Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Manipulación de Objetos mediante sistema de visión artificial y el robot IRB120 + Inmoov-SR

Autor: Sergio Reñones Domínguez

Tutor/es: Rafael Barea Navarro

2017

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

“Manipulación de objetos mediante sistema de visión artificial y el robot IRB120 + Inmoov-SR”

Sergio Reñones Domínguez

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Manipulación de objetos mediante sistema de visión artificial y el robot IRB120 + Inmoov-SR"

Autor: Sergio Reñones Domínguez

Tutor: Rafael Barea Navarro

TRIBUNAL:

Presidente: J. Antonio Jiménez Calvo

Vocal 1: Manuel Ocaña Miguel

Vocal 2: Rafael Barea Navarro

FECHA: 18/09/2017

Agradecimientos.

Me gustaría agradecer en primer lugar a la universidad de Alcalá de Henares por brindarme el material necesario para desarrollar el proyecto. Por otro lado, me gustaría dar las gracias a toda la gente que confía en mí y se encuentra a mi lado cada día. A mi familia, sois un apoyo incondicional y sé que siempre podré contar con vosotros. Mis amigos, los de siempre, todos en mayor o menor medida habéis compartido mi alegría y mi sufrimiento y sin vosotros no habría sido capaz de cumplir esta etapa, espero teneros siempre conmigo.

"La vida no debería ser un viaje hacia la tumba con la intención de llegar a salvo con un cuerpo bonito y bien conservado, sino más bien llegar derrapando de lado, entre una nube de humo, completamente desgastado y destrozado, y proclamar en voz alta: ¡Uf! ¡Vaya viajecito!"

Hunter S. Thompson

Contenido General

Lista de Figuras

Lista de Tablas

Resumen

La idea principal del proyecto es realizar el diseño e implementación de una aplicación informática centrada en el ámbito de la robótica, más concretamente se basa en la creación de un sistema de visión que realizará el tratamiento de una imagen correspondiente a la zona de trabajo de un brazo robótico IRB-120 de ABB, con la finalidad de colocar diferentes objetos simples en ésta y que el robot proceda a su clasificación por colores en los depósitos previamente prefijados teniendo en cuenta la altura de dichos objetos para manipularlos de forma correcta.

Palabras Clave

-Visión artificial

-IRB 120

-Sistema

-Manipulación objetos

-Tratamiento color

Abstract

The main idea of the Project is to design and implement an informatic application focused on the robotics field, more specifically it is based on the creation of a visual system that performs the image treatment corresponding to the work area of an IRB-120 robotic arm from ABB, with the purpose of placing different simples objects in it and that the robot classify them by colours on the storages previously prefixed taking care about their height to manipulate them correctly.

KeyWords

-Artificial vision

-IRB 120

-System

-Object manipulation

-Color treatment

Resumen Extendido

El proyecto se fundamenta en la implementación de un sistema de visión artificial en la GUI de MATLAB basado en el tratamiento de una imagen por colores (en concreto azul, rojo y verde), de forma que podamos detectar los colores y formas de los objetos que se encuentren en la zona de trabajo del robot IRB-120 de ABB para posteriormente clasificarlos en los diferentes depósitos colocados a la izquierda de éste.

El desarrollo de la aplicación se realiza en 3 etapas, en la primera se ejecuta un procedimiento de conexión con los diferentes programas implicados y se permite al usuario, en caso de que lo desee, realizar un proceso de calibración para adaptarse a la zona de trabajo por si se hubiese producido alguna modificación. Durante la segunda etapa se realizará el análisis sobre la imagen y se almacenará el destino de los centroides de los objetos en cuestión junto con sus respectivos valores de altura, cabe destacar que se podrán modificar ciertos parámetros relacionados con la detección de los colores en función de las características lumínicas de la zona de trabajo, todo ello en tiempo real a medida que vamos ejecutando el programa general. La última etapa estaría destinada al conjunto de sentencias que mandan la orden de recoger dichos objetos con el brazo robótico mediante protocolo TCP/IP y el uso de datagramas.

Diferenciaremos en todo momento dos modos de funcionamiento, en el modo de simulación, la obtención de las imágenes y la secuencia de instrucciones llevada a cabo por el brazo robótico serán ejecutados por los programas VREP y ROBOTSTUDIO respectivamente. Por otro lado, el modo real utilizará imágenes tomadas en el entorno de trabajo real y ejecutará las instrucciones del programa en RAPID del brazo robótico mediante el uso del programa ROBOTSTUDIO, dirigido a través del armario de control IRC5 que suministrará energía a los diferentes sensores y motores de la máquina. El sistema en ambos casos cuenta con unas configuraciones de entorno propias aunque basadas en los mismos conceptos que iremos explicando a lo largo del proyecto.
El proyecto ha sido diseñado para poder trabajar de forma efectiva con varias herramientas diferentes que puedan ser acopladas al robot, por lo que trabajaremos de forma independiente con la ventosa, el gripper y la mano Inmoov y los objetos a recoger por dichas herramientas dependerán precisamente de ellas pudiendo encontrarnos con fichas rectangulares o cilindros de diferentes alturas dependiendo de cada caso.

Los programas utilizados en tiempo real para el desarrollo del proyecto son MATLAB, ROBOTSTUDIO y VREP. MATLAB es la base de programación de la aplicación, utilizamos la herramienta de interfaz gráfica GUI para crear un espacio de trabajo sencillo y funcional para el usuario, desarrollamos a su vez el tratamiento de la imagen y gestionamos además las diferentes conexiones con el resto de programas utilizados para enviar y recibir la información necesaria en cada caso. VREP se utiliza para obtener imágenes del espacio de trabajo en modo simulado, que funcionará de forma análoga a la imagen obtenida de la cámara (Kinect o web) aunque su interpretación será ligeramente diferente. ROBOTSTUDIO por último es el programa encargado de ejecutar las instrucciones de movimiento en modo simulado enviadas a través de Matlab en forma de sockets, en el modo real en cambio el código RAPID será procesado por el armario de control IRC5.

1.- Introducción

1.1.- Robótica y Sociedad

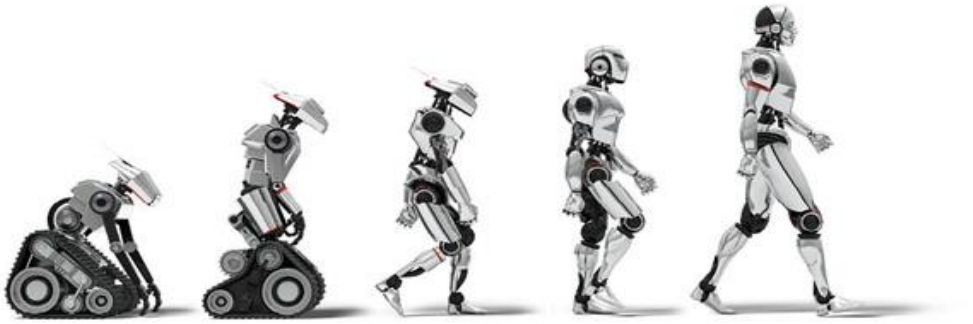

Figura 1.1.- *Evolución de la robótica*

La robótica se ha convertido en uno de los campos de aplicación y estudio más importantes en muchos de los sectores de nuestra sociedad (en nuestro caso nos centraremos en la industria). Además, en los últimos años está jugando un papel muy intenso en nombre del progreso y se ha convertido en parte fundamental de cara a la automatización de procesos, de manera que no se contempla en la sociedad actual actividad industrial en la que se rechace el uso de estos aparatos, sin necesidad de esfuerzo por parte de seres humanos que vayan más allá de la supervisión o el control. Por otro lado, la robótica industrial presenta una amplia gama de posibilidades para el futuro, y se estima que, en pocos años, se habrán alcanzado metas que incluso en estos momentos se nos asemejan imposibles.

En la actualidad, los avances en visión artificial y robótica son conducidos hacia el campo de la inteligencia artificial, permitiendo prever la disponibilidad de robots dotados con gran flexibilidad y capacidad de adaptación al entorno, ya que, en disposición de una cámara que conforme el sistema de visión y la implementación de nuevos sistemas de sensores e adquisición de datos las posibilidades son casi infinitas.

Figura 1.2.- *Cadena automovilística*

La IFR [1] (International Federation of Robotics) clasifica según la UNE-EN ISO 8373 un robot industrial como un robot de más de tres ejes, reprogramable, multiaplicación, móvil o no y destinado a aplicaciones de automatización industrial. La IFR además, contabiliza las áreas de aplicación (mostradas en la tabla 1.1.- Áreas de aplicación robots industriales en España) y el número de robots acumulados en los últimos 12 años, donde indiscutiblemente el liderazgo pertenece, con datos del 2013, al continente asiático (seguido muy por detrás por Europa y después América) y aunque históricamente Japón posee con mucho el mayor número de robots industriales en funcionamiento (300.000) y se considera el país más automatizado, se ha visto superado por el mercado Chino [2] que casi ha triplicado el volumen de ventas anunciado en 2012, tanto es así que por cada 5 robots que se vendieron ese año en el mundo, uno se instaló en China. España por su parte instaló en 2008 2.461 unidades logrando un histórico total de 29.029 desde 1997, de los cuales 11.539 fueron proporcionados por la empresa ABB, la marca del robot que utilizaremos en nuestra aplicación.

Aplicaciones	a 31/12/07	Altas 08	Bajas 08	TOTAL
Manipulación y carga/descarga de máquinas	10127	1094	263	10958
Soldadura	13759	772	597	13934
Materiales	1128	117	55	1190
Montaje y desmontaje	1165	139	149	1155
Otros	1131	74	69	1136
Sin especificar	391	265	0	656
TOTAL	27701	2461	1133	29029

Tabla 1.1.- *Áreas de aplicación robots industriales en España*

1.2.- Objetivos

-El objetivo principal del proyecto es desarrollar una aplicación con las características propias de un sistema real y, por lo tanto, con los retos que ello plantea abordando aun así una primera fase basada en la simulación de dicho sistema donde podamos abordar además la manera en la que interconectamos los diferentes programas que lo integran.

-Por otro lado, se intentará profundizar en el manejo de herramientas que trabajan con el procesamiento de imágenes y, en general, con el concepto de visión artificial.

-Por último, plantear el establecimiento de un procedimiento de aplicación utilizando la interfaz gráfica de MATLAB para generar secuencias de movimiento del brazo robótico IRB120 en tiempo real y utilizar la herramienta de la mano robótica.

1.3.- Estructura del Proyecto

A fin de esclarecer el desarrollo del proyecto y los contenidos que se van a tratar en cada caso observaremos la siguiente tabla en donde expondremos los apartados generales del mismo:

APARTADO	CONCEPTO
1	INTRODUCCIÓN
2	HERRAMIENTAS UTILIZADAS
3	ARQUITECTURA GENERAL DEL SISTEMA
4	DESARROLLO DE LA APLICACIÓN. RESULTADOS
5	MANUAL DE USUARIO
6	CONCLUSIONES
7	PLIEGO DE CONDICIONES
8	PLANOS
9	PRESUPUESTO
10	BIBLIOGRAFÍA

Tabla 1.2.- *Estructura del proyecto*

2.- Herramientas Utilizadas

2.1.- Matlab

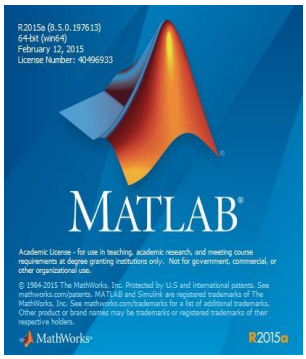

Figura 2.1.- *Logotipo MATLAB*

MATLAB [3] es un entorno de programación interactivo de alto nivel dedicado a la computación numérica, análisis y visualización de datos, importación de los mismos desde otros entornos o su exportación a dispositivos externos entre otros. MATLAB debe su nombre a la aplicación de diseño inicial, 'MAtrix LABoratory' o 'laboratorio de matrices', ya que fue desarrollado en principio para proporcionar una herramienta sencilla para el cálculo matricial. Ahora en cambio, podemos usar MATLAB para un sinfín de aplicaciones diferentes que engloban desde el procesamiento y medición de señales, sistemas de control, análisis numérico enfocado a diferentes sectores como la biología o las finanzas, etc. y es por eso por lo que es una herramienta utilizada actualmente por más de un millón de científicos e ingenieros tanto en el ámbito estudiantil como laboral.

Es importante destacar que éste, es un entorno que permite gran variabilidad y puede dar solución a problemas de temáticas muy diferentes mediante el uso de las TOOLBOXES, un conjunto de funciones ya implementadas para una tarea específica. Además, cuenta con un lenguaje de programación propio (lenguaje M) para ampliar y adaptar código nuevo. Por último, aunque dispone de muchas otras herramientas internas de gran utilidad como SIMULINK, destacaremos en nuestro caso GUIDE (entorno de desarrollo GUI), el cual nos permite diseñar una interfaz gráfica de usuario para una aplicación específica a la vez que nos genera automáticamente el código de MATLAB necesario para construirla, pudiendo así moldear su comportamiento a nuestro gusto.

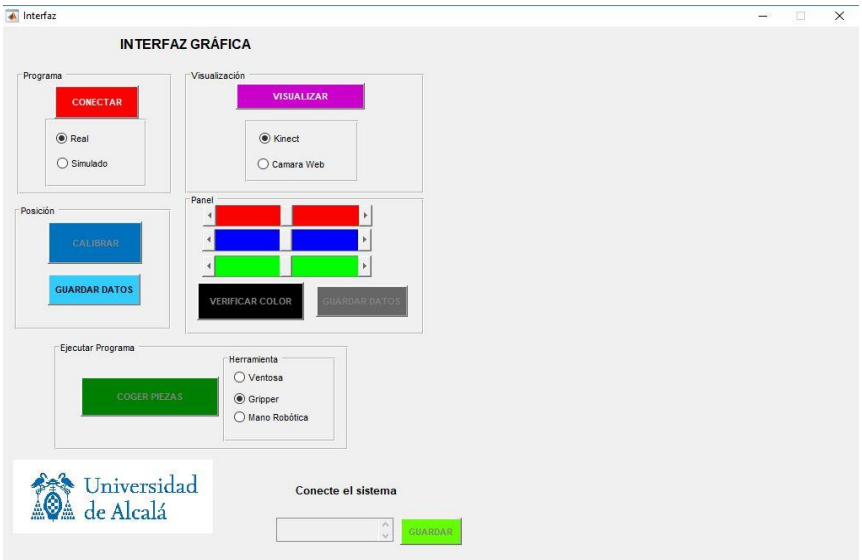

Figura 2.2.- *Ejemplo GUI MATLAB*

La versión utilizada en nuestro caso para el desarrollo de la aplicación será la 'R2015a – academic use' y las toolbox principales (además de las que vienen por defecto) son 'Image Analysis Toolbox' y 'Robotic System Toolbox'.

2.2.- V-REP

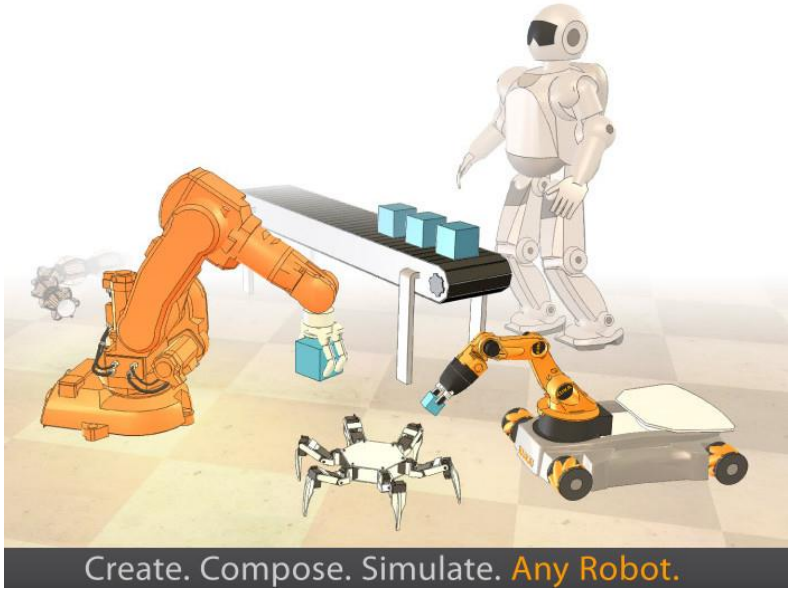

Figura 2.3.- *Entorno V-REP*

VREP [5] es un entorno libre de desarrollo integrado y simulación, resulta una herramienta perfecta para la educación relacionada con la robótica puesto que permite el desarrollo rápido de algoritmos, la simulación de sistemas de automatización y su control y monitorización remotos. Además, puede utilizarse como una herramienta autónoma o ser incrustada fácilmente en aplicaciones de nivel superior, también goza de gran variabilidad puesto se basa en una arquitectura de control distribuido de forma que cada modelo (objeto) puede ser controlado individualmente mediante scripts, complementos, nodos ROS, un cliente API remoto u otro tipo de soluciones personalizadas y los controladores pueden escribirse en C, C++, Python, Java o Matlab entre otros.

