Diseño de Software - Informe

CONTROL DE VERSIÓN

|  |  |  |  |
| --- | --- | --- | --- |
| VERSIÓN | FECHA | COMENTARIO | ARCHIVO |
| 1.2 | 07/06/2022 |  |  |

INDICE

[1. Introducción 2](#_Toc105562214)

[2. Programas y librerías utilizados 2](#_Toc105562215)

[3 . Interfaz de Usuario 3](#_Toc105562216)

[3.1 Diseño Gráfico de la Interfaz 3](#_Toc105562217)

[3.1.1 Idea Preliminar 3](#_Toc105562218)

[3.1.2 Resultado Final 5](#_Toc105562219)

[3.2 Arquitectura del Software 7](#_Toc105562220)

[3.2.1 Programa de la App de MATLAB 7](#_Toc105562221)

[3.2.2 Programa Clase de Control Principal 12](#_Toc105562222)

[3.2.4 Programa Clase LibXimc 16](#_Toc105562223)

[4. Reality Capture 18](#_Toc105562224)

[4.1 Procesado de Imágenes 19](#_Toc105562225)

[4.1.1 Orientación de imágenes 19](#_Toc105562226)

[4.1.2 Reconstrucción 3D 20](#_Toc105562227)

[4.1.3 Texturizado 20](#_Toc105562228)

[4.2 Pruebas 21](#_Toc105562229)

[4.2.1 Prueba desenfoque de imágenes 21](#_Toc105562230)

[4.2.2 Prueba reemplazo de imágenes 22](#_Toc105562231)

[5. Unreal Engine 24](#_Toc105562232)

[5.1 Lumen 24](#_Toc105562233)

[5.2 Nanite 24](#_Toc105562234)

[5.3 Virtual Shadow Map 25](#_Toc105562235)

[6. Bibliografía 26](#_Toc105562236)

# 1. Introducción

Este capítulo describe el proceso llevado a cabo para diseñar e implementar el software de control para el sistema autónomo de captación de imágenes. Se describe la arquitectura de software y el proceso de diseño de la interfaz de usuario, necesaria para la comunicación entre el proceso y el operario. Se detalla además el funcionamiento del código y su comunicación con la app desarrollada. También se investiga el proceso de creación de modelos 3D de la figura objetivo a partir de conjuntos de datos de imágenes, así como la posible automatización de este proceso. Por último, se especifican las pruebas experimentales que se han realizado para determinar posibles fallos en la toma de imágenes e identificar soluciones para optimizar el modelo emitido.

# 2. Programas y librerías utilizados

MATLAB ha sido elegido como programa para el desarrollo de la interfaz de usuario. Esto se debe a la facilidad que dispone para crear interfaces gráficas y para el procesamiento de sistemas de visión artificial. El uso de MATLAB también supone una facilidad añadida a la hora de realizar experimentos con componentes de laboratorio. Varios *add-ons* han sido utilizados para controlar distintos aspectos del sistema. En primer lugar, el *add-on* de control para Arduino ha sido utilizado para controlar el sistema preliminar de iluminación en el prototipo inicial. En segundo lugar, *Image Acquisition Toolbox* junto con la librería *TISimaq* han sido usados para captar las imágenes de la cámara y guardarlas en el PC de destino. En tercer lugar, la función *vrjoystick* ha sido utilizada para realizar la conexión con el joystick y captar los datos de posición asociados a este. Por último, se ha utilizado una librería especializada llamada *libximc* para el control de los motores lineares. Esta librería está escrita en C y sus funciones son llamadas en MATLAB por medio de la función *calllib*.

En cuanto al procesamiento de imágenes, la generación de modelos 3D de las piezas objetivo se realizan utilizando *Reality Capture*. Se trata de un software de fotogrametría capaz de identificar puntos homólogos entre una colección de fotografías introducidas, crear una nube de puntos, y relacionar estos puntos mediante vectores para completar la geometría y crear el modelo 3D, así como la textura final. Este modelo puede entonces pasarse a otro software de procesado 3D como *Unreal Engine*, donde se puede renderizar la pieza. Este proceso fotogramétrico puede automatizarse mediante la programación de un algoritmo que envíe una cadena de comandos en CLI (Command-Line Interface) a *Reality Capture*.

# 3 . Interfaz de Usuario

Se ha diseñado e implementado en MATLAB una interfaz de usuario para facilitar el control manual de todos los componentes del sistema por parte del operador, así como la puesta en funcionamiento del sistema autónomo de captación de imágenes. Este aparatado detallará el proceso de diseño gráfico de la interfaz, además de exponer el funcionamiento programático del sistema de *Callbacks* que hacen funcionar los diferentes apartados de la app de MATLAB, junto con el funcionamiento de las clases de control del sistema, y de comunicación con la librería de control de los motores.

