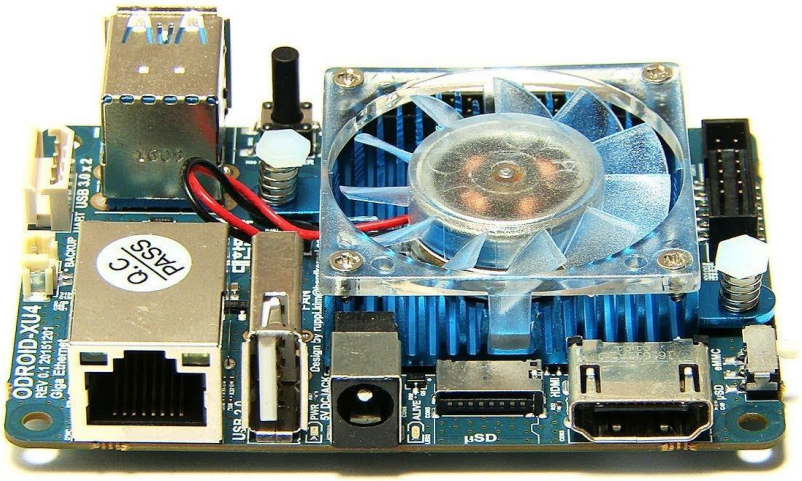


Figura 2. 1 OdroidXU4

El Odroid-XU4 es una placa con arquitectura ARM, una de las arquitecturas más utilizadas para dispositivos móviles e integrados de 32 bits. Existen algunas diferencias entre los ordenadores de uso común y un dispositivo ARM, algunas de ellas son:

- La velocidad del procesador ARM no es comparable con un procesador Intel.
- Debido a la eficiencia de la CPU, el XU4 puede ofrecer un gran tiempo de respuesta tan rápido como un ordenador potente.
- Los sistemas operativos disponibles para XU4 están muy optimizados, ya que se benefician de contribuyentes de código abierto. Los autores mantienen un repositorio de GitHub.

Odroid-XU4 contiene muchas de las conexiones físicas de un ordenador. Posee 2 puertos USB 3.0, un puerto USB 2.0, una conexión Ethernet con velocidades de transferencia de Gigabit, un conector HDMI para monitores de 720p y 1080p, un conector de alimentación de corriente continua de 5V/4A. Además, la Odroid XU4 contiene un GPIO de 42 pines, un conector RTCC externo, conector de módulo eMMC y una ranura dedicada para la tarjeta microSD.

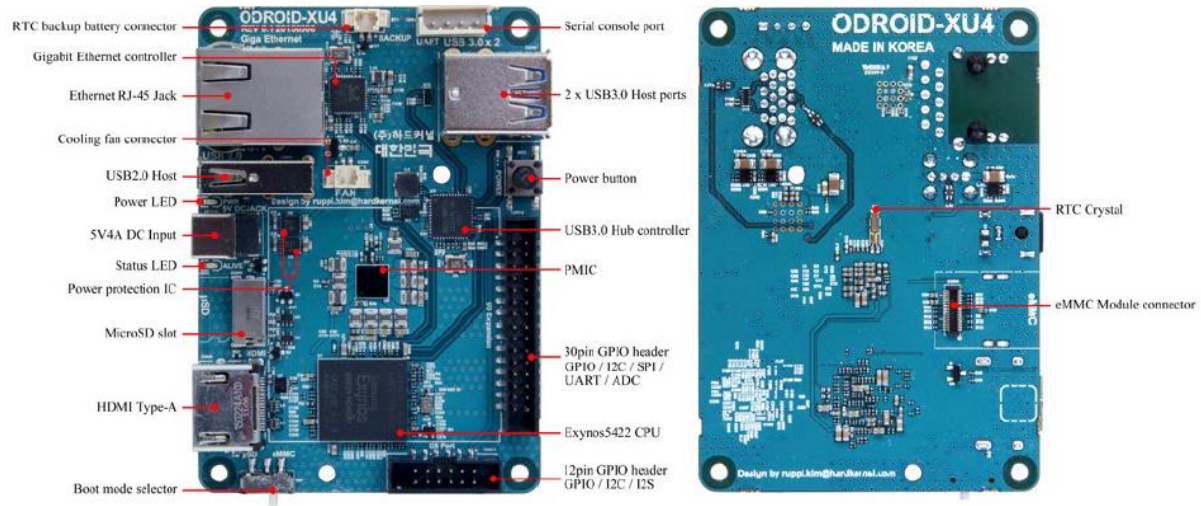


Figura 2. 2 Conjunto de componentes en la OdroidXU4

La conexión de la placa con internet se ha realizado mediante Ethernet, situándose cerca del router. No obstante, algunas pruebas del laboratorio se han realizado utilizando el adaptador Wifi mediante USB.

Los puertos USB se han utilizado para conectar el teclado, el ratón y la cámara UEYE XS, descrita en el siguiente apartado.

Tal y como se ha mencionado anteriormente, la OdroidXU4 posee dos headers de GPIOS, uno de 30 pines y otro de 12. El primero de ellos puede ser utilizado como GPIO, IRQ, SPI o ADC, mientras que el segundo se utiliza como GPIO, I2S o I2C. A la hora de utilizar los pines como GPIO hay que tener en cuenta que se utilizan a 1.8V, así como los ADC tienen una entrada limitada a 1.8V. Sin embargo, si algún sensor necesita un voltaje más elevado, los puertos GPIO pueden elevar su nivel de voltaje hasta los 3.3V o 5V.

En cuanto a las formas de trabajo que se pueden dar con esta placa, destacan las mostradas en la siguiente figura. Concretamente, para la realización del proyecto se ha implementado la forma 1 (izquierda), ya que resultaba más cómodo según el entorno de trabajo doméstico.

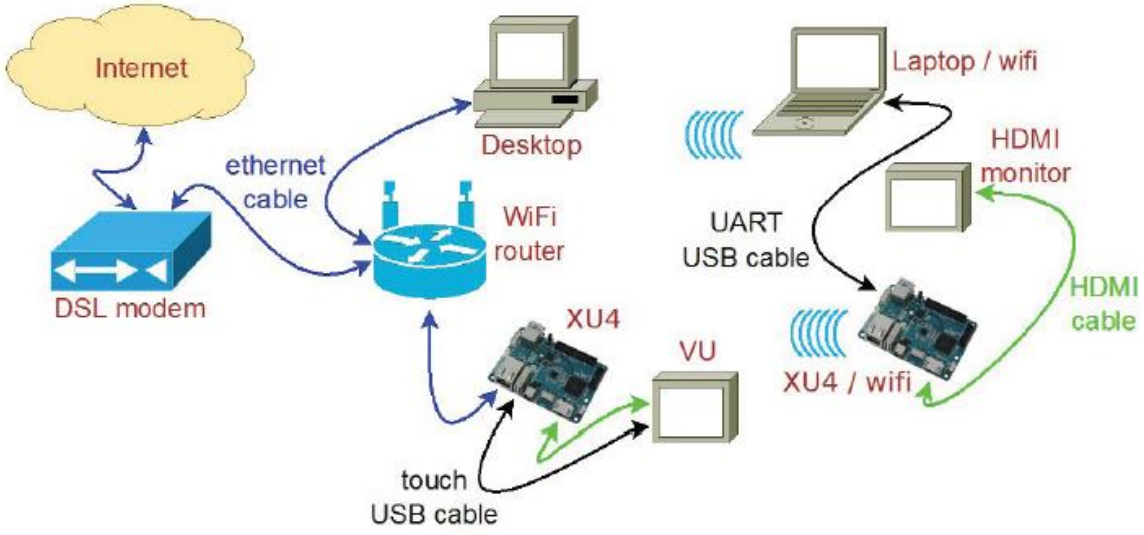


Figura 2. 3 Configuraciones de conexión en el entorno de trabajo

Por seguridad, se ha instalado un ventilador vía USB debajo de la OdroidXU4 con objeto de disipar mejor el calor, ya que el tiempo de funcionamiento del dispositivo en el proyecto es elevado. Con esto, favorecemos una mayor refrigeración del dispositivo y evitamos que existan daños en los pines inferiores de la placa con el roce de la superficie de trabajo.

2.2 CÁMARA UEYE XS

Este trabajo fin de máster consiste en la detección de piezas basándose en técnicas de visión artificial. El elemento fundamental que consta como partida en cualquier proyecto de visión por computador es la cámara. En este caso, se ha decidido usar la cámara UEYE XS.

Esta cámara con el sensor CMOS de 5,04 megapíxeles de ON Semiconductor y un tamaño de píxel de 1,4um, ofrece la máxima precisión de detalle en los colores y una calidad de imagen nítida y transparente. Las características técnicas de la cámara se muestran a continuación:

CARACTERÍSTICAS DE LA CÁMARA UEYE XS 2	
Interfaz	USB 2.0
Tipo de sensor	CMOS
Fabricante	ON Semiconductor
Frecuencia de imagen	15 fps
Resolución	2592 x 1944 (5 Mpx)
Superficie óptica	3,62 mm x 2,72 mm

Esta cámara ofrece tomas óptimas en todas las condiciones lumínicas gracias a las funciones automáticas de exposición, ganancia, compensación contraluz y balance de blancos. Posee una fácil integración gracias a su reducido tamaño y una carcasa ligera. Además, gracias a la interfaz USB 2.0 es ideal para sistemas embebidos.

La imagen del cable de conexión entre el ordenador y la cámara se muestra a continuación:

Cabe destacar la importancia de conectar la cámara al puerto 2.0 de la OdroidXU4, puesto que se han observado varis problemas de comunicación cuando la cámara se encuentra conectada a los dos puertos 3.0 disponibles.

En cuanto a las aplicaciones, destaca su uso en sistemas de control de acceso, biometría, sistemas embebidos, tecnología médica o webcams.

El software disponible para la cámara UEYE XS 2 se encuentra disponible tanto en Windows como en Linux, siendo este último el sistema operativo con el que se trabajará. Dicho software se encuentra en la página oficial.

Para más información sobre la instalación de los drivers y la configuración de los archivos, consultar el anexo de instalación de la cámara.

Capítulo 3

SOFTWARE Y TECNOLOGÍAS IMPLEMENTADAS

El conjunto de tecnologías que se implementan en este proyecto constituye el esqueleto principal de su desarrollo, pues proporcionan la base de comunicación entre cada uno de los programas utilizados.

Entre ellos destaca la tecnología OPCUA, que actúa como base y puente entre todas las tecnologías utilizadas. Adicionalmente, se ha utilizado OpenCV como software de adquisición, tratamiento y procesado de imágenes. En cada uno de los apartados siguientes se desarrollarán en profundidad.

Además, se han utilizado programas como el PuTTY o WinSCP para transferir archivos desde el ordenador anfitrión hasta la Odroid. Estas comunicaciones se realizan por ssh, en las que es necesario saber la IP de la Odroid. Con este método se facilita la transferencia de archivos de forma biridireccional, pudiéndose ejecutar comandos desde el programa PuTTY.

3.1 INTRODUCCIÓN A LA TECNOLOGÍA OPC

La utilización de los sistemas de automatización basados en los PC y software en el ámbito de la automatización industrial se incrementó considerablemente desde principios de los años noventa. Se realizó un gran esfuerzo por estandarizar el desarrollo del software que permite el acceso a la gran cantidad de datos de distintos sistemas de bus, protocolos e interfaces.

Proveedores de Human Machine Interface (HMI) y Supervisor de Control y Adquisición de Datos (SCADA) presentaban estos mismos problemas. Empresas como Rockwell Software, entre otras 22 empresas, empezaron a definir un estándar para controladores de dispositivos que proporciona un acceso estándar a los datos en el campo de la automatización en sistemas basados en Windows.

El resultado fue la creación de la fundación OPC. Una de las razones de su éxito fue la disminución de características principales y la restricción a la definición de API utilizando las tecnologías de Microsoft Windows (COM) y COM distribuido (DCOM). OPC es el estándar aceptado por sistemas SCADA, HMI, sistemas de control distribuido (DCS) y sistema de ejecución de fabricación (MES).

3.2 CLASIC OPC

En base a los distintos requerimientos dentro de las aplicaciones industriales, se han desarrollado tres principales especificaciones o módulos de OPC:

- **Acceso a datos (DA)**: la interfaz de acceso a datos permite leer, escribir y supervisar todas las variables que contienen datos del proceso. Los clientes OPC DA pueden seleccionar qué variables desean leer, escribir o mandar al servidor.