Utilizaremos en nuestro sistema la versión VREP PRO EDU descargada de la web Coppelia Robotics para crear la estación en modo simulado donde colocaremos los objetos con la forma y tamaño que queramos que trabaje nuestro robot, es decir, realizará la función de la cámara Kinect transcurriendo el modo simulación.

COPPELIA ROBOTICS

Figura 2.4.- *Logotipo COPPELIA ROBOTICS*

2.3.-ABB

ABB es una compañía líder mundial en automatización e ingeniería eléctrica que proporciona tecnologías para la generación, transporte y distribución de energía de forma eficiente y segura, así como ayuda a incrementar la productividad de las industrias y colabora con diferentes instituciones y servicios públicos para conseguir sus objetivos con el mínimo impacto ambiental. Encontramos por lo tanto que comprende desde la fabricación de elementos sencillos como interruptores de iluminación o cables hasta sistemas robóticos suministrando tanto el software como el hardware necesario para su debida utilización.

2.3.1.- IRB-120

Figura 2.5.-*Robot IRB-120*

Como elemento principal de nuestra aplicación y eje global del proyecto dispondremos del brazo robótico IRB-120 (de configuración articular) disponible en la Universidad de Alcalá de Henares, el robot industrial más pequeño de ABB que tan sólo pesa 25Kg puede levantar unos 3Kg de carga. Al ser una versión reducida con respecto a otros modelos más comerciales en la industria goza de las dimensiones óptimas para el proyecto y además mantiene sus características de precisión y movilidad. El brazo dispone de 6 GDL (grados de libertad), es decir de 6 articulaciones. Los últimos 3GDL forman la muñeca esférica ya que está formada por una articulación rotacional (eje4), una articulación angular (eje5) y otra articulación rotacional (eje6). Los otros 3 GDL están formados por una articulación rotacional (en la base o eje 1) y dos articulaciones angulares (eje 2 y 3). Estos 6 ejes de movimiento están limitados de la siguiente forma:

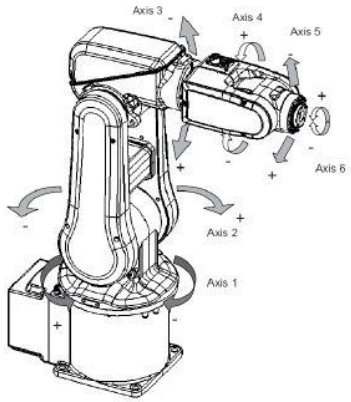

Figura 2.6.- *Ejes IRB-120*

	Rango (º)	Velocidad Máxima (º/s)
Axis 1: Rotación	+165 a -165	250
Axis 2: Brazo	+110 a -110	250
Axis 3: Brazo	+70 a -110	250
Axis 4: Muñeca	+160 a -160	320
Axis 5: Doblar	+120 a -120	320
Axis 6: Girar	+400 a -400	420

Tabla 2.1.-*Limitación ejes IRB-120*

2.3.2.- IRC-5 y FlexPendant

Deberemos contar a su vez con varios aparatos imprescindibles para el correcto funcionamiento del equipo en modo real. En primer lugar, el armario de control IRC5 es donde tiene lugar el procesamiento de la información de manera que ejecuta en tiempo real el código del programa y lo sincroniza con los dispositivos externos disponibles. Además, sirve de alimentación e intercambio bidireccional de información con éstos, incorporando 16 entradas y 16 salidas digitales y una analógica en el rango de 0-10V. Este armario de control es de tipo portable y funciona como su propio cerebro, otorgándole precisión y fiabilidad a su uso, puede ser utilizado con brazos robóticos de hasta 8kg.

Figura 2.7.- *Armario de control IRC-5*

En segundo lugar, el IRC5 incluye una herramienta muy útil denominada FlexPendant que, entre otras funcionalidades, nos permite:

- Cargar mediante un USB 2.0 el programa en RAPID (eliminando la parte creada para la simulación) con formato txt.

-Crear nuestros propios programas y depurarlos desde la misma herramienta.

-Interrumpir y reanudar su ejecución según las necesidades del usuario o para realizar comprobaciones de seguridad.

-Mover cada uno de los ejes del robot de forma manual utilizando el joystick que encontramos en la parte frontal.

Figura 2.8.- *FlexPendant*

2.3.3.- Robotstudio

Contaremos a su vez también con una licencia de estudiante de la versión 5.15 de Robotstudio, el software de ABB que permitirá crear una estación con las mismas características que robot real para que ambos respondan de la misma forma ante órdenes idénticas, será en el código RAPID (propio del programa) en el que debamos diferenciar las señales de simulación de las de ejecución real, además permitirá crear diversas situaciones de trabajo y la realización de gran variedad de pruebas de funcionamiento para estar seguros de cuál será la respuesta del brazo robótico en todo momento.

2.4.- Espacio de Trabajo

Destacaremos los elementos propios de nuestro espacio de trabajo que no hemos clasificado anteriormente, entrarían dentro de éste en primer lugar las herramientas que acoplaremos a nuestro brazo robótico (con el enganche de 40 mm), que además utilizan un circuito neumático para desarrollar su función, ya se trate de la ventosa, el gripper o la mano Inmoov, aunque a lo largo del proyecto daremos mayor prioridad a esta última, puesto que aunque hayamos realizado la mayor parte de pruebas con la ventosa, trabajar utilizando la herramienta de la mano es uno de los elementos objetivos principales del proyecto.

2.4.1.- Ventosa

Figura 2.9.- *Ventosa SMC ZPT32BN-B01*

La característica principal de esta herramienta será la capacidad para ejercer succión sobre el objeto que deseemos transportar, con ello conseguiremos un agarre firme, preciso y seguro, aunque deberemos asegurarnos en todo momento de que la superficie de contacto con la ventosa sea lo suficientemente lisa y grande para que ejerza su función adecuadamente.

Para nuestro caso contaremos con la ventosa modelo SMC ZPT32BN-B01, de 35 mm de diámetro y que añadirá una altura total al TCP del robot de 165 mm actuando de la siguiente forma:

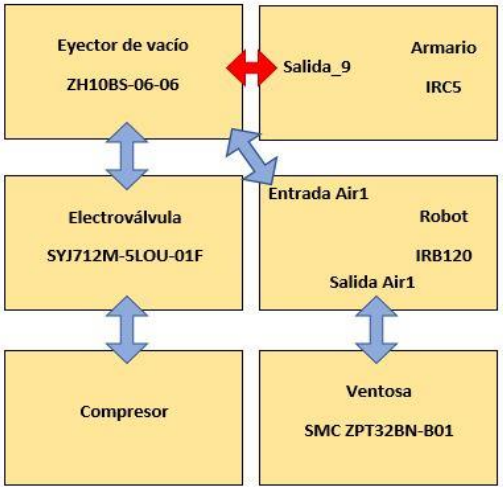

Figura 2.10.- *Esquema conexiones ventosa*

Gracias al código en RAPID diseñado, el armario de control IRC5 generará la señal de succión por la E/S nº 9, la cual está conectada con la electroválvula señalada para dejar pasar o no el aire comprimido. El aire a presión circulará entonces por la electroválvula y llegará al eyector de vacío tipo Venturi que, como su propio nombre indica, se encargará de generar internamente el vacío para (a través de unos conductos de plástico) llegar a la ventosa conectada al robot y succionar el objeto en cuestión. Cuando deshabilitemos la E/S nº 9 la electroválvula cerrará el paso de aire y se dejará de generar el vacío en el objeto por lo que caerá por su propio peso en el lugar deseado por el usuario.

2.4.2.- Gripper

La característica principal de esta herramienta es la de poder aproximarnos a la aplicación de manipulación y transporte de objetos. Para asegurar un agarre preciso y seguro las dimensiones de los objetos que utilicemos deben tener unas dimensiones similares al espacio entre las pinzas cuando se encuentra en posición cerrada, puesto que si es muy pequeño el gripper no sería capaz de agarrarlo y si, en cambio es demasiado grande, las pinzas no lo abarcarían pudiendo caerse o resultar dañados tanto el objeto como la herramienta.

Figura 2.11.- *Gripper SCHUNK-GBW44*

Para nuestro caso contaremos con el gripper modelo SCHUNK-GBW44, formado por dos pinzas con forma rectangular y dimensiones 20x15x70 mm que se han diseñado para permitir una separación entre las mismas de 30 mm, además la herramienta añadirá una altura al TCP del brazo robótico de 226 mm actuando de la siguiente forma:

El armario IRC5, basándose en el código en RAPID diseñado, generará las señales necesarias para la abertura y cierre (que se dirigirán a la electroválvula para dar o no paso al aire comprimido) por las E/S 10 y 11. A diferencia de la ventosa, para esta herramienta dispondremos de dos señales de actuación, la primera está destinada a abrir las pinzas mientras la segunda ejecuta la función de agarre por lo que, siempre que trabajemos con el gripper mantendremos una de las dos activas y será importante que en ningún momento se encuentren operando ambas.

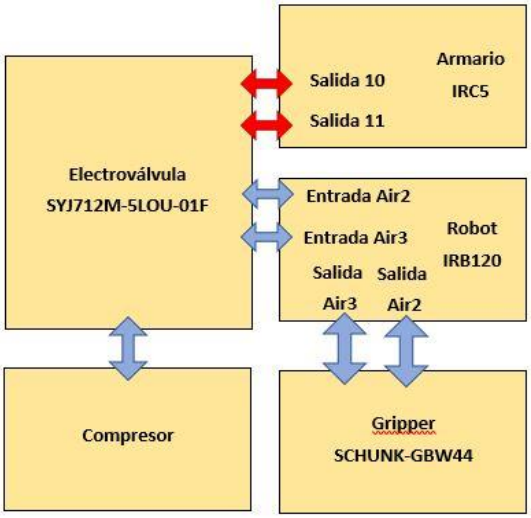

Figura 2.12.- *Esquema conexiones gripper*

2.4.3.- Mano Inmoov-SR

La mano robótica articulada que utilizaremos para el proyecto forma parte de un robot humanoide articulado creado por el artista francés Gael Langevin, el cual ha publicado todos los diseños de las piezas que conforman el robot para que puedan ser reproducidas en plástico utilizando simplemente una impresora 3D, además se utilizarán varios varios servos, cables, una fuente de alimentación y algunas placas Arduino que controlan la robótica para construirlo basándose en el concepto "Do It Yourself". Nos basaremos en el diseño, fabricación, modelado y control de la herramienta desarrollados en los TFG's de los compañeros Álvaro Bailón [11] y Sergio Rodríguez [10].

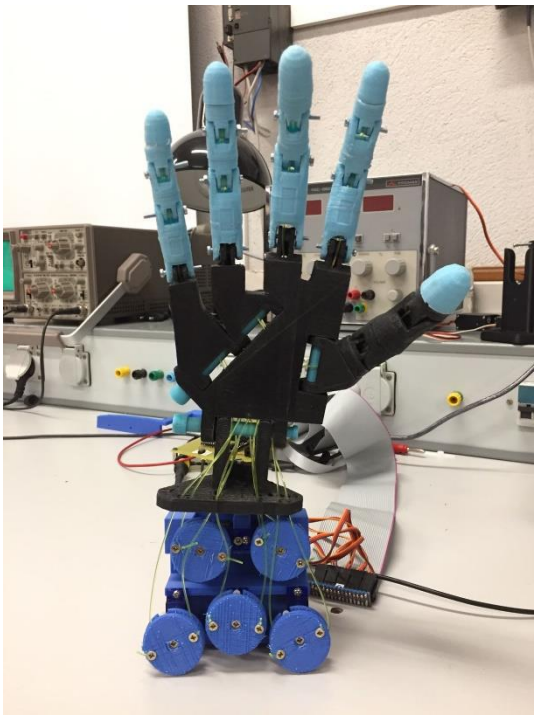

Figura 2.13.- *Mano Inmoov-SR*

Figura 2.14.- *Imagen fotorrealista en Solidworks*

A modo de resumen adjuntaremos una tabla con las señales de E/S necesarias para la manipulación de cada una de las herramientas herramientas:

Herramienta	Modelo	Señales de manipulación
Ventosa	SMC ZPT32BN-B01	9
Gripper	SCHUNK-GBW44	10, 11
Mano robótica	Inmoov-SR	13, 14, 15, 16

Tabla 2.2.*-Señales E/S que activan cada herramienta*

2.4.4.- Cámaras Web y Kinect

Hablaremos en este apartado de las cámaras web y Kinect, para el primer caso sería válida cualquier cámara web con una resolución estándar de 640x480 píxeles ya que mismamente en el caso de este proyecto para realizar las pruebas iniciales de la configuración del dispositivo utilizamos una app para utilizar la cámara de un dispositivo móvil, aunque finalmente utilizaríamos la cámara instalada en el laboratorio de marca Logitech y modelo C-310. La cámara Kinect por otra parte es la que nos permite captar una imagen de profundidad del espacio de trabajo además de la imagen en color estándar a la que estamos acostumbrados, utilizamos el modelo Kinect for Windows con una resolución de 640x480 píxeles.

Figura 2.15.- *Cámara Logitech C310*

Figura 2.16.- *Cámara Kinect*

Un detalle importante a tener en cuenta utilizando la cámara Kinect es que en algunas partes de la imagen de profundidad aparece como si los objetos estuvieran a 0mm, es un error de la cámara debido a que al colocar algunos obstáculos el patrón de puntos se coloca sobre ellos y no permite ver lo que se encuentra detrás. En definitiva, como el sensor de la cámara y el que proyecta el patrón no están en la misma posición tras el objeto queda una zona donde no se detectan puntos.

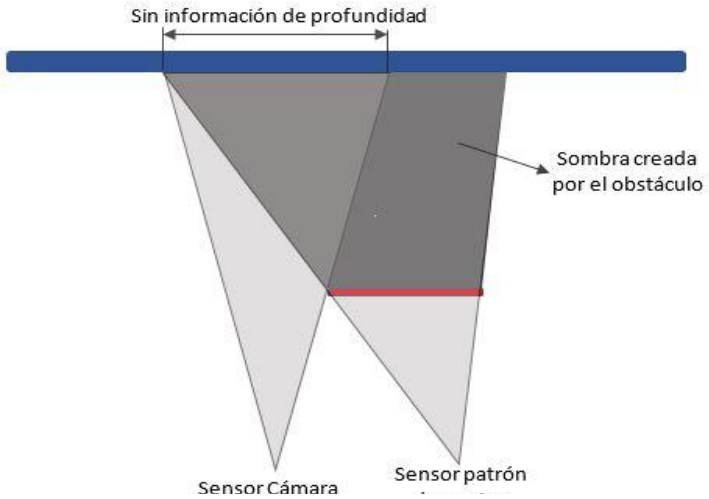

Figura 2.17.- *Diagrama defecto Kinect*

2.4.5.- Otros

Por último, simplemente nombrar el resto de útiles que han sido necesarios durante el desarrollo de la aplicación, empezando por la mesa de trabajo, los objetos de diferentes formas y colores que serán detectados en la imagen y por supuesto el computador que soporta todo el sistema. Se adjuntan para finalizar un par de imágenes que muestran en conjunto todo lo anteriormente relatado.