## 3.1 Diseño Gráfico de la Interfaz

En esta sección se describe el proceso de diseño gráfico de la interfaz de usuario. Se empieza explorando la elaboración de un modelo preliminar de la interfaz, para finalmente comparar este diseño inicial con el resultado final.

### 3.1.1 Idea Preliminar

Un proceso sistémico fue utilizado para el desarrollo de la interfaz de usuario. El sistema debía ser capaz de cumplir con los siguientes objetivos: Dar acceso al control manual de los motores, permitir al usuario controlar manualmente la longitud de onda de la luz proyectada sobre la obra, disponer al operador de un modo de variación de los parámetros de la cámara, incluyendo la velocidad de obturación, apertura, ISO y distancia de campo (control del foco), así como procurar la habilidad de tomar fotos, y facilitar la introducción de los parámetros iniciales para el inicio del sistema autónomo de toma de fotos. Se identificó que estos objetivos podían agruparse en cuatro funciones básicas: el control de los motores, el control del sistema de iluminación, el control del sistema de toma de fotos, y el control del sistema autónomo. Basado en estas funciones, se desarrolló el diseño inicial mostrado en la Imagen 3.1.

La interfaz se compone de cuatro zonas principales, que se encargan individualmente de cumplir cada una de las cuatro funciones. La zona de control de los motores se encarga de dar acceso al usuario al control manual de los actuadores lineares. Está compuesta por tres partes: la gráfica de posición de los ejes X e Y, la gráfica de posición del eje Z y los recuadros de *input* de la posición del efector. Las gráficas de posición de los ejes proporcionan al usuario la posición actual del efector en los ejes correspondientes, también

|  |
| --- |
|  |
| Imagen 3.1: Esquema de Idea Preliminar para la Interfaz de Usuario. Fuente: Elaboración propia |

incluyen dos botones en cada eje: uno para aumentar el valor de posición, y otro para disminuirlo. El usuario también es capaz de seleccionar un punto de la gráfica con el ratón, al cual se moverá el efector. Las gráficas están en milímetros. Los recuadros de input de la posición del efector sirven para permitir al usuario introducir una posición tridimensional, a la cual se desplazará el efector. Dispone de dos botones: “Actualizar Posición”, el cual es presionado para activar el movimiento, y “STOP”, el cual detiene el movimiento.

La zona de control del sistema de iluminación solo dispone de tres componentes: el *slider* de valores de longitud de onda, el LED de estado y el botón de “Secuencia”. El *slider* se utiliza para cambiar la longitud de onda de la luz producida por la luminaria utilizada, el LED de estado se activa cuando la luz está encendida y el botón de “Secuencia” activa un *loop* que enciende la luminaria en todas las longitudes de onda disponibles una a una.

La zona de control del sistema de toma de fotos se compone de tres partes. La primera parte sirve para ajustar los valores de Velocidad de Obturación, Apertura e ISO. El usuario puede desplazar el *slider* para cambiar los parámetros. La siguiente parte controla la distancia de campo proporcionada por el foco de la cámara. De la misma, este valor se ajusta utilizando una barra deslizante. Esta sección también incluye un botón de “Actualizar”, que mueve el foco a la posición especificada, y un botón de “AutoFocus”, que adapta el foco automáticamente a la geometría de la figura sujeto. La última parte comprende el sistema de captura, compuesto por dos recuadros donde se introduce el nombre de la imagen y la localización de destino del archivo; y por un botón de “Captura”, que al ser presionado realiza una fotografía con los parámetros establecidos.

La última es la zona de control del sistema autónomo. Este conjunto de componentes está diseñado para permitir al usuario introducir los parámetros iniciales con los cuales el sistema empezará el funcionamiento del proceso automático de toma de fotos. Está compuesto por tres partes. La primera parte se encarga de disponer al usuario de una interfaz para introducir tres grupos de coordenadas tridimensionales. El primer parámetro es la “Posición Esquina de Cuadricula”, el cual se refiere a la posición en milímetros en el cual se encuentra la esquina del cuadro bidimensional desde la cual debe iniciarse la toma de datos. El segundo parámetro es el “Tamaño de Cuadrícula”. Se refiere al tamaño en pasos del marco cuadrangular en el cual se tomarán las fotos. El último parámetro es el “Tamaño de Pasos”. Sirve para introducir la distancia que cubrirá cada paso en cada una de las tres dimensiones. La segunda parte de la zona consiste en una gráfica donde se muestra la posición actual del efector, y se visualiza el recorrido que ha llevado a cabo el sistema. Por último, la tercera parte se compone de un recuadro donde se puede introducir la localización de destino de los archivos, y de dos botones: “Start”, el cual empieza el sistema autónomo, y “Stop”, el cual lo detiene.