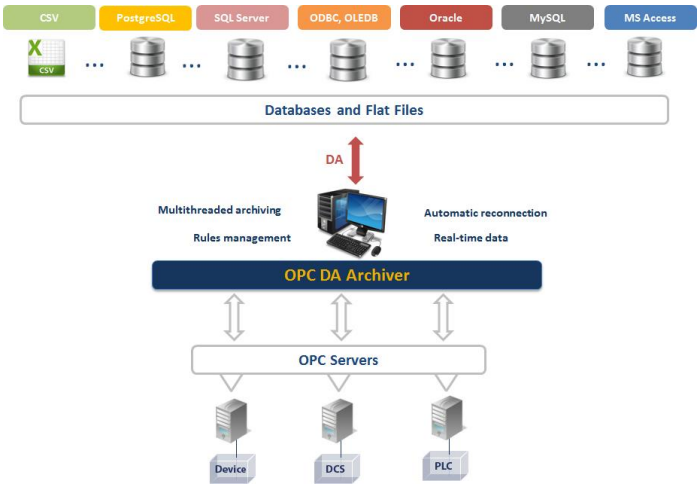


Figura 3. 1 Módulo de acceso a datos (DA)

El cliente agrupa el conjunto de elementos OPC de manera que tengan la misma configuración y tiempo de actualización. También se pueden leer los datos cuando únicamente cuando se produzcan cambios en los valores de las variables. El cliente OPC crea una serie de objetos en el servidor, tal y como se muestra a continuación:

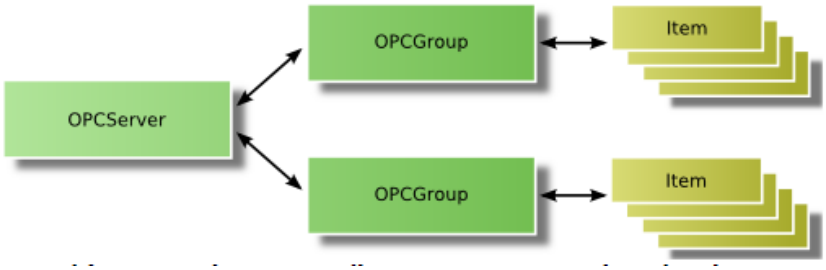


Figura 3. 2 Objetos creados por el cliente OPC

- **Alarmas y eventos (A&E)**: este módulo permite crear y recibir las distintas notificaciones de eventos y alarmas en el proceso. Los eventos son notificaciones únicas que informan al cliente de que un hecho ha tenido lugar, mientras que las alarmas son notificaciones que informan al cliente sobre un cambio del proceso. El proceso de este módulo se muestra a continuación:

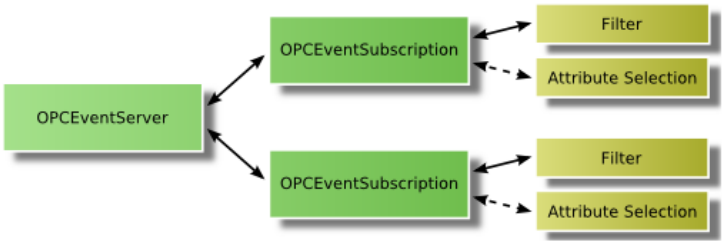


Figura 3. 3 Subscripción del cliente OPC al servidor

El cliente OPC se conecta creando un objeto de la clase OPCEventServer, y genera un OPCEventSubscription para recibir los mensajes. Cada subscripción tendrá asignadas unas reglas o directrices marcadas por el usuario.

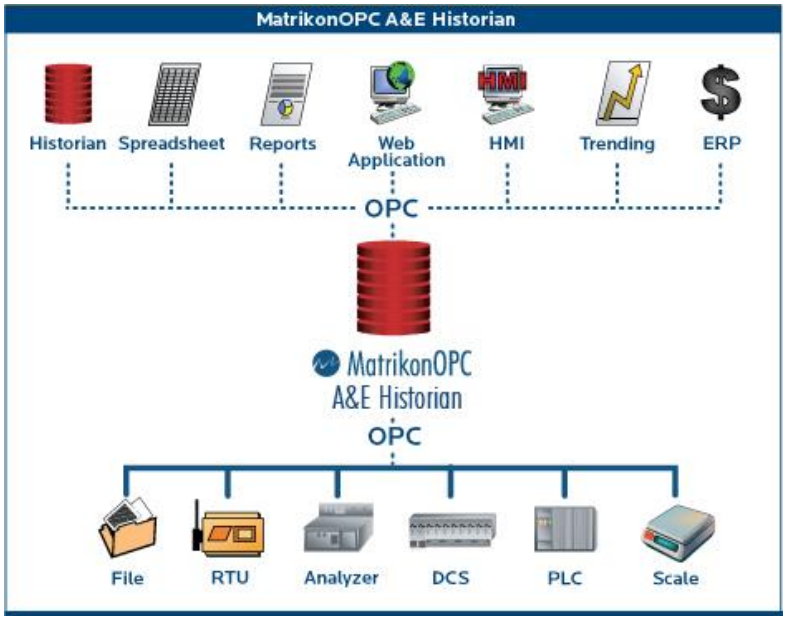


Figura 3. 4 Interconexión de elementos que forman el servidor OPC

En esta imagen se puede ver cómo estarían los diferentes elementos conectados en un proceso industrial teniendo en cuenta el protocolo OPC. Los elementos de más bajo nivel, conectados directamente con los robots, motores o actuadores vuelcan toda la información a los niveles superiores como históricos o informes.

- **Acceso a datos históricos (HDA)**: este último módulo proporciona acceso a todos los datos que se encuentran almacenados. El cliente OPC se conecta a través de un objeto OPCHDAServer en el servidor HDA, que permite leer y actualizar el conjunto de históricos. Además, un objeto OPCHDABrowser recorre todo el conjunto de espacio de direcciones del servidor HDA.

OPC se basa en una comunicación cliente-servidor para intercambiar información. El servidor OPC encapsula la fuente de información del proceso como un dispositivo y hace que su información se encuentre disponible mediante su interfaz. Por otro lado, el cliente OPC se conecta al servidor OPC, pudiendo acceder a todos los datos. En la siguiente figura se observa un ejemplo de clientes y servidores OPC:

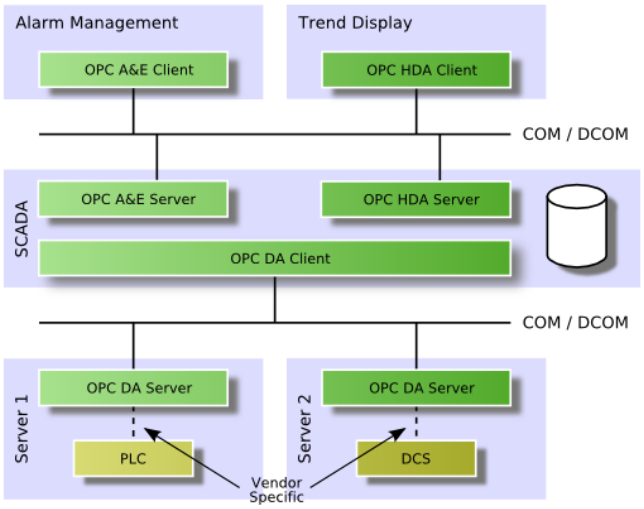


Figura 3. 5 Niveles de implementación del OPC

Tal y como se puede apreciar, se muestran las distintas fases de implementación de OPC, desde el nivel más bajo donde se encuentran los PLC o DCS, hasta el gestor de alarmas o el acceso a los datos históricos.

La ventaja de este enfoque es la reducción de las API, no teniendo que definir un protocolo de red o mecanismo de comunicación entre procesos. En cuanto a las desventajas, destacan la dependencia de Windows como plataforma y los problemas de DCOM, ya que es difícil de configurar, posee grandes tiempos de espera y no se puede usar para comunicar por Ethernet.

3.3 EVOLUCIÓN A OPC-UA

El antiguo protocolo OPC utilizaba cadenas o identificadores que era único para todo el servidor, ya que sólo había un espacio de nombres. Además, los servidores tenían una jerarquía simple. El hecho de crear carpetas completas parar crear ID de elementos únicos provoca una ralentización y cadenas redundantes.

Las desventajas de OPC mencionadas anteriormente sugieren una evolución de este protocolo a una mejora, llamada OPCUA. Los requisitos más importantes de OPCUA son los siguientes:

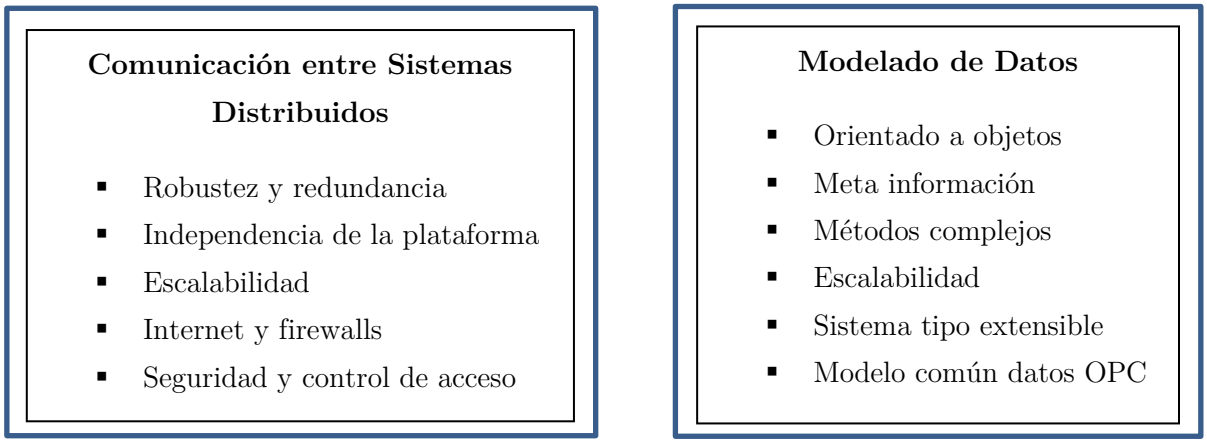


Figura 3. 6 Características principales del OPCUA

La primera versión de OPC UA define un protocolo TCP binario de forma optimizada, así como un conjunto de estándares de Internet aceptados como servicios web, XML y HTTP. El modelado de datos define un conjunto de reglas y componentes necesarios para establecer un modelo de información, ya que los servicios están descritos de manera abstracta.

En OPC UA, cada una de las entidades localizadas en el espacio de direcciones es un nodo. Cada nodo tiene un identificador formado por tres elementos:

-NamespaceIndex: el conjunto de nombres se almacena en una matriz de espacios de nombres. Cada uno de los espacios de nombres tiene un índice que sirve para identificar y optimizar la transferencia y el procesamiento de información.

-Tipo de identificador: puede ser un valor numérico, una cadena o un valor opaco (ByteString). Cuando es necesario ahorrar memoria se utilizan identificadores de nodo numéricos.

-Identificador: se trata del identificador para un nodo en el espacio de direcciones.

La arquitectura de comunicaciones OPC UA se define en la siguiente imagen.

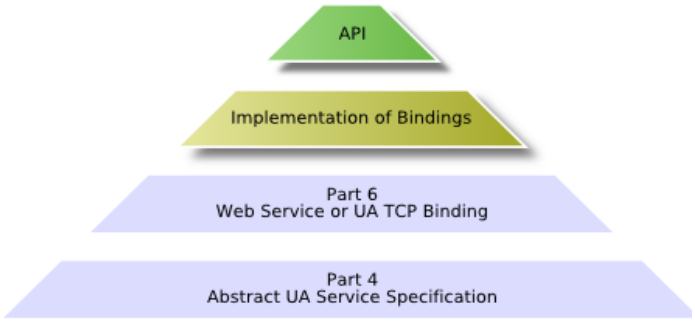


Figura 3. 7 Capas que forman parte del OPCUA

Los servicios de la parte 4 representan el conjunto de interacciones entre el cliente y las aplicaciones del servidor. Por otra parte, en la parte 6 se definen los mecanismos de seguridad de los mensajes y su transporte.

OPCUA establece un modelo de objetos y nodos que forman parte del espacio de direcciones dentro del protocolo:

- **Modelo de Objetos**: el objetivo principal del espacio de direcciones es proporcionar una forma en la que los servidores representen objetos a los clientes. Estos objetos son definidos mediante variables y métodos. Cuando los servicios quieren acceder a los objetos y sus componentes, pueden llamar a un método o recibir eventos del objeto.