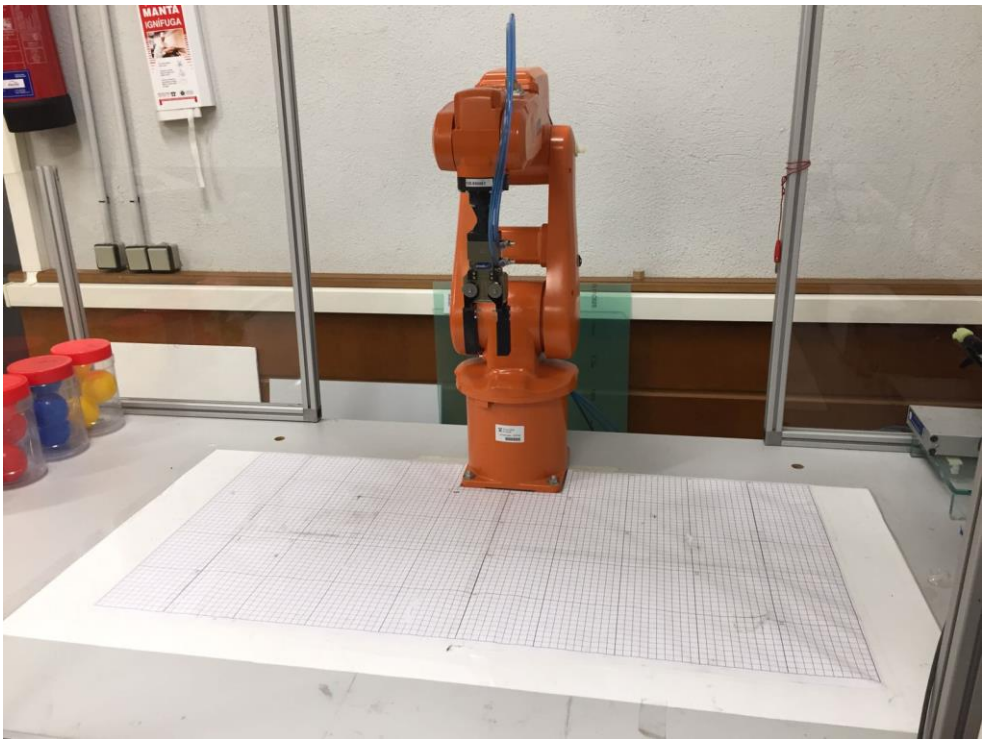

Figura 2.18.- *Espacio de trabajo*

Figura 2.19.- *Ordenador de trabajo*

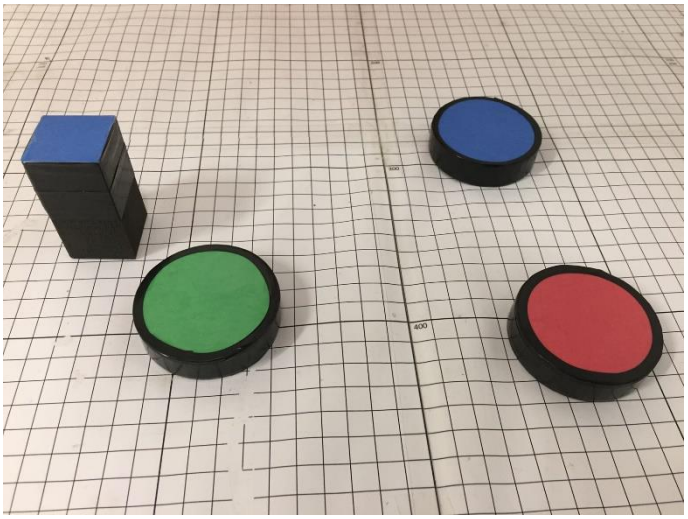

Figura 2.20.- *Objetos a detectar*

3.- Arquitectura General del Sistema

3.1.- Esquema General

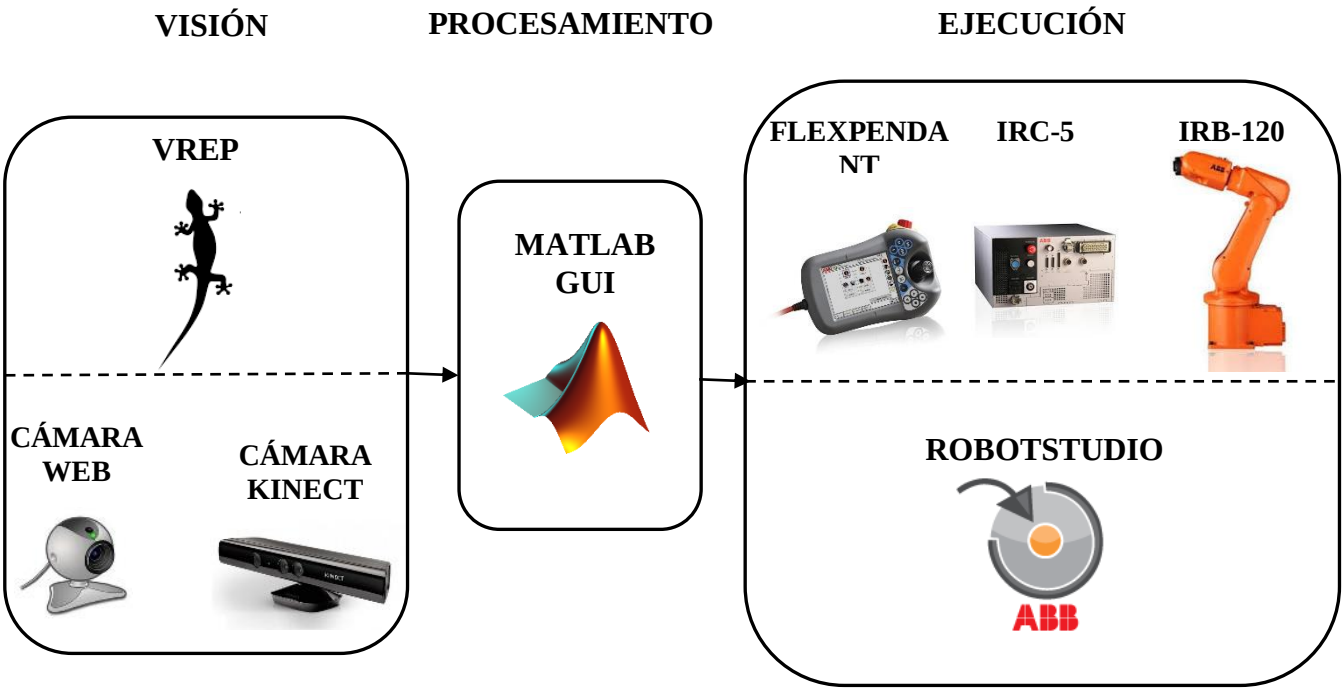

Figura 3.1.- *Diagrama esquema general*

3.2.- Sistema de Visión

3.2.1.- Captación de Imágenes

Como ya hemos explicado en apartados previos, una parte fundamental del proyecto consiste en discernir en todo momento los diferentes modos de operación, diferenciando en primera instancia si queremos operar en modo real o simulado y posteriormente si en cualquiera de estos casos se actuará con datos de la cámara web o la Kinect.

Instalación Cámaras Web y Kinect

Para el caso en modo real deberemos instalar y configurar los dispositivos para que sean incorporados en MATLAB correctamente, lo que haremos de un modo similar en ambos casos. Todos los paquetes de soporte contienen los archivos de Matlab necesarios para usar la toolbox con los adaptadores que dispongan nuestras cámaras y teniendo en cuenta que la resolución en ambas es de 640x480 píxeles. El procedimiento a seguir es seleccionar la pestaña “HOME” y en el desplegable “Add-Ons” seleccionar el apartado “Get Hardware Support Packages” como se muestra en la figura 3.2:

Figura 3.2.-*Captura Pantalla de Inicio Matlab*

Una vez seleccionada se nos abrirá una ventana emergente donde deberemos seleccionar el paquete que deseemos instalar (para iniciar la instalación deberemos estar registrados como usuarios en Mathworks [3]). De esta manera instalamos los siguientes paquetes:
-DCAM Hardware
-Os Generic Video Interface
-Kinect for Windows Sensor
Utilizamos el primer paquete para poder adquirir imágenes desde la cámara de un dispositivo móvil, lo que nos resultó tremendamente útil durante las pruebas iniciales, además fue necesario instalar tanto en el ordenador como en el dispositivo la aplicación "DroidCam", que nos ofrece la posibilidad de comunicarlos vía WIFI o como elegimos en nuestro caso, mediante un USB 2.0. El segundo paquete es necesario para adquirir imágenes de cualquier interface de video genérico, que incluye cámaras con adaptadores Linux ('linuxvideo'), Macintosh ('macvideo') o Windows ('winvideo') como en el caso que nos aplica. El tercero será necesario para adaptar la cámara Kinect y poder recibir la información de profundidad y color que nos ofrece.

Secuencia de comandos utilizados

A continuación desarrollaremos un pequeño listado con los comandos más comunes utilizados para la adquisición de imágenes y administración de los diferentes dispositivos [3]:

-Imaqreset: Fuerza a MATLAB a resetear las conexiones con el puerto USB, resulta útil si acabas de conectar un dispositivo y el programa no lo identifica.

```
>> imaqreset
```

-Imaqhwinfo: De forma general muestra la información de los adaptadores disponibles:

```
>> imaqhwinfo
ans =
   InstalledAdaptors: {'dcam'  'kinect'  'winvideo'}
      MATLABVersion: '8.5 (R2015a)'
        ToolboxName: 'Image Acquisition Toolbox'
     ToolboxVersion: '4.9 (R2015a)'
```

También podemos ver las propiedades de algún adaptador específico e incluso de un dispositivo concreto dentro de este adaptador de la siguiente forma:

```
>> imaqhwinfo('winvideo',1)
ans =
          DefaultFormat: 'RGB24_640x480'
     DeviceFileSupported: 0
              DeviceName: 'DroidCam Source 3'
                DeviceID: 1
   VideoInputConstructor: 'videoinput('winvideo', 1)'
  VideoDeviceConstructor: 'imaq.VideoDevice('winvideo', 1)'
       SupportedFormats: {'RGB24_640x480'}
```

-Videoinput (): Inicializaremos la cámara identificando el adaptador y el dispositivo indicados, además podremos añadir si procede otros parámetros como la resolución. Ejecutaremos la sentencia y se la asignaremos a una variable para poder referirnos a ella posteriormente:
>>vid=videoinput('kinect',1)

-Start (): Inicia la cámara para comenzar a adquirir imágenes de ella.
>>start(vid)

-Preview (): Nos permite visualizar las imágenes que está generando la cámara.
>>preview(vid)

-Getsnapshot (): Devuelve una instantánea del objeto de video seleccionado en formato de workspace, es necesario que el objeto haya sido inicializado aunque no que lo esté previsualizando.
>>img=getsnapshot(vid);

-Imagesc (), imtool(), image(): Visualiza una imagen guardada como workspace en un figure, sirve para ver la imagen, obtener información acerca de los píxeles o incluso guardarla aunque también se puede utilizar el comando imwrite(img,'fichero.jpg')
>>imagesc(img)

-Save: Aunque no es propiamente un comando exclusivo para el guardado de imágenes con formato .mat, será usado continuamente a lo largo del programa con este fin.
>> save 'ImagenesKinect.mat' alignedColorImage flippedDepthImage

Modo simulación

Mientras el modo real consiste en captar las imágenes en tiempo real de las cámaras en función de las condiciones de trabajo, en modo simulación el proceso es significativamente diferente debido a que, la forma en la que simularemos la cámara web será cargando en el sistema imágenes previamente almacenadas con formato JPG. Por otro lado, la forma en la que simularemos la cámara Kinect consta de un proceso más complicado que explicaremos a continuación:

Según lo explicado en anteriores apartados, VREP nos permitirá realizar la simulación del entorno de trabajo (aunque obviaremos colocar el robot para no sobrecargar la imagen, en su lugar colocaremos un cilindro en la localización de su base) para así poder recrear cualquier situación desde el mismo ordenador de trabajo. Para ello, será de suma importancia establecer una conexión segura y fiable entre los programas MATLAB y VREP, por lo que en la web de coppelia robotics) [6] existe un apartado diseñado para habilitar las funciones API ('Interfaz de Programación de Aplicaciones') de forma remota. En concreto nos interesa la sección de la conexión con MATLAB, en ella se nos requerirán 3 archivos:

-RemoteApiProto.m
-remApi.m
-remoteApi.dll, remoteApi.dylib o remoteApi.so dependiendo de la versión que utilicemos (en nuestro caso remoteApi.dll).

Todos ellos están disponibles en el propio ordenador una vez instalamos VREP dentro del directorio *programming/remoteApiBindings/Matlab*, utilizaremos a su vez como base el programa SimpleTest.m y lo adaptaremos a nuestras necesidades, añadiendo el código necesario para captar las imagenes de los "VisionSensor" creados en la estación y almacenándolas en el sistema en caso de que no se produzca ningún error en la ejecución del programa, destacar que VREP debe estar simulando (sacando imágenes por los "VisionSensor") en el momento de arrancar el código [5]. Renombraremos entonces el Script como ObtenerImagenesVrep2.m, el

cual adjuntamos a continuación (obviando las primeras líneas de comentario hechas por los creadores):

```
function simpleTest()
    disp('Program started');
  vrep=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
vrep.simxFinish(-1)
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
if (clientID>-1)
     [res,objs]=vrep.simxGetObjects(clientID,vrep.sim_handle_all,vrep.simx_opmode_oneshot_wait);
    if (res==vrep.simx_error_noerror)

[err_left,camhandle_left]=vrep.simxGetObjectHandle(clientID,'StereoSensor_leftSensor',vrep.simx_opm
ode_oneshot_wait);

[errorCode_left,resolution_left,img_left]=vrep.simxGetVisionSensorImage2(clientID,camhandle_left,0,vr
ep.simx_opmode_oneshot_wait);

[err_right,camhandle_right]=vrep.simxGetObjectHandle(clientID,'StereoSensor_rightSensor',vrep.simx_
opmode_oneshot_wait);

[errorCode_right,resolution_right,img_right]=vrep.simxGetVisionSensorImage2(clientID,camhandle_ri
ght,0,vrep.simx_opmode_oneshot_wait);

[err_depth,camhandle_depth]=vrep.simxGetObjectHandle(clientID,'StereoSensor',vrep.simx_opmode_on
eshot_wait);

[errorCode_depth,resolution_depth,img_depth]=vrep.simxGetVisionSensorImage2(clientID,camhandle_
depth,0,vrep.simx_opmode_oneshot_wait);

        save ImagenesVrep.mat img_left img_right img_depth
    else
        fprintf('Remote API function call returned with error code: %d\n',res);
    end
    vrep.simxFinish(clientID);
else
    disp('Failed connecting to remote API server');
end
vrep.delete(); % explicitely call the destructor!

    disp('Program ended');
end
```

A modo de ejemplo adjuntaremos a su vez el código necesario para la verificación de las condiciones de operación y la adquisición de las imágenes para cada caso

```
global original Kinect Camara real simulado
if Kinect
if real
colorVid = videoinput('kinect',1,'RGB_640x480');
start(colorVid);
colorImage=getsnapshot(colorVid);
depthVid = videoinput('kinect',2,'Depth_640x480');
start(depthVid);
depthImage = getsnapshot(depthVid);
 [alignedColorImage,flippedDepthImage] = alignColorToDepth(depthImage,colorImage,depthVid);
 original=alignedColorImage;
 save 'ImagenesKinect.mat' alignedColorImage flippedDepthImage
elseif simulado
```

```
ObtenerImagenesVrep2;
original=img_left;
flipped_depth=imrotate(img_depth,180);
end
elseif Camara
if real
vid = videoinput('winvideo', 2);
start(vid);
original=getsnapshot(vid);
elseif simulado
  original=imread('Imagen1.JPG');
end
end
```

3.2.2.- Configuración del Espacio de Trabajo

Resulta imprescindible para nuestra aplicación identificar la relación entre las imágenes y espacio de trabajo para que el sistema localice los objetos de forma óptima y así el robot pueda manipularlos correctamente. Deberemos tener en cuenta no sólo que la cámara no se va a encontrar siempre en el mismo punto exacto ni va a mirar desde exactamente la misma angulación, aunque se mantenga colocada en una misma posición, sino que además se va a trabajar con diferentes cámaras y por lo tanto con distintas resoluciones, por lo que el sistema deberá adaptarse a las necesidades de cada tipo de operación.

El proceso de calibración consiste en la detección previa de dos objetos (fichas) de color azul colocados de forma que además de realizar la conversión de píxeles a milímetros delimiten el área de trabajo del robot, por lo que se deberán posicionar las referencias delimitando una zona no muy alejada de éste donde no tenga problema para coger objetos de cierta altura. De esta manera diseñaremos el proceso de calibración con objeto de que no se configure mediante unos parámetros fijos, sino que los mantendremos accesibles a nivel de código para modificarlos sobrescribiendo las variables de trabajo guardadas como “datos_mm.mat”.