### 3.1.2 Resultado Final

El sistema diseñado es implementado en MATLAB utilizando *App Designer*. Este es un programa incluido en MATLAB que permite al desarrollador crear aplicaciones e interfaces de usuario para toma de datos, interacción con actuadores, simulación y representación gráfica de datos a tiempo real. El resultado se expone en la Imagen 3.2.

Como se puede apreciar la mayoría de componentes se han dispuesto de la forma diseñada. La interfaz se divide en cuatro secciones de la misma forma que el diseño preliminar. Las principales diferencias son las siguientes. En el recuadro de control de motores se han añadido dos botones, uno para activar el control por Joystick, y otro para pausarlo. Cuando el joystick es activado, el botón de activación es utilizado para terminar por completo el envío de comandos a los motores. Los *spinners* que se utilizan para introducir la coordenada a la cual el efector debe moverse también actualizan su valor a tiempo real para mostrar la posición actual de los motores.

|  |
| --- |
| Graphical user interface, application, Excel  Description automatically generated |
| *Imagen 3.2: Interfaz de Usuario Final elaborada en App Designer. Fuente: Elaboración propia* |

En el apartado de iluminación se ha añadido también un *spinner* paraintroducir el valor de espera entre el encendido de LEDS en la secuencia. El control de la cámara se ha implementado de igual manera excepto por dos componentes: el foco se controla mediante la especificación del número de pasos que debe dar el servo integrado al objetivo y la espera entre pasos; y se ha añadido un botón para abrir y cerrar el Live View, que enseña una vista preliminar de la información que recibe la cámara.

Finalmente, en cuanto al apartado de control del sistema autónomo, se ha añadido una barra de progreso, que enseña el número de fotografías realizadas y restantes para terminar la toma; los *switches* para activar el sistema multiespectral y el control de la base rotatoria (para el escáner de esculturas); y los *spinner* para introducir un punto inicial de continuación, en caso de que la toma de datos se pare de forma imprevista y deba ser reanudada. No se ha incluido la gráfica donde se muestra la posición de la cámara dentro del recuadro objetivo.

## 3.2 Arquitectura del Software

En este apartado trata la estructura del sistema de software que permite el funcionamiento de la interfaz de usuario. Como se ha descrito en el apartado anterior, la interfaz ha sido implementada utilizando *App Designer.* Las apps creadas con este programa funcionan a través de un sistema de *Callbacks.* Esto significa que cada componente de la interfaz tiene una función asociada que es ejecutada cuando el usuario interactúa con el componente. En este caso, se han creado dos clases de MATLAB para simplificar el código: *PhotogrammetryRobot.m* y *LibXimc.m*. *PhotogrammetryRobot.m* se encarga del control de todos los componentes del robot. Al llamar una instancia de esta clase, se crea un nuevo objeto ‘robot’, al cual se le pueden enviar comandos para activar elementos individuales del robot, o empezar procesos donde se coordinen varios elementos. *LibXimc.m* se encarga del control básico de los motores. Contiene varias funciones especializadas para inicializar los elementos, cambiar parámetros y enviar comandos. Para esto, se comunica con la librería *libximc.dll*, la cual está escrita en C.

### 3.2.1 Programa de la App de MATLAB

La App desarrollada contiene métodos privados, los cuales son funciones que solo se utilizan internamente; métodos estáticos, los cuales son funciones que no requieren de la introducción de una referencia propia como argumento; y los *callbacks*, los cuales son funciones que se ejecutan al interactuar con los componentes. A continuación se exponen algunas estas funciones principales.

##### Inicialización del programa

|  |  |
| --- | --- |
| *Text  Description automatically generated* | *A screenshot of a computer  Description automatically generated with medium confidence* |
| *Imagen 3.3: Función StartUp. Fuente: Elaboración propia.* | *Imagen 3.4: Función privada initializevalues. Fuente: Elaboración propia.* |

La inicialización del programa se lleva a cabo por dos funciones: startupFcn e initializevalues. startupFcn es un *callback* nativo a *App Designer* que se ejecuta automáticamente al iniciarse la aplicación. Es el primer programa en ejecutarse. Como se puede apreciar en al Imagen 3.3, tan solo hace dos operaciones: crea una instancia de la clase *PhotogrammetryRobot.m* (llamándola *pgrobot*) y ejecuta la función *initializevalues (Imagen 3.4).* Esta segunda función es una función privada que se encarga de inicializar los valores iniciales de los diversos componentes de la App.

Utiliza otras dos funciones para funcionar: *flaglampstatus* (la cual es una función estática) y *updatecomponents* (la cual es otra función privada).