  Cada uno de los elementos de este modelo se encuentra representado en el espacio de direcciones como nodos. Cada uno de los nodos se asigna a una variable y método.

- **Modelo de Nodo**: los objetos se representan como un conjunto de nodos en el espacio de direcciones, caracterizados por atributos y referencias.

  OPCUA establece ocho clases distintas de nodos. Cada nodo es una instancia de las clases de nodos en el espacio de direcciones. En la siguiente figura se muestra el conjunto de clases de nodo:

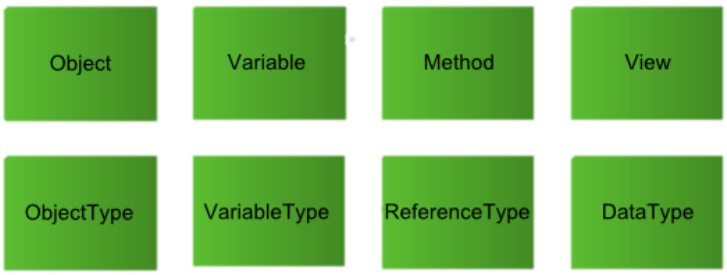


Figura 3. 8 Distintas clases de nodos en un servidor OPCUA

- **Atributos:** Los atributos son el conjunto de datos que describen los nodos. Los clientes, a través de servicios de lectura, escritura o consulta pueden acceder a los valores de los atributos. Las definiciones de los atributos son conocidas directamente por los clientes, pero no se encuentran visibles en el espacio de direcciones.

  El atributo consiste en un identificador, un nombre, una descripción, tipo de datos y un indicador.

- **Referencias:** Las referencias sirven para relacionar distintos nodos entre sí. Las referencias se definen como instancias de nodos ReferenceType, a diferencia de los atributos.

- **Variables:** Las variables se utilizan para la representación de valores. Se definen dos tipos de variables: propiedades y datos variables.
  - Propiedades: difieren de los atributos debido a que el servidor las define y agrega. Un nodo y sus propiedades siempre deben situarse en el mismo servidor. Los atributos forman parte de todos los nodos de una clase de nodo, y se definen mediante la especificación OPC UA, mientras que las propiedades pueden ser definidas por el servidor.

- Variables de datos: representan el contenido de un objeto. Un objeto está formado por un conjunto de variables y métodos. El nodo de un objeto no proporciona un valor, mientras que los nodos de variables sí.

3.4 CONFIGURACIÓN DE LA CONEXIÓN

OPC UA ofrece una funcionalidad denominada suscripción. Un cliente puede suscribirse a distintos tipos de información proporcionada por un servidor OPC UA. Mediante la suscripción, las fuentes de información se agrupan formando una notificación.

Una suscripción debe ser creada a través de una sesión. Para crear una nueva sesión, es necesario establecer un canal seguro entre cliente y servidor.

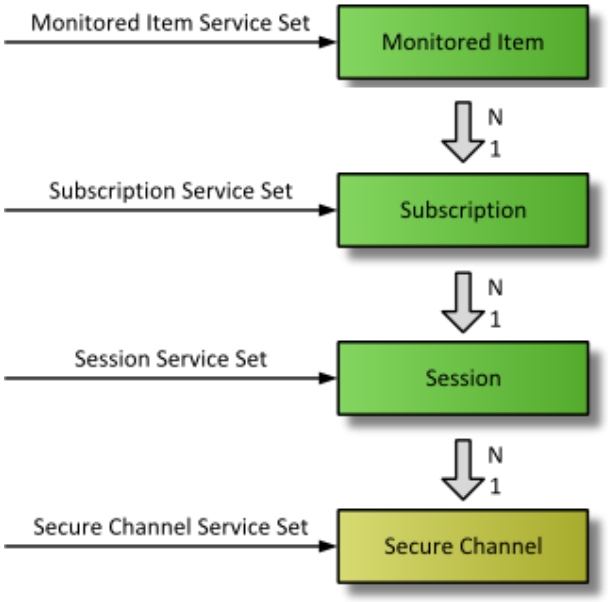


Figura 3. 9 Contexto de suscripción a cambios y eventos

Existen tres tipos de cambios mediante los cuales el cliente se puede suscribir al agregar elementos nuevos a la suscripción:

-Suscripción a los cambios de los datos de las variables.

-Suscripción a eventos de objetos.

-Suscripción a valores agregados en base a los valores de las variables.

El tiempo de muestreo define la velocidad en la que el servidor actualiza los cambios del conjunto de datos. El muestreo se puede configurar más rápido que la notificación que realiza el cliente. El servidor admite la cola de muestras de datos y eventos. Cuando los datos se entregan al cliente, la cola se vacía.

La conexión a un servidor requiere la dirección de red, protocolo y dirección de seguridad. Toda esta información se almacena en un punto final, que contiene:

-URL (protocolo y dirección de red).

-Seguridad: conjunto de protocolos de seguridad y la clave.

-Autentificación del usuario.

Capítulo 4

DESARROLLO E IMPLEMENTACIÓN

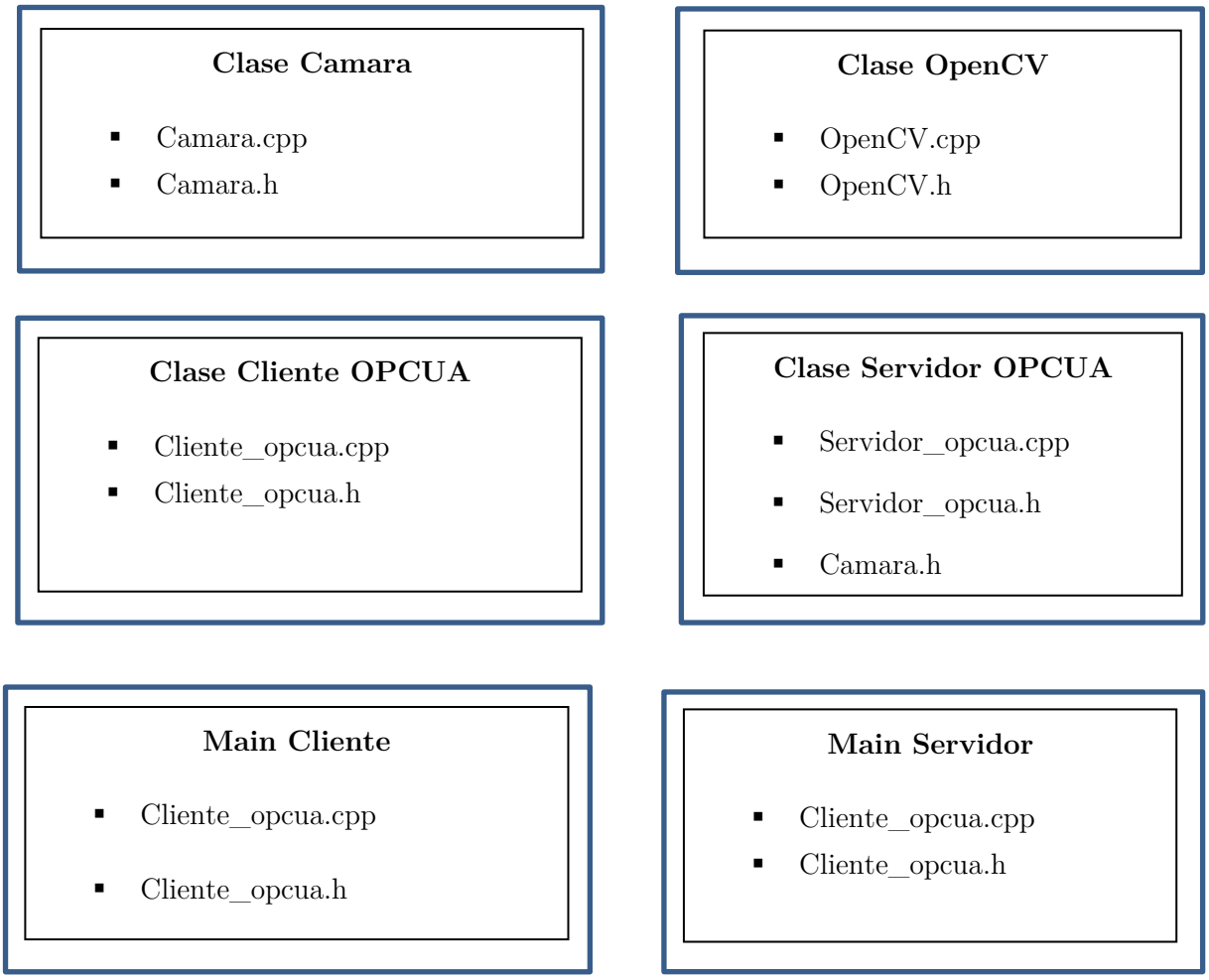
Este capítulo explica los procedimientos que se han seguido para realizar la comunicación mediante el servidor OPCUA, así como la adquisición y tratamiento de imágenes utilizando la herramienta OpenCV.

Todas estas herramientas deben estar conectadas entre sí, siendo necesario desarrollar mecanismos que permitan integrar todo el código en varios ficheros. Por esta razón se explicará el procedimiento de forma jerárquica, de tal forma que se analice el problema de forma más general, llegando a profundizar hasta los módulos más pequeños implementados.

4.1 PLANTEAMIENTO DEL PROBLEMA

La integración de información en un servidor partiendo de un algoritmo basado en visión artificial conlleva a plantearse una estrategia de diseño del problema que sea lo más modular posible y que cumpla con el objetivo deseado.

El esquema general del proceso se describe en la siguiente la Figura 4.1. Este esquema se compone de cuatro elementos principales que interactúan entre sí: cámara, OpenCV, cliente y servidor. Cada uno de ellos está desarrollado mediante archivos distintos siguiendo la siguiente distribución:



Tal como se puede observar, cada parte principal del proyecto constituye una clase en C++, que depende de su archivo principal y de las librerías necesarias para realizarlo. Las clases cámara como la clase OpenCV constituyen la base a partir de la cual, tanto el cliente como el servidor hacen uso de ellas para su implementación.

Todo es gestionado por el programa main. Debido a que el cliente y servidor funcionan de forma independiente, el programa final se ejecuta en dos terminales distintos, pudiendo observar el comportamiento y los cambios de cada uno. Es por lo que existen dos programas principales, uno para el cliente y otro para el servidor.

En los siguientes apartados se explica cada una de las partes, aunque como se observa el e esquema anterior, la relación entre ellas es fundamental para que el programa funcione.

4.2 ADQUISICIÓN DE IMÁGENES

El comienzo del proceso tiene lugar a través del algoritmo de adquisición de imágenes. El objetivo propuesto es detectar la presencia de monedas en la zona de estación de trabajo. Dicha estación se encuentra en una mesa, en la que la cámara UEYE XS se encuentra fija por un soporte vertical, de tal forma que el eje focal de la cámara se encuentra perpendicular a la superficie de trabajo. La imagen siguiente muestra un esquema de la estación:

La cámara se encuentra situada a unos 15 cm de altura. Se ha considerado esta altura óptima para trabajar con un número de monedas suficientes.

La adquisición de imágenes es un proceso delicado en el que factores como la iluminación ambiental, orientación de los objetos y color de la superficie de trabajo juegan un papel fundamental.

Para calibrar todos estos parámetros físicos, se han hecho varias pruebas en distintas condiciones ambientales, así como el color escogido para la superficie. Para la elección del color de la superficie de trabajo se ha tenido en cuenta el color de los objetos que se desean detectar, ya que es importante que exista un contraste claro. Dado que las monedas son de color ocre y bronce, los colores que mejores resultados han dado han sido el blanco y el azul oscuro.

Una vez instalada la cámara, se ha programado siguiendo las funciones proporcionadas por la librería de instalación, de tal forma que cumpla con los objetivos deseados. Uno de los parámetros más importantes a elegir es el tamaño de la imagen en píxeles, ya que no todos los formatos proporcionados por la librería son compatibles.

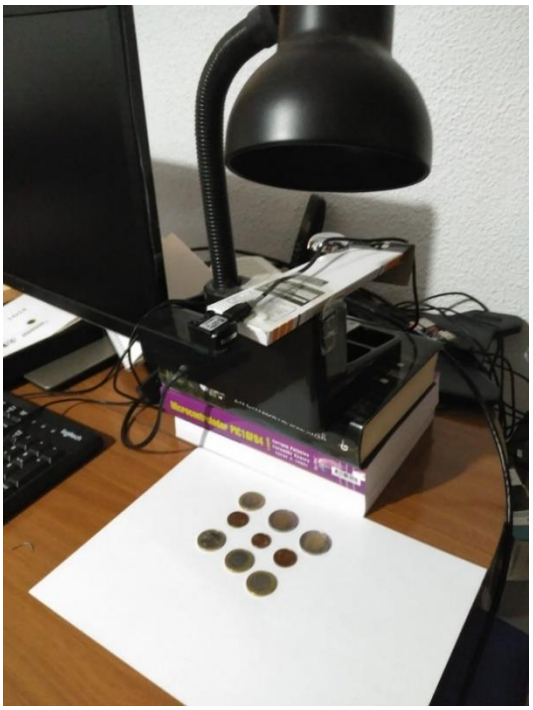


Figura 4. 1 Superficie de trabajo

Teniendo en cuenta la altura focal de la cámara, el tamaño de la imagen seleccionado ha sido de 1080x720 píxeles. A continuación, se expone la implementación del código para la adquisición de imágenes:

- **Camera.h**: la clase cámara está compuesta por dos métodos. El método de adquisición realiza una llamada al método de inicializar la cámara.

```
using namespace std;

class Camera
{

public:

    SENSORINFO sInfo;
    void adquisition();
    void initializeCameraInterface(HIDS* hCam_internal);
};
```

Figura 4. 2 Código Camera.h

- **Camera.cpp**: todos los archivos cuentan con una cabecera donde se documenta un resumen del documento. En la parte superior se encuentran todas las librerías necesarias, así como constantes.

```
//!-----------------------------------------------------------------------------
//! @file Camera.cpp
//! @class Camera
//! @brief Clase que implementa la UEYE XS
//! @author David Castillo Saez
//! @date 29/08/2018
//!-----------------------------------------------------------------------------
#include <iostream>
#include <stdio.h>
#include <stddef.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <time.h>
#include <ctime>
#include "camera.h"
//-----------------------------------------------------------------------------
//                              DEFINICION DE CONSTANTES
//-----------------------------------------------------------------------------

#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_BLUE    "\x1b[34m"
#define ANSI_COLOR_MAGENTA "\x1b[35m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_RESET   "\x1b[0m"
using namespace std;
```

Figura 4. 3 Código Camera.cpp

El primer método se encarga de la adquisición de la foto, en el cual se inicializan las variables necesarias como el identificador de la cámara. A continuación, se llama al método de inicializar la cámara, encargado de configurar todos los parámetros de calibración, modo de color y frecuencia de trabajo. A esta función se le pasa como parámetro el identificador de la cámara.

Cada una de las funciones implementadas retorna un variable de tipo INT. La mayoría de las funciones devuelven *IS_SUCCESS* o *IS_NOT_SUCCESS* en función del éxito que tengan, por lo que después de cada una de ellas, se muestra en pantalla el resultado, de forma que el usuario tenga conocimiento del resultado. El nombre de la variable escogida que almacena dicho resultado es 'nRet'.

```
void Camera::adquisition(){
unsigned t,t1; //definicion de tiempos
// ----------------------------------------------------------------------------------------
SENSORINFO sInfo;
cout<<"INICIO..."<<endl;
system("/etc/init.d/ueyeusbdrc start");
sleep(1);
t=clock();
HIDS hCam = 0;
int memID = 0;
char* pMem=NULL;
initializeCameraInterface(&hCam);//LLAMADA A LA CONFIGURACION DE LA CÁMARA
is_AllocImageMem(hCam, 1280, 720 ,16, &pMem, &memID);//1980x1080
INT nRet = is_SetImageMem(hCam, pMem, memID);
cout<<"SE VA A INICIAR EL PROGRAMA..."<<endl;
//PROCESO DE CAPTURA DE FOTOS
INT displayMode = IS_SET_DM_DIB;
nRet=is_SetDisplayMode (hCam, displayMode);
printf("Status displayMode %d\n",nRet);
nRet = is_FreezeVideo(hCam, IS_WAIT);
cout<<"Status is_FreezeVideo"<<nRet<<endl;//El encendido de la camara ha tenido exito
system("echo 1 > /sys/class/gpio/gpio29/value");
IMAGE_FILE_PARAMS ImageFileParams;
//      INT is_SaveImage(hCam,"/usr/local/share/ueye/bin/imagen.jpeg");
ImageFileParams.pwchFileName =L"./usr/local/share/ueye/bin/tfm/imagen.png";
ImageFileParams.pnImageID = NULL;
ImageFileParams.ppcImageMem = NULL;
ImageFileParams.nQuality = 98;
ImageFileParams.nFileType = IS_IMG_PNG;
```

Figura 4. 4 Clase que implementa la cámara UEYE XS

A través de este método se obtiene la información del sensor, así como si la frecuencia de la cámara se ha establecido correctamente. Por último, se selecciona el modo de color. Este parámetro depende del tipo de imagen que se quiera obtener posteriormente. Debido a que se ha escogido un formato de imagen .jpg, el color de imagen es seleccionado en la librería es '*COLORMODE_BAYER*'.

```
void Camera::initializeCameraInterface(HIDS* hCam_internal){
    int nMemoryId;
    int colorMode;
    INT nRet = is_InitCamera (hCam_internal, NULL);
    if (nRet == IS_SUCCESS){
        cout << "Camera initialized!" << endl;
    }
    is_GetSensorInfo(&hCam_internal, &sInfo);
        int  m_nSizeX,m_nSizeY, mirSizex,mirSizey;
        m_nSizeX = sInfo.nMaxWidth;
        m_nSizeY = sInfo.nMaxHeight;
        mirSizex = sInfo.nMaxWidth;
        mirSizey = sInfo.nMaxHeight;
        cout<< "SizeX " <<m_nSizeX<<endl;
        cout<< "SizeY " <<m_nSizeY<<endl;
        cout<< "Sizex " <<mirSizex<<endl;
        cout<< "Sizey " <<mirSizey<<endl;
    UINT nPixelClockDefault =21;
    nRet = is_PixelClock(*hCam_internal, IS_PIXELCLOCK_CMD_SET, (void*)
    if (nRet == IS_SUCCESS){
        cout << "Camera pixel clock succesfully set!" << endl;
    }else if(nRet == IS_NOT_SUPPORTED){
        cout << "Camera pixel clock setting is not supported!" << endl;
    }
    colorMode =IS_COLORMODE_BAYER;
    nRet = is_SetColorMode(*hCam_internal,colorMode);
    if (nRet == IS_SUCCESS){
        cout << "Camera color mode succesfully set!" << endl;
    }
}
```

Figura 4. 5 Función que inicializa la cámara

Por último, se ejecuta el proceso de guardado de las imágenes. Cada vez que se produce la adquisición de una imagen, se guardan tres fotos en los formatos: jpg, png y bmp en el directorio de trabajo específico, de tal forma que pueda ser accesible por el algoritmo de reconocimiento.

El proceso de adquisición de imágenes se realizará periódicamente, atendiendo a un periodo de muestreo en el que el usuario podrá cambiar tanto el número de monedas existentes como su valor. Dicho período de muestreo se puede modificar. En la explicación del código principal de ejecución de profundizará más sobre este tema.

En las Figuras 4.6 y 4.7 se muestran varios ejemplos de adquisición de monedas:

Figura 4. 6 Ejemplo de imagen con fondo blanco

Figura 4. 7 Ejemplo de imagen con fondo azul oscuro

Las imágenes anteriores pertenecen a pruebas realizadas para determinar el color del fondo de la superficie de trabajo, así como la luminosidad. Con ambos colores se han obtenido resultados muy parecidos, por lo que se ha implementado la ejecución final con ambos.

4.3 ALGORITMO DE RECONOCIMIENTO DE IMÁGENES

Todo proceso de reconocimiento de imágenes requiere un entorno visual que posea unas características de luminosidad adecuadas al propósito de su análisis. Esta es la principal dificultad que se expone en este apartado. Se describirán los algoritmos implementados y los ajustes realizados para la detección de monedas.

```
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <stdio.h>
#include <vector>
#define tresh 100
#define max_tresh 255
using namespace cv;
using namespace std;
class OpenCV
{
public:
    //---------------------------------------------------------------------------
    //                          DECLARACION DE LA CONSTANTES
    //---------------------------------------------------------------------------
    RNG rng;
    ofstream archivo;                          //Archivo donde se guardaran los datos
    Mat src,src_gray,gray,gauss,original,canny,salida,image_recortada;
    Mat canny_output,fondo,result,diff,dst;
public:
    void savedFiles();
    void detection(int *cont_m1,int *cont_m2,int *cont_m3,int *cont_m4,int *cont_m5,int *cont_m6,int *cont_m7,

};
```

Figura 4. 8 Definición de variables y métodos de la clase

El algoritmo de reconocimiento de las imágenes permite identificar al usuario el número de monedas de cada valor que se encuentran en la superficie de trabajo, y, por tanto, el valor total. Esta operación resulta interesante cuando se extrapola a distintos ámbitos de utilidad, como puedan ser una clasificación de objetos que se desplazan a lo largo de una cinta transportadora en un proceso industrial.

A continuación, se muestra un ejemplo del algoritmo con una imagen adquirida.

Figura 4. 9 Imagen obtenida con la cámara

La implementación de la clase OpenCV se ha realizado en C++ usando la siguiente estrategia:

- **Establecer región de interés**: esta operación consiste en recortar la imagen que se ha adquirido a través de la cámara, obteniendo así la zona donde se encuentran las monedas que van a ser detectadas. Para ello se ha definido un rectángulo con las dimensiones deseadas, y se le ha pasado como parámetro a la imagen original.
- **Preprocesamiento**: esta técnica tiene como objetivo detectar y eliminar los fallos que puedan existir en la imagen en la medida de lo posible. Dentro de las diversas técnicas existentes, se ha convertido la imagen a escala de grises con la función:

  ```
  cvtColor(image_recortada,gray, COLOR_BGR2GRAY);
  ```

  donde los parámetros son la imagen recortada, el nombre de la imagen destino transformada a escala de grises y el parámetro específico de la transformación.

Figura 4. 10 Imagen transformada a escala de grises

```
//!-------------------------------------------------------------------------------------------------
//! @file OpenCV.cpp
//! @class OpenCV
//! @brief Clase OpenCV que realiza la deteccion de monedas
//! @author David Castillo Saez
//! @date 29/08/2018
//!-------------------------------------------------------------------------------------------------
#include <vector>
#include "OpenCV.h"
/*-----------ALGORITMO DE DETECCION DE MONEDAS-------------*/
void OpenCV::detection(int *cont_m1,int *cont_m2,int *cont_m3,int *cont_m4,int *cont_m5,int *cont_m6,int
   *cont_m1=*cont_m2=*cont_m3=*cont_m4=*cont_m5=*cont_m6=*cont_m7=*cont_m8=0;
    archivo.open("fichero.txt");
    vector <Vec3f> circles;
    vector<vector<Point>>contornos;
    vector<Vec4i> hierarchy;
     original=imread("p34.png",CV_LOAD_IMAGE_COLOR);
    original = imread("/usr/local/share/ueye/bin/tfm/imagen.jpg",CV_LOAD_IMAGE_COLOR);
    //Abre la imagen que se encuentra en la ruta de la variable buffer
    //Por lo tanto, a partir de ese punto se define una zona de 100 pixeles de ancho por 150 de alto.
    Rect myrect(250,10,900,700);//Definimos un rectangulo para la region de interes:Establece una región
    Mat image_recortada=original(myrect);
```

Figura 4. 11 Implementación del algoritmo de detección

A continuación, se ha filtrado la imagen con objeto de reducir el ruido existente. Para ello se ha aplicado la expresión:

```
GaussianBlur(gray, gauss,Size(3,3),0,0);
```