La forma de calcular la posición de los centroides de los objetos de calibración como se muestra en la figura 3.2 se efectúa de la misma manera en la que se procederá a la hora de detectar los objetos que deseemos manipular con el robot por lo que para entender el proceso recomendamos avanzar hasta el apartado 3.2.3 Procesado Digital de Imágenes.

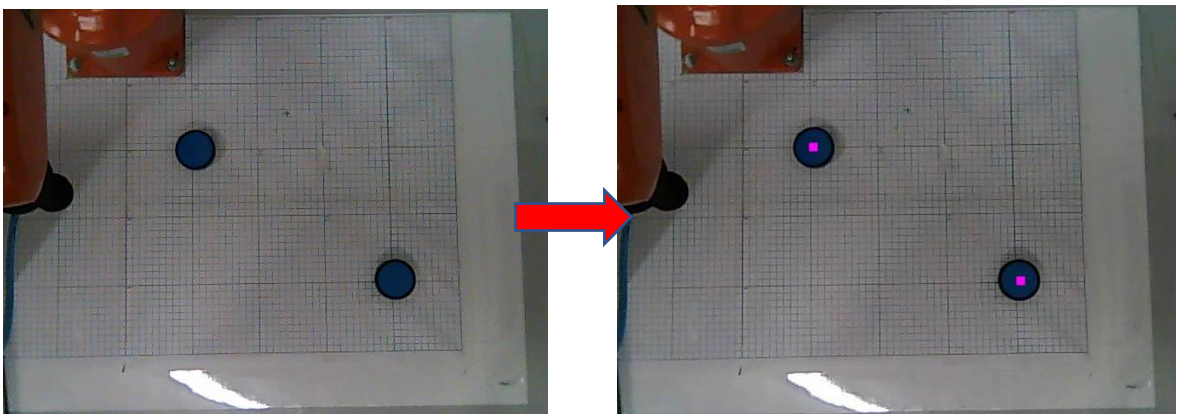

Figura 3.3.*-Detección de objetos para la calibración*

Por defecto, el sistema utiliza los parámetros de calibración establecidos la última vez que se ejecutó por lo que, si deseamos recalibrar el sistema deberemos modificar las variables con la nueva posición establecida y ejecutar desde la interfaz gráfica el protocolo de calibración

guardando posteriormente los cambios. El cálculo que se realizará para conocer la posición de cada objeto que se desee detectar será el siguiente (ejemplo para objetos de color azul):

```
YAzul(n)=(((x(n)-pixX_ref1)*(mmX_ref2-mmX_ref1))/(pixX_ref2-pixX_ref1))+ mmX_ref1;
XAzul(n)=(((y(n)-pixY_ref1)*(mmY_ref2-mmY_ref1))/(pixY_ref2-pixY_ref1))+mmY_ref1;
```

Cabe destacar que para que exista una correspondencia entre los sistemas de coordenadas establecidos entre los píxeles de MATLAB y las posiciones en ROBOTSTUDIO deberemos intercambiar en esta parte del código como se muestra las coordenadas X e Y.

3.2.3.- Procesado Digital de Imágenes

Tratamiento de la Imagen

Un PDI o Procesamiento Digital de Imágenes está basado en el análisis y manipulación de imágenes mediante computadora, puede considerarse como un procesamiento digital en 2 dimensiones que nos sirve para descubrir o resaltar información contenida en una imagen. En concreto, ya que nuestra aplicación se centra en el tratamiento del color es interesante realizar previamente una pequeña introducción:

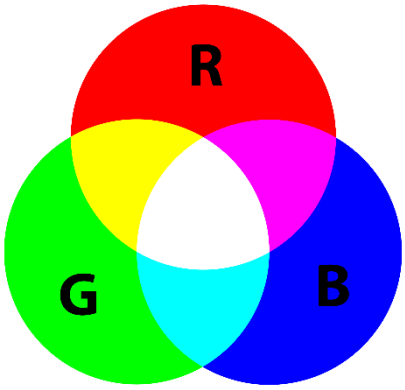

Figura 3.4.-*Concepto RGB*

RGB es el concepto utilizado para referirse a un modelo cromático que representa los distintos colores a partir de la mezcla de los tres primarios, de ahí su nombre RGB (Red-Green-Blue), o lo que es lo mismo, el modelo RGB se fundamenta en la síntesis aditiva del color, por la cual la luminosidad de los colores primarios en diferentes proporciones forma el resto de colores. Debido a esto en informática muchos lenguajes de programación siguen este sistema y asignan cuantitativamente un valor por color, cuanto más alto sea mayor será la intensidad en la mezcla. De esta manera cada imagen está formada por 3 matrices de dimensiones iguales a su resolución, la primera representa al color rojo, la segunda al verde y la tercera al azul.

Los procesos principales que aplicaremos en nuestra imagen y que explicaremos a continuación han sido desarrollados por Javier Ribera Díaz [8] en su TFG titulado “Manipulación de objetos en una cinta transportadora mediante un sistema de visión artificial y brazo robot IRB120”, utilizaremos como ejemplo el tratamiento realizado para hallar los objetos de color rojo sobre la siguiente imagen:

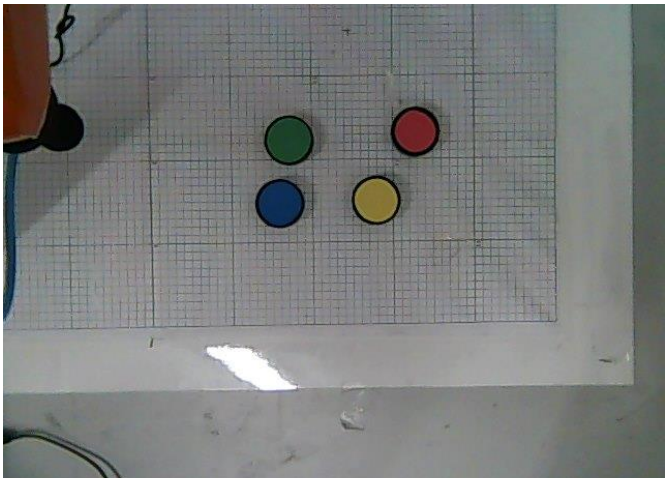

Figura 3.5.- *Imagen para tratamiento color rojo*

1.-En primer lugar, tras obtener la imagen sobre la que deseamos realizar el tratamiento (la cual almacenamos en nuestro programa como la variable global 'original') ejecutaremos la sentencia rgb2gray para convertirla en escala de grises (imagen 'gris') y con ello eliminar la saturación y el color en la imagen conservando la luminancia. Por otro lado extraeremos de la imagen 'original' la componente 1 del RGB, es decir, los valores de intensidad de color rojo de la imagen

```
img = original;
gris = rgb2gray(original
rojo = img(:,:,1);
```

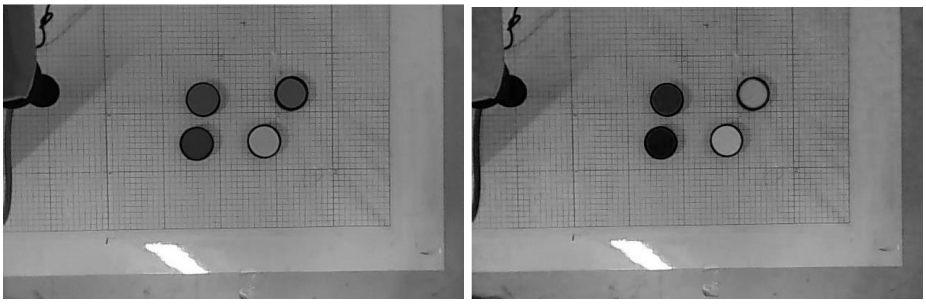

Figura 3.6.-*Imágenes gris y rojo*

2.- Una vez obtenida la información sobre el color rojo le restaremos a ésta la imagen en escala de grises previamente obtenida de manera que sólo aparezcan en la nueva imagen 'resta' los datos correspondientes a los objetos de interés, en este caso los de color rojo, además aumentaremos la intensidad de este efecto multiplicando el valor resultante por un factor 3 de valor fijo.

```
img_rojo = cat(1,rojo);
resta = (img_rojo - gris)*3;
```

Figura 3.7.-*Imagen resta*

3.- A continuación, se realiza una binarización de la imagen según el comando im2bw, lo que a grandes rasgos se puede asemejar a una comparación entre los valores intensidad del color rojo de la imagen con un valor que se obtiene en tiempo real del slide correspondiente en el GUI, si el valor del píxel es mayor que el límite establecido (acotado entre 0 y 1) le corresponderá el color blanco (1) y de lo contrario se le asignará el color negro (0). Durante la ejecución del programa podremos realizar esta operación varias veces hasta quedarnos con la imagen 'bin' que refleje correctamente los objetos que estamos intentando destacar y descarte el resto.

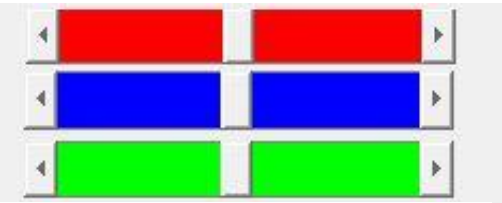

Figura 3.8.- *GUI herramienta slides*

```
v=get(handles.sliderRojo,'Value');
bin= im2bw(resta, v);
```

Figura 3.9.-*Imágenes bin con v igual a 0.2 y 0.6*

4.- Como podemos observar en las imágenes anteriores, al binarizar una imagen podemos encontrarnos con una pequeña parte de ruido debida a la pérdida de algunos pixeles, por ello la forma y por lo tanto el centroide de los objetos que deseamos localizar pueden verse afectados, así que será necesario realizar sobre éstas las dos operaciones básicas de la morfología aplicada al procesamiento de imágenes, las cuales se caracterizan porque el estado de cualquier pixel de la imagen de salida viene definido por la operación que se aplica al pixel correspondiente y a sus vecinos de la imagen de entrada, estas operaciones son:

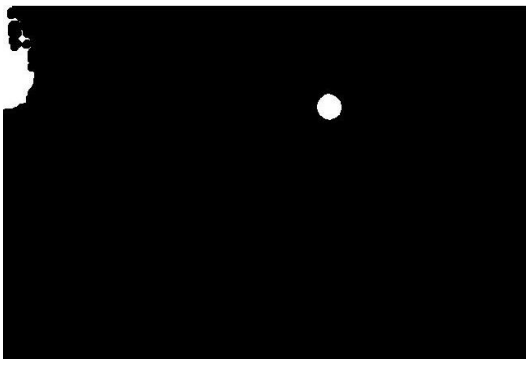

Figura 3.10.- *Imagen bin tras filtro de erosión*

-Erosión: Este filtro se ejecutará en primer lugar tiene la función de eliminar los píxeles de los límites de los objetos, de esta forma si uno de los pixeles colindantes al objeto tiene valor 0 el filtro establecerá el pixel de estudio en 0. Cabe destacar que en ambos filtros se utilizará previamente la función 'strel' para crear un elemento estructural morfológico plano que funcionará de superficie sobre la que trabajarán ambos filtros.

```
ele = strel('disk', 5);
bin = imerode(bin, ele);
```

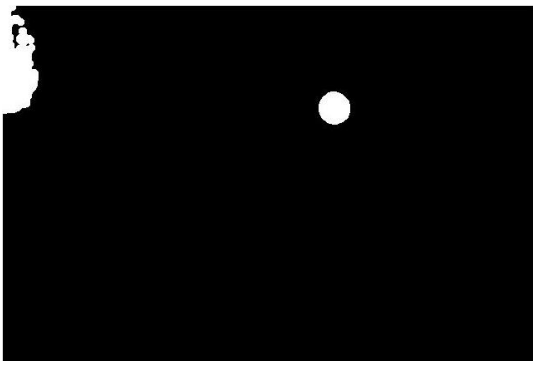

Figura 3.11.- *Imagen bin tras filtro de dilatación*

-Dilatación: Ejecutaremos este filtro en segundo lugar para añadir píxeles en los límites de los objetos, de esta forma si uno de los pixeles colindantes al objeto tiene valor 1 el filtro establecerá el pixel de estudio en 1.

```
ele2 = strel('disk', 5);
bin = imdilate(bin, ele2);
```

5.- Por último, se limpiarán los bordes de error de los objetos con el comando 'imclearborder', que suprime las estructuras más ligeras que estén conectadas al borde de la imagen. Con esto daríamos por concluido el tratamiento realizado sobre la imagen y sólo sería necesario analizar y administrar el número de superficies 'sin agujeros' presentes en ella, los cuales aparecerán etiquetados en la matriz 'L' a partir de la cual podremos sacar propiedades tales como el centroide o la orientación.

```
bin = imclearborder(bin);
[B, L] = bwboundaries(bin,'noholes');
imageR=bwlabel(bin,8);
prop = regionprops(L, 'all');
```

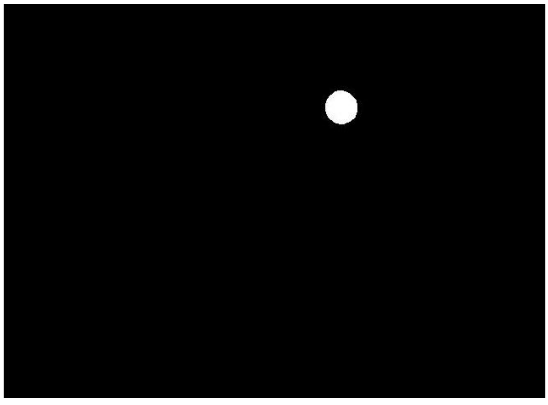

Figura 3.12.- *Imagen bin tras filtro final*

Cálculo de la orientación de los objetos

De cara a la aplicación con el uso de la herramienta gripper, donde las fichas no van a ser circulares sino cuadradas y por lo tanto toma importancia tener en cuenta la orientación de los objetos, se ha desarrollado un pequeño código para poder conocerla de una forma sencilla. Aprovechando el código explicado previamente para conseguir el tratamiento de la imagen, nos valdremos de la última línea mostrada donde puede observarse que hemos almacenado las propiedades de los objetos detectados dentro de la variable tipo estructura llamada 'prop' ya que, a partir de ésta, podremos hallar una serie de parámetros interesantes:

La forma en la que el sistema interpreta un objeto [3] (organizado dentro de una región determinada) es como en la figura 3.13, de forma que genera una elipse caracterizada por dos ejes, uno mayor y otro menor. Gracias a esto es posible calcular la orientación, ya que si el objeto estuviese girado, el valor que nos aportaría la sentencia 'prop.Orientation' es un escalar que especifica el ángulo existente entre el eje x y el eje mayor de la elipse, el cual puede variar entre $\pm 90^{o}$. Por otro lado, hemos comprobado empíricamente que las piezas se nos muestran giradas siempre $\pm 45^{o}$, por lo que hemos desarrollado la siguiente corrección:

```
ang(x)=prop(x).Orientation;
ang(x)=ang(x)+45;
if ang(x)>90
   ang(x)=ang(x)-90;
end
```

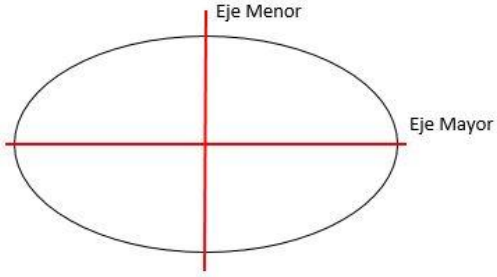

Figura 3.13.- *Elipse caracterizada por ambos ejes*

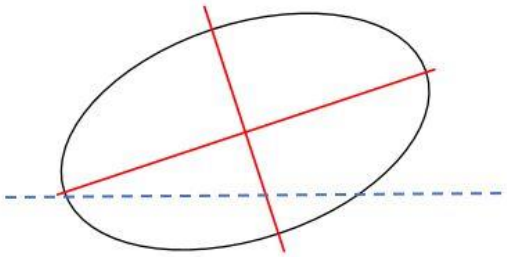

Figura 3.14.- *Diagrama ejemplo orientación*

3.2.4.- Cálculo de la Altura y Posición Final

Hasta ahora sólo nos hemos preocupado en un procesamiento de la imagen dedicado exclusivamente a identificar las diferentes fichas que se mostrasen en ella, pero existe además un análisis fundamental que consistirá en captar y procesar una imagen de profundidad la cual llevaremos a cabo cuando ejecutemos el programa con la cámara Kinect tanto en modo simulación como en real. Esto será imposible de realizar cuando trabajemos con la cámara web porque no puede arrojarnos esos datos, por lo que en ese caso los incorporaremos manualmente (todos los objetos deberán tener la misma altura).