##### Funciones privadas y estáticas

Todas las funciones han sido diseñadas para cumplir un objetivo simple cada una. Como se puede apreciar en la Imagen 3.5, la función *updatecomponents* simplemente actualiza el valor de todos los *spinner* para cada uno de los ejes, para finalmente actualizar el gráfico que muestra la posición de la cámara. Cada una de estas actualizaciones se realiza llamando a otras funciones, como por ejemplo *updatespinnervalue* (Imagen 3.7). Como se puede ver, esta función lee el estado calibrado del eje introducido desde el objeto *pgrobot*, para después ajustar el valor del *spinner* igualándolo al valor leído, siempre que esté entre los límites establecidos.

|  |  |
| --- | --- |
|  |  |
| *Imagen 3.5: Función updatecomponents. Fuente: Elaboración propia.* | *Imagen 3.6: Función flaglampstatus. Fuente: Elaboración propia.* |
|  | |
| *Imagen 3.7: Función updatespinnervalue. Fuente: Elaboración propia.* | |

Las funciones estáticas desarrolladas para esta *App* ejecutan operaciones más simples. Por ejemplo, como se puede apreciar en la Imagen 3.6, *flaglampstatus* se encarga de cambiar el color del componente lámpara introducido a rojo o verde, dependiendo de si el estado de la variable booleana ‘*flag’* introducida es negativo o positivo, respectivamente. Otra función privada interesante es la función *setautosystem* (Imagen 3.8). Esta función lee los valores de todos los componentes involucrados en el ‘setup’ del sistema autónomo, los asigna a las variables correspondientes, y los envía al objeto *pgrobot* ejecutando a su vez la función *pgrobot.startautosystem,* encontrada dentro de *PhotogrammetryRobot.m.* A su vez, si el *switch* de base rotatoria está en posición ‘On’ el programa ejecutará la función *pgrobot.startcircularautosystem* en su lugar.

|  |
| --- |
|  |
| *Imagen 3.8: Función setautosystem. Fuente: Elaboración propia.* |

##### Callbacks

Las funciones *callback* se ejecutan al pulsar un botón específico o interactuar con un *slider* o *spinner* correspondiente. Para esta *App,* han sido diseñadas para ser lo más concisas posible. Como se puede apreciar, se hace uso de funciones privadas para simplificar el código. Por ejemplo, *updatecomponents* y *updatespinnervalue* se utilizan en numerosas ocasiones.

|  |  |
| --- | --- |
|  |  |
| *Imagen 3.9: Función updatebuttonvaluechanged. Fuente: Elaboración propia.* | *Imagen 3.10: Funciones movimiento lateral. Fuente: Elaboración propia.* |

En la Imagen 3.9 se exhibe el *callback ‘updatebuttonvaluechanged’*. Este callback se activa cuando se presiona el botón “Nueva Posición”. La función lee los valores de los tres spinner con las coordenadas objetivo, y ejecuta la función *pgrobot.moveefector*, que mueve los tres motores simultáneamente para llevar al efector a la posición de destino.

Otros *callbacks* interesantes son los enseñados en la Imagen 3.10. Estas funciones se activan al presionar los botones “más (+)” y “menos (-)” de cada eje. El método ejecuta el comando “leftmotor” o “rightmotor” desde el objeto *pgrobot.* A continuación, utiliza la función *updatespinnervalue* para actualizar la posición del slider y spinner correspondientes al eje (‘Z’ en este caso).

|  |  |
| --- | --- |
|  |  |
| *Imagen 3.11: Función joystickmotorbuttonpuched. Fuente: Elaboración propia.* | *Imagen 3.12: Botón empezar control joystick* |

La función *joystickcontrolbuttonpushed* (Imagen 3.11) sirve para activar o desactivar el control de los motores utilizando el joystick. Se activa al pulsar el botón mostrado en la Imagen 3.12. El *callback* cambia la apariencia y disponibilidad de uso del botón al detectar un cambio de valor. Después, utiliza la función *joystickmotorcontrol* desde el objeto *pgrobot* para empezar a leer la posición del joystick y transmitirla a los motores. Como se puede apreciar, el componente del botón es pasado mediante la variable “components” a la función secundaria. Esto es porque su estado es comprobado por dicha función periódicamente para detener su propia ejecución si es necesario.

|  |
| --- |
|  |
| *Imagen 3.13: Función capturebuttonpushed. Fuente: Elaboración propia.* |

La Imagen 3.13 enseña el código de la función accionada al pulsar el botón “Capturar Fotografía”. Como se aprecia, el sistema lee los parámetros necesarios y utiliza la función *pgrobot.capture* para realizar la fotografía utilizando *Image Aquisition Toolbox.*

|  |  |
| --- | --- |
|  |  |
| *Imagen 3.14: Función startbuttonpushed. Fuente: Elaboración propia.* | *Imagen 3.15: Botón empezar secuencia. Fuente: Elaboración propia.* |