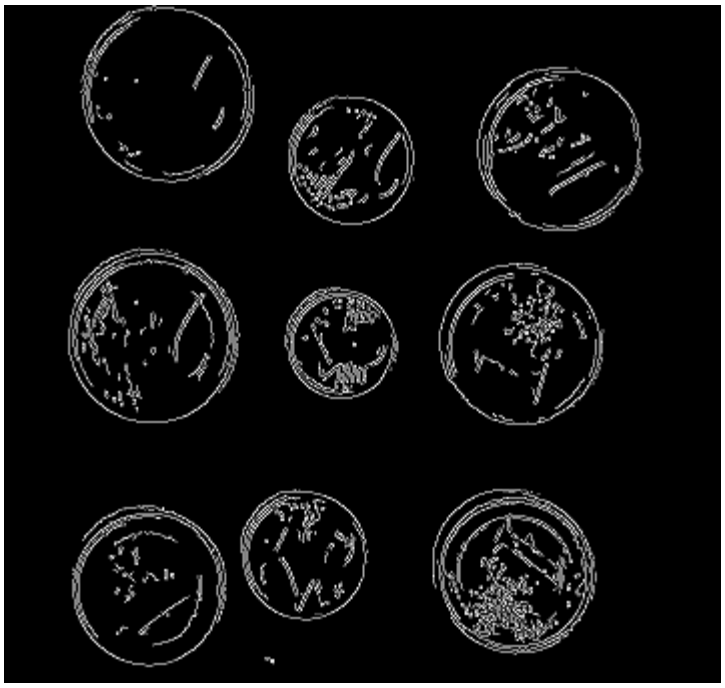


Figura 4. 12 Algoritmo de Canny para detección de bordes

```
namedWindow( "Imagen Recortada", CV_WINDOW_AUTOSIZE );
imshow("Imagen Recortada", image_recortada);
cvtColor(image_recortada,gray, COLOR_BGR2GRAY);
GaussianBlur(gray, gauss,Size(3,3),0,0);                              //Se aplica un suavizado Gaussi
imshow("suavizado", gauss);

threshold(gray, diff, 45, 100,0 );
imshow("Resta", diff);
Canny(gauss,canny_output, 25,50);
namedWindow( "Canny", CV_WINDOW_AUTOSIZE );
imshow("Canny", canny_output);
HoughCircles( gauss, circles, CV_HOUGH_GRADIENT, 1, 30, 100, 30, 50, 100);//Se usa la imagen gauss
cout << "Numero de Monedas Detectadas : " << circles.size()<<endl;//Mostramos el número de monedas
char buff[100];
static int tipo;
findContours(gray, contornos, hierarchy,  RETR_EXTERNAL,CV_CHAIN_APPROX_SIMPLE);          //Buscamo
cout<<"Numero de contornos: "<<contornos.size()<<endl;
namedWindow( "Contours", CV_WINDOW_AUTOSIZE );
float vect_diam[circles.size()];
float vect_centro[circles.size()];
```

Figura 4. 13 Detección del número de monedas

Donde se indica el tamaño del núcleo que se utilizará en el filtro, así como las desviaciones estándar. En base a varias pruebas realizadas, los parámetros indicados han sido los más adecuados.

- **Segmentacion**: consiste en la división de la imagen digital en varios objetos. El objetivo consiste en simplificar la imagen de partida por otra que sea más fácil de analizar. Existen diversas técnicas de segmentación, dentro de las cuales se ha implementado un thresholding. El thresholding es el método de segmentación que consiste en separar las regiones basándose en la diferencia de intensidad entre píxeles vecinos, pudiendo distinguir los píxeles del objeto y los del fondo. Para ello, se utiliza un umbral determinado en base a pruebas de ensayos que se adecue mejor a la situación particular. La función implementada es:

```
threshold(gray, diff, 45, 100,0 );
```

Después, el objetivo es detectar los bordes de las monedas. Para ello se ha implementado el detector Canny, pasándole el umbral inferior de detección y el tamaño del núcleo (matriz):

```
Canny(gauss,canny_output, 25,50);
```

Como se puede apreciar, se detectan los bordes externos de las monedas, así como algunas de las siluetas internas.

El siguiente paso consiste en encontrar los contornos externos. Para ello se ha utilizado la función 'findContours', cuyos parámetros son la imagen fuente y destino, así como el tipo de contornos que se desean encontrar. El programa considera un contorno como cerrado. Por ello, se han generado dos vectores de tamaño el número de contornos o círculos detectados, para almacenar tanto el centro como el radio de cada moneda. La implementación se muestra a continuación:

```
for(int i = 0; i < circles.size(); i++ )
   {
     sprintf(buff,"%s" "%d","Moneda",i+1);
     Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
     Point a(-30,40);
     float radius = cvRound(circles[i][2]);
     float diametro=radius*2*FACTOR;
     vect_diam[i]=diametro;
     circle( image_recortada, center, 3, Scalar(0,255,0), -1, 8, 0 );
     circle( image_recortada, center, radius, Scalar(0,0,255), 2.5, 8,0);
     putText(image_recortada, buff,center-a,FONT_HERSHEY_SIMPLEX,0.5,CV_RGB(0,0,255),2 );
```

Figura 4. 14 Bucle para cálculo de centros y radios

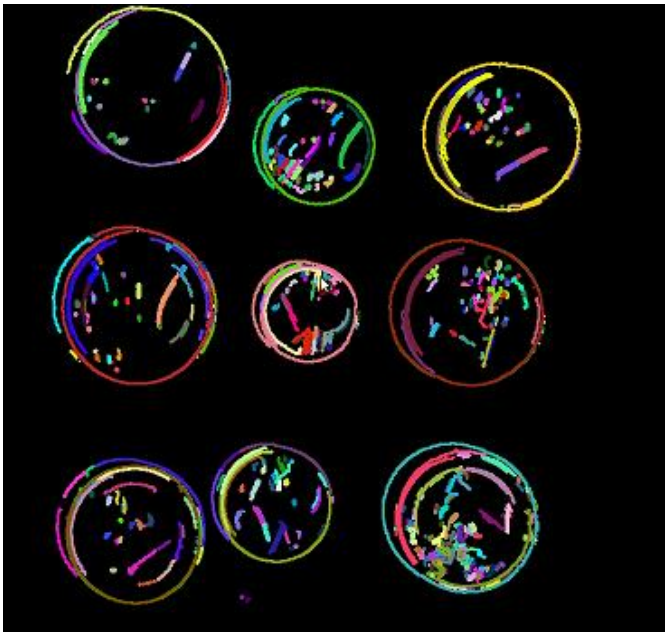


Figura 4. 15 Contornos dibujados de las monedas detectadas

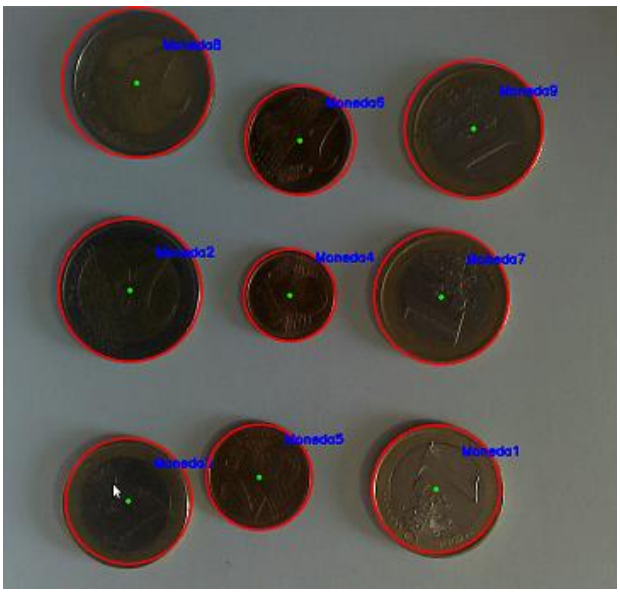


Figura 4. 16 Resultado del algoritmo implementado

Es importante detectar en la imagen que se muestra por pantalla qué moneda corresponde a cada iteración del bucle. Para ello se ha creado un buffer donde se almacenará la concatenación del nombre de la moneda. En cada iteración del bucle, se calcula tanto las coordenadas [x,y] del centro como el diámetro, ambas medidas en píxeles.

Es importante tener en cuenta que la discriminación del tamaño de las monedas se realiza mediante el diámetro calculado en milímetros, no en píxeles. Por ello, es necesario saber la conversión que existe entre puntos del mundo (milímetros) y puntos de la imagen (píxeles).

Basándose en las medidas del sensor de la cámara, el tamaño definido de la imagen en píxeles, y las medidas de alto y ancho reales medidas en milímetros que abarca la foto, se ha calculado el factor de conversión, siendo este 0,151. De esta forma, ya se puede establecer una relación directa entre píxeles y milímetros.

```
if(vect_diam[i]>=15 && vect_diam[i]<=16.5){
            cont_m1++;
            archivo<<"MONEDA 1 CENTIMO: "<<vect_diam[i]<<" "<< vect_centro[i]<<endl;
            //Se almacena el centro en el archivo
            cout << "MONEDA " <<i+1<< "es de 1 CENTIMO: "<< "Diametro : \n" << diametro << endl;
       }
    else if(vect_diam[i]>16.5 && vect_diam[i]<=19.33){
            cont_m2++;
            archivo<<"MONEDA 2 CENTIMOS: "<<vect_diam[i]<<" "<< vect_centro[i]<<endl;
            cout << "MONEDA " <<i+1<< "es de 2 CENTIMOS: "<< "Diametro : \n" << diametro << endl;
    }
    else if(vect_diam[i]>19.33 && vect_diam[i]<=21.6){
            cont_m3++;
            archivo<<"MONEDA 5 CENTIMOS: "<<vect_diam[i]<<" "<< vect_centro[i]<<endl;
            cout << "MONEDA " <<i+1<< "es de 5 CENTIMOS: "<< "Diametro : \n" << diametro << endl;
    }

    else if(vect_diam[i]>21.6 && vect_diam[i]<=22.75){
            cont_m5++;
           archivo<<"MONEDA 20 CENTIMOS: "<<vect_diam[i]<<" "<< vect_centro[i]<<endl;
           cout << "MONEDA " <<i+1<< "es de 20 CENTIMOS: "<< "Diametro : \n" << diametro << endl;
    }
```

Figura 4. 17 Discriminación de monedas por radio

En la imagen anterior se muestra un ejemplo de la discriminación del tamaño de las monedas. Por pantalla se mostrarán los datos calculados para cada moneda. Además, cada uno de los datos será guardado en un fichero .txt, de tal forma que pueda ser consultado o exportado a un programa de base de datos.

A continuación, se muestra un resultado en forma de imagen basado en el algoritmo de detección:

Figura 4. 18 Dibujo del contorno de círculos

Cada una de las monedas identificadas se enumeran con una etiqueta y un número, pudiendo distinguir los valores de sus centros y radios en el archivo generado.

4.4 IMPLEMENTACIÓN DEL SERVIDOR OPCUA

El servidor constituye uno de los elementos fundamentales de este trabajo fin de máster. Basándose en el concepto teórico de OPCUA explicado en el capítulo anterior, se ha implementado en este apartado un archivo capaz de establecer un servidor OPCUA, recopilando toda la información importante sobre el proceso de adquisición y reconocimiento de imágenes.

En este apartado se explica cómo se ha implementado el programa principal llamado '*main_server.cpp*', que constituye el programa principal a partir del cual se ejecuta el servidor del programa. Dicho programa contiene las clases servidor, cámara y OpenCV, estas dos últimas explicadas anteriormente.

El servidor establece un modo de comunicación TCP/IP seguro formado por un nodo. Por defecto, la dirección IP en la que se establece el servidor es en la del propio host, pudiendo cambiarla por otra distinta. El puerto donde se establece la comunicación es el 4840.

En dicho nodo se han definido y añadido distintas tipologías de variables o tags. Cada tag pertenece a un tipo de moneda, de tal forma que su valor se irá actualizando conforme las detecte el algoritmo de reconocimiento de OpenCV.

La siguiente figura muestra la estructura principal del archivo 'main_server.cpp':

```
UA_Int32 cont_m1;                                   //Contador de monedas de 1 centimo
UA_Int32 cont_m2;                                   //Contador de monedas de 2 centimos
UA_Int32 cont_m3;                                   //Contador de monedas de 5 centimos
UA_Int32 cont_m4;                                   //Contador de monedas de 10 centimos
UA_Int32 cont_m5;                                   //Contador de monedas de 20 centimos
UA_Int32 cont_m6;                                   //Contador de monedas de 50 centimos
UA_Int32 cont_m7;                                   //Contador de monedas de 1 euro
UA_Int32 cont_m8;                                   //Contador de monedas de 2 euros
char buffer[100];
void initproceso();
void closeconfiguration();
void proc(Server_Opcua* Opcua1,OpenCV* Opencv1, Camera *camera1);            //Prototipo Hilo
int main(){
Server_Opcua* Opcua1=new Server_Opcua();
OpenCV* Opencv1=new OpenCV();
Camera *camera1=new Camera();
thread hilo(proc, Opcua1, Opencv1, camera1);
hilo.join();
return 0;
}
```

Figura 4. 19 Código main_server.cpp

Se han definido las variables globales de los contadores. La tipología de estas variables debe adaptarse a opcua. Se han definido dos métodos que implementan el arranque y el cierre de la aplicación. Dichos métodos contienen la configuración de los parámetros de la cámara, así como de habilitar/deshabilitar los GPIOs utilizados.

Dentro del programa principal se ha instanciado cada una de las clases que se han implementado. Después, se ha definido un hilo de ejecución paralelo al main que se encargará de realizar todo el proceso del servidor. Por ello, es necesario pasarle como argumentos el nombre de la función que va a ejecutar (proc), y cada una de las instancias de las que va a hacer uso. Por último, es necesario hacer un join del hilo para que el programa espere a la finalización del hilo.

La función que ejecuta el hilo creado se muestra a continuación:

```
void proc(Server_Opcua* Opcua1,OpenCV* Opencv1,Camera *camera1){
    initproceso();
    HIDS hCam=10;
    int memID=1;
    Opcua1->AddTags(&cont_m1,&cont_m2,&cont_m3,&cont_m4,&cont_m5,&cont_m6,&cont_m7,&cont_m8);
    for(;;){
    cout<<RED<<"INICIO DEL HILO"<<RESET<<endl;
    camera1->adquisition(&hCam,&memID);
    //cout<<RESET<<"1"<<endl;
    Opencv1->detection(&cont_m1,&cont_m2,&cont_m3,&cont_m4,&cont_m5,&cont_m6,&cont_m7,&cont_m8);
    Opcua1->TagsAssigment(&cont_m1,&cont_m2,&cont_m3,&cont_m4,&cont_m5,&cont_m6,&cont_m7,&cont_m8);
    cout<<"CAMBIA LAS MONEDAS DE POSICION"<<endl;
    sleep(10);
    waitKey(1000000);
    }
    closeconfiguration();               //Finaliza la configuracion de los dispositivos
  }
```

Figura 4. 20 Función de implementación del hilo

La clase servidor se encuentra estructurada de la siguiente forma:

- Librerías y definición de constantes
- Variables públicas y privadas usadas en la clase
- Métodos implementados en la clase.

```
#define     T_MUESTREO      10                  //Tiempo de muestreo del proceso
#define     M1CENTIMO       0.01                //Valor de moneda de 1 centimo
#define     M2CENTIMOS      0.02                //Valor de moneda de 2 centimos
#define     M5CENTIMOS      0.05                //Valor de moneda de 5 centimos
#define     M10CENTIMOS     0.1                 //Valor de moneda de 10 centimos
#define     M20CENTIMOS     0.2                 //Valor de moneda de 20 centimos
#define     M50CENTIMOS     0.5                 //Valor de moneda de 50 centimos
#define     M1EURO          1                   //Valor de moneda de 1 euro
#define     M2EUROS         2                   //Valor de moneda de 2 euros
#define     DOWN_1CENTIMO   10                  //Rangos inferiores y superiores
```

Figura 4. 21 Valores monetarios y constantes

Primero se definen las constantes que se van a utilizar en el programa, de forma que se encuentre lo más parametrizado posible. Se ha definido el tiempo de muestreo, que es el tiempo que se permite al usuario cambiar de posición las monedas antes de que se ejecute otra iteración del bucle. Además, se han definido tanto el valor de cada tipo de moneda y los rangos superiores e inferiores de estas.

```
//---------------------------------------------------------------------------
//                          DECLARACION DE LA CLASE
//---------------------------------------------------------------------------

class Server_Opcua
{
public:
    //-------------------DEFINICIÓN DE VARIABLES PUBLICAS Y COMPARTIDAS-----
    bool conexion;        //1 si el servidor esta conectado, 0 si no esta conectado.
    bool server_connected;//1 si el servidor se encuentra conectado, 0 si no se encuentra conectado
    bool client_connected;//1 si el cliente se encuentra conectado, 0 si no se encuentra conectado.
    bool photo_adquisition//1 si se esta realizando la captura de una imagen, 0 si no.
    unsigned t0,t1;        //Definicion de los tiempos del programa (inicio y fin de proceso)
    UA_VariableAttributes attr1,attr2,attr3,attr4,attr5,attr6,attr7,attr8;
    UA_Variant value1,value2,value3,value4,value5,value6,value7,value8;
```

Figura 4. 22 Variables públicas de la clase servidor

A continuación, se han definido variables para identificar el estado de conexión del servidor y cliente, medidas de tiempos y las variables propias de la librería opcua para identificar los valores y atributos del nodo.

En la Figura 4.23 se definen los distintos métodos que implementa el servidor. Cada uno de ellos se rige por un propósito específico. Se han contemplado métodos de iniciar la configuración, establecer conexión, añadir los tags, asignar tags, actualizar tags y cerrar el servidor.

```
//-------------------------------------------------------------------------
//                    M É T O D O S   P Ú B L I C O S
//-------------------------------------------------------------------------
    //! @brief Inicializa la aplicación del servidor
    //! @param <Poner los parametros necesarios>
    //! @return <none>
      void initConfiguration();

    //! @brief Finaliza la aplicación del servidor
    //! @param <none>
    //! @return <none>
      void closeServer(void);

    //! @brief Realiza la conexion del servidor
    //! @param <Se indicara tanto el puerto como la direccion IP>
    //! @return <none>
      bool doConexion();

    //! @brief Añade los distintos Tags al servidor OPCUA
    //! @param <Se le pasa como parametros cada uno de los contadores>
    //! @return <none>
      void AddTags(UA_Int32 *cont_m1,UA_Int32 *cont_m2,UA_Int32 *cont_m3,UA_Int32 *cont_m4
```

Figura 4. 23 Métodos públicos de la clase servidor

Algunos de estos métodos modifican las variables, por lo que son necesario pasarlas por dirección.

En cuanto al archivo Server_Opcua.cpp, se han definido en la cabecera las variables globales necesarias en la implementación:

```
//-------------------------------------------------------------
//                          DEFINICION DE CONSTANTES
//-------------------------------------------------------------
using namespace std;
UA_ServerConfig *config=UA_ServerConfig_new_default();
UA_Server *server=UA_Server_new(config);
UA_Boolean running=true;
```

Figura 4. 24 Definición de constantes en la clase servidor

Se ha definido un servidor con una configuración por defecto, y una variable llamada 'running' que permite identificar si el servidor se encuentra en marcha.

La siguiente imagen muestra la implementación del método añadir tags:

```
void Server_Opcua::AddTags(UA_Int32 *cont_m1,UA_Int32 *cont_m2,UA_Int32 *cont_m3,UA_Int32 *cont_m4
    /* Añade tag monedas de 1 centimo a un nodo del servidor*/
    UA_VariableAttributes attr1 = UA_VariableAttributes_default;
    attr1.displayName = UA_LOCALIZEDTEXT("en-US", "monedas 1 centimo");
    attr1.accessLevel=UA_ACCESSLEVELMASK_READ|UA_ACCESSLEVELMASK_WRITE;
    UA_Int32 monedas_1 = 0;
    UA_Variant_setScalar(&attr1.value, &monedas_1, &UA_TYPES[UA_TYPES_INT32]);
    UA_NodeId newNodeId1 = UA_NODEID_STRING(1, "monedas 1 centimo");
    UA_NodeId parentNodeId1 = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId1 = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableType1 = UA_NODEID_NULL; /* take the default variable type */
    UA_QualifiedName browseName1 = UA_QUALIFIEDNAME(1, "monedas 1 centimos");
    UA_Server_addVariableNode(server, newNodeId1, parentNodeId1, parentReferenceNodeId1,
    browseName1, variableType1, attr1, NULL, NULL);
```

Figura 4. 25 Método para añadir tags al servidor

Como se puede observar, el trozo de código representa cómo se añaden un tag correspondiente a una moneda, en este caso de 1 céntimo. Para ello, se debe establecer un atributo distinto para cada tag y un identificador de nombre. El acceso de permiso es de lectura y escritura, con objeto de que tanto el cliente como el servidor puedan leer/escribir en dicho tag.

El nombre de la variable será "monedas_1", y se debe inicializar antes de usarla. Posteriormente, este tag se asocia a un nodo concreto y se establece el browsename.

```
UA_Variant_init(&value1);
UA_Int32  c1=(UA_Int32)(*cont_m1);
UA_Variant_init(&value2);
UA_Int32 c2=(UA_Int32)(*cont_m2);
UA_Variant_init(&value3);
UA_Int32 c3=(UA_Int32)(*cont_m3);
UA_Variant_init(&value4);
UA_Int32 c4=(UA_Int32)(*cont_m4);
UA_Variant_init(&value5);
UA_Int32  c5=(UA_Int32)(*cont_m5);
UA_Variant_init(&value6);
UA_Int32 c6=(UA_Int32)(*cont_m6);
UA_Variant_init(&value7);
UA_Int32 c7=(UA_Int32)(*cont_m7);
UA_Variant_init(&value8);
UA_Int32  c8=(UA_Int32)(*cont_m8);
```

Figura 4. 26 Declaración de los contenedores de tags

Para asignar una variable a un nodo es necesario definir un contenedor. El contenedor puede ser una variable de tipo escalar, una cadena de caracteres o un array.

Una vez realizado el algoritmo de reconocimiento ya se han detectado los tipos de monedas que existen en la superficie de trabajo, por lo que es necesario actualizar el valor de cada uno de los tags en el nodo.

```
void Server_Opcua::TagsAssigment(UA_Int32 *cont_m1,UA_Int32 *cont_m2,UA_Int32 *cont_m3

    *mutex1.lock();
    UA_Variant_setScalarCopy(&value1,cont_m1,&UA_TYPES[UA_TYPES_INT32]);
    UA_Server_writeValue(server,UA_NODEID_STRING(1,"monedas 1 centimo"),value1)

    //*El procesado cuenta las monedas de 1 centimo y actualiza el tag del servidor*/
    UA_Variant_setScalarCopy(&value2,cont_m2,&UA_TYPES[UA_TYPES_INT32]);
    UA_Server_writeValue(server,UA_NODEID_STRING(1,"monedas 2 centimos"),value2);

   //*El procesado cuenta las monedas de 1 centimo y actualiza el tag del servidor*/
    UA_Variant_setScalarCopy(&value3,cont_m3,&UA_TYPES[UA_TYPES_INT32]);
    UA_Server_writeValue(server,UA_NODEID_STRING(1,"monedas 5 centimos"),value3);


    //*El procesado cuenta las monedas de 1 centimo y actualiza el tag del servidor*/
    UA_Variant_setScalarCopy(&value4,cont_m4,&UA_TYPES[UA_TYPES_INT32]);
    UA_Server_writeValue(server,UA_NODEID_STRING(1,"monedas 10 centimos"),value4);
```

Figura 4. 27 Método de asignación de tags

Uno de los problemas que se pueden plantear cuando se comparten variables entre varios hilos de ejecución es la condición de carrera, es decir, un comportamiento indeseado debido a la no sincronización en la actualización de la información. Por ello, es necesario proteger a las variables compartidas mediante un mecanismo llamado mútex. Este mútex protege toda la parte de asignación y actualización de los tags, de la forma que sólo pueda estar un hilo accediendo y modificando dichas variables en un instante de tiempo.

Por último, es necesario liberar la memoria de los contenedores creados en el proceso. Para ello, el método 'freeVariants' se encarga de eliminarlos mediante:

```
void Server_Opcua::freeVariants(){


    UA_Variant_deleteMembers(&value1);
    UA_Variant_deleteMembers(&value2);
    UA_Variant_deleteMembers(&value3);
    UA_Variant_deleteMembers(&value4);
    UA_Variant_deleteMembers(&value5);
    UA_Variant_deleteMembers(&value6);
    UA_Variant_deleteMembers(&value7);
    UA_Variant_deleteMembers(&value8);
```

Figura 4. 28 Borrado de los contenedores

Dentro de los métodos declarados, existe un método de arranque que se encarga de la configuración de parámetros y de la exportación de los GPIOs.

Cada controlador GPIO gestiona 32 señales digitales. El uso de diodos leds como salidas digitales en la OdroidXU4 conlleva utilizar la tecnología GPIO (General Pin Input/Output), es decir, pines de entrada/salida con propósito general. Como los pines del encapsulado del procesador comparten funciones (pinmux), sólo hay acceso a un conjunto limitados de señales E/S.

En general, cada pin del procesador puede desempeñar distintas funciones (pinmux). Las funciones por defecto de los pines se pueden cambiar accediendo al módulo de control del procesador.

A la hora de realizar el conexionado eléctrico sin dañar la Odroid, hay que tener en cuenta las siguientes limitaciones eléctricas para los diferentes pines:

- El nivel de tensión de 0V (tierra) se encuentra en los pines 2, 28 y 30.
- El nivel de tensión de 5V, disponible siempre que la Odroid se encuentre en funcionamiento, se puede extraer del pin 1.
- El nivel de tensión de 1.8V se obtiene del pin 29, cuya corriente máxima disponible es de 250mA.

Los pines GPIO funcionan a una tensión máxima de 3.3 V, por lo que no se conectan a 5V. Además, como los pines GPIO pueden absorber una corriente máxima limitada, es necesario utilizar resistencias limitadoras. Para el caso de los LEDs implementados, las resistencias utilizadas han sido de 1K.

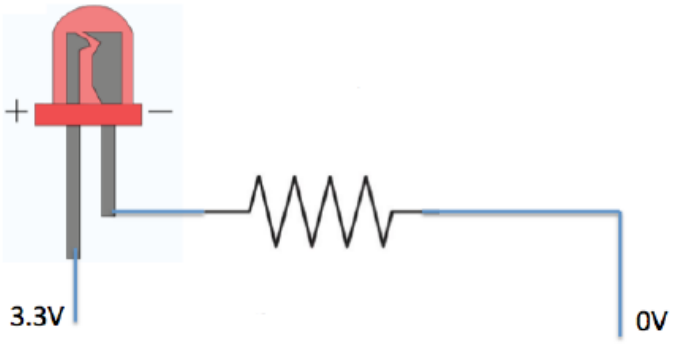


Figura 4. 29 Esquema básico de conexión de un LED

Sin embargo, este esquema no es muy eficaz cuando se requiere controlar una carga. Es por ello por lo que surge la necesidad de aislar el circuito de conmutación utilizando un transistor. Se ha utilizado un transistor MOSFET capaz de controlar cargas de 60V y 2A. Estas características serán suficientes para el propósito deseado. El esquema de montaje se muestra a continuación:

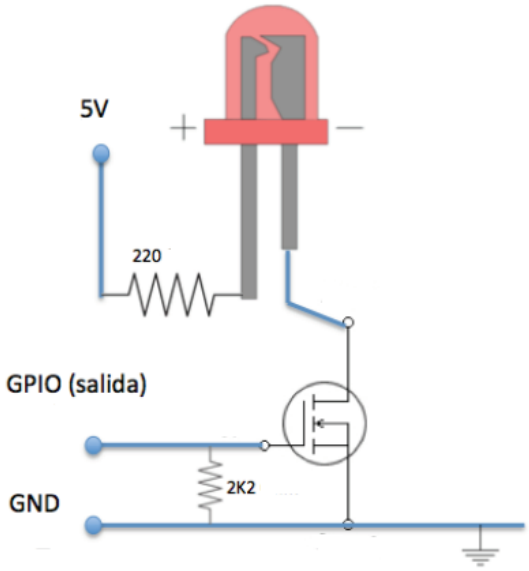


Figura 4. 30 Montaje del LED con transistor en un GPIO

Tal y como se ha mencionado anteriormente, los pines de la Odroid poseen múltiples configuraciones, por lo que es necesario configurar aquellos que se deseen utilizar con dicha función. El SO de Ubuntu Mate incluye los manejadores de dispositivos necesarios para manejar cómodamente los pines GPIO.

El usuario debe leer y escribir en los archivos situados en la ruta: /sys/class/gpio , y a partir de ahí, gestionar mediante línea de comandos las diferentes entradas y salidas digitales. Los valores de interés del "sysfs" de los pines GPIO que se han utilizado son:

- Direction: define la dirección de un pin, ya sea como entrada (in) o salida (out).
- Value: es el valor del pin. Puede adoptar dos valores. 0 como nivel bajo y 1 como nivel alto. Si el pin se encuentra configurado como salida, se puede escribir el valor.

Cabe destacar que la numeración de los pines físicos de la Odroid no coincide con el número asociado de los GPIOs localizados en el sysfs. Esto es importante, pues un error eléctrico de conexión puede provocar la inhabilitación total del pin. Por ello, se ha consultado el manual del dispositivo, donde se muestra la tabla de conversión del header:

WiringPi GPIO#	NAME(GPIO#)			NAME(GPIO#)	WiringPi GPIO#
	5.0 V Power	1	2	Ground	
AIN0	ADC_0.AIN0 (ADC#0)	3	4	UART_0.CTSN (#173)	1
0	UART_0.RTSN (#174)	5	6	UART_0.RXD (#171)	16
12	SPI_1.MOSI (#192)	7	8	UART_0.TXD (#172)	15
13	SPI_1.MISO (#191)	9	10	SPI_1.CLK (#189)	14
10	SPI_1.CSN (#190)	11	12	PWRON(Input 1.8V ~ 5V)	
2	GPIO (#21)	13	14	I2C_1.SCL (#210)	9
7	GPIO (#18)	15	16	I2C_1.SDA (#209)	8
3	GPIO (#22)	17	18	GPIO (#19)	4
22	GPIO (#30)	19	20	GPIO (#28)	21
26	GPIO (#29)	21	22	GPIO (#31)	23
AIN3	ADC_0.AIN3 (ADC#3)	23	24	GPIO (#25)	11
5	GPIO (#23)	25	26	GPIO (#24)	6
27	GPIO (#33)	27	28	Ground	
	1.8 V Power	29	30	Ground	

Figura 4. 31 Tabla de pines de la OdroidXU4

El LED que muestra el estado de adquisición de imágenes se representará mediante color azul Para esta indicación se ha usado el gpio29, correspondiente al pin número 21 de la placa. El led que indica el intercambio de información entre cliente y servidor será de color naranja, localizado en el pin 20. Una vez configurados. Tras iniciarse la cámara, el resultado se puede observar en la Odroid en la siguiente imagen:

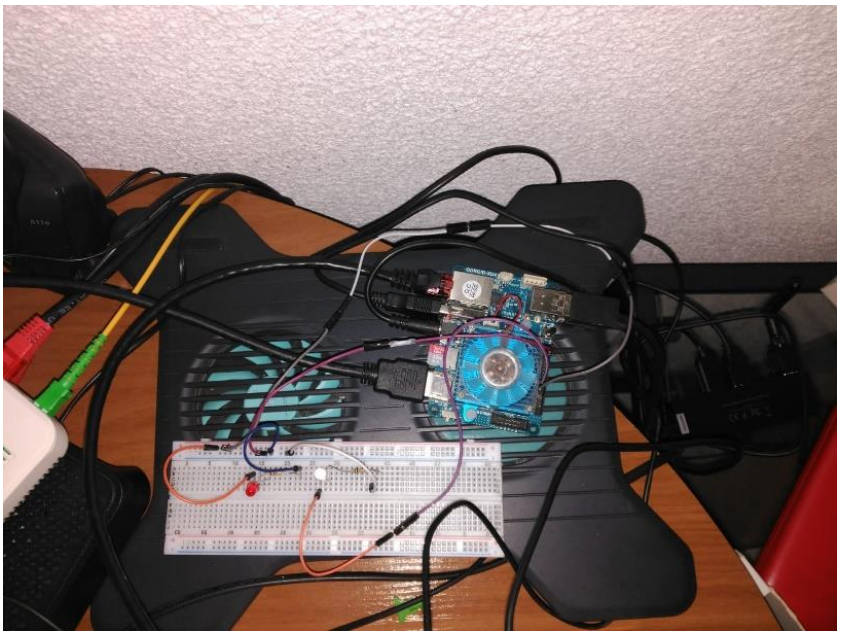


Figura 4. 32 Activación de los GPIOs

4.5 IMPLEMENTACIÓN DEL CLIENTE OPCUA

La implementación del cliente se ha realizado de forma equivalente al servidor. El código se encuentra dividido en varios métodos. Para una mejor visualización, la ejecución del cliente se realiza de forma paralela al servidor, es decir, en otro terminal.

Para lograr una mayor seguridad en la conexión, se ha establecido un método de autentificación en el que el cliente debe introducir sus credenciales: usuario y contraseña. De esta manera, si alguno de las credenciales es erróneo, el programa no permite realizar la conexión al servidor. Al igual que sucede con el servidor, es necesario declarar unos contenedores de tags así como la inicialización de estos, en los que cada contenedor estará asociado a un valor de una moneda.

```
int main(int argc, char *argv[])

{
signal(SIGINT, stopHandler);
UA_StatusCode status;
UA_ClientConfig config= UA_ClientConfig_default;
config.timeout=1000;
UA_Client *client = UA_Client_new(config);
UA_Variant value;
UA_Variant_init(&value);
UA_Variant value1;
UA_Variant_init(&value1);
UA_Variant value2;
UA_Variant_init(&value2);
UA_Variant value3;
UA_Variant_init(&value3);
UA_Variant value4;
UA_Variant_init(&value4);
UA_Variant value5;
UA_Variant_init(&value5);
UA_Variant value6;
UA_Variant_init(&value6);
UA_Variant value7;
UA_Variant_init(&value7);
```

Figura 4. 33 Código del main del cliente

El procedimiento de ejecución del cliente se basa en un bucle infinito, es decir, el cliente siempre estará ejecutando el mismo algoritmo cuando se encuentre conectado. Este bucle se basa en la condición de la variable 'running'.

Posteriormente, se declara una variable de estado que permite identificar si la conexión se realiza con éxito. La función de conexión tiene como argumentos el propio cliente, la dirección del hostname y el puerto en el que se realiza la conexión. Por defecto el servidor OPC-UA se conecta en el puerto 4.840.

```
while(running)
{
system("clear");//Limpia Pantalla
UA_StatusCode status = UA_Client_connect(client, "opc.tcp://localhost:4840");//Trata De Establecer Conexión
if(status != UA_STATUSCODE_GOOD) {//Si Falla Conexión Parpadea y Avisa
            UA_LOG_ERROR(logger, UA_LOGCATEGORY_CLIENT, "Not connected. Retrying to connect in 1 second");
            usleep(100000);
            continue;
        }
if(status != UA_STATUSCODE_GOOD) {//Si Falla Conexión Elimina Cliente
UA_Client_delete(client);
return status;
}

if(status = UA_STATUSCODE_GOOD) {//Si Conexión Con Exito , Actualiza Datos cada segundo

            UA_LOG_ERROR(logger, UA_LOGCATEGORY_CLIENT, "Conectado con éxito al servidor");

            usleep(100000);//Intevalo de Actualización ns

            continue;

        }
```

Figura 4. 34 Bucle de ejecución del cliente

A continuación, se evalúan una serie de condiciones para ir realizando de forma secuencial tanto la inicialización como la asignación de tags. La lectura de los tags del servidor según sus nodos se realiza analizando los valores que puede adoptar la variable 'status'. El cliente leerá los valores de los atributos para cada tipo de moneda.

```
status = UA_Client_readValueAttribute(client, UA_NODEID_STRING(1, "monedas 1 centimo"), &value);
if(status == UA_STATUSCODE_GOOD &&
UA_Variant_hasScalarType(&value, &UA_TYPES[UA_TYPES_INT32])) {
cout<<  "Monedas 1 centimo: "<< *(UA_Int32*)value.data<<endl;

}
```

Figura 4. 35 Lectura de los tags del servidor por parte del cliente

Si en algún momento la conexión entre cliente y servidor falla, el cliente tratará de reconectarse. Esto forma parte también de un mecanismo de seguridad, mostrándose los mensajes de error por pantalla.

```
if(status == UA_STATUSCODE_BADCONNECTIONCLOSED) {
            UA_LOG_ERROR(logger, UA_LOGCATEGORY_CLIENT, "Connection was closed. Reconnecting ...");
            continue;
        }
```

Figura 4. 36 Reconexión del cliente

Cada interacción del bucle del cliente tiene un tiempo de espera predefinido por el usuario. Actualmente dicho valor es de 0.1 segundo, es decir, el cliente actualiza sus valores en dicho tiempo. Después, se eliminan de la memoria las variables de los contenedores y el cliente.

Por último, se ha registrado en el servidor un tag que contiene la imagen extraída por la cámara en escala de grises. Los contenedores del servidor OPCUA pueden almacenar matrices unidimensionales. Para poder almacenar una imagen en un tag del servidor OPCUA hay que pasarla de matriz bidimensional conteniendo los pixels (capturada por la cámara o bien de un fichero) a matriz unidimensional.

Para ello, es necesario dimensionar tanto las filas como columnas de la imagen, así como el tipo de dato correspondiente al píxel de la imagen.

```
        UA_VariableAttributes vAttr = UA_VariableAttributes_default;
vAttr.arrayDimensions=(UA_Int32*)UA_Array_new(2,&UA_TYPES[UA_TYPES_INT32])
                    vAttr.arrayDimensions[0]=imagen.rows;
                    vAttr.arrayDimensions[1]=imagen.cols;
                       vAttr.arrayDimensionsSize = 2;
  vAttr.displayName = UA_LOCALIZEDTEXT("en-US", "matriz bidimensionall");
  vAttr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;
```

Ahora, se inicializa la matriz con las filas y columnas>:

```
                UA_Int32  matriz [imagen.rows*imagen.cols]] ;
```

```
int i;

for(i=0;i<imagen.rows*imagen.cols;i++)

{matriz[i]=0;}
```

Después, se asigna la matriz a la variable del tag. Una vez creado el modelo del tag que soporta el array y capturada la imagen, se pasa al tag a través de la función del servidor write.

El último paso consiste en declarar un array unidimensional para almacenar la imagen y copiar el array al contenedor.

Capítulo 5

RESULTADOS

En este capítulo se muestran los resultados principales que se han obtenido tras la implementación del código completo.

Con objeto de que la visualización de los resultados sea más clara, se ejecuta el código en dos terminales distintas. La primera de ellas constituye la implementación del cliente y la segunda la implementación del servidor.

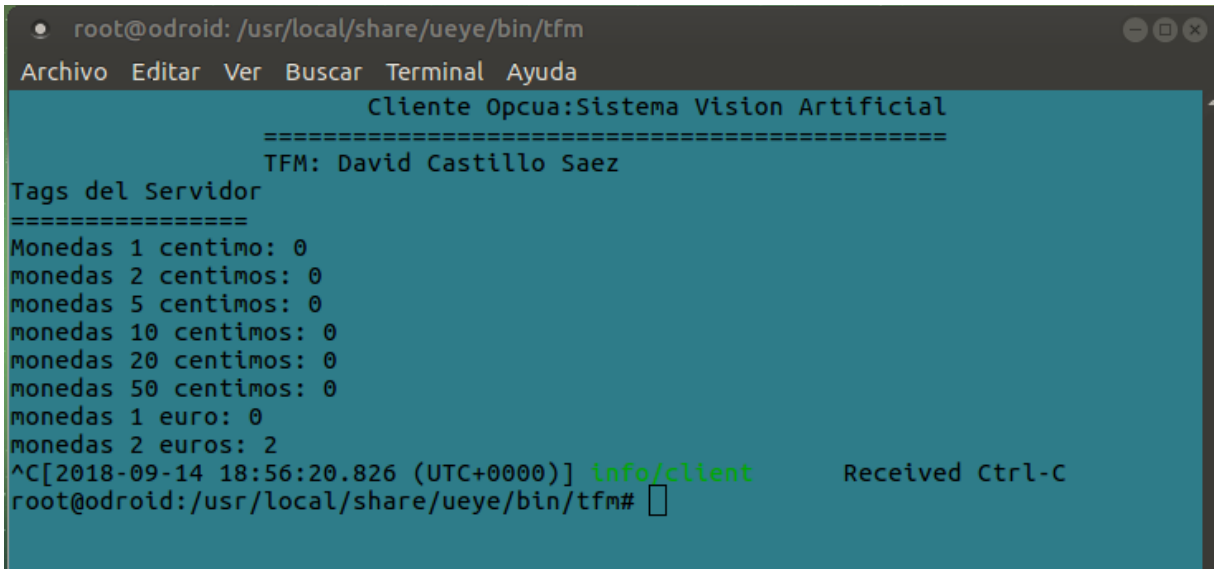


Figura 5. 1 Terminal de implementación del cliente OPC-UA

```
root@odroid:/usr/local/share/ueye/bin/tfm# ./tfm
Comienzo del Proceso. Iniciando Configuracion...
[2018-09-14 18:56:05.721 (UTC+0000)] info/network       TCP network layer listen
ing on opc.tcp://odroid:4840/
Starting ueyeusbd... ueyeusbd is already running.
Por Favor, Espere
echo: write error: Device or resource busy
echo: write error: Device or resource busy
echo: write error: Device or resource busy
CONFIGURACION DE ARRANQUE CON EXITO
[2018-09-14 18:56:05.981 (UTC+0000)] info/session       Connection 0 | SecureCha
nnel 0 | Session 00000001-0000-0000-0000-000000000000 | AddNodes: No TypeDefinit
ion; Use the default TypeDefinition for the Variable/Object
[2018-09-14 18:56:05.981 (UTC+0000)] info/session       Connection 0 | SecureCha
nnel 0 | Session 00000001-0000-0000-0000-000000000000 | AddNodes: No TypeDefinit
ion; Use the default TypeDefinition for the Variable/Object
[2018-09-14 18:56:05.982 (UTC+0000)] info/session       Connection 0 | SecureCha
nnel 0 | Session 00000001-0000-0000-0000-000000000000 | AddNodes: No TypeDefinit
ion; Use the default TypeDefinition for the Variable/Object
```

Figura 5. 2 Inicio del proceso del Servidor OPC-UA

En la imagen anterior se muestra la información del servidor al inicio del programa, así como un aviso del establecimiento de un canal seguro de comunicaciones. A continuación, se inicia la cámara y los algoritmos de adquisición y reconocimiento.

```
Camera initialized!
Camera color mode succesfully set!
SE VA A INICIAR EL PROGRAMA...
Status displayMode 0
Status is_FreezeVideo0
Status is_ImageFile0
Status is_ImageFile0
Status is_ImageFile0
Execution Time: 1.41622
Numero de Monedas Detectadas : 2
Numero de contornos: 1
MONEDA 1es de 2 EUROS: Diametro :
27.18
MONEDA 2es de 2 EUROS: Diametro :
25.972
 Existen 0 monedas de 1 centimo
 Existen 0 monedas de 2 centimos
 Existen 0 monedas de 5 centimos
 Existen 0 monedas de 20 centimos
 Existen 0 monedas de 1 euro
 Existen 2 monedas de 2 euros
Los TAGS se han añadido correctamente
CAMBIA LAS MONEDAS DE POSICION
```

Figura 5. 3 Adquisición y Reconocimiento de monedas

En las imágenes anteriores, se muestra como el cliente actualiza el número de monedas detectado por el servidor.

En la imagen siguiente se cancela la ejecución del cliente con la orden signal.

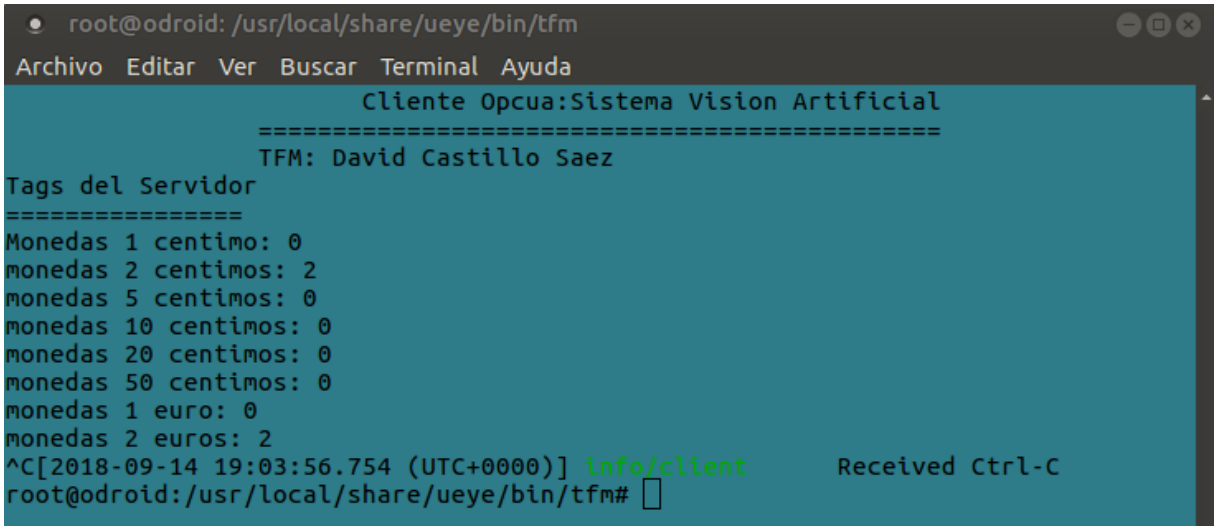


Figura 5. 4 Cancelación del cliente con la orden signal

Capítulo 6

CONCLUSIONES

Este trabajo fin de máster engloba todo el ámbito relacionado con la parte de informática industrial impartida en este máster, así como los conceptos relacionados con la visión artificial adquiridos en la asignatura de visión por computador.

El comienzo del trabajo ha resultado bastante tedioso, puesto que la puesta en marcha de un entorno siempre ocasiona distintos problemas que no se solucionan de una forma trivial. Esto entra dentro del concepto de los sistemas empotrados, en los que cada sistema tiene un hardware diferente y es necesario adaptarse a él. Por otra parte, la instalación y configuración de cada uno de los softwares requiere especial atención, ya que al haberse realizado de forma correcta ha permitido una localización sencilla de los distintos archivos, tanto para el usuario como para cualquier persona externa que comience a trabajar con él.

Respecto al algoritmo de reconocimiento de monedas, se han logrado identificar cada uno de los tipos de monedas seleccionados para esta aplicación en función de su diámetro. Esta idea parece relativamente sencilla cuando uno plantea el problema por primera vez. Sin embargo, la diferencia de tamaño entre las monedas no supera los 2mm en la gran mayoría de los casos. Por tanto, se trata de una aplicación en la que un mínimo error en la adquisición de imágenes puede provocar fallos en el reconocimiento de las monedas.

Es por esta razón que uno de los puntos más críticos de este trabajo fin de máster son las condiciones ambientales a las que se encuentra sometida la estación de trabajo. Se requiere de un color y textura que no refleje, así como una luz natural no directa que produzca sombras. Este problema puede eliminarse con la introducción de sistemas de iluminación propios en sistemas de fotografía, aunque en la práctica no se han implementado por falta de medios.

El cliente y servidor han hecho uso de la mayoría de las características expuestas en el capítulo teórico sobre el protocolo OPC-UA. El cliente es capaz de conectarse al servidor, y en caso de que surja algún problema, reintenta la conexión. Se ha emulado un comportamiento en el cual el servidor realiza el procesamiento de la información y se la muestra al cliente.

Una de las ventajas del código implementado es su estructura. Se encuentra divido en varias clases, en las que cada una consta de su librería y su archivo principal. Además, cada método y función se encuentran debidamente comentadas, facilitando la lectura y compresión del usuario. Por último, el código se encuentra parametrizado, pudiéndose ampliar en un futuro de una manera sencilla.

En cuanto a las posibles líneas futuras de este trabajo, existen numerosas aplicaciones relacionadas con la visión artificial que se pueden implementar con el cliente y servidor. El método se ha realizado a través de capturas de imágenes, pero se podría realizar mediante vídeo en directo, detectar posibles diferencias entre imágenes o reconocimiento facial.

La librería que implementa el cliente y servidor se encuentra en constante evolución al tratarse de un código abierto. En este trabajo se comenzó a trabajar con la primera versión, finalizando con tercera versión, siendo ésta la actual. Una de las mejoras podría ser la conexión basada en la suscripción, así como la utilización de los plugins de encriptación. Este método no se ha implementado al tratarse de una comunicación asíncrona, pero dependiendo de la aplicación a implementar podría ser un punto de vista a considerar.

BIBLIOGRAFÍA

[1] José Simó (2017). Interfaces Físicos y Sistemas Empotrados: instalación del entorno. Universidad Politécnica de Valencia

[2] José Simó (2018). Interfaces Físicos y Sistemas Empotrados Preparación de la imagen del sistema. Universidad Politécnica de Valencia.

[3] Procolo OPC-UA [Online]. Disponible: http://documentation.unified-automation.com/uasdkcpp/1.5.4/html/index.html

[4] "Ueye XS" [Online]. Disponible: https://es.ids-imaging.com/xs-home.html.

[5] Rob Roy, Venkat Bommakanti (2017). User Manual OdroidXU4. South Korea: HardKernel.