Utilizando la cámara Kinect, ya sea en modo real o simulado, la forma de calcular la altura de los objetos es parecida pero discerniendo algunos detalles importantes. Básicamente en ambos se recorta un cuadrado con el comando imcrop de la imagen de profundidad, con los puntos xmin e ymin colocados 20 pixeles por debajo del centroide inicial de la figura y de 60 pixeles tanto de ancho como de largo, en segundo lugar restaremos el valor que nos aporta el centroide del objeto al valor mínimo encontrado en el cuadrado (que se debe corresponder con algún punto de la mesa de trabajo). Lo que cambia entre ambos métodos es la forma de tratar esos datos a nivel de código y la interpretación que se hace sobre ellos como explicaremos a continuación:

Código	Explicación

if auxA
distancia=[];
for i = 1:auxA
imagenAux = imcrop(flipped_depth,[(int16(xB(i))-20) (int16(yB(i))-20) 60 60]);
maxdist = min(max(imagenAux));
distancia(i)= flipped_depth(int16(yB(i)),int16(xB(i)))-maxdist(:,:,1);
for j = 1:11
if d(j)==distancia(i)
distancia(i)=a(j);
end
end
end
distanciaA = double(distancia);
end
	Para el modo simulado, en VREP hemos probado empíricamente los valores que arroja el sistema para las fichas de cada una de las alturas disponibles (de 10 a 100 mm) y los hemos almacenado dentro del array de valores 'd' comprobando que éste no es completamente lineal, después comparamos en bucle el valor obtenido en el análisis con estos posibles resultados para igualar en consecuencia la variable referida a la altura de cada objeto con el correspondiente valor del array 'a', el cual nos muestra la equivalencia real de altura.

if auxA
distancia=[];
for i = 1:auxA
imagenAux = imcrop(flippedDepthImage,[(int16(xB(i))-20) (int16(yB(i))-20) 60 60]);
maxdist = max(max(imagenAux));
distancia(i)=maxdist(:,:)-flippedDepthImage(int16(yB(i)),int16(xB(i))) ;
if distancia(i)>=a(1) && distancia(i)<(a(1)+5)
distancia(i)=a(1);
else
for j = 2:11
if distancia(i)>=(a(j-1)+5) && distancia(i)<=(a(j)+4)
distancia(i)=a(j);
end
 	
end
if distancia(i)>=(a(11)+5)
distancia=a(11);
end
end
end
distanciaA = double(distancia);
end
	Para el modo real, como dispondremos del mismo tipo de fichas y necesitamos valores fijos de altura (en escalones de 10 en 10mm), en forma de código y valiéndonos de nuevo del array 'a', estableceremos un redondeo sencillo por aproximación y haremos que el máximo tamaño sea el de 100mm.
	El código funciona de forma parecida al anterior, simplemente modificamos de forma leve la forma de adquirir los datos, pero posteriormente en este caso el redondeo establece que si un objeto dista menos de 5 valores por debajo o 4 por encima del valor de 'a' se seleccione como el valor de altura del objeto.

Tabla 3.1.- *Cálculo altura en modo real y simulado*

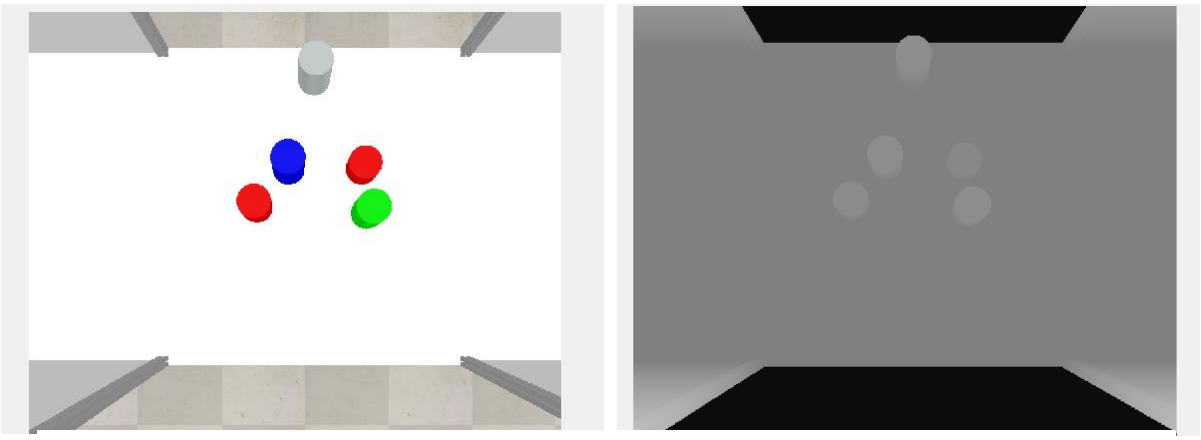

Figura 3.15.- *Imagen diferentes alturas*

Corrección en altura:

Basándonos en el desarrollo del TFG realizado por Álvaro Fernández Expósito [9], si procediésemos a posicionar el robot en el punto del centroide inicial obtenido por el sistema nos daríamos cuenta de que, dependiendo de la altura del objeto en cuestión, se produciría un error más o menos significativo producido por el siguiente efecto:

La cámara no es capaz de interpretar que el objeto tiene cierta altura y debido a esto y a la propia altura de la cámara el centroide observado aparece desviado ligeramente, aunque con un sencillo cálculo trigonométrico podremos realizar el ajuste.

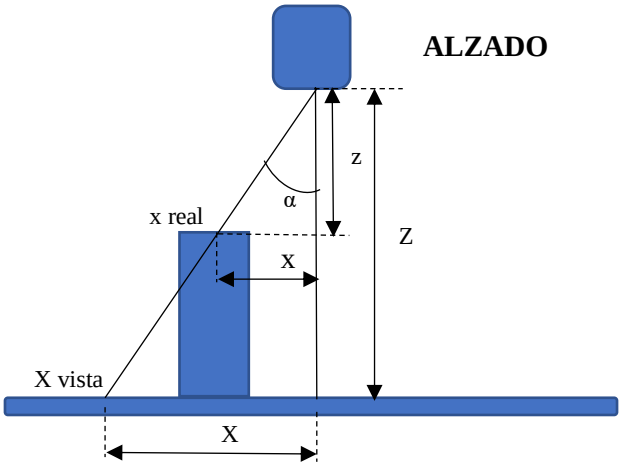

Figura 3.16.- *Diagrama alzado*

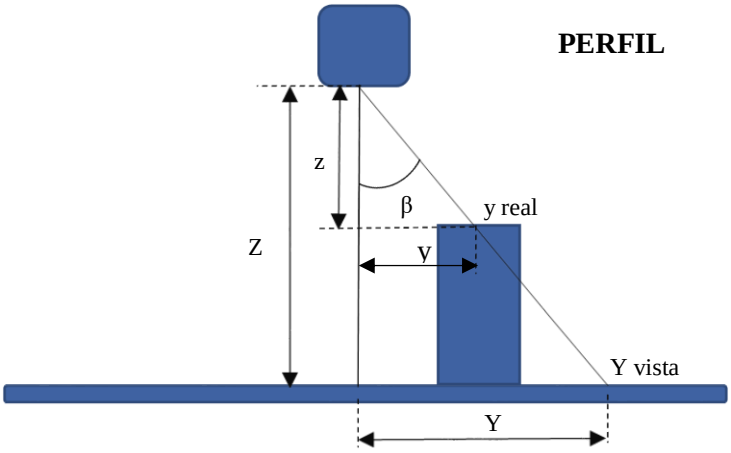

Figura 3.17.- *Diagrama perfil*

$$\tan\alpha = \frac{x}{z} = \frac{X}{Z};\ x = \frac{Xz}{Z}$$

$$\tan\beta = \frac{y}{z} = \frac{Y}{Z};\ y = \frac{Yz}{Z}$$

Deberemos tener en cuenta a su vez que, con objeto de aproximarnos lo mayor posible a la posición real del objeto, interpretaremos que la cámara se sitúa en el centro de nuestra zona de trabajo, por lo que las referencias de los cálculos dependerán de la resolución de la cámara y se almacenarán como 'xmed' e 'ymed'. De esta manera el código será el siguiente:

```
for i=1:auxA
xBREAL(i)=(xB(i)-xmed)*(Acamara-distanciaA(i))/Acamara+xmed;
yBREAL(i)=(yB(i)-ymed)*(Acamara-distanciaA(i))/Acamara+ymed;
end
```

Las variables 'xBREAL' e 'yBREAL' simbolizan la posición final real en pixeles de los objetos azules, es decir, su centroide una vez realizado el ajuste sobre el que posteriormente se realizará la equivalencia en milímetros en el script 'ConvPixMm_AZUL.m'.

3.3.- Comunicación Entre Robot y Usuario

La conexión e interactuación entre el robot (mediante ROBOTSTUDIO y el armario de control IRC5) y el usuario (manejando la GUI de MATLAB) será estrictamente necesaria y de vital importancia para nuestro proyecto, por lo que deberemos asegurar que se establezca de forma precisa y segura. A nivel de hardware la conexión se realizará mediante un cable ethernet en el caso que nos queramos comunicar con el IRC5, en cambio no hará falta ningún elemento adicional para realizar la conexión con ROBOTSTUDIO, puesto que se encuentra en el mismo ordenador y por lo tanto trabajaremos en la red local. Destacar además que la conexión se basa en la previamente desarrollada en el TFG de Javier Ribera Díaz [8]*: "Manipulación de objetos en una cinta transportadora mediante un sistema de visión artificial y brazo robot IRB120*", que a su vez se ha apoyado en el proyecto: *"Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station",* de Marek Jerzy Frydrysiak [7].

3.2.4.- Comunicación Mediante Sockets

Un socket de comunicación es un tipo especial de manejador de ficheros que permite el intercambio de información entre dos programas que pueden (o no) estar en computadoras diferentes. En concreto nos centraremos en establecer la conexión mediante el conjunto de protocolos TCP/IP (Protocolo de Control de Transmisión / Protocolo de Internet) que aúna todas las reglas y requisitos para implementar una arquitectura cliente-servidor entre los equipos de la red.

Envío de sockets mediante datagramas

El sistema de Marek Jerzy Frydrysiak [7] se desarrolló con objeto de establecer una comunicación a bajo nivel mediante datagramas para transmitir informaciones muy distintas entre sí. Un datagrama ejemplo que define la posición u orientación de un objeto tiene la siguiente estructura:

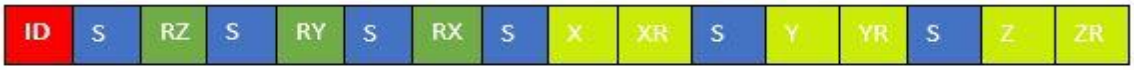

Figura 3.18.- *Diagrama ejemplo estructura datagrama*

-ID, Byte de identificación: Informa del tipo de secuencia enviada.
-S, Valor de signo: Informa sobre el sentido de los giros o los desplazamientos que deberán ejecutarse, 1 para positivo y 0 para negativo.
-Rz Ry Rx, Valores de rotación del efector final (grados): Informan del valor de rotación en ángulos de Euler xyz.
-X Xr Y Yr Z Zr, Valores de posición del efector final (milímetros): Informan del valor de posición en dos bytes de información separando las centenas (X, Y Z) de decenas y unidades (Xr, Yr, Zr)

Cabe destacar que para el desarrollo de nuestra aplicación precisamos de unos datagrama con estructura mucho más sencilla en donde simplemente en unos enviaremos los datos de posición y orientación que deberá tomar el efector final de nuestro brazo robótico y en otros se enviarán datos sencillos que, a nivel de código, se interpretarán en ROBOTSTUDIO como veremos a continuación.

Intercambio de información entre MATLAB y ROBOTSTUDIO

1.- Envío de datos de MATLAB al robot:

Se han prefijado una serie de funciones que conforman los datagramas adaptados a las necesidades del proyecto y creados para enviar la información desde MATLAB hasta el robot, ya sea a través del IRC5 en el sistema real como en ROBOTSTUDIO [4] para el sistema simulado:

Sentencia	ID	Descripción
robot.connect	1	Establece la conexión entre los sistemas
robot.rotation	2	Realiza un giro del efector final en los ejes x, y o z
robot.position	3	Realiza una traslación del efector final en los ejes x, y o z
robot.TCProtandpos	4	Ejecuta sentencia de movimiento del efector final atendiendo tanto a la rotación como a la traslación
robot.TCPtool	6	Ejerce control sobre la herramienta utilizada, ya sea para abrir o cerrar la mano como para activar la ventosa entre otros
robot.TCPFunction	7	Utilizado como apoyo para lanzar alguna función auxiliar desde MALAB

Tabla 3.2.- *Descripción funciones de los datagramas*

A modo de ejemplo adjuntamos el código de la función robot.TCPtool(), que envía el datagrama con la función 'fwrite':

```
function TCPtool(r,x)
    D6=uint8([6 x]);
    fwrite(r.conexion,D6);
    pause(2);
    fwrite(r.conexion,D6);
    pause(2);
end
```

El datagrama será recibido por el robot mediante la función 'DataProcessingAndMovement', que, obviando la parte del código dedicada a la detección de errores, se basa en un bucle continuo que se encarga de recibir los datos de entrada:

```
WHILE connectionStatus DO
      SocketReceive socketClient \Data:=receivedData \ReadNoOfBytes:=19 \Time:= WAIT_MAX;
      stringHandler(receivedData);
    ENDWHILE
```

Una vez recogido, la sentencia 'stringHandler' procesa el datagrama almacenado en la variable 'receivedData' y entra en una función de SWITCH CASE donde se analizará el ID del datagrama y en función de éste se actuará de la manera pertinente en cada caso. Continuando con el ejemplo anterior, para la función robot.TCPtool(), el ID correspondiente según indicamos previamente sería el 6 por lo que accedería a la siguiente parte del código:

```
CASE 6: !CONTROL HERRAMIENTA

    IF Data2Process{2} = 0 THEN  !(Activa succion de ventosa)
     WaitTime 1;
     SetDO CogerFicha,1;
     SetDO CogerFicha,0;
     SetDO DO10_9,1; !ACTIVAR SEÑAL VENTOSA REAL
     WaitTime 1;


    ELSEIF Data2Process{2} = 1 THEN  !(Desactiva succion de ventosa)
     WaitTime 1;
     SetDO SoltarFicha,1;
     SetDO SoltarFicha,0;
     SetDO DO10_9,0; !DESACTIVAR SEÑAL VENTOSA REAL
     WaitTime 1;

    ELSEIF Data2Process{2} = 2 THEN  !(Activa Manejo del Gripper)
     WaitTime 1;
     SetDO CerrarGripper,1;
     SetDO CerrarGripper,0;
     SetDO DO10_11,1; !ACTIVAR SEÑAL GRIPPER REAL
     WaitTime 1;

    ELSEIF Data2Process{2} = 3 THEN  !(Activa Manejo del Gripper)
     WaitTime 1;
     SetDO AbrirGripper,1;
     SetDO AbrirGripper,0;
     SetDO DO10_11,0; !DESACTIVAR SEÑAL GRIPPER REAL
     WaitTime 1;

      ELSEIF Data2Process{2} = 4 THEN  !(Activa Manejo de mano Inmoov)
     WaitTime 1;
     SetDO CerrarMano,1;
     SetDO CerrarMano,0;
     SetDO DO10_11,1; !ACTIVAR SEÑAL MANO REAL !!!!POR DETERMINAR
     WaitTime 1;

    ELSEIF Data2Process{2} = 5 THEN  !(Activa Manejo de mano Inmoov)
     WaitTime 1;
     SetDO AbrirMano,1;
     SetDO AbrirMano,0;
     SetDO DO10_11,0; !DESACTIVAR SEÑAL GRIPPER REAL  !!!!POR DETERMINAR
     WaitTime 1;
    ENDIF
```

4.- Desarrollo de la Aplicación. Resultados

A continuación, desarrollaremos el proceso de ejecución del proyecto, las distintas posibilidades que nos ofrece y los resultados que nos arroja.