El sistema autónomo es accionado al pulsar el botón “Empezar”, en la esquina inferior derecha de la interfaz (Imagen 3.15). Al accionarse, el *callback* “startbuttonpushed” es ejecutado. Esta función es muy parecida a la función que controlaba el estado del botón de accionamiento del joystick, ya que se encarga de variar el estado y la apariencia del botón cuando este es presionado. Sin embargo, el componente varía del estado “Empezar” al estado “Pausa”, en vez de al estado “Terminar”. Para lograr esto, se utiliza una variable booleana llamad “autoSystemFlag”, la cual permite que la función *setautosystem* (Imagen 3.8) solo sea activada si su valor es positivo (una vez haya terminado el proceso, o se pulse el botón “Terminar”).

|  |  |
| --- | --- |
|  |  |
| *Imagen 3.16: Botón cerrar app. Fuente: Elaboración propia.* | *Imagen 3.17: Función closeappbuttonpushed. Fuente: Elaboración propia.* |

Una vez se ha terminado de utilizar la interfaz, se debe cerrar utilizando el botón “Close App”, situado en la esquina superior derecha. Como se muestra en la Imagen 3.17, esto permite la activación de la función *pgrobot.closerobot,* la cual cierra todos los componentes.

### 3.2.2 Programa Clase de Control Principal

El programa *PhotogrammetryRobot.m* es una clase la cual contiene todas las funciones que interactúan con los componentes físicos del robot: los motores, la luminaria, el joystick y la cámara. Por lo tanto, se encarga de la comunicación entre los *callbacks* de la interfaz y el robot.

##### Inicialización de la clase

La función de inicialización describe como crear una instancia de objeto de la propia clase fuera de este programa. En este caso, como se muestra en la Imagen 3.18, la función *initizalizeproperties* es ejecutada inmediatamente al crear una instancia. En la Imagen 3.19 se puede apreciar cómo funciona la función *initizalizeproperties.* En primer lugar, se ejecutan cuatro funciones, cada una de las cuales activa cada componente del robot. La función *connectarduino* activa la luminaria, ya que esta es controlada por un Arduino UNO. También define el valor de varias variables que se utilizan luego. Las variables *relayPin, ledWavelength* y *rPrevious* se utilizan para interactuar con el sistema de iluminación. La variable *calibrationSettings* es utilizada para determinar los valores de calibración para los motores, entre el número de pasos totales máximos y la distancia real en milímetros.

|  |  |
| --- | --- |
|  |  |
| *Imagen 3.18: Función inicialización clase. Fuente: Elaboración propia.* | *Imagen 3.19: Función initializeproperties. Fuente: Elaboración propia.* |
|  | |
| *Imagen 3.20 Fragmento función connectmotors. Fuente: Elaboración propia.* | |

La función *connectmotors*, de la cual se enseña un fragmento en la Imagen 3.20, crea primero una instancia de la clase LibXimc.m. Luego, utiliza el comando calllib para interactuar con la librería *libximc.dll* y configurar cada uno de los motores con su puerto correspondiente.

##### Funciones principales

Algunas funciones importantes incluyen *movemotor*, la cual envía el comando ‘move\_calb’ al eje especificado, moviendo el efector a la posición especificada en dicho eje. Como se aprecia, esto se consigue utilizando la función *inputcommand*, utilizada desde el objeto *ximc*, creado utilizando la clase *LibXimc.m*.

|  |  |
| --- | --- |
|  | |
| *Imagen 3.21: Función movemotor. Fuente: Elaboración propia.* | |
|  |  |
| *Imagen 3.22: Función rightmotor. Fuente: Elaboración propia.* | *Imagen 3.23: Función shiftmotor. Fuente: Elaboración propia.* |

De similar forma funcionan las funciones *rightmotor* y *shiftmotor* (Imágenes 3.22 y 3.23). La función *rightmotor* mueve el efector a la “derecha” por el tiempo especificado por la variable “wait”. Esencialmente, mueve el stepper que controla el motor lineal en sentido horario durante el lapso correspondiente. *Shiftmotor* funciona de forma muy similar a *movemotor*, pero en vez de especificar una posición concreta, se define como argumento de entrada una cierta distancia “shift” con la cual el efector debe desplazarse desde la posición actual.

|  |
| --- |
|  |
| *Imagen 3.24: Función joystickmotorcontrol. Fuente: Elaboración propia.* |

La función *joystickmotorcontrol* se encargar de leer la posición actual del joystick en los tres ejes (X, Y y Z), y trasladar ese valor a los motores. Para ello utiliza la función privada *joycontrolaxis*, la cual mueve los motores a una dirección u a otra si el valor recibido es mayor o menor a un determinado *threshold*. Como se puede apreciar (Imagen 3.24), esta función utiliza otra función llamada *checkpausestopstatus*, la cual toma como input los botones de “start” y de “pausa” pasados como argumento desde la interfaz, y decide si continuar, parar o pausar el programa dependiendo de su estado actual.