Para comenzar el desarrollo del sistema deberemos inicializar por defecto los programas de MATLAB, VREP y ROBOTSTUDIO de la siguiente forma:

Primero arrancaremos el programa VREP el cual, según lo explicado, sólo será necesario en el caso de que deseemos simular la captación de las imágenes a través de la cámara Kinect. De ser así deberemos mantener activas para poder simularlas cuando se precise la escena propia de calibración y la creada específicamente para la verificación de los objetos (figura 4.1).

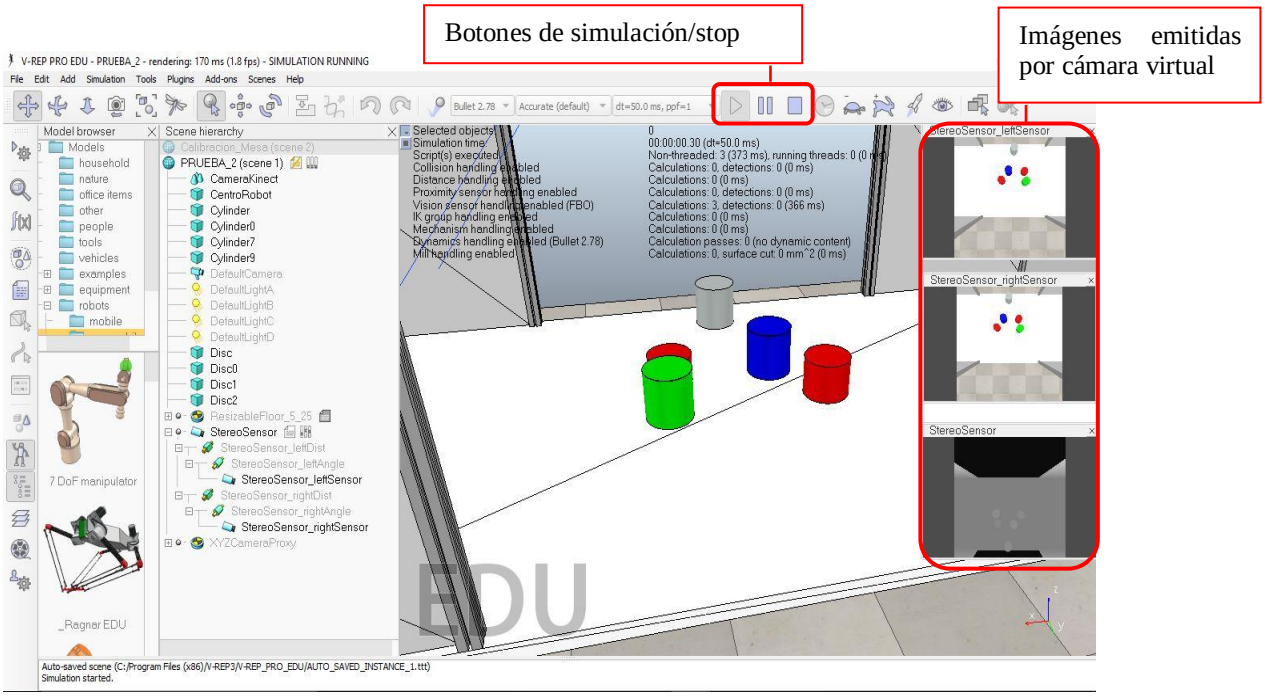

Figura 4.1.- *VREP pantalla estación*

Posteriormente inicializaremos el programa ROBOTSTUDIO y realizaremos la acción de Unpack and Work de la estación deseada según la aplicación (hemos diseñado varias estaciones según la herramienta y los objetos del espacio de trabajo). Ahora, en función de si trabajaremos con el robot real o mediante el simulador, en el módulo en RAPID 'TCPIPConnectionHandler' escribiremos la dirección IP correspondiente, que será 127.0.0.1 para el modo simulación y 172.29.28.185 para la estación real (y por lo tanto tendremos que estar conectados mediante un cable de red). En el caso de trabajar en modo real también deberemos modificar las señales necesarias a efectos de simulación, es decir, en nuestro caso las correspondientes a los componentes inteligentes, que dejaremos comentadas a fin de guardar el programa después y cargarlo en el FlexPendant. El programa se guarda desde la pestaña vertical del controlador haciendo click derecho sobre el icono T_ROB1.

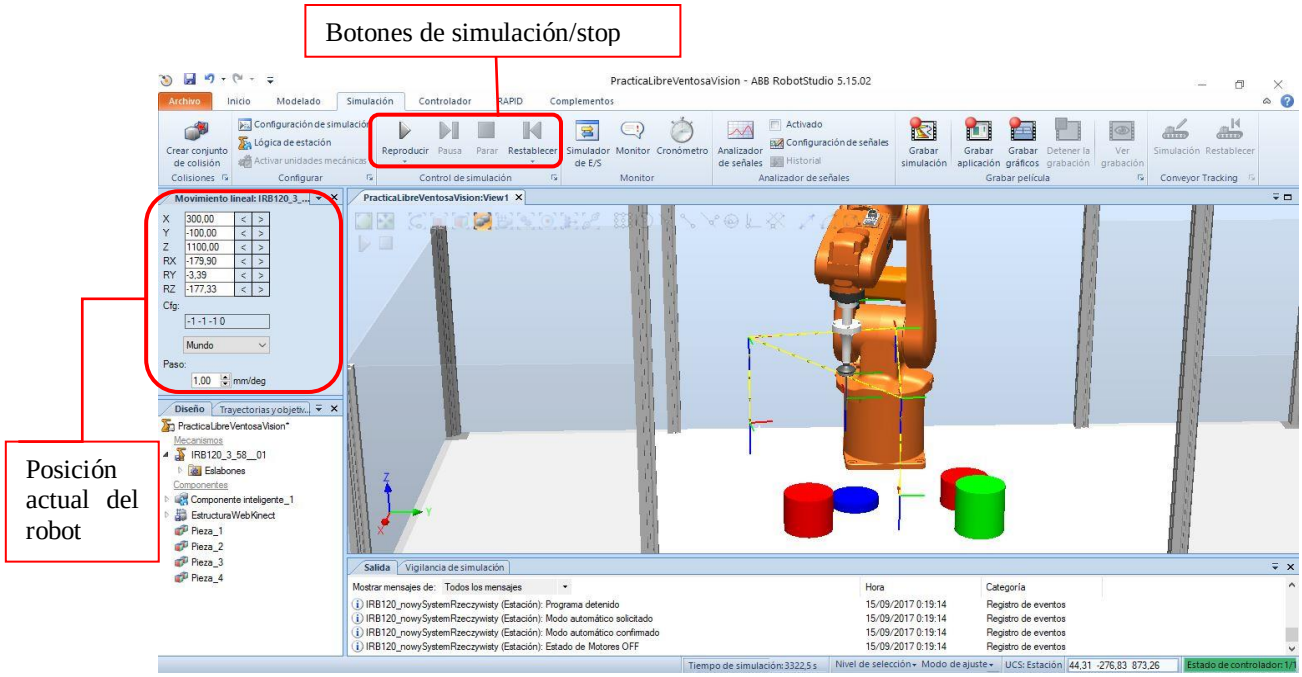

Figura 4.2.- *Robotstudio pantalla estación*

Habiendo concluido los pasos anteriores, procederemos a arrancar la GUI de MATLAB apareciéndonos un interfaz con el siguiente aspecto:

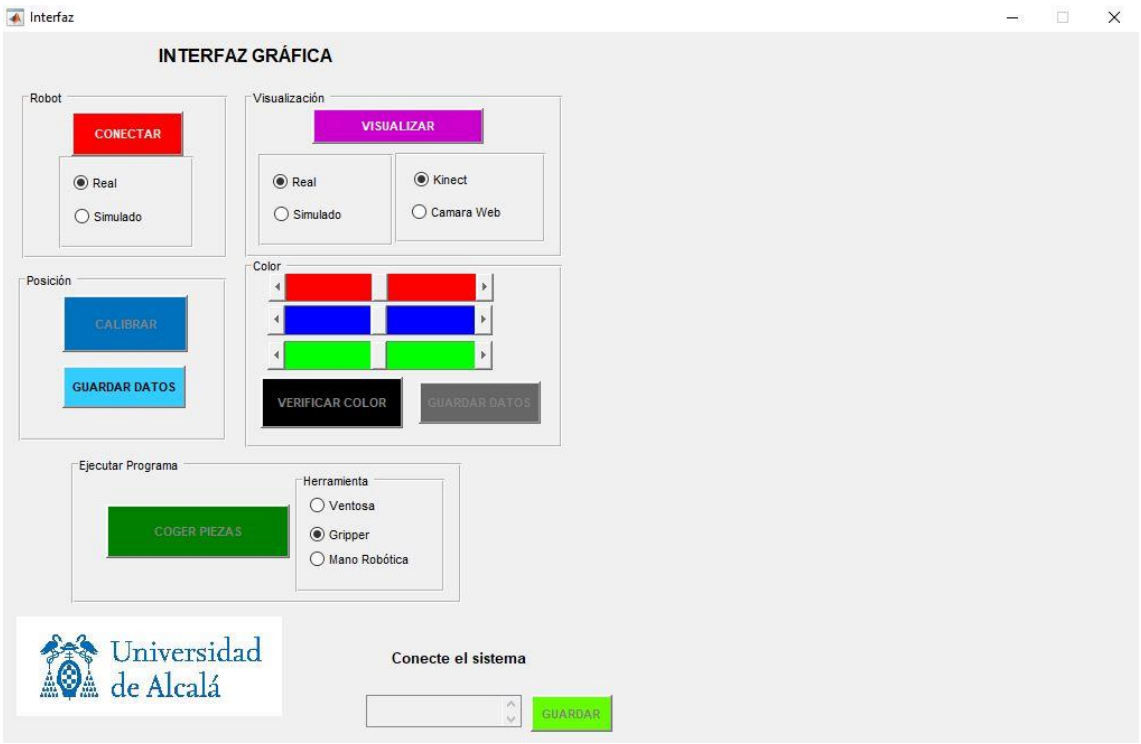

Figura 4.3.- *GUI pantalla de inicio*

Como podemos observar, la mayoría de opciones hasta el momento aparecen deshabilitadas y por defecto, los mensajes informativos del sistema localizados en la parte inferior del interfaz aconsejan establecer la conexión con el resto de programas, por lo que antes de hacer click sobre el botón 'CONECTAR' seleccionaremos las opciones de operación deseadas, ya sea utilizar las cámaras web o Kinect tanto en modo real como simulado, trabajar en el simulador del robot o en la estación real o seleccionar la herramienta con la que manipularemos los objetos.

Una vez realizada la conexión nos aparecerá el mensaje 'Conexión Completada', por lo que a partir de este punto podremos proceder directamente a la verificación de las figuras en el espacio de trabajo o, por el contrario, realizar una serie de acciones previas a ésta que veremos a continuación:

-En primer lugar, el sistema nos permite visualizar aquello que el sistema de visión está captando, tanto la imagen de color como la de profundidad si existe, partiendo de la premisa de que en modo simulación nos mostrará simplemente la imagen que hayamos asignado mientras que si utilizamos las cámaras, nos muestra un video a tiempo real de lo que está ocurriendo en el espacio de trabajo. Si pulsamos una vez sobre el botón 'Visualizar' aparecerá la imagen y simplemente volviéndolo a pulsar ésta desaparecerá.

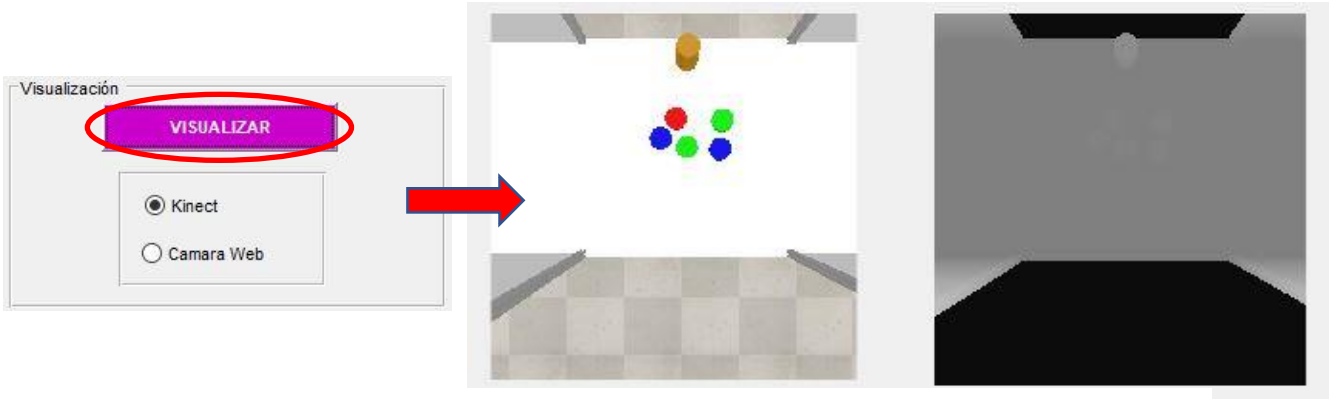

Figura 4.4.- *GUI sentencia visualizar*

-En segundo lugar, tendremos la posibilidad de realizar el proceso de calibración y, basándonos en lo anteriormente explicado, para ejecutarlo correctamente deberemos colocar dos piezas azules que funcionen como referencias del sistema en las posiciones indicadas (que deberán ser las mismas que las almacenadas en código). Una vez pulsemos el botón 'CALIBRAR' el sistema nos mostrará una imagen en la que aparecerán resaltados los centroides de ambas figuras y además el mensaje de confirmación 'Calibración Terminada' aunque ésta no será efectiva hasta no guardar los datos mediante el botón correspondiente del panel. Cabe destacar que, si no se realiza ninguna calibración, el sistema trabajará con los datos de operación obtenidos la última vez que se ejecutó.

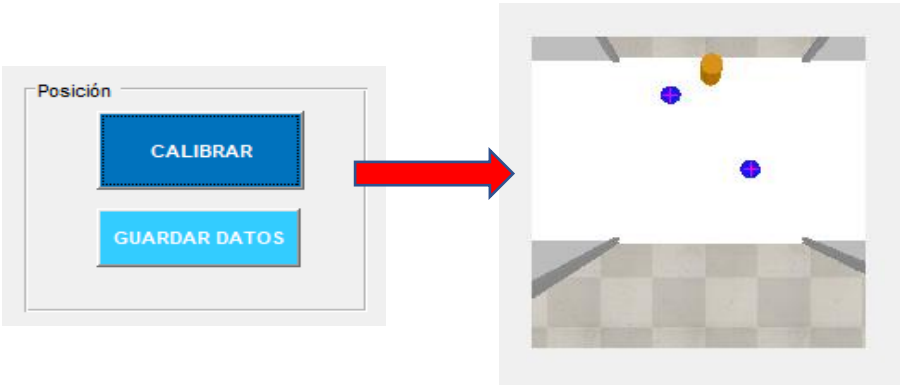

Figura 4.5.- *GUI sentencia calibrar*

El siguiente paso será proceder a realizar la verificación de las posiciones y alturas de los objetos en cuestión, para lo cual simplemente pulsaremos sobre el botón de 'VERIFICAR COLOR' y en función del tipo de cámara e independientemente de si es real o simulado se nos abren dos posibilidades:

-Si utilizamos la cámara web nos encontraríamos ante la particularidad de que el sistema no es capaz de calcular la altura de las piezas, por lo que en este paso se activaría una celda en la parte inferior del interfaz en donde, según está indicado deberemos introducir el valor de la altura de los objetos en cuestión, el cual debe ser común a todos para simplificar el proceso. Posteriormente guardamos el dato pulsando sobre el botón 'GUARDAR' y entonces el sistema reanudará la ejecución de la verificación, detectando así las piezas disponibles.

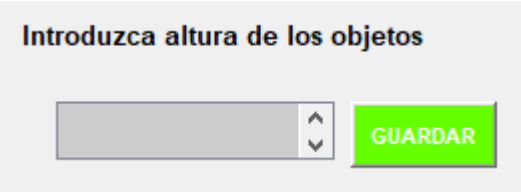

Figura 4.6.- *GUI altura objetos camWeb*

-Si por el contrario utilizamos la cámara Kinect, el sistema reconocería la altura de los objetos por sí solo según lo explicado en apartados anteriores por lo que directamente nos aparecerían las imágenes correspondientes según los objetos que aparezcan con cada uno de sus centroides resaltados.

Se puede dar el caso de que, una vez finalizada la verificación (lo que ocurre cuando aparece el mensaje 'Verificación Terminada') nos demos cuenta de que alguna de las piezas no se ha reconocido correctamente como en la figura 4.7, en donde, a pesar de haber una pieza de color rojo en el espacio de trabajo ésta no ha sido detectada. El procedimiento de actuación en este caso consistiría en desplazar el slide del color correspondiente de forma que modifiquemos el umbral de binarización de detección, además como ayuda nos aparecerá con cada cambio la imagen binarizada según el parámetro en cuestión para que podamos observar qué cambios se producen hasta llegar al punto deseado por el usuario. Después solo tendríamos que guardar los datos y realizar de nuevo la verificación para que el sistema almacene la posición del nuevo objeto. Este problema puede surgir análogamente en caso de que durante la calibración el sistema no reconozca alguna de las piezas azules o detecte más de dos objetos, en tal caso actuaríamos de la misma forma que lo explicado para la verificación.

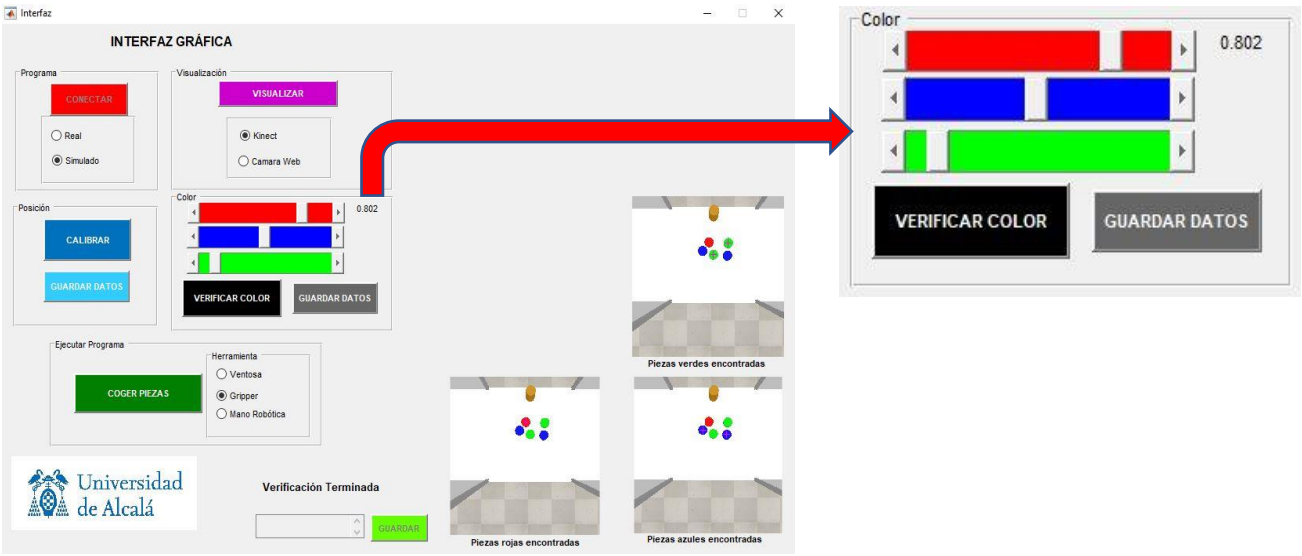

Figura 4.8.- *GUI verificación con objetos rojos corregida*

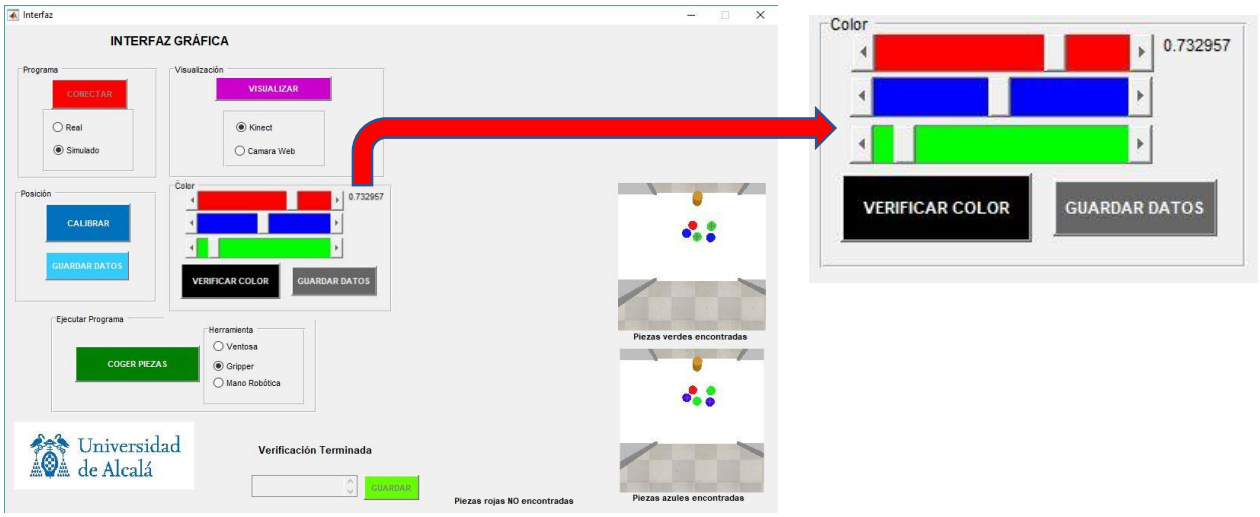

Figura 4.7.- *GUI Verificación sin objetos rojos*

Una vez terminado el proceso de verificación el sistema está preparado para enviar las posiciones de los objetos detectados (ya corregidos y convertidos de píxeles a milímetros) al robot, acción que ejecutaremos seleccionando en primer lugar la herramienta que deseamos utilizar comprobando que coincida con la que se encuentra conectada al robot. Posteriormente haremos click sobre el botón 'COGER PIEZAS' (el cual aparecerá habilitado una vez se hayan realizado los pasos previos) y se iniciará la secuencia de movimientos del robot por colores en el orden rojo, azul y verde partiendo desde una posición de reposo. Adjuntamos como ejemplo la sentencia correspondiente a los objetos de color rojo e imágenes del robot en simulación recogiendo y depositando una pieza (figura 4.9):

```
pos_reposo=[180 0 180 0 -370 300];
robot.TCProtandpos(pos_reposo);
pause(1);
%Primero se recogerán los objetos de color Rojo
for n=1:auxR
  robot.TCPtool(1);
   robot.TCProtandpos([180 0 180 round(XRojoREAL(n)) round(YRojoREAL(n)) 300]);
   robot.TCProtandpos([180 0 180 round(XRojoREAL(n)) round(YRojoREAL(n)) distanciaR(n)]);
   pause(1);
  robot.TCPtool(0);
  pause(1);
   robot.TCProtandpos([180 0 180 round(XRojoREAL(n)) round(YRojoREAL(n)) 300]);
   pause(1);
    %Dejamos el cilindro en una posición determinada
  robot.TCProtandpos([180 0 180 Xdepo_R Ydepo_R 300]);
   pause(1);
   robot.TCProtandpos([180 0 180 Xdepo_R Ydepo_R distanciaR(n)]);
   pause(1);
   robot.TCPtool(1);
   pause(1);
   robot.TCPFuncion(0);
   pause(1);
   robot.TCProtandpos([180 0 180 Xdepo_R Ydepo_R 300]);
   pause(1);
    robot.TCProtandpos(pos_reposo);
end
```

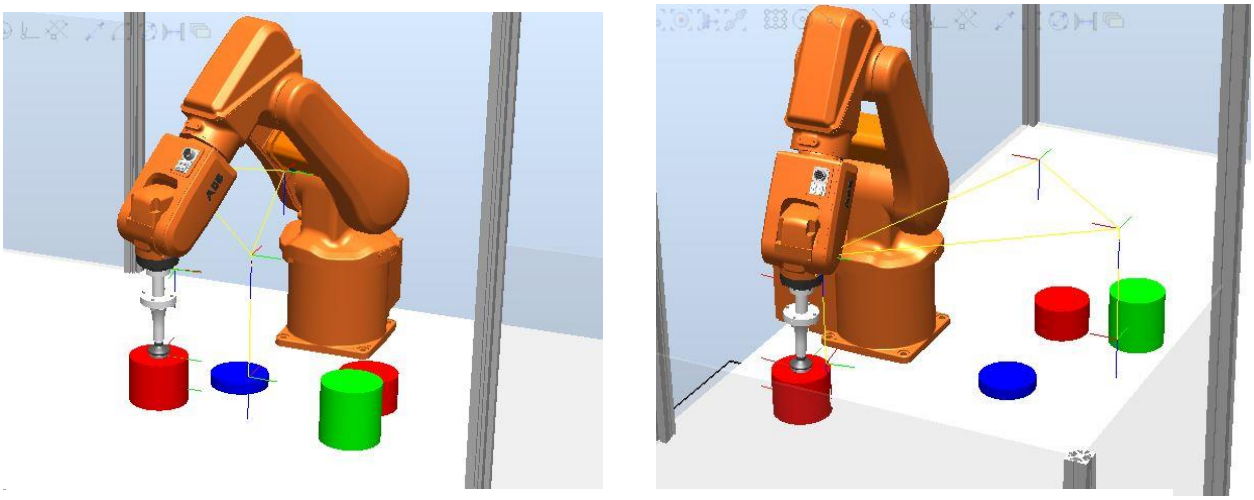

Figura 4.8.- *Robotstudio manipulación de objetos*

Cada vez que llega un nuevo objeto al depósito del color indicado éste se borra en simulación mediante la sentencia TCPFuncion (0) para que no se solape con otros posibles objetos, en modo real en cambio estaremos atentos de recogerlo para que no se produzcan colisiones. Además, cabe destacar que al tener configurado en simulación una copia exacta del robot de trabajo del laboratorio, en ambos casos éstos actuarán de forma exacta, lo que resultó muy útil durante la fase de pruebas del sistema puesto que nos asegurábamos de cumplir con las especificaciones.
Acabada la secuencia de movimientos llevada a cabo por el robot, ya sea en modo simulación o en el entorno real de trabajo, éste quedará de nuevo en la posición de reposo y en el interfaz aparecerá el mensaje 'Aplicación Finalizada', por lo que las piezas en cuestión habrán sido debidamente colocadas en sus respectivos depósitos y daremos por concluido el trabajo.

5.- Manual de Usuario

Los pasos a seguir para realizar un correcto uso de la aplicación, teniendo en cuenta la variabilidad de opciones de las que disponemos, serían los siguientes:

1.- Inicializar independientemente las estaciones específicas de trabajo de VREP y ROBOTSTUDIO y en MATLAB más concretamente, inicializaremos el script 'Interfaz.m' seleccionando además el directorio donde hallamos dispuesto el resto de archivos de código del programa. Por otro lado conectaremos las cámaras Web y Kinect al ordenador y sobre la pantalla de comandos de MATLAB escribiremos el código "imaqreset" para resetear los objetos de adquisición de imágenes existentes en memoria y que detecte los dispositivos sin problema. De esta manera tendremos a nuestra disposición todas las opciones disponibles que nos brinda el sistema (si sabemos seguro que no vamos a utilizar alguna opción en concreto simplemente obviaremos esa parte).

2.- Si deseamos trabajar con el robot simulado simplemente seleccionaremos en el interfaz, dentro del panel robot, la opción de simulado y a su vez en ROBOTSTUDIO comenzaremos la simulación. En caso de querer trabajar con el robot real deberemos, en primer lugar, seleccionar la opción de real dentro del panel robot y posteriormente acudiremos al ROBOTSTUDIO en el cual, si no hemos realizado este proceso anteriormente, deberemos acudir a la pestaña RAPID y comentar las líneas referidas a elementos de simulación. Además, en el módulo 'TCPIPConnectionHandler' cambiaremos la dirección IP en red local (127.0.0.1) a la propia de la estación de trabajo real (172.29.28.185). Por último, en la misma pestaña RAPID, haremos click derecho sobre el icono T_ROB1 y guardaremos el programa para pasarlo vía USB al FlexPendant y cargarlo desde allí.

3.- Ahora, seleccionaremos las opciones de visualización oportunas en la interfaz, ya sea trabajar con la cámara Web o la Kinect tanto en el modo real como en el simulado. Posteriormente haremos click sobre el botón 'CONECTAR' y esperaremos a que se produzca la conexión entre los sistemas.

4.- A modo de comprobación de que las imágenes se están captando correctamente, podemos pulsar sobre el botón 'VISUALIZAR', aunque si estamos simulando la cámara Kinect deberemos elegir la estación y comenzar previamente la simulación. En el resto de opciones no habría un paso previo requerido.

5.- En este paso tendremos la posibilidad de realizar una calibración del espacio de trabajo para posteriormente localizar de manera precisa los objetos a detectar en el espacio de trabajo, aunque el sistema trabaja por defecto con los datos almacenados la última vez que se guardó por lo que si las condiciones son las mismas podríamos saltarnos este paso. De considerarla necesaria colocaríamos las dos piezas azules en las posiciones indicadas en el workspace 'datos_mm.mat' (que además son accesibles a modificación a nivel de código) y haríamos click sobre el botón 'CALIBRAR', si esta no se completase moveríamos el slide azul para modificar el umbral de detección hasta que apareciesen las dos imágenes. Una vez aparezca el mensaje 'Calibración Terminada' podríamos guardar los datos para que el sistema los tome como válidos.

6.- Continuamos con uno de los pasos principales del sistema, colocaremos en el espacio de trabajo los objetos a detectar o cambiaremos la estación de simulación en VREP y posteriormente haremos click en el botón 'VERIFICAR COLOR', en el caso de trabajar con la cámara Web deberemos indicar a su vez la altura de los objetos en la parte inferior del interfaz y darle al botón 'GUARDAR'. Si alguno de los objetos en cuestión no se detectase correctamente deberemos modificar el umbral de detección del color en cuestión (de la misma forma que se hizo durante la calibración) y observar la imagen binarizada hasta que aparezca resaltado, cuando esto ocurra haremos click sobre el botón 'GUARDAR DATOS' y volveremos a 'VERIFICAR COLOR'. El paso queda completo cuando se señalan los centroides de los objetos en cuestión y por lo tanto se ha calculado su posición en milímetros dentro del espacio de trabajo.

7.- Por último, tan sólo seleccionaremos el tipo de herramienta con la que manipularemos los objetos (que tiene que ser el mismo que el utilizado en la estación de ROBOTSTUDIO) y haremos click sobre el botón 'COGER PIEZAS'. Una vez hecho esto el robot comenzará a ejecutar las sentencias de movimiento necesarias para desplazar los objetos detectados en el paso anterior a sus respectivos depósitos. Cabe destacar que si utilizamos el sistema real y disponemos de más de una pieza del mismo color será necesario que, una vez el robot la deje en el depósito correspondiente, la apartemos de éste para que la siguiente se coloque en la misma posición de forma óptima.

6.- Conclusiones

El sistema se fundamenta en la multiplicidad de opciones a la hora de adquirir las imágenes y a su vez en la forma en la que, una vez recogidos los datos de éstas se convierten para poder desarrollar la aplicación de forma automatizada, aunque dirigida por un operador a través de una interfaz gráfica.

De esta manera la exactitud del sistema resulta ser directamente proporcional a la calidad de la adquisición de las imágenes y al tratamiento que establezcamos sobre éstas, por lo que preferentemente trabajaremos con las imágenes adquiridas en cada momento en lugar de almacenarlas para trabajar con ellas en el momento que se precise, ya que MATLAB reduce la calidad de estas imágenes y por lo tanto la precisión de nuestro sistema, aunque admitiremos cierta tolerancia que se incrementará sobre todo en función de la altura de los objetos y lo alejados que éstos se encuentren del centro del espacio de trabajo.

Otro de los pilares sobre los que se fundamenta el proyecto es la conjunción sincronizada de los diferentes elementos que actúan orquestados por la interfaz gráfica desarrollada en MATLAB, que nos permite comunicar imágenes, videos y datos entre varios programas y dispositivos diferentes funcionando simultáneamente entre ellos.