##### Sistema Autónomo

El sistema automático de toma de fotografías es capaz de tomar fotos de forma ortogonal, utilizando los tres ejes X, Y y Z. El usuario introduce los datos de posición inicial, número de pasos por cada eje, distancia entre pasos y dirección de los archivos resultantes. A partir de esto, el sistema toma una fotografía por cada punto en la cuadrícula especificada. Si valor del switch de multiespectral es positivo, el sistema toma tres capturas por punto: una para luz ultravioleta, otra para luz visible y otra para luz infrarroja.

|  |
| --- |
|  |
| *Imagen 3.25: Fragmento función startautosystem. Fuente: Elaboración propia.* |

Como se observa en la Imagen 3.26, esto se consigue utilizando un sistema de bucles ‘for’ anidados. El programa se ejecuta tantas veces en cada eje como número de pasos se hallan especificado, contando de más a menos. Si el valor de la variable temporal es menor que uno, el efector se mueve un paso en el eje perpendicular, hasta que se haya completado la geometría.

|  |
| --- |
|  |
| *Imagen 3.26: Fragmento función startautosystem. Fuente: Elaboración propia.* |

Por cada paso, se ejecuta el código mostrado en la Imagen 3.27. Como se puede apreciar, en esta función también se utiliza el método *checkpausestopstatus*, para comprobar el estado del botón de inicio y determinar si se debe parar el programa. A continuación, se determina el nombre de la captura basándose en el conteo actual, se toma y guarda la fotografía en la dirección especificada y se actualiza la barra de progreso.

|  |
| --- |
|  |
| *Imagen 3.27: Fragmento función startautosystem. Fuente: Elaboración propia.* |

##### Cierre de componentes

Los componentes son dejados de utilizar por el programa utilizando la función “closerobot”. Esta función sirve para que el sistema pueda ser inicializado de nuevo en otra instancia sin tener que desconectar y volver a conectar los componentes. Como se observa (Imagen 3.28) el programa cierra los motores uno a uno, para finalmente ejecutar la función *ximc.unloadlib,* la cual cierra la instancia creada de la librería *libximc.* Finalmente borra las propiedades privadas que administraban el resto de componentes.

|  |
| --- |
|  |
| *Imagen 3.28: Función closerobot. Fuente: Elaboración propia.* |

### 3.2.4 Programa Clase LibXimc

La clase *LibXimc.m* crea instancias que son capaces de comunicarse con los archivos de la librería *libximc.dll*, la cual está escrita en C y sirve para enviar comandos a los motores lineares. De esta forma, esta clase es una interfaz intermedia que facilita la comunicación entre la clase de control principal *PhotogrammetryRobot.m* y los motores.

##### Inicialización de la clase

La clase se inicializa llamando a la función *loadlib*. Un fragmento de esta función se muestra en la Imagen 3.30. En primer lugar, el método localiza los archivos de la librería requerida. El programa utiliza entonces el comando *loadlibrary* para importar e interpretar los archivos a MATLAB.

|  |  |
| --- | --- |
|  |  |
| *Imagen 3.29: Función inicialización clase. Fuente: Elaboración propia.* | *Imagen 3.30: Fragmento función loadlib. Fuente: Elaboración propia.* |

##### Funciones principales

Hay dos funciones principales que operan de forma similar: *getsettings* y *inputcommand*. Su funcionamiento consiste en recopilar una serie de parámetros, determinar en base a estos que información debe ser enviada a los motores, compilarla para formar el comando requerido, y utilizar el comando *calllib* para enviar la información.

La diferencia principal entre ambas funciones principales es que *getsettings* devuelve el valor actual de la configuración del motor, dependiendo de que configuración se requiera, mientras que *inputcomand* tan solo envía el comando para hacer actuar al motor, y no devuelve ningún valor. En la Imagen 3.31 se muestra un fragmento de la función *inputcommand,* donde se muestra como se envía el comando requerido utilizando *calllib.*

|  |
| --- |
|  |
| *Imagen 3.31: Fragmento función inputcommand. Fuente: Elaboración propia.* |

##### Cierre de la clase

La clase se cierra utilizando el método unloadlib. Esta función es muy sencilla, y tan solo consiste de la utilización de la función *unloadlibrary*, añadiendo seguridad en caso de que surja un error por si, por ejemplo, la instancia de la librería ya estuviese cerrada. Se muestra en la Imagen 3.32.

|  |
| --- |
|  |
| *Imagen 3.32: Función unloadlib. Fuente: Elaboración propia.* |

# 4. Reality Capture

Reality Capture es un software de fotogrametría para crear modelos digitales 3D a partir de fotografías o escáneres láser. Funciona tomando diferentes fotografías en diferentes posiciones a un objeto, y el programa busca cada píxel de cada imagen y todas las otras imágenes, con esto crea una base de datos de características. Una forma de un ángulo de una foto se hace coincidir con otra de otra foto y eso hace que se sepa dónde, en un espacio 3d, estaba la cámara cuando se tomó la imagen. Una vez se sabe dónde están todas las cámaras se puede triangular y crear un modelo tridimensional de todos los puntos.

Dentro de Reality Capture, encontramos una pantalla con lo necesario para poder empezar a trabajar. Arriba a la izquierda encontramos donde se sitúan las fotos que vamos a usar (Imagen 4.1), justo debajo están las propiedades (Imagen 4.2), y en tercer lugar tenemos un viewport donde se encontrará el modelo 3D o la nube de puntos que obtenemos de las fotos (Imagen 4.3).

|  |  |
| --- | --- |
| *Graphical user interface, application  Description automatically generated* | *A computer screen capture  Description automatically generated with low confidence* |
| *Imagen 4.1: Localización fotografías a utilizar. Fuente: Elaboración propia* | *Imagen 4.2: Propiedades. Fuente: Elaboración propia* |
|  |  |
| *A screenshot of a computer  Description automatically generated with medium confidence* | |
| *Imagen 4.3: Localización Modelo 3D resultante. Fuente: Elaboración propia* | |

## 4.1 Procesado de Imágenes

El proceso a seguir para generar el modelo 3D en Reality Capture a partir de imágenes consta de los siguientes pasos:

### 4.1.1 Orientación de imágenes

En este primer paso el programa aplica procedimientos de orientación relativa de pares de imágenes a partir de un número de puntos de paso, resección espacial para orientar una imagen cuando ya se dispone de puntos objeto conocidos, y ajuste de haces de todo el bloque para refinar las orientaciones y transformar las coordenadas al sistema objeto. Con esto se obtienen dos cosas, la orientación de las cámaras desde donde se han tomado las imágenes, y la identificación de los puntos homólogos creando una nube de puntos.

|  |
| --- |
| A picture containing outdoor object, night, dark, night sky  Description automatically generated |
| *Imagen 4.4: Nube de puntos pieza A. Fuente: Elaboración propia* |

### 4.1.2 Reconstrucción 3D

En este punto se crea el modelo 3D. La obtención de coordenadas 3D se crea al medir las coordenadas de los homólogos en un mínimo de dos imágenes en las que aparezcan. Y la malla 3D por la cual se configuran los parámetros antes, se hace a partir de la unión triangulada de la nube de puntos.

|  |
| --- |
|  |
| *Imagen 4.5: “Solid” pieza A. Fuente: Elaboración propia* |

### 4.1.3 Texturizado

En lo que destaca la fotogrametría por encima del resto de técnicas como podría ser el escáner láser, es en crear texturas fotorrealistas en el modelo. Cuando se tenga la malla creada se reproyectan las texturas en el modelo 3D y se crean los mapas de UVS de la malla del objeto. Finalmente, la reconstrucción 3D se exporta hacía el software donde se recrean las luces y se renderiza.

|  |
| --- |
| A picture containing text  Description automatically generated |
| *Imagen 4.6: “Textura” pieza A. Fuente: Elaboración propia* |

## 4.2 Pruebas

Teniendo todo esto en cuenta, hemos hecho una serie de pruebas para ver los límites que podemos tener a la hora de trabajar con Reality Capture.

### 4.2.1 Prueba desenfoque de imágenes

En esta primera prueba queríamos ver qué pasaba cuando las fotos salen desenfocadas y cómo interpreta esa información el software acompañado del resto de imágenes bien enfocadas. Para esta primera prueba, hicimos una secuencia de fotografías, y elegimos al azar diferentes imágenes para reemplazarlas por sus equivalentes, pero desenfocadas. El primer paso fue recrear en 3D la primera pieza usando la secuencia de imágenes sin variar.

|  |
| --- |
|  |
| *Imagen 4.7: Resultado pieza imágenes enfocadas. Fuente: Elaboración propia* |

Una vez hecho esto repetimos el proceso, pero cambiando las fotografías por las desenfocadas.

|  |
| --- |
|  |
| *Imagen 4.8: Resultado pieza imágenes desenfocadas. Fuente: Elaboración propia* |

Lo que podemos ver al comparar los dos ejemplos es que las imágenes desenfocadas no terminan de transmitir información al software, por lo que pueden ocurrir dos cosas, en caso de que tenga otras imágenes en las que apoyarse recreará la zona y creará una textura desenfocada, como la imagen, en ese punto, en caso de no tener suficientes imágenes en las que apoyarse no tendrá nada de información por lo que creará una especie de parche en la geometría y texturizar con la imagen desenfocada.

### 4.2.2 Prueba reemplazo de imágenes

En esta prueba lo que queríamos demostrar es si podríamos reemplazar fotos concretas de una secuencia por otras fotos individuales. Así en caso de que por cualquier motivo hubiera una fotografía que se corrompiese o no valiera, se pueda volver a tomar sin tener que crear una nueva secuencia y tomar todas las imágenes de nuevo.

Al igual que en la anterior, hicimos una secuencia de imágenes completa de 840 fotografías con el robot de fotogrametría, y después elegimos al azar varias fotografías cogiendo sus coordenadas y volviendo a tomarlas para guardarlas de manera individual. Primero se recreó en Reality Capture la figura 3D del modelo, al que podemos llamar original, sin cambiar ninguna fotografía (Imagen 4.7).

|  |
| --- |
| A picture containing text, electronics, display  Description automatically generated |
| *Imagen 4.9: Resultado pieza fotografías originales. Fuente: Elaboración propia.* |

Y después se recreó el modelo 3D cambiando las imágenes del primero por las segundas que habíamos tomado (Imagen 4.8).

|  |
| --- |
|  |
| *Imagen 4.10: Resultado pieza fotografías reemplazadas. Fuente: Elaboración propia* |

Como se puede apreciar en las imágenes no se representa ninguna diferencia entre los modelos. A lo único que habría que prestar atención es a la orientación del objeto en Reality Capture.

# 5. Unreal Engine

Unreal Engine 5 es un motor de desarrollo de Epic Games. Con UE5 se pueden crear entornos interactivos de manera fotorrealista a tiempo real gracias a elementos presentes en esta nueva actualización como nanite o el sistema de iluminación lumen. Una vez tengamos el modelo 3D de la obra, se envía a este motor para renderizarlo, creando primero las luces y el entorno donde se va a situar la obra. Gracias a los nuevos avances de Unreal se pueden conseguir resultados hiperrealistas en un sistema de renderizado a tiempo real.

## 5.1 Lumen

Lumen es un sistema de iluminación global dinámico, en el que la luz indirecta se adapta sobre la marcha a los cambios en la luz directa. La ventaja de lumen es que no hay que esperar a procesar los mapas de luz o crear puntos de reflexión.

|  |
| --- |
| A living room with a red couch  Description automatically generated with medium confidence |
| *Imagen 5.1: Renderizado en UE5 de sala de estar utilizando Lumen. Fuente: unrealengine.com, 2022* |

## 5.2 Nanite

Nanite permite crear entornos con enormes cantidades de polígonos manteniendo el frame-rate y la fidelidad del modelo 3D. Nanite recrea y procesa solo el detalle que se puede percibir, eliminando tiempo de trabajo del motor como procesar los mapas de normales y crear manualmente los niveles de detalle. Sin embargo, también tiene sus limitaciones, este sistema no puede usarse con materiales translúcidos o enmascarados ni con mapas de normales, ya que no los necesita porque usa la propia geometría.

|  |
| --- |
| A picture containing text  Description automatically generated |
| *Imagen 5.2: Renderizado dinámico de 4096 estatuas de 1,000,310 polígonos cada una con Nanite y Lumen. Fuente: Elaboración propia* |

## 5.3 Virtual Shadow Map

Virtual shadow map es la nueva generación de sombreado dinámico a tiempo real diseñado específicamente para trabajar con lumen y nanite. Trae sombras de gran calidad tanto para calidad cinematográfica como para grandes mundos abiertos.

|  |
| --- |
|  |
| *Imagen 5.3: Sombreado a tiempo real utilizando Lumen y Virtual Shadow Map. Fuente: unrealengine.com, 2022* |

# 6. Bibliografía

José Manuel Pereira Uzal(29/11/2013)*.*Modelado 3D en patrimonio cultural por técnicas de structure from motion. *Instituto andaluz del patrimonio histórico.*

*Lumen global illumination and reflections*. in Unreal Engine | Unreal Engine 5.0 Documentation. (2022, April 13). Retrieved June 4, 2022, from <https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/>

*Nanite virtualized geometry*. in Unreal Engine | Unreal Engine 5.0 Documentation. (2022, April 6). Retrieved June 6, 2022, from <https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/>   
*Virtual shadow maps*. in Unreal Engine | Unreal Engine 5.0 Documentation. (2022, April 5). Retrieved June 6, 2022, from https://docs.unrealengine.com/5.0/en-US/virtual-shadow-maps-in-unreal-engine/