7.- Pliego de Condiciones

Describiremos a lo largo de este capítulo las características de las herramientas utilizadas para el desarrollo de la aplicación:

HARDWARE

1.- Ordenador portátil Lenovo z500
-Intel Core i7-3632QM 2.2 GHz, RAM de 16GB
-Windows 10, 64 Bit
-NVIDIA GeForce GT 645M

2.- Brazo robótico IRB-120 de ABB
-Controlador IRC-5
-25Kg de peso, levanta cargas de hasta 3kg
-6GDL, muñeca esférica

3.- Cámara Logitech C310
-Resolución 640x480 píxeles
-Enfoque y apertura automáticos

4.- Cámara Kinect for Windows
-Compatible con PC mediante uso de cable USB
- 640x480 píxeles con 32-bit de color @30fps
1280x960 RGB @12fps
RAW YUV 640x480 @15fps
-Por defecto trabaja dentro del rango de 0.8 a 4m de forma óptima, a partir de 8m no recoge datos

5.- Sistema neumático de succión

6.- Herramientas mano Inmoov, ventosa y gripper

SOFTWARE

1.- S.O. Windows 10

2.- MATLAB R-2015a

3.- VREP PRO EDU

4.- ABB Robotstudio 5.15.02

5.- Microsoft Office Student 2016

8.- Planos

Adjuntaremos en este capítulo el código base de la interfaz gráfica en MATLAB sobre el que se fundamenta el sistema general sin profundizar en cada una de las funciones y scripts propias utilizadas, además obviaremos las líneas de código iniciales comentadas por el autor y generadas automáticamente.

```
function varargout = Interfaz(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Interfaz is made visible.
function Interfaz_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Interfaz (see VARARGIN)

% Choose default command line output for Interfaz
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%%%Botones de interfaz desactivados al iniciar el programa
set(handles.coger,'Enable','off');
set(handles.verificar,'Enable','off');
set(handles.calibrar,'Enable','off');
set(handles.Visualizar,'Enable','on');
set(handles.guardarconf,'Enable','off');
set(handles.guardarpos,'Enable','off');
set(handles.valorVerde,'Enable','off');
set(handles.guardar_altura,'Enable','off');
set(handles.estado,'String','Conecte el sistema');
%%%Imagenes de logos de instituciones academicas
uah=imread('LogoUAH.jpg');
axes(handles.uah);
imshow(uah);
axis off;
global R1X R1Y R2X R2Y b c
global real simulado
load ('datos.mat')
```

```
b=1;
c=1;
% UIWAIT makes Interfaz wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Interfaz_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in calibrar.
function calibrar_Callback(hObject, eventdata, handles)
% hObject    handle to calibrar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%%%%%%%%%%%%%%%%%Captura de imagen para calibracion
%%%%%%%%%%%%%%%%%%%%%%

set(handles.estado,'String','Realizando Calibración');
global real simulado Kinect Camara originalcal varcal
varcal=1;
if simulado
 if Camara
originalcal=imread('Calibracion2.JPG');
 elseif Kinect
ObtenerImagenesVrep2;
originalcal=img_left;
 end
elseif real
  if Camara
vid = videoinput('winvideo', 2);
start(vid);
originalcal=getsnapshot(vid);
  elseif Kinect
 colorVid = videoinput('kinect',1,'RGB_640x480');
start(colorVid);
colorImage=getsnapshot(colorVid);
depthVid = videoinput('kinect',2,'Depth_640x480');
start(depthVid);
depthImage = getsnapshot(depthVid);
 [alignedColorImage,flippedDepthImage] =
alignColorToDepth(depthImage,colorImage,depthVid);
 originalcal=alignedColorImage;
  end
end
axes(handles.ImagenReal);
imshow(originalcal);
axis off;
Calibracion;
for n=1:numb
plot(xB(n),yB(n),'-m+');
x(n)=round(xB(n));
y(n)=round(yB(n));
end
```

```
if numb == 2
%%%%%%Asignacion de valores en pixeles a variables globales
R1X=x(1);
R1Y=y(1);

R2X=x(2);
R2Y=y(2);
save 'datosaux.mat' R1X R2X R1Y R2Y
set(handles.estado,'String','Calibración Terminada');
save 'info.mat' originalcal
else
set(handles.estado,'String','Calibración no Completada');
end
% --- Executes on button press in coger.
function coger_Callback(hObject, eventdata, handles)
% hObject    handle to coger (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global robot
load('destino.mat');
load('destinoREAL.mat');
load('numobjetos.mat');
load('alturas.mat');
load ('posdepositos.mat');
ventosa=get(handles.Ventosa,'Value');
gripper=get(handles.Gripper,'Value');
mano=get(handles.Mano,'Value');

if ventosa == 1
    Interfaz_Ventosa;
elseif gripper == 1
    Interfaz_Gripper;
elseif mano == 1
    Interfaz_Mano;
else
    disp('Herramienta no seleccionada');
end
set(handles.estado,'string','Manipulación Conlcuida');



% --- Executes on slider movement.
function sliderRojo_Callback(hObject, eventdata, handles)
% hObject    handle to sliderRojo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Asignación parámetros de lectura de imagen
valorR=get(hObject,'Value');
set(handles.valorRojo,'String',valorR);
Trat_ROJO;
axes(handles.imagenRoja);
imshow(bin);
axis off
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function sliderRojo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderRojo (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function sliderAzul_Callback(hObject, eventdata, handles)
% hObject    handle to sliderAzul (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global varcal original originalcal
if varcal
original=originalcal;

end
valorA=get(hObject,'Value');
set(handles.valorAzul,'String',valorA);
Trat_AZUL;
axes(handles.imagenAzul);
imshow(bin);
axis off
% --- Executes during object creation, after setting all properties.

function sliderAzul_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderAzul (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on slider movement.
function sliderVerde_Callback(hObject, eventdata, handles)
% hObject    handle to sliderVerde (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

valV=get(hObject,'Value');
set(handles.valorVerde,'String',valV)
Trat_VERDE;
axes(handles.imagenVerde);
```

```
imshow(bin);
axis off
% --- Executes during object creation, after setting all properties.
function sliderVerde_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderVerde (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in verificar.
function verificar_Callback(hObject, eventdata, handles)
% hObject    handle to verificar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load('sliders.mat');
load ('datos.mat');
load('datos_mm.mat');
cla(handles.ImagenReal);
cla(handles.estado);
cla(handles.imagenRoja);
cla(handles.imagenAzul);
cla(handles.imagenVerde);
set(handles.coger,'Enable','on');
set(handles.sliderRojo,'Value',sliderRojo);
set(handles.sliderAzul,'Value',sliderAzul);
set(handles.sliderVerde,'Value',sliderVerde);
set(handles.textoRojo,'String','');
set(handles.textoVerde,'String','');
set(handles.textoAzul,'String','');

 global  original varcal
 global XVerde YVerde XAzul YAzul XRojo YRojo
 global auxA auxR auxG real simulado Camara
 global xR yR xB yB xG yG
 %Inicializamos algunas variables
XAzul=[];YAzul=[];XVerde=[];YVerde=[];XRojo=[];YRojo=[];
XAzulREAL=[];YAzulREAL=[];XVerdeREAL=[];YVerdeREAL=[];XRojoREAL=[];YRo
joREAL=[];
distanciaR=[];distanciaA=[];distanciaG=[];
auxR=[];auxG=0;auxA=0;varcal=0;
%%% Referencias de espacio de trabajo
mmX_ref1;
mmY_ref1;

mmX_ref2;
mmY_ref2;
%%% Referencias en pixeles de espacio de trabajo
pixX_ref1=R1X;
pixY_ref1=R1Y;

pixX_ref2=R2X;
pixY_ref2=R2Y;
%Equivalencia en simulación para las alturas de los objetos
d=[0 2 3 4 5 7 8 9 11 12 13];
```

```
a=[0 10 20 30 40 50 60 70 80 90 100 110];
Acamara=984.04;
Verificacion;
if Camara
set(handles.estado,'String','Introduzca altura de los objetos');
set(handles.Altura,'Enable','on');
set(handles.guardar_altura,'Enable','on');
uiwait(gcf)
end
set(handles.guardar_altura,'Enable','off');
%set(handles.estado,'String','Realizando Verificación');
Trat_VERDE;       %Tratamiento de las imagenes (procesamiento de
Verde)
if numg>0
        Centrid_VERDE;    % Cálculo de centroides
        ConvPixMm_VERDE; %Ploteo de Centroides
       % clearvars n xG yG plot %Se limpian las variables
       clearvars n plot
        numg=numg-1;
        %image(original,'Parent',handles.imagenverde);
        % axes(handles.imagenverde);
    %%%Desplegar mensaje de estado de piezas
        set(handles.textoVerde,'String','Piezas verdes encontradas');
        axis off;
else
    set(handles.textoVerde,'String','Piezas verdes NO encontradas');
    axis off;
end

Trat_ROJO;       %Tratamiento de las imagenes (procesamiento de Rojo)
if numr>0
        Centrid_ROJO;    % Cálculo de centroides
        ConvPixMm_ROJO;  %Ploteo de centroides
       % clearvars n xR yR plot %Se limpian las variables
       clearvars n plot
        numr=numr-1;
        %image(original,'Parent',handles.imagenverde);
        % axes(handles.imagenverde);
        %%%Desplegar mensaje de estado de piezas
        set(handles.textoRojo,'String','Piezas rojas encontradas');
        axis off;
else
    set(handles.textoRojo,'String','Piezas rojas NO encontradas');
    axis off;
end

Trat_AZUL; %Tratamiento de piezas azules
if numb>0
        Centrid_AZUL;    % Cálculo de centroides
        ConvPixMm_AZUL;
       % clearvars n xB yB plot %Se limpian las variables
       clearvars n plot
        numb=numb-1;
        %image(original,'Parent',handles.imagenverde);
        % axes(handles.imagenverde);
        set(handles.textoAzul,'String','Piezas azules encontradas');
        axis off;
else
    set(handles.textoAzul,'String','Piezas azules NO encontradas');
    axis off;
end
```

```
save numobjetos.mat auxR auxG auxA
save alturas.mat distanciaR distanciaA distanciaG
save 'destinoREAL.mat' XAzulREAL YAzulREAL XVerdeREAL YVerdeREAL
XRojoREAL YRojoREAL
save 'destino.mat' XAzul YAzul XVerde YVerde XRojo YRojo
set(handles.estado,'string','Verificación Terminada');

% --- Executes on button press in CONECTAR.
function CONECTAR_Callback(hObject, eventdata, handles)
% hObject    handle to CONECTAR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.verificar,'Enable','on');
set(handles.guardarconf,'Enable','on');
set(handles.guardarpos,'Enable','on');
set(handles.calibrar,'Enable','on');
set(handles.Visualizar,'Enable','on');
set(handles.CONECTAR,'Enable','off');
set(handles.estado,'string','Espere un momento');
global real simulado original Camara Kinect
real=get(handles.buttonreal,'value');
simulado=get(handles.buttonsimulado,'value');
Kinect=get(handles.Kinect,'Value');
Camara=get(handles.CamaraWeb,'Value');
Conectar; %Conectar con robot
set(handles.estado,'string','Conexión Completada');
if simulado
 if Camara
original=imread('imagen1.JPG');
set(handles.estado,'string','Conexión Completada');
 elseif Kinect
ObtenerImagenesVrep2;
original=img_left;
 end
end

% --- Executes on button press in Visualizar.
function Visualizar_Callback(hObject, eventdata, handles)
% hObject    handle to Visualizar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global b c original Kinect Camara real simulado
%load ('ImagenesVrep.mat');
Kinect=get(handles.Kinect,'Value');
Camara=get(handles.CamaraWeb,'Value');
real=get(handles.buttonreal,'value');
simulado=get(handles.buttonsimulado,'value');
if Kinect
if real
if b
colorVid = videoinput('kinect',1,'RGB_640x480');
start(colorVid);
axes(handles.ImagenRGB);
himage1=image(zeros(640, 480, 3),'parent', handles.ImagenRGB);
preview(colorVid,himage1);

depthVid = videoinput('kinect',2,'Depth_640x480');
start(depthVid);
axes(handles.ImagenDepth);
himage2=image(zeros(640, 480, 3),'parent', handles.ImagenDepth);
preview(depthVid,himage2);
```

```
axis off
b=0;
else
cla(handles.ImagenRGB);
cla(handles.ImagenDepth);
 b=1;
end
elseif simulado
   if b
   ObtenerImagenesVrep2;
   flipped_depth=imrotate(img_depth,180);
    axes(handles.ImagenRGB);
    imshow(img_left);
    axes(handles.ImagenDepth);
    imshow(flipped_depth);
    axis off
    b=0;
    else
cla(handles.ImagenRGB);
cla(handles.ImagenDepth);
 b=1;
end
end
end
if Camara
if real
if c
vid = videoinput('winvideo', 2);
start(vid);
axes(handles.ImagenRGB);
%%Importante para visualizar video en un axes especifico
himage=image(zeros(640, 480, 3),'parent', handles.ImagenRGB);
preview(vid,himage);
c=0;
else
cla(handles.ImagenRGB);
c=1;
end
elseif simulado
 if c
    axes(handles.ImagenRGB);
    imshow(original);
    c=0;
    else
cla(handles.ImagenRGB);
cla(handles.ImagenDepth);
 c=1;
end
end
end

% --- Executes during object creation, after setting all properties.
function Altura_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Altura (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in guardarconf.
function guardarconf_Callback(hObject, eventdata, handles)
% hObject    handle to guardarconf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sliderRojo=get(handles.sliderRojo,'Value');
sliderVerde=get(handles.sliderVerde,'Value');
sliderAzul=get(handles.sliderAzul,'Value');
save 'sliders.mat' sliderRojo sliderVerde sliderAzul
set(handles.estado,'string','Datos guardados Correctamente');

% --- Executes on button press in guardarpos.
function guardarpos_Callback(hObject, eventdata, handles)
% hObject    handle to guardarpos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load ('datosaux.mat');
save 'datos.mat' R1X R2X R1Y R2Y
set(handles.estado,'string','Datos guardados Correctamente');

% --- Executes on button press in guardar_altura.
function guardar_altura_Callback(hObject, eventdata, handles)
% hObject    handle to guardar_altura (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
uiresume(gcf)
set(handles.estado,'string','Datos guardados Correctamente');
```

9.- Presupuesto

Detallaremos en este capítulo los costes teóricos asociados al proyecto, incluyendo tanto los costes materiales como las tasas profesionales

COSTES MATERIALES

Objeto	Cantidad	Precio UD (€)	Precio TOTAL (€)
PC Lenovo z500	1	900	900
Cámara Kinect	1	159.79	159.79
Cámara Logitech C310	1	40	40
IRB-120 ABB	1	10954	10954
Mano Inmoov	1	1000	1000
Microsoft Office Student 2016	1	0	0
VREP PRO EDU	1	0	0
Matlab R2015a	1	0	0
Robotstudio ABB	1	0	0
TOTAL			13053.79

Tabla 7.1.- *Costes materiales*

TASAS PROFESIONALES

Objeto	Tiempo (meses)	Coste UD (€/mes)	Coste TOTAL (€)
Ingeniería	4	1350	5400
Escritura	1	800	800
TOTAL			6200

Tabla 7.2.- *Tasas profesionales*

COSTES TOTALES

Una vez analizados los diferentes costes por separado, simplemente para calcular el coste general del proyecto sumaremos los parciales totales y aplicaremos el IVA, con carácter general del 21%:

Objeto	Coste (€)	IVA Asociado(€)	Coste TOTAL (€)
Costes Materiales	13053.79	2741.30	15795.09
Tasas Profesionales	6200	1302	7502
TOTAL			23297.09

Tabla 7.3.- *Costes totales*

10.- Bibliografía

[1] Estudio estadísticas de robótica 2009 por AER-ATP.

[2] Actualidad y perspectivas de la robótica, UNMSM – Facultad Ingeniería Industrial

[3] MathWorks.com

[4] Manual operador Robotstudio

[5] V-REP User Manual

[6] http://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm
https://definicion.de/rgb/

[7] "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station", de Marek Jerzy Frydrysiak.

[8] *TFG: "Manipulación de objetos en una cinta transportadora mediante un sistema de visión artificial y brazo robot IRB120*" de Javier Ribera Díaz.

[9] TFG: "Implementación de un sistema de visión para control del brazo robot IRB120" de Álvaro Fernández Expósito.

[10] TFG: "Diseño, fabricación y control de la mano robótica Inmoov-SR para el brazo IRB120" de Sergio Rodríguez.

[11] TFG: "Modelado y control de la mano Inmoov-SR acoplada al brazo robot IRB120 en robotstudio" de Álvaro Bailón.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR