UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Diseño de una estación de trabajo para el Robot IRB120. Control de cinta transportadora mediante IRC5"

2015 Andrés Senén Estremera

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Diseño de una estación de trabajo para el Robot IRB120. Control de cinta transportadora mediante IRC5"

Autor:	Andrés Senén Estremera
Universidad:	Universidad de Alcalá de Henares
País:	España
Profesor Tutor:	Rafael Barea Navarro

Presidente: Marta Marrón Romera

Vocal 1: Ernesto Martín Gorostiza

Vocal 2: Rafael Barea Navarro

CALIFICACIÓN:.....

FECHA:.....

Agradecimientos

El talento, en buena medida, es una cuestión de insistencia. (Francisco Umbral).

Quisiera agradecer el apoyo de mi familia, por estar siempre a mi lado, a mis amigos, a los que me han ayudado en estos 5 años y a los que llevan conmigo mucho más tiempo porque gracias a ellos es todo más divertido, y a los profesores, por su dedicación y ayuda.

En especial quería dedicarle este proyecto a mi madre. Gracias por tu enorme paciencia, mamá.

Contenido general

Lista de Figuras

Figura 10.4. Diagrama de flujo – Reconocimiento de referencias

Figura 10.5. Diagrama de flujo – Calibración por defecto.

Lista de tablas

Resumen

El objetivo de este proyecto es desarrollar un sistema de comunicación entre una maqueta que hará las veces de cinta transportadora, un sensor de proximidad, una aplicación basada en la visión artificial y el brazo robot IRB120 que permita a este interactuar con los círculos que circulen por la cinta y manipularlos de distinto modo dependiendo del color que sean.

Palabras clave

- Visión Artificial
- Robótica
- Socket de comunicación
- Maqueta
- Detección de objetos

Abstract

The objective of this project is to develop a communication system between a model that will serve as conveyor belt, a proximity sensor, an application based on computer vision and robot arm IRB-120 to allow this to interact with circles travelling on the model and manipulate differently depending on the color of the object.

Keywords

- Computer Vision
- Robotics
- Socket
- Model
- Object Detection

1. Introducción

1.1. Tecnología y sociedad.

La tecnología es un término que viene del griego, y podría traducirse como "el estudio de la destreza". La tecnología es el conjunto de conocimientos técnicos, científicamente ordenados, que permiten diseñar y crear bienes y servicios que facilitan la adaptación al medio ambiente y satisfacer tanto las necesidades esenciales como los deseos de la humanidad.

Desde la implantación a principios del siglo XX del sistema Ford, o el uso de cadenas de montaje, la tecnología y la industria han ido cada vez más de la mano desde su auge en la revolución industrial acontecida en la segunda mitad del siglo XVIII.
El progresivo aumento de las posibilidades tecnológicas que comenzó con la creación del primer robot industrial en 1937 ha provocado una especialización cada vez mayor en torno a la fabricación de objetos en las cadenas de montaje de prácticamente todas las fábricas del mundo. Así como antes este primer robot solo era capaz de apilar unos bloques de madera según unos patrones pre-programados ahora tenemos nos encontramos con robots que interactúan con el entorno y son capaces de reaccionar a tiempo real según lo que ocurre a su alrededor gracias en gran medida a la visión artificial.

La visión artificial a pesar de su relativa temprana fase de investigación y desarrollo ha supuesto un gran avance para implementar nuevos sistemas de sensores o de adquisición de datos así como una gran propulsión en el campo de la inteligencia artificial.
Gran parte del éxito de esta disciplina reside en las bibliotecas de librerías que cada día se van actualizando, incorporando nuevas funciones y desarrollándose nuevas aplicaciones de éstas.
Si añadimos a un sistema con una cámara capaz de recibir estímulos visuales de cualquier tipo un brazo robótico básico como es el IRB120 podemos obtenemos un nuevo sistema capaz de reaccionar físicamente a esos estímulos. Las aplicaciones son infinitas.

Este proyecto se va a basar en mostrar la aplicación directa de un sistema de este tipo acoplada a una maqueta que podría hacer las veces de cadena de montaje en miniatura. El reconocimiento de objetos a través de una cámara y un programa basado en las librerías de código libre OpenCV (a través de EMGU) en lenguaje C# para poder detetctar objetos, formas, imágenes, etc., así como las órdenes que se asignarán al brazo robótico para conseguir interactuar con dichos objetos que se encuentran en movimiento a través de la utilización de sensores de detección

1.2. Estado del arte

Es común encontrar numerosos sistemas de visión artificial en la industria que se utilicen como complemento en cadenas de montaje. Ofrecen una solución rápida y sin contacto para numerosos problemas.

1.2.1. Ensamblado de Teselas

Un ejemplo de aplicación muy avanzada de visión artificial y robótica puede ser el Ensamblado de Teselas. El servicio eVis-Enginyeria Visual, integrado en el INIT, ha realizado una colaboración con las empresas Disinel S.L. y Tecnopamic S.A. sobre ensamblado automático de teselas decorativas en piezas cerámicas por medio de visión artificial y un brazo robotizado.

Figura 1.1. Sistema de teselado

El objetivo de este sistema se basa en la construcción de una máquina ensambladora que consta de dos cintas transportadoras, dos sistemas de visión por ordenador y un robot manipulador.

Funcionamiento básico:

- Una cinta transporta teselas decorativas de varios colores y formas, la otra cinta transporta piezas cerámicas con unos huecos donde hay que pegar las teselas.

- El primer sistema de visión identifica y sigue las teselas en base a su forma y color.

- El segundo sistema de visión identifica y sigue las piezas en base a la disposición de sus huecos, su forma y su relieve.

- Las piezas cerámicas aparecen en grupos o modelos, en cualquier orientación. Un

modelo puede constar de 4 a 8 piezas diferentes.

- El ordenador de visión envía las coordenadas de una tesela y de un hueco a un brazo robotizado, el cual realiza el ensamblado encolando la tesela y pegándola sobre la pieza.

- Se ha desarrollado un método de calibración para relacionar las coordenadas de cámara, las de la cinta y las del robot manipulador. Esta calibración se basa en una plantilla situada en las cintas y en interpolación por splines bicúbicos.

Figura 1.2. Teselado

Características:

- Precisión de 1 mm.
- Prácticamente 100% de aciertos en la detección y clasificación de teselas y piezas cerámicas.
- Entorno de usuario mostrando el proceso de imagen de ambos sistemas de visión.
- Pantalla táctil para selección de modelos, puesta en marcha y apagado.

Video demostrativo: http://vimeo.com/20785957

El resultado es un sistema de gran precisión, velocidad y acierto.

2. Herramientas utilizadas

En este punto se realizará un breve resumen acerca de las herramientas, tanto software como hardware, utilizadas para la realización de la aplicación. Estas herramientas han sido seleccionadas de manera objetiva con la intención de garantizar una mayor calidad en el proyecto.

2.1. Microsoft Visual Studio

Visual Studio.NET es un conjunto de herramientas de desarrollo para la construcción de aplicaciones.

Soporta múltiples lenguajes de programación como C++, Java, C#, Visual Basic.NET, F#, Python, Ruby, PHP; así como entornos de desarrollo web como ASP.NET o Django entre otros.

Este software permite a los desarrolladores crear aplicaciones, aplicaciones web, y servicios web soportables por la plataforma .NET. De esta manera, se pueden desarrollar aplicaciones que se intercomuniquen entre estaciones de trabajo, dispositivos móviles y páginas web.

Muchas son las versiones de Visual Studio desde su lanzamiento en 1997, pero el cambio más significativo se da en el año 2002. Este año supuso la introducción de la plataforma .NET de Microsoft. Se trata de una plataforma de ejecución intermedia multilenguaje, lo que significa que los programas que se desarrollan en esta plataforma se compilan en un lenguaje intermedio y no en lenguaje máquina.

En las aplicaciones realizadas en este formato, el código no se convierte a lenguaje máquina hasta que se ejecuta, con lo que el código puede ser totalmente independiente de plataforma.

Actualmente la última versión estable del software es Visual Studio 2013, y es el software utilizado en el desarrollo de más de la mitad de la aplicación que tiene como objeto este proyecto.

2.1.1. Plataforma .NET

Es conveniente entrar en más detalle de todo el potencial de esta plataforma.

.NET conduce a la tercera generación de Internet. La primera generación se caracterizó por trabajar con información estática que podía ser consultada a través de exploradores. La segunda generación consistía en que las aplicaciones pudieran interactuar con las personas. Y por último, la tercera generación, se basa en aplicaciones capaces de interactuar con otras aplicaciones

Precisamente, el principio de .NET es que los sitios Web aislados y los diferentes dispositivos trabajen conectados a través de internet. Esto se consigue gracias a la aceptación de los estándares basados en XML (Extensible Markup Language).

Microsoft .NET extiende las ideas de Internet y sistema operativo haciendo de la propia Internet la base de un nuevo sistema operativo. Esto permite a los desarrolladores crear programas que aprovechen al máximo los dispositivos así como la conectividad de la red y sus aplicaciones.

Para ello proporciona una plataforma que incluye los siguientes componentes:

- Un conjunto de servicios que actúan como bloques de construcción para el sistema operativo de Internet

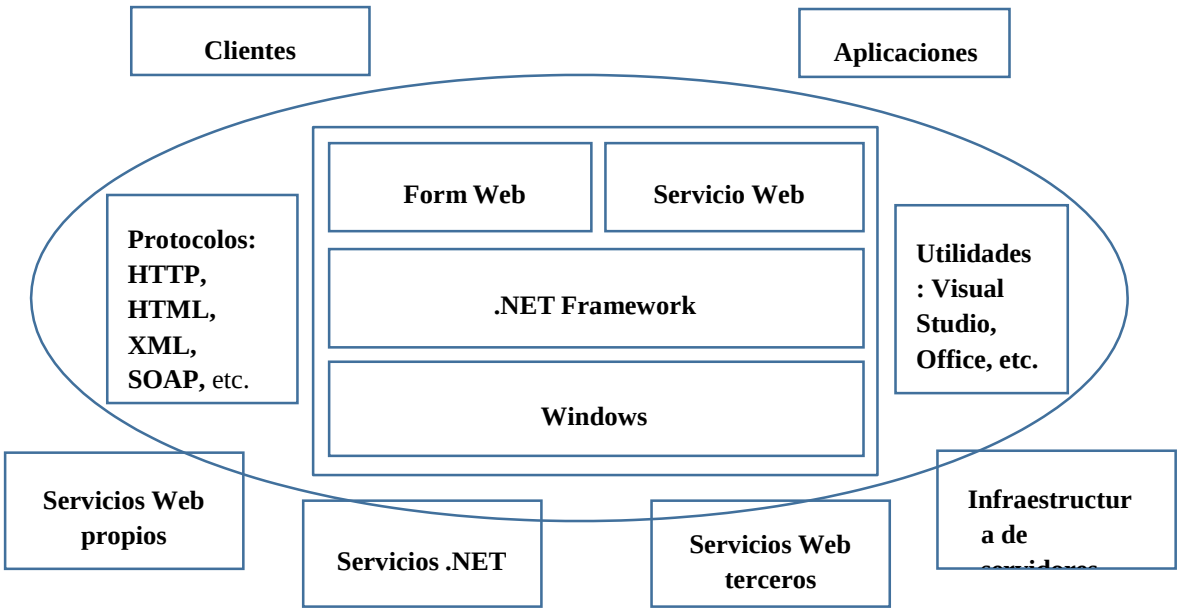


Figura 2.1. Bloques de la plataforma .NET

- Software de dispositivos .NET
- Herramientas de programación para crear servicios Web XML con soporte multilingüe: Visual Studio.NET y .NET Framework
- Infraestructura de servidores, incluyendo Windows y .NET Enterprise Servers.

Pero también se requiere de una infraestructura, no solo para facilitar el desarrollo de aplicaciones, sino también para hacer que el proceso de encontrar un servicio Web e integrarlo en una aplicación resulte transparente para usuarios y desarrolladores. Esta estructura nos la proporciona .NET Framework:

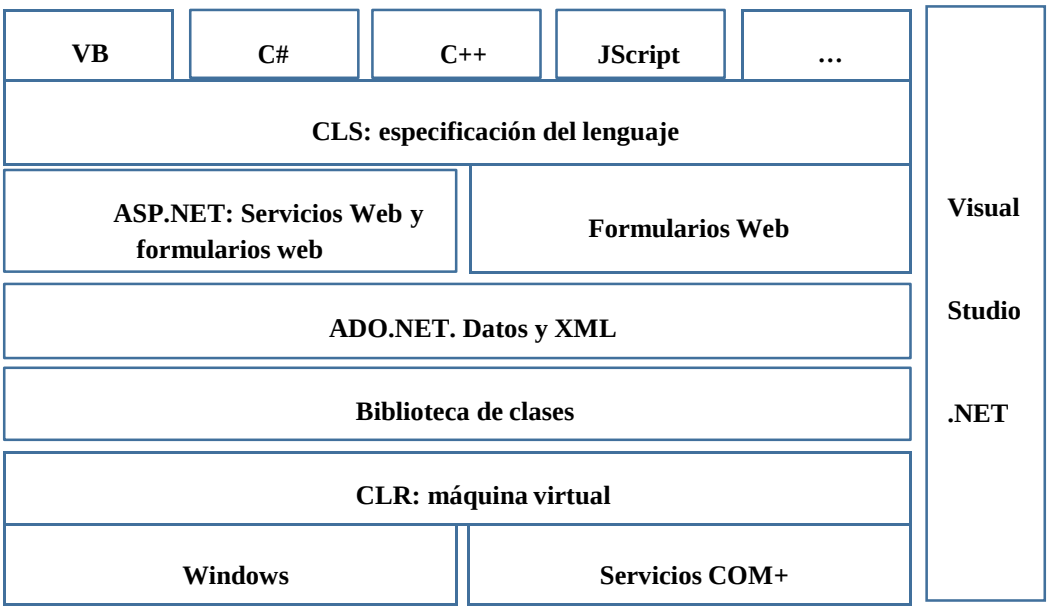


Figura 2.2. Estructura de la plataforma .NET

.NET Framework proporciona un entorno unificado para todos los lenguajes de programación.

Microsoft ha incluido en este marco los lenguajes Visual Basic, C#, C++ y JavaScript. Lo más llamativo quizá sea la posibilidad de escribir parte de una aplicación en un C# y la otra parte en Visual Basic. Sin embargo para que esto pueda ocurrir se propone la especificación común para los lenguajes (CLS). Para eso, define un conjunto de tipos de datos comunes que indican que tipos de datos se pueden manejar, la forma de declararlos y de utilizarlos.

Así aunque cada lenguaje .NET use un sintaxis diferente para cada tipo de datos, por ejemplo en VB se utiliza Integer para un número entero de 32 bits y en C# se utiliza Int, estos nombres no son más que sinónimos para el tipo común System.Int32. De esta manera se permite la interoperabilidad entre lenguajes.

.NET Framework proporciona un entorno de ejecución llamado CLR (Common Language Rutine). Se trata de una máquina virtual que administra la ejecución del código y proporciona servicios que hacen más cómodo el desarrollo.

Cuando se compila el código escrito, el compilador lo traduce a un código intermedio denominado MSIL (Microsoft Intermediate Language). Esto supone que cualquier lenguaje .NET puede implementarse en cualquier plataforma (Intel, Motorola…) que tenga instalada una máquina virtual .NET.

Esquema orientativo:

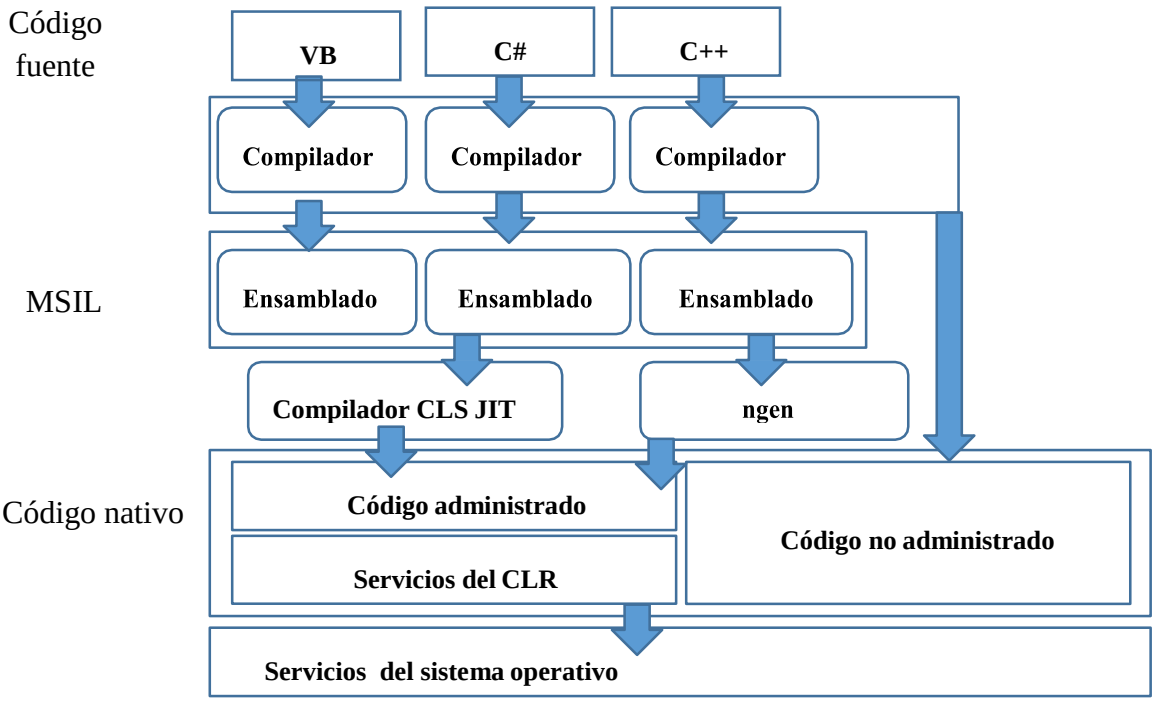


Figura 2.3. Esquema orientativo de uso de la Plataforma .NET

2.1.2. Lenguaje C#

El lenguaje C# es un lenguaje de programación orientado a objetos y ha sido diseñado por Microsoft en la iniciativa .NET.

Es sencillo y sigue el mismo patrón de los lenguajes de programación modernos. Incluye un amplio soporte de estructuras, manipulación de objetos… Las clases son la base de los lenguajes de programación orientado a objetos, y C# contiene las herramientas para definir nuevas clases, sus propiedades y sus métodos, así como la capacidad de implementar encapsulación, herencia y polimorfismo que son los tres pilares de la programación orientada a objetos.

C# posee un nuevo estilo de documentación XML que se añade a lo largo de la aplicación, también provee soporte para estructuras.

Pero, ¿por qué C#? La razón fundamental es que se diseñó para la plataforma .NET y es capaz de aprovechar todo su potencial. También añadir que es un lenguaje "limpio" en el sentido de que, al no tener que proporcionar compatibilidad hacia etapas anteriores, se ha dedicado más tiempo al diseño y se ha hecho especial hincapié en su simplicidad.

2.2. Compañía ABB

ABB es una empresa líder mundial en ingeniería eléctrica y automatización.

La compañía es el producto de la unión en 1988 de Asea y BBC. Actualmente la sede central se encuentra en Zúrich, Suiza, cuenta con más de 150.000 empleados y está presente en más de 100 países.

Tiene una gran oferta en la que se puede diferenciar cinco divisiones: Power Products, Power Systems, Discrete Automation and Motion y Low Voltage Products.

En estas divisiones se ofrecen desde interruptores de iluminación, hasta robots industriales, grandes transformadores eléctricos o sistemas de control capaces de gestionar grandes redes eléctricas o industrias.

Como se ha mencionado, ABB suministra robots industriales con los que se mejora la productividad de las empresas, cuenta con una oferta de más de 25 tipos de robots distintos para las diferentes utilidades que se le puede dar en cada tipo de industria.

Este gran abanico de posibilidades que ofrece ABB le convierte en la empresa líder de suministro de robots en el mundo.

Además este suministro de robots se incrementa cada año debido a la gran necesidad de las empresas de evolucionar hacia una eficiencia energética y tecnológica, que les mantenga dentro del mercado.

Figura 2.4. Cadena de producción robots ABB

En la imagen se puede ver como toda una cadena de montaje de coches está llevada a cabo por robots de ABB. Se puede vislumbrar la importancia de los robots en este tipo de sistemas.

2.2.1 Software Robot Studio

RobotStudio es el software de simulación y programación offline que permite crear y simular estaciones robóticas industriales en 3D en un ordenador. Sus características aumentan la variedad de tareas posibles para llevar a cabo mediante un sistema de robótico, como la capacitación, la programación o la optimización.

Algunos de ellos ofrecen la reducción de riesgos, una puesta en marcha más rápida o el cambio de formato más corto. Además, cuando se utiliza RobotStudio con los controladores reales, se conoce como "modo en línea".

El software está basado en el controlador virtual de ABB, que es, una copia simulada del software real que utilizan los robots de ABB. Gracias a este software se puede ejecutar en nuestro ordenador un sistema robótico que se haya diseñado antes de ejecutarlo en el robot real, con lo que se evitan costes innecesarios.

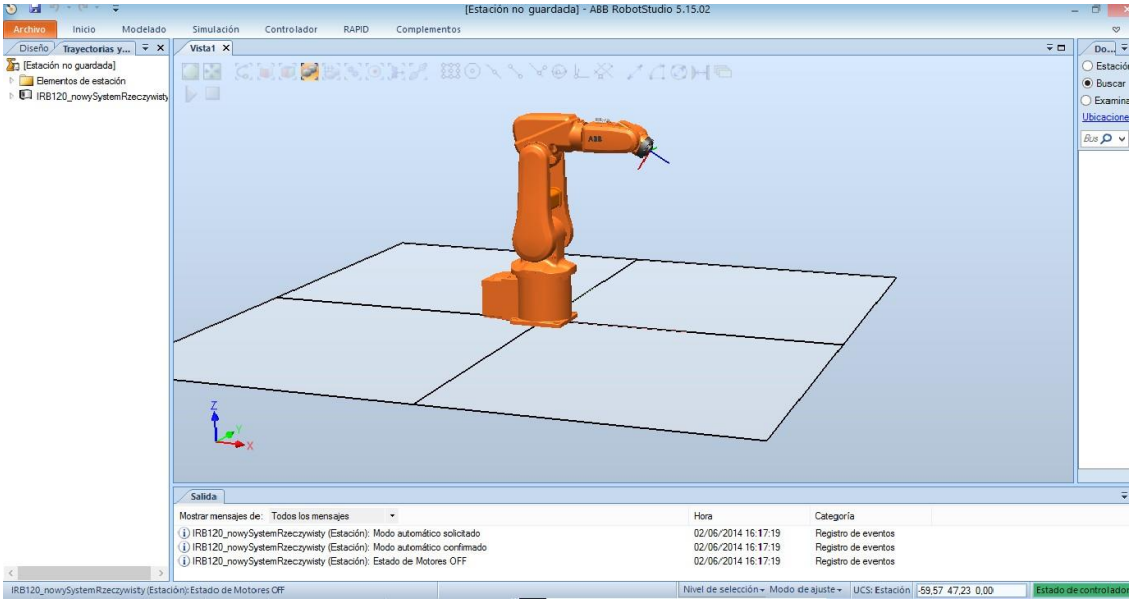


Figura 2.5. Interfaz principal Robtstudio

RobotStudio y su lenguaje de programación RAPID, tienen gran importancia en el desarrollo de este proyecto, pero toda la información necesaria del mismo, está completamente explicada y comentada en el proyecto *"Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station"* de Marek Frydrysiak, por lo que no se entrará en explicaciones detalladas del software.

2.2.2. Brazo robótico IRB-120

Para llevar a cabo la parte final de la aplicación, se requiere un brazo robótico que ejecute la aplicación que se ha desarrollado. Este brazo robótico, de la compañía ABB, es el modelo IRB-120 y a continuación se detallan algunas de sus características principales.

Figura 2.6. Robot ABB IRB-120

Como se ha dicho, se dispone de un brazo robótico de tamaño reducido pero suficiente para el objetivo que se persigue. La estructura del brazo es la siguiente:

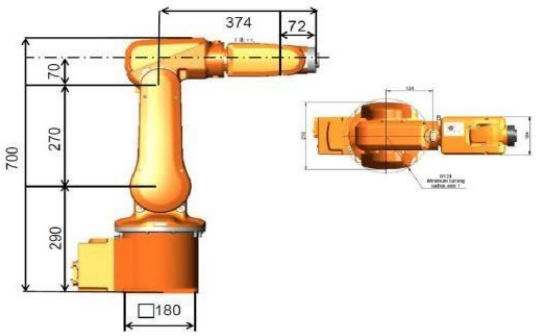


Figura 2.7. Dimensiones robot ABB IRB-120

Además posee 6 ejes, los tres primeros servirán para establecer la posición del efector final (o herramienta) y los tres últimos para determinar la orientación del mismo.

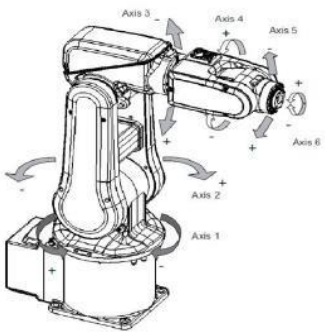


Figura 2.8. Articulaciones robot ABB IRB-120

2.3. Librerías utilizadas

Este capítulo tendrá varios objetivos. El primero de ellos será dar una noción sobre lo que es Open CV y su importancia en esta clase de sistemas. Se explicará, también, lo que es Emgu CV y la necesidad de su uso. Y por último se incluirá una guía de instalación para las librerías de Emgu CV, con las que el usuario podrá instalarlas.

2.3.1. Open CV

Open CV es una biblioteca libre de visión artificial creada en el año 1999 por Intel. En ella se pueden encontrar más de 500 funciones que abarcan un gran abanico de posibilidades, como el reconocimiento de objetos, reconocimiento facial, calibración de cámaras, visión estérea y visión robótica.

Su gran capacidad permite que se emplee en una gran cantidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto, además, es gracias a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación.

Una de las limitaciones, es que esta librería no se puede implementar de forma directa en una aplicación de Windows Forms, escrita en lenguaje C# por lo que se necesita una solución práctica con la que se consiga poder hacer uso de esta librería tan potente

2.3.2. Descripción del problema

Mediante el software Visual Studio 2013 se realiza una aplicación de Windows Forms escrita en C#.

Como la aplicación final tendrá por objeto principal el procesamiento de imágenes, es recomendable la utilización de las funciones contenidas en la biblioteca "Open CV" ya que están diseñadas para tal propósito.

Como ya se ha mencionado, Open CV es una biblioteca de visión artificial donde se puede encontrar una serie de funciones que permiten una gran cantidad de posibilidades, como el reconocimiento de objetos, reconocimiento facial, calibración de cámaras, visión estérea y visión robótica.

El principal inconveniente es que no es posible implementar de forma directa las funciones de Open CV en la aplicación, con lo que hay que ayudarse de algún tipo de "plataforma envoltorio" con la cual, poder invocar a dichas funciones de manera directa.

La implementación de estas funciones de Open CV está explicada en el proyecto de Álvaro Fernández Expósito, *Implementación de un sistema de visión para control del brazo robot IRB120.*

2.4. Maqueta

Para transportar los círculos se necesitará usar una cinta transportadora, para este uso se puede acoplar la maqueta que crea Diego Sánchez Navarro en su proyecto "*Diseño, construcción y programación de una maqueta de un sistema de identificación y clasificación industrial"*. Se deben hacer varios cambios con respecto a la maqueta inicial para que se pueda utilizar de la manera deseada.

2.4.1. Materiales

- Tambor que actúa como eje tractor, fabricado en acero con un diámetro de 35mm y una longitud de 70mm.
- Rodillo que actúa como eje de transmisión del movimiento.
- Rodamientos para unir el tambor a la estructura de 19mm de diámetro y 8mm de ancho.
- Varias piezas de metacrilato unidas entre sí con cola de contacto y forradas con aironflix.
- Un sistema de acoplamiento que permite transmitir la tracción del motor al tambor de la cinta transportadora.
- Una correa de transmisión encargada a medida con el suficiente agarre para que los objetos no resbalen por ella de 6.5cm de ancho y 134cm de largo.
- Motor K14 2841 con una reductora acoplada que funciona a 24 voltios con una velocidad fija a esos 24 voltios de 38rpm. Para sujetarlo a la base de la maqueta se ha construido una estructura de aluminio que se atornilla al motor apoyada en una base de metacrilato para obtener la altura necesaria para que esté alineado con el eje del tambor.

Figura 2.9. Motor empleado para mover la cinta

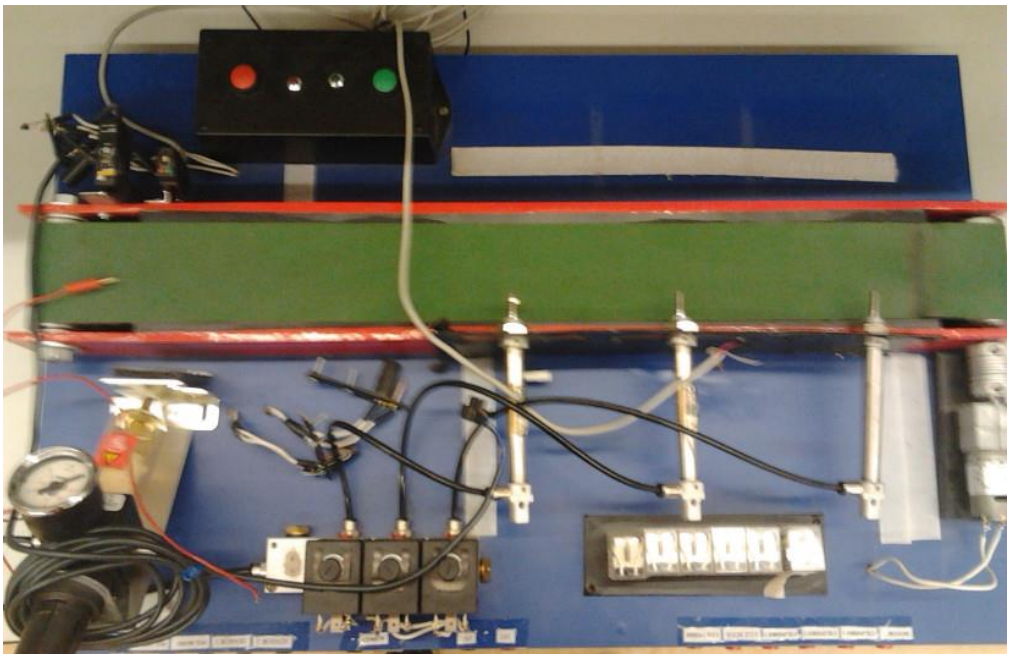


Figura 2.10. Maqueta inicial

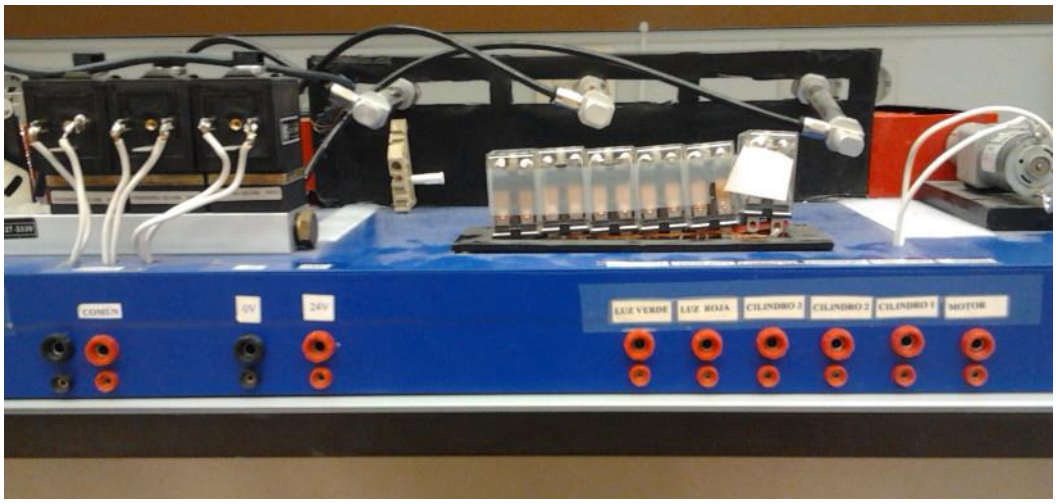


Figura 2.11. Conexiones disponibles

2.4.2. Metodología

- El armario de control IRC5 genera la señal de movimiento de la cinta transportadora a través de la SALIDA 15.

- El sensor de tipo capacitivo E2K-C25MF1 conectado a la ENTRADA 14 genera un 1, una tensión de valor 24V, cuando detecta un objeto en sus proximidades.

- Mediante la comunicación entre el programa RAPID de RobotStudio y el programa

principal en C# se para la cinta al generar el sensor un 1 en la ENTRADA 14.
El movimiento de la cinta transportadora se vuelve a reanudar cuando el sensor devuelve un 0 en la ENTRADA 14, es decir, el brazo robot retira el objeto de la cinta y lo sitúa en su posición de paletizado.

2.4.3 Montaje

Para que se acople al uso que se le quiere dar hay que eliminar y añadir varios elementos en la maqueta.

1. Se eliminan todos los sistemas de válvulas neumáticas así como el pulsador, los relés, los sensores ópticos y las conexiones presentes en la parte inferior de la maqueta para quedarse solamente con el motor acoplado a la reductora y la correa de transmisión.

2. Se conecta el motor a un relé y se lleva el positivo a la conexión con la salida DO10_15 del brazo robot y el negativo a la conexión lateral que pone 0V.

3. Se lleva la alimentación a la maqueta desde el brazo robot, tanto los 24V como el común a las conexiones laterales de 24V y 0V respectivamente.

4. Se conecta el sensor a la conexión 0V y a los 24V como corresponde y la señal de detección se lleva a la entrada DIO10_14 del brazo robot.

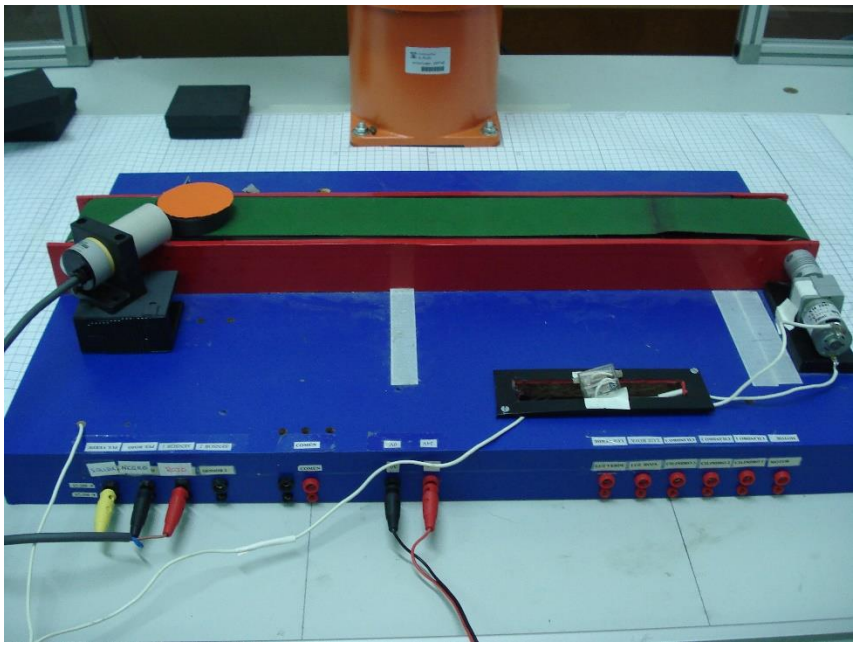


Figura 2.12. Maqueta final

2.4.4. Conexiones de la maqueta

Como se ha mencionado antes, se utiliza la Salida Digital 15, para el control del motor de la cinta, y la Entrada Digital 14, conectada al sensor, para la detección de piezas.

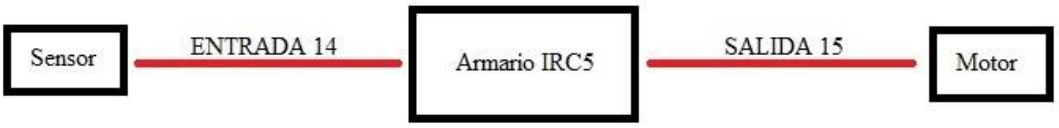


Figura 2.13. Conexión entre IRC5, motor y sensor

Sin embargo, las conexiones habilitadas en un principio solo abarcan hasta la salida digital 9 por lo que se realiza una ampliación de estas, desde la 13 a la 16, tanto en SALIDAS como en las ENTRADAS, y así dejar disponibles más conexiones en caso de necesitarse.

Para realizar esta ampliación se ha de instalar un cableado que contenga ocho nuevos elementos (cuatro entradas y cuatro salidas) para acoplarse al conector de 42 pines XS7 presente en el armario de control IRC5 que a su vez está conectado internamente con la unidad de E/S DSQC 652 en la cual se pueden encontrar las 16 salidas y entradas digitales que tiene el armario de control. Estas nuevas entradas y salidas se conectarán internamente de la siguiente forma.

In/Out	Número de I/O	Pin XS7
In	13…16	35…38
Out	13…16	15…18

Tabla 2.1. Conexiones ampliadas

Los nuevos cables se unirán al conector XS7 mediante un conector DB9 con pines del 1 al 8. A este conector DB9 se le acoplarán cables de diferentes colores que se detallan en las tablas 2.2 y 2.3 así como el pin del conector DB9 al que corresponden.

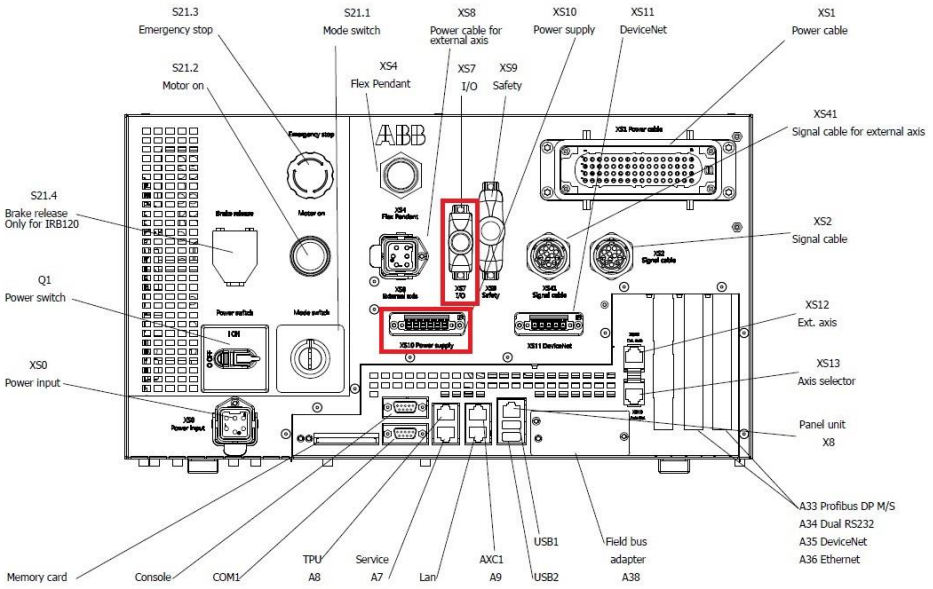


Figura 2.14. Conexiones IRC5

Figura 2.15. Conexión E/S ampliada

Para distinguir con claridad que cable es cada número de entrada o de salida, a la hora de conectar el motor y el sensor, se han puesto unas fundas de diferentes colores:

Nº SALIDA	Pin DB9	Color de la funda	Pin XS7
13	1	Blanco-Naranja	15
14	2	Naranja	16
15	3	Blanco-Verde	17
16	4	Verde	18

Tabla 2.2. Colores de las salidas

Nº ENTRADA	Pin DB9	Color de la funda	Pin XS7
13	5	Blanco-Azul	35
14	6	Azul	36
15	7	Blanco-Marrón	37
16	8	Marrón	38

Tabla 2.3. Colores de las entradas

Por tanto, se suelda la alimentación del motor con el cable blanco-marrón y la salida del sensor con el de color azul. Los otros colores se dejan al aire para que se puedan usar más adelante en otras aplicaciones.

Además, también se ha ampliado la conexión XS10 Power Supply con dos cables en los pines 4 (positivo) y 5 (masa) para proporcionar alimentación de 24 voltios a la maqueta.

Los cables que salen del armario IRC5 se unen mediante una clavija a otros con más recorrido.

Para que se trabaje con más seguridad se ha situado en el extremo de los cables unas bananas que evitan el contacto directo con los propios cables

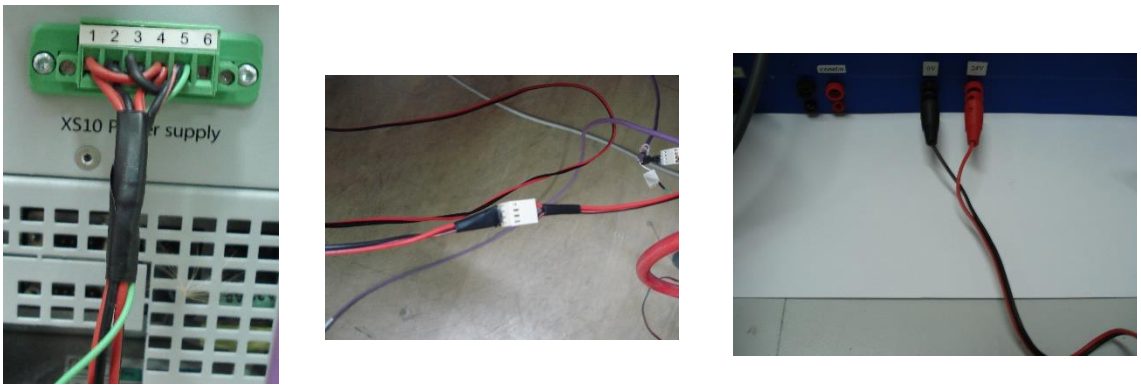


Figura 2.16. Conexiones XS10

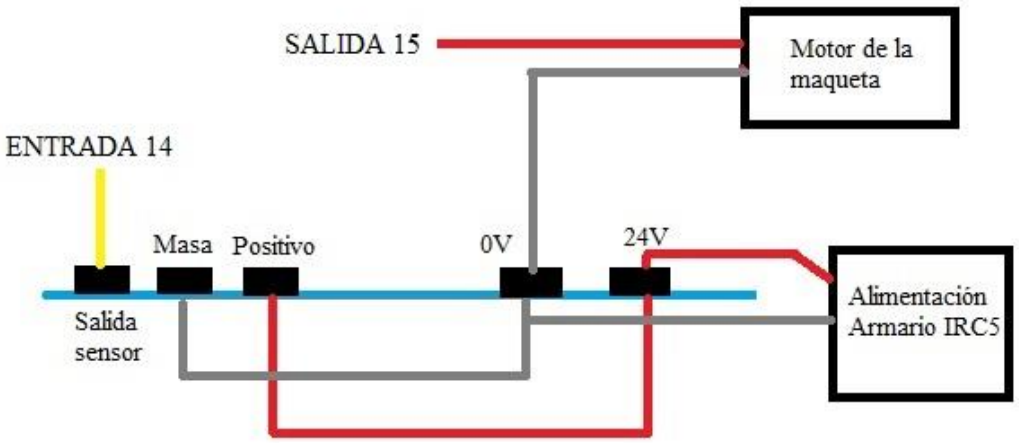


Figura 2.17. Conexiones de la maqueta y el sensor

2.5. Herramienta Ventosa

El efector final del brazo robótico deber ser capaz de coger objetos. Es por ello que se ha diseñado un sistema de succión ejercida a través de un compresor.

2.5.1. Materiales

- Eyector de vacío tipo Venturi Ref.: EZH10BS-06-06
- Ventosa de 32 mm de diámetro Ref.: ZPT32BN-B01
- Electroválvula 24Vcc Ref.: SYJ712M-5LOU-01F
- Conector con cable para electroválvula Ref.:SY100-68-A-30
- Silenciador Ref.: AN10-01

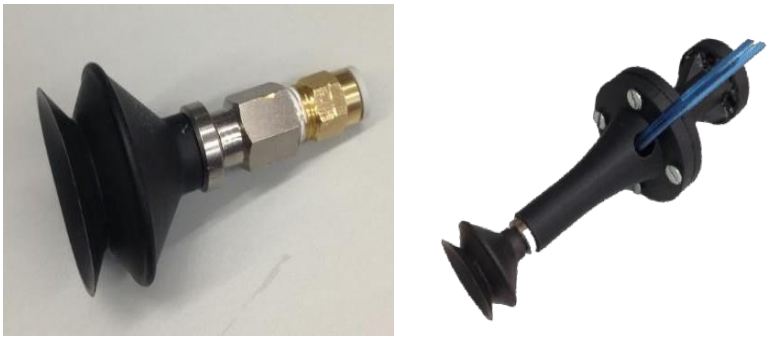


Figura 2.18. Ventosa

2.5.2. Metodología

- El armario de control IRC5 genera la señal de succión no succión a través de la E/S 9.
- Dicha señal activa la electroválvula SYJ712M-5LOU-01F
- La salida de la electroválvula SYJ712M-5LOU-01F se conecta al eyector de vacío tipo Venturi EZH10BS-06-06
- La salida del eyector de vacío se conecta a la ventosa ZPT32BN-B01 a través de los circuitos internos del robot IRB120

2.5.3. Esquema gráfico

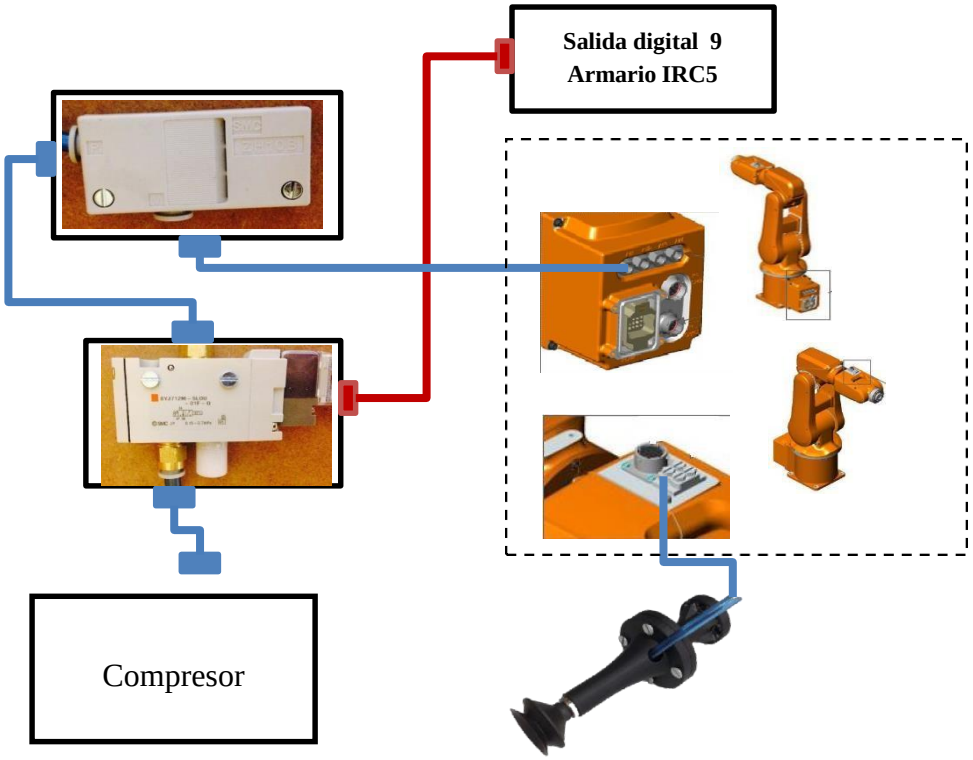


Figura 2.19. Esquema gráfico

2.6. Sensor capacitivo

Para detectar la llegada de un objeto al final de la cinta transportadora se utilizará un sensor de proximidad E2K-C25MF1.

Figura 2.20. Sensor E2K-C25MF1

Para que el sensor esté a la altura del paso de los objetos se necesitará un soporte que lo eleve, este será diseñado en SolidWorks e impreso en la impresora 3D del centro.

El sensor no tendrá posición fija y se podrá situar donde se quiera, por ejemplo, a partir de la mitad de la cinta en adelante para que el objeto tenga un mayor recorrido en la cinta, recomendándose que la posición en la que se sitúe no esté muy cerca del extremo al haber problemas de iluminación en dicho punto.
En las siguientes imágenes se puede ver como el brazo robótico alcanza el círculo en dos posiciones distintas de parada determinadas por el sensor.

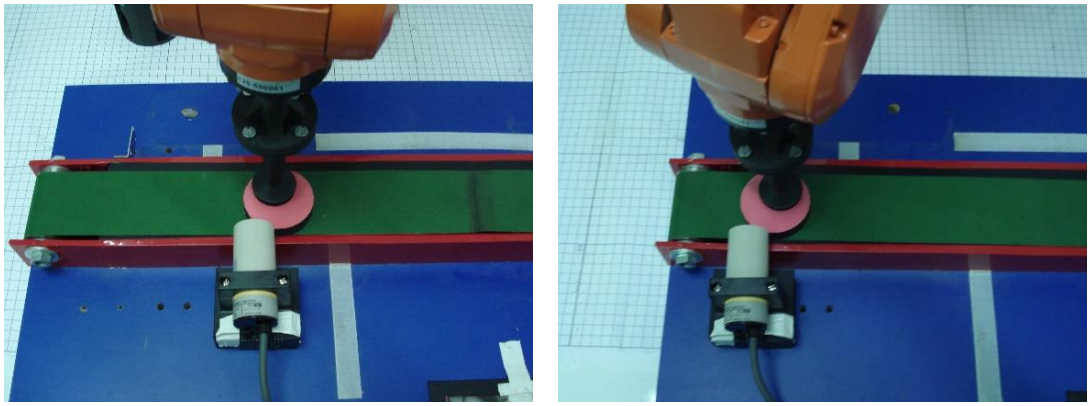


Figura 2.21. Distintas posiciones del sensor

2.6.1. Características del sensor

- Este sensor provoca un 1 (tensión a 24V) cuando detecta un objeto en sus proximidades y 0 cuando no detecta nada.
- Tiene un alcance entre 3mm y 25mm
- Su tensión de alimentación puede estar entre los 10 y los 30 voltios.
- La configuración de salida es PNP
- Su frecuencia máxima de interrupción es de 70Hz.
- La carcasa está fabricada en ABS (Acrilonitrilo butadieno estireno), un plástico muy resistente a los impactos y su peso es de 0.23Kg.

2.6.2. Montaje

Las conexiones del sensor con la cinta y el armario IRC5 están esquematizadas en el apartado 2.4.4.

1. El sensor E2K-C25MF1 consta de tres cables para realizar su conexión. Uno de color marrón (acabado en un protector rojo), uno de color azul (acabado en un protector negro) y otro de color negro (acabado en un protector amarillo).

2. Conectamos el cable marrón a la conexión lateral de 24V, el cable azul a la conexión lateral de 0V y el cable negro a la entrada 14, es decir, al cable azul de la ampliación de E/S que hemos realizado en el armario de control.

2.6.3. Esquemas del soporte

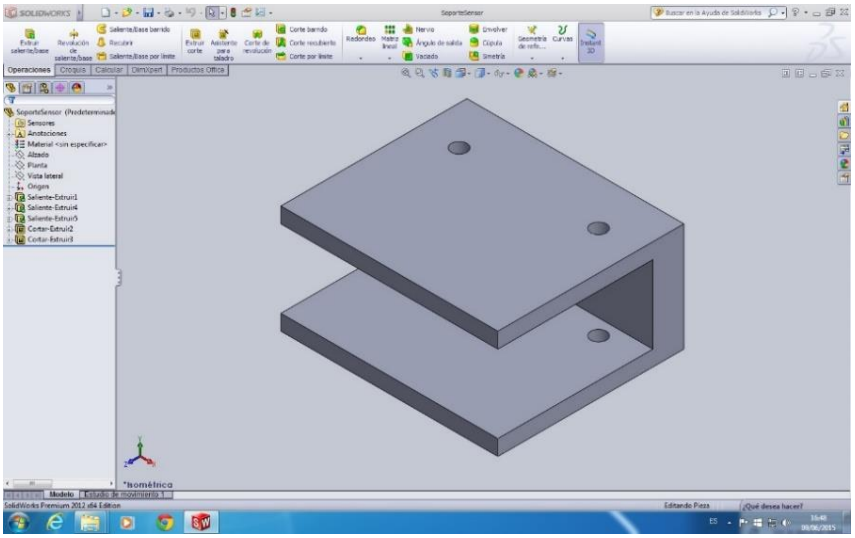


Figura 2.22. Soporte en SolidWorks

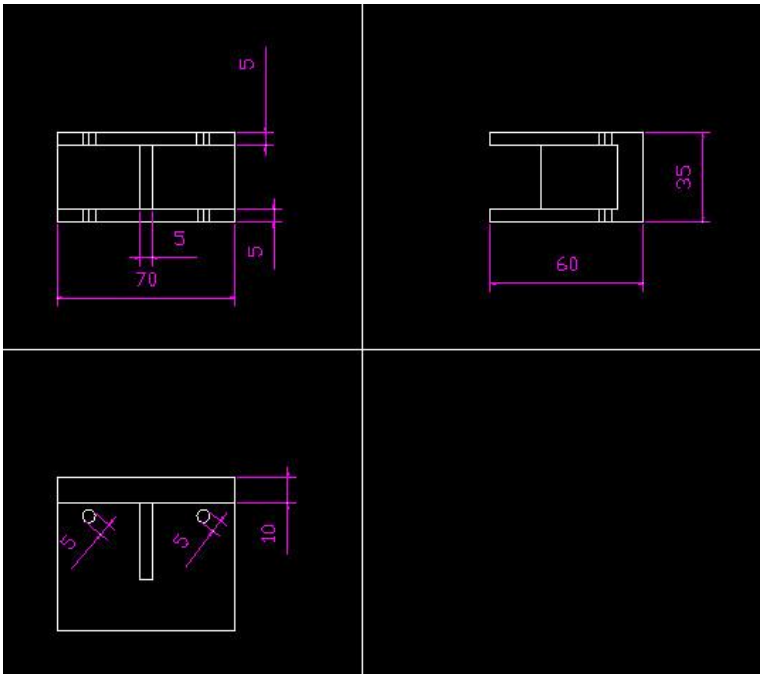


Figura 2.23. Croquis del soporte

3. Arquitectura general del sistema

3.1. Esquema orientativo

- Adquisición de imágenes desde cámara web USB.
- Envío de información mediante socket de comunicación.
- Procesamiento de imágenes. Open CV – EMGU CV
- Envío de información del sensor a través de DIO10_14.
- Comunicación entre software y salida 15.

Figura 3.1. Arquitectura general de la aplicación

3.2. Introducción

En los capítulos venideros se comentarán los distintos elementos que conforman el esquema orientativo presentado en el apartado anterior. Si bien no se entrará en profundidad en ellos, se analizarán las funciones de cada uno.

El bloque que atañe principalmente a esta aplicación es la comunicación entre el software del robot y el sistema de visión con la cinta y el sensor por lo que estará explicado de forma más extensa. También se entrará en profundidad en los cambios realizados en el procesamiento de imágenes respecto al proyecto del que se toma referencia [3].

Se puede ver, en el esquema anterior, que el bloque del sistema de visión está formado por una cámara web (hardware) y un ordenador en el que están instalados los elementos software citados en el capítulo de herramientas utilizadas.

El otro bloque está formado por el software del robot y el sistema de visión en comunicación con la cinta transportadora y el sensor de proximidad.

3.3. Adquisición de imágenes

Para ello se necesita una cámara USB que obtenga imágenes del espacio de trabajo del brazo robótico, pudiendo así detectar los objetos que tiene próximos para su posterior interactuación con ellos.

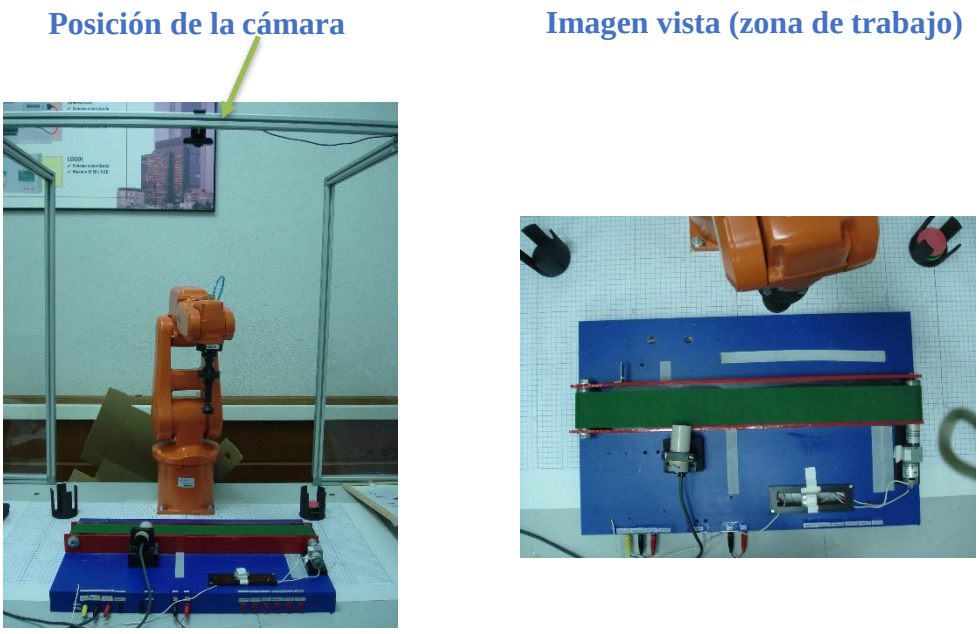


Figura 3.2. Adquisición de imágenes

3.4. Reconocimiento del espacio de trabajo

Puesto que el objetivo final es permitir al robot interactuar con los objetos detectados, es necesario hacer una calibración directa entre la imagen que envía la cámara USB y la posición real en el sistema de coordenadas del robot en la que se encuentra esa área vista por la cámara. Para ello se usan puntos de referencias inmutables que se reconocen antes de buscar otros objetos.

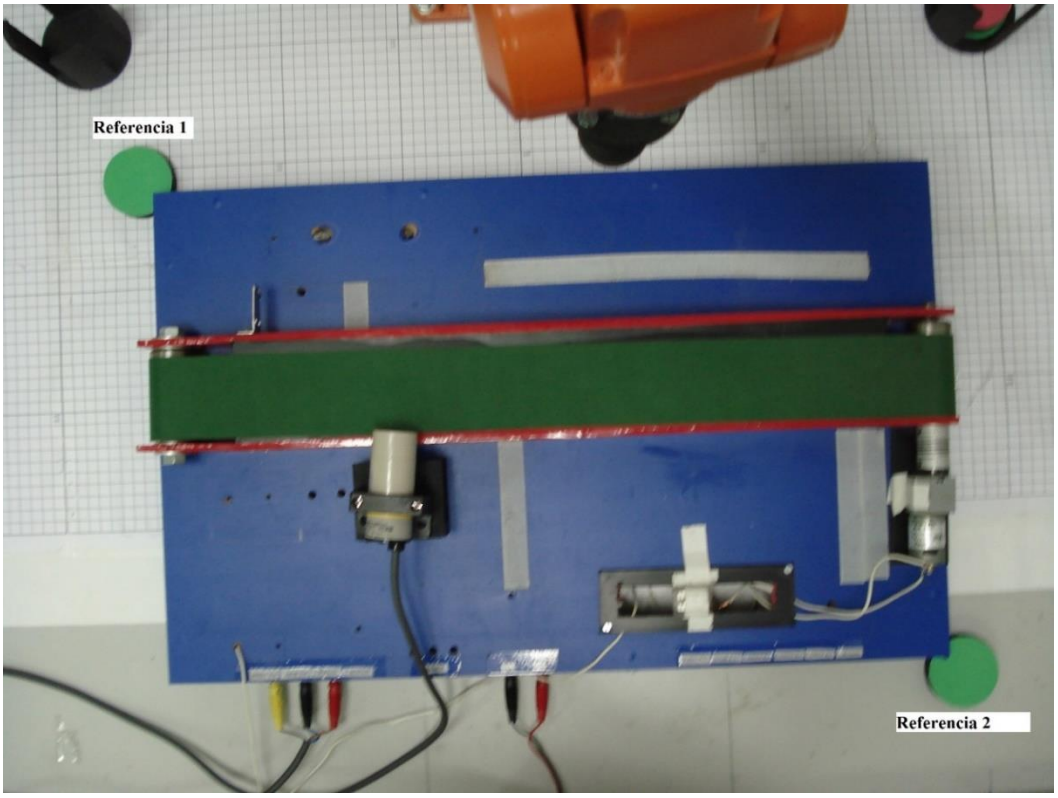


Figura 3.3. Reconocimiento del espacio de trabajo

También se pueden dar esos puntos de referencias inmutables a través de unos datos incluidos en un fichero a los cuales accederá el sistema de visión.
Con esto se evita el tener que realizar la calibración "manual" cada vez que se arranque el programa para su uso.

3.5. Detección de objetos

Una vez reconocida el área de trabajo, se puede pasar a reconocer los objetos que están en ella. Este trabajo de ha orientado a la búsqueda de círculos de distintos colores.

Los objetos se mueven por la cinta de tal forma se pueden detectar los objetos en movimiento, sin embargo, obligaremos a que la cinta está parada para lo que nos ayudamos del sensor colocado al final de esta. Cuando un objeto llega al final de la cinta esta se para y se pone en funcionamiento el brazo robot.
Siempre debemos tener en cuenta que la altura que configuramos en la interfaz del programa es la altura de la cinta transportadora más la propia altura del objeto.

El brazo robot está programado para que mueva círculos, todos de la misma altura y por separado. Solo puede haber un círculo dentro del espacio de trabajo que reconoce el sistema de visión.

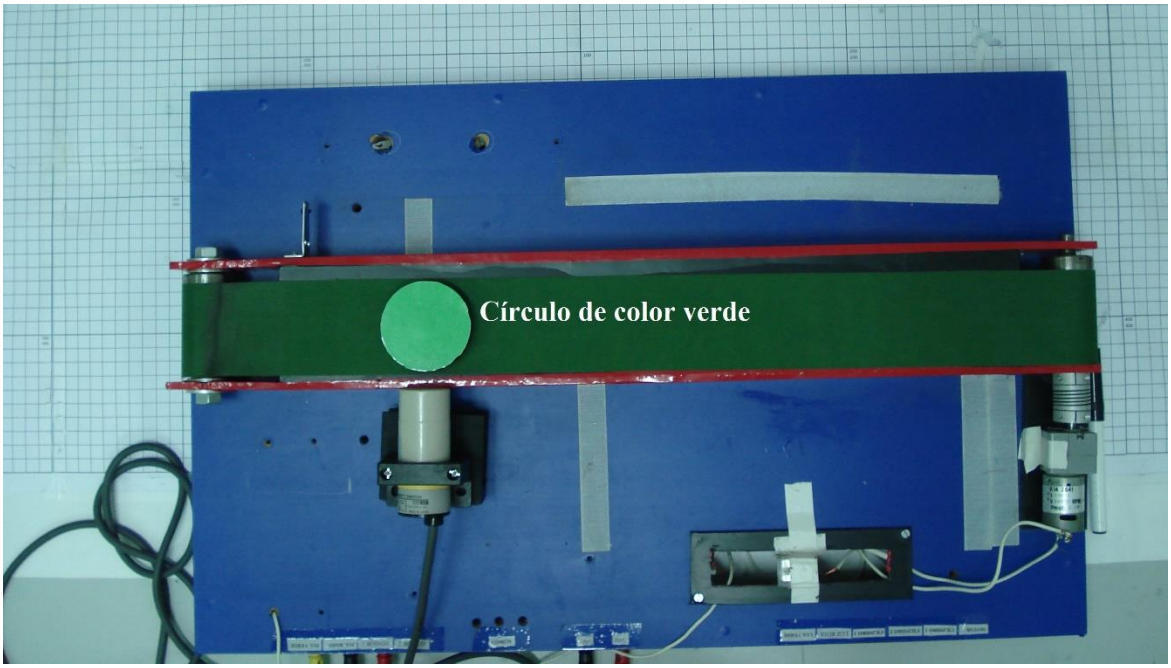


Figura 3.4. Detección de objetos

3.6. Interactuación con los objetos

Se han configurado dos modos de interactuación con los objetos, modo manual y modo automático.

En el modo manual la cinta se para una vez llega el círculo a las proximidades del sensor y el usuario tendrá que pulsar un botón presente en la interfaz para que el brazo robot interactúe con él.

En el modo automático el movimiento del brazo robot es inmediato una vez el sensor detecta que le ha llegado un círculo sin que el usuario tenga que realizar ninguna acción. Una vez que se retira el círculo que detecta el sensor la cinta se volverá a poner en marcha y se detendrá de nuevo al interactuar el nuevo círculo con el sensor. El sistema vuelve a

detectar que pieza es la que está ahora interactuando con el sensor y manda al robot que la mueva a la posición que tengamos asignada para cada color que tenga el círculo que se detecta y así sucesivamente.

La interactuación con los objetos se llevará a cabo mediante succión provocada por una ventosa situada en el extremo del brazo robot.

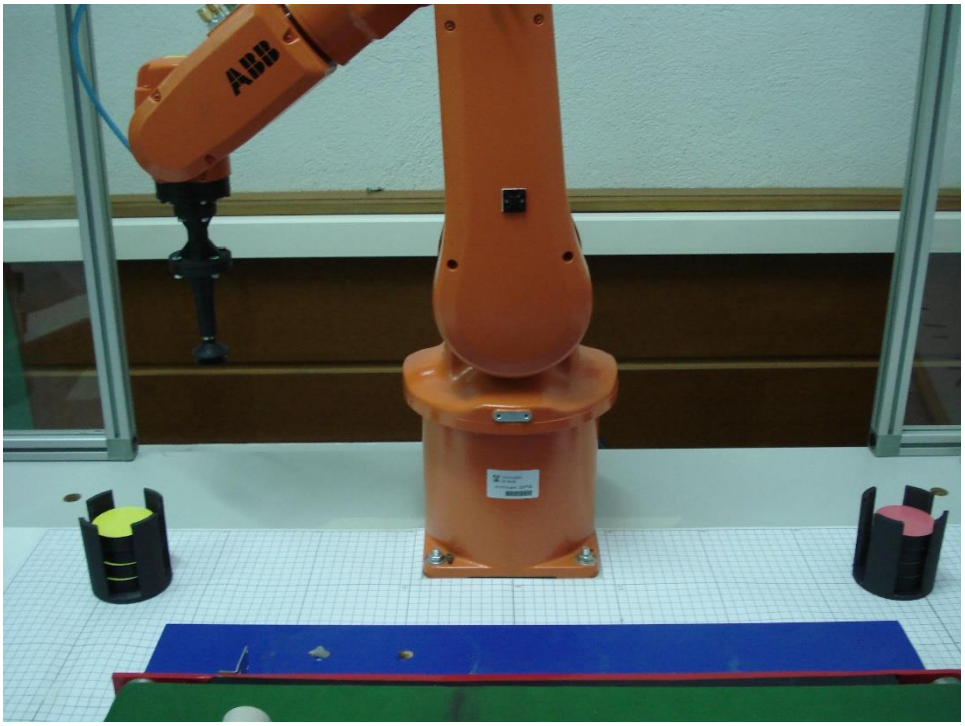


Figura 3.5 Interactuación con los objetos

4. Sistema de visión para el control del IRB-120.

4.1. Detección de círculos

Hay infinitas posibilidades de objetos con los que puede interactuar el brazo robótico IRB120 siempre que cumplan unas condiciones de tamaño y peso máximos.

El proyecto [3] se centró en la detección de rectángulos y círculos por lo que se aprovechará ese trabajo para la detección de círculos de distintos colores, amarillo, rosa y verde.

Como ya se ha dicho, estos pasos están explicados en [3] por lo que aquí simplemente detallamos el resultado final del proceso.

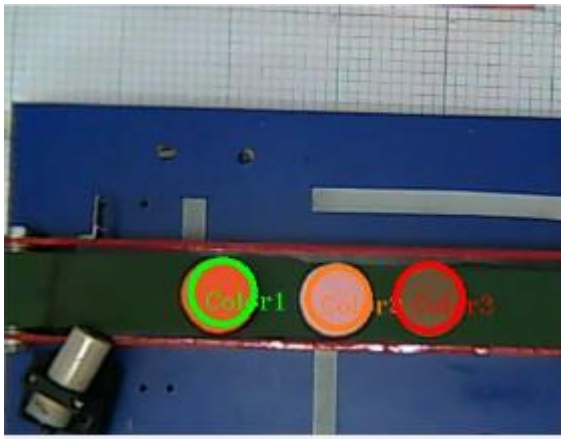


Figura 4.1. Detección correcta de círculos.

Se comprueba que se han detectado únicamente 3 círculos, del mismo tamaño y diferentes colores.

4.2. Manipulación automática de objetos

Es necesario retocar una serie de funciones y añadir algunas líneas de código para realizar la manipulación automática de los distintos círculos de colores que circulan por la cinta.

Basta con añadir la función RobotT2in (para mover círculos), que ya se encuentra implementada, en la parte del código donde se realiza la detección de círculos.
Además, se introducen tres variables tipo bool que nos permitirán distinguir tres colores

distintos para realizar tres tipos de trayectorias distintas en la función Robot2Tin.

Estas funciones de movimiento se realizarán tras pulsar el botón Modo Automático y haya pasado un mínimo de 250 frames, tiempo máximo que tarda un objeto en transcurrir desde el inicio de la cinta transportadora hasta la posición más alejada en la que se puede encontrar el sensor si la pieza está situada en la cinta antes de que esta se empiece a mover.

Para refrescar la adquisición de datos por la cámara basta con añadir el incremento de Procesamiento_Imagen++ al terminar la función de procesamiento del frame obligando así a que el próximo objeto identificado sea 250 frames después.
Esta condición se debe escribir al terminar de analizar cada color por separado para que cada vez que se detecte un color realice el movimiento acorde al tipo que corresponda.

En modo manual basta con añadir las instrucciones de movimiento en el botón Modo Manual y distinguir entre tres tipos de colores como se ha hecho en el modo automático.

4.3. Reconocimiento del espacio de trabajo

Hasta el momento se ha hablado del reconocimiento de objetos en la imagen, de sus coordenada X e Y en píxeles, ángulos de giro etc...

La aplicación debe pasarle al robot la posición en milímetros dentro de la zona de trabajo del mismo, para que pueda interactuar con los objetos adecuadamente.

Realmente un reconocimiento del espacio de trabajo podría efectuarse una única vez si la cámara con la que se trabajase estuviese siempre fija, y no tuviese opción a moverse.
Se puede hacer coincidir el primer píxel que ve la cámara con el punto de trabajo en coordenadas XY que corresponde dicho píxel. Luego se puede medir que área en milímetros alto*ancho aparece en la imagen, y calcular la transformación:

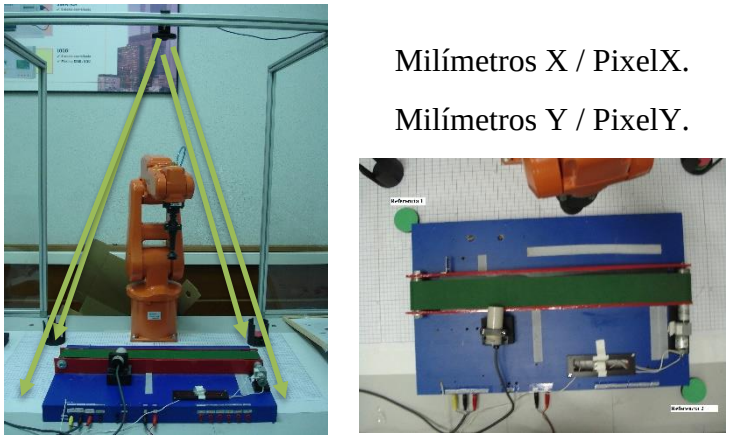


Figura 4.2. Espacio de trabajo

En la práctica esta no es la situación en la que se puede encontrar el entorno de trabajo. El más mínimo cambio en la posición de la cámara, o del robot se traduce en errores de posición de los objetos.

Si la cámara apunta 1 cm más hacia cualquier dirección, el error de posición que tendrá la aplicación será de al menos 1cm en dicha dirección. El cambio de un modelo de cámara por otro puedo hacer que debido a las diferencias en especificaciones técnicas de las diferentes cámaras (diferentes resolución o diferente apertura de la lente) puede llegar a causar errores muchos mayores.

4.4. Calibración del espacio de trabajo.

Como en la detección de cajas y de círculos, las opciones de calibrado del espacio de trabajo, sus problemas y correcciones están explicadas en el mismo proyecto al que ya he hecho referencia antes.

Sin embargo para que se adquieran los objetos con corrección se han cambiado las referencias a unas que abarquen un mayor área.

El área de trabajo que dará delimitada por las 2 referencias:

Referencia 1: mmX = 150, mmY = -350.

Referencia 2: mmX = 600, mmY = 400.

El área final de trabajo es de 750x550mm^2.

En las siguientes imágenes se muestra como se efectúa el reconocimiento del espacio de trabajo, y de la delimitación del mismo.

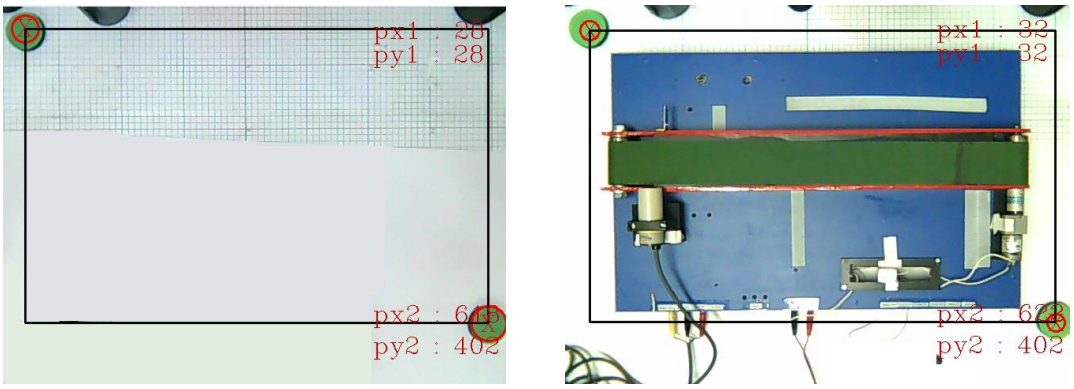


Figura 4.3. Reconocimiento de las referencias y delimitación del área de trabajo.

Para que la aplicación transcurra con mayor fluidez se ha incluido la calibración por defecto.
Esta se lleva a cabo con la adquisición de cuatro referencias incluidas en un fichero llamado datosdefecto, estas referencias serán el pixel al que equivale mmx y mmy de la calibración manual.

Estas referencias son:

P1		**P2**	
X	Y	X	Y
32	32	622	402

Una vez que se inicia el socket se cargan estas referencias por lo que se podrá pasar directamente al desarrollo del programa sin tener que estar calibrando cada vez que se inicie.

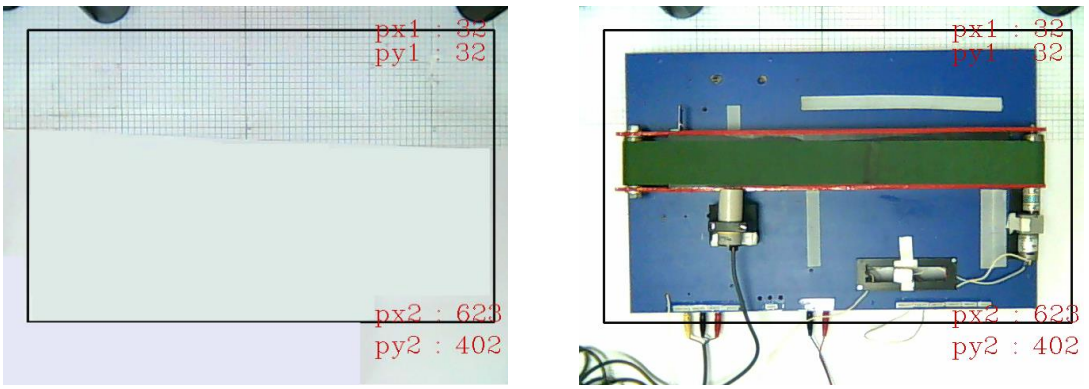


Figura 4.4. Delimitación del área de trabajo por defecto

Sin embargo, se ha dejado la opción de calibrar por si se quiere más precisión en el espacio de trabajo. Además se ha incluido una opción de guardar las referencias obtenidas al calibrar para poder usarlas después si se quisiera.

Estas nuevas referencias se guardan con los datos del último frame antes de que se pulse el botón calibrar por segunda vez en otro archivo llamado datoscalibracion y se actualizan cada vez que se termina de calibrar. Así en caso de fallar la calibración se puede acceder a un frame anterior que puede ser correcto. En caso de que se obtenga una calibración correcta no habrá mucha diferencia entre el frame que se guarda y el que se obtiene, siendo un error que se puede asumir.

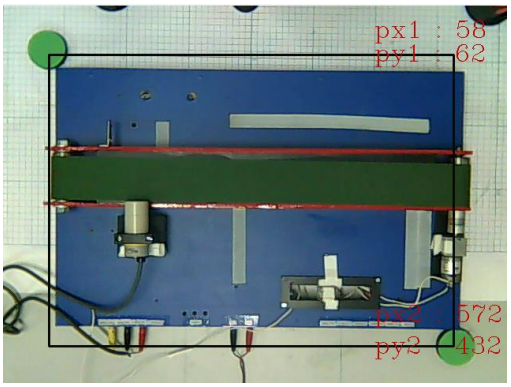


Figura 4.5. Captura usando los datos obtenidos en Last Calibrate

Se debe tener cuidado con estos ficheros si se va a cambiar el ordenador en el que se va a usar el programa puesto que están referenciados a un ordenador y a una carpeta en concreto.

Si se cambia de ordenador se ha de encontrar el directorio de la carpeta con la que trabajemos y sustituirlo por la dirección que se tenía antes.

5. Modelado de la estación

Hasta ahora se ha hablado únicamente de lo relativo al programa en Visual C#. Sin embargo, este programa no puede hacer mover el brazo robótico IRB120 por sí mismo.

Se necesita un programa que cargar en el armario de control IRC5 del brazo robótico que esté esperando órdenes del programa principal en C#.

Es aquí donde entra en juego el socket de comunicación que se ha proporcionado antes del comienzo de este proyecto, y en el cual se ha basado esta aplicación para la comunicación con la estación del robot.

Las especificaciones técnicas del mismo se pueden encontrar de forma concisa en la memoria del proyecto *Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station* de Marek Frydrysiak.

5.1. Herramienta ventosa

Se parte de una estación y programa en RAPID de [9]. En él se encuentra como herramienta y efector final del brazo robótico un portarotuladores.

Se debe por tanto realizar un modelo 3D de la herramienta que se va a utilizar en este proyecto, la ventosa.

Este modelo 3D se debe incluir en la estación, y agregarse como herramienta y efector final.

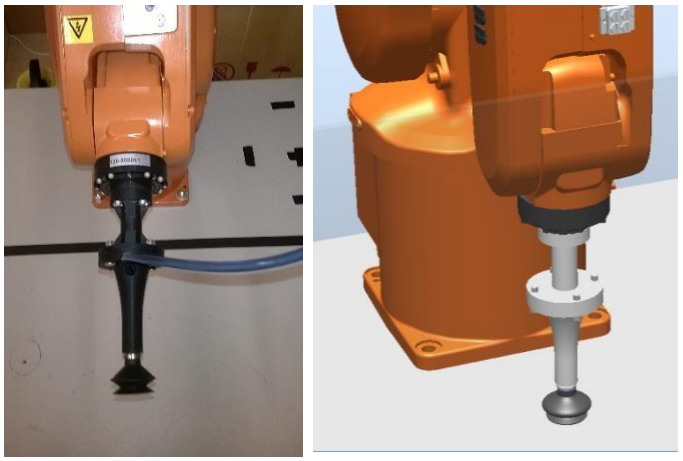


Figura 5.1. Modelos real y virtual de la herramienta ventosa.

En el módulo **"CalibData"** del programa Rapid de la estación que se ha partido es necesario incluir el modelo de herramienta (con sus medidas, inercias, pesos etc...).

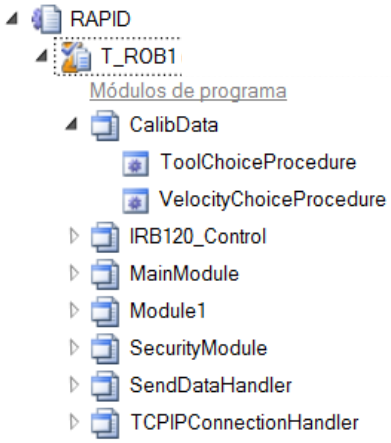


Figura 5.2. Ruta de acceso al módulo RAPID CalibData

```
PERS tooldata Ventosa:=[TRUE,[[0,0,160],[1,0,0,0]],[1,[0,0,60],[1,0,0,0],0,0,0]];
```

El Socket de comunicación por defecto adopta como herramienta principal (tool_selected=1) el portarotuladores. Esto debe cambiarse por la nueva herramienta ventosa:

```
PROC ToolChoiceProcedure (num tool_selected)

    IF tool_selected = 1 THEN
         tool := [TRUE,[[0,0,160],[1,0,0,0]],[1,[0,0,100],[1,0,0,0],0,0,0]]; !Ventosa

     ELSEIF tool_selected = 0 THEN
         tool := [ TRUE, [ [0, 0, 0], [1, 0, 0 ,0] ], [0.001, [0, 0, 0.001], [1, 0, 0,
         0], 0, 0, 0] ];
         !None

     ENDIF

 ENDPROC
```

5.2. Succión ventosa

Como se ha detallado en el capítulo de herramientas utilizadas, la succión de la ventosa se activa a través de la señal digital **D010_9** del armario IRC5.

Es necesario añadir un datagrama a la lista de datagramas a los que responde el programa en Rapid del proyecto citado. Este datagrama simplemente será capaz de activar o desactivar la succión a través de una orden del programa principal en C#.

Si esta salida se encuentra el nivel alto (1), se estará succionando aire a través de la ventosa. Si por el contrario se encuentra a nivel bajo (0), permanecerá en reposo.

Figura 5.3. Datagrama de succión/no succión.

ID	**Nombre:** **Tipo de dato:** **Valor:** **Información:**	Identificador del datagrama Byte 7 Contiene el identificador del datagrama auxiliar de succión.
S	**Nombre:** **Tipo de dato:** **Valor:** **Información:**	Succión / No Succión Byte 0 ó 1 Si S=1, se activa la salida digital D010_9 y se succionará a través de la ventosa. Si S=0, no hay succión.

Para incluir el datagrama en el programa en RAPID es necesario modificar el módulo **"IRB120_Control"** y concretamente en la función **stringHandler**. Se añadirá el caso nuevo con identificador 7.

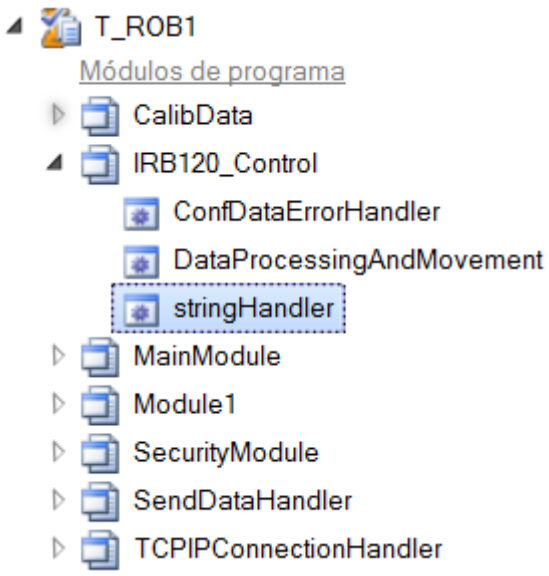


Figura 5.4. Acceso a la función stringHandler.

El código añadido a la función es el siguiente:

```
CASE 7:

    IF Data2Process{2} = 0 THEN  !!attacher

                    SetDO DO10_9, 1; !!succion ventosa

    ELSEIF Data2Process{2} = 1 THEN  !!dettacher

                    SetDO DO10_9, 0; !! stop succion ventosa

    ENDIF
```

5.3. Movimiento de la cinta transportadora

Para poder realizar la aplicación de forma virtual se debe incluir la maqueta de la cinta en RobotStudio, para ello se crea la maqueta en SolidWorks y se incluye en el programa como una herramienta a través de la biblioteca.

No se ha de incluir la maqueta en ninguna parte del programa RAPID ni se debe acoplar al brazo robótico de ninguna forma.

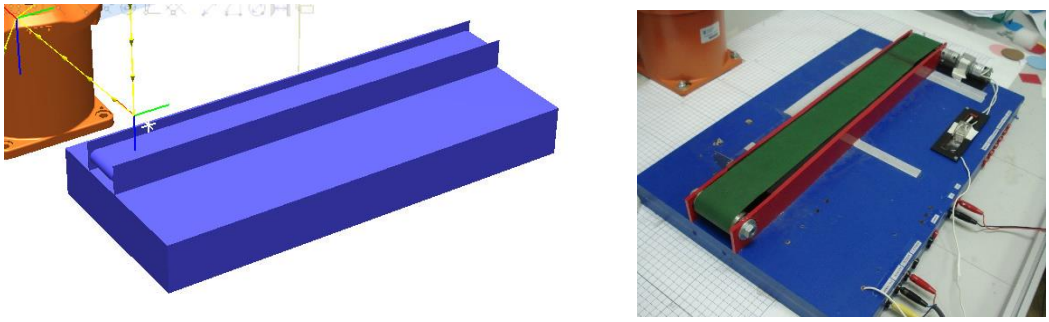


Figura 5.5. Modelos real y virtual de la maqueta.

Como se ha detallado en el capítulo de herramientas utilizadas, el movimiento de la cinta se activa a través de la señal digital **D010_15** del armario IRC5, que pone en marcha o apaga el motor que mueve la cinta.
El sensor envía un nivel alto o nivel bajo a la entrada digital **DI10_14**.

En este caso, como en el de la succión de la ventosa, también se añadirá un datagrama a la lista de datagramas a los que responde el programa en RAPID del anterior proyecto.
Este datagrama será capaz de activar o desactivar el motor a través de una orden del programa principal en C# y a su vez también podrá activar o desactivar el movimiento de la cinta a través del motor dependiendo del estado en el que se encuentre el sensor.

Si esta salida es (3), se estará poniendo en marcha el motor. Si se encuentra en (2), el motor permanecerá apagado por lo que la cinta no se moverá.

En este mismo datagrama se ha incluido la comunicación entre el sensor y la cinta transportadora de tal forma que si la salida es (1) se espera hasta que el sensor no detecte nada en sus proximidades y solo entonces se pone en funcionamiento el motor. Si se encuentra a nivel bajo (0), se espera hasta que el sensor detecta un objeto en sus proximidades y manda un nivel alto al programa, solo entonces el motor se apaga. El nivel alto está programado como medida de seguridad por si el brazo robot no adquiere la pieza correctamente y sigue en la misma posición.

Figura 5.6. Datagrama de movimiento/no movimiento

ID	**Nombre:** **Tipo de dato:** **Valor:** **Información:**	Identificador del datagrama Byte 8 Contiene el identificador del datagrama auxiliar de succión.
S	**Nombre:** **Tipo de dato:** **Valor:** **Información:**	Movimiento/No movimiento Byte 0 ó 1 Si S=3, se activa la salida digital D010_15 y se moverá la cinta transportadora. Si S=2, no hay movimiento. Si S=1, se espera a que la entrada digital DI10_14 sea 0 y activa la salida digital D010_15 que moverá la cinta transportadora. Si S=0, se espera a que la entrada digital DI10_14 sea 1 y desactiva la salida digital D010_15 lo que parará la cinta transportadora

Para incluir el datagrama en el programa en RAPID es necesario modificar el módulo **"IRB120_Control"** y concretamente en la función **stringHandler**. Se añadirá el caso nuevo con identificador 8.

```
CASE 8:

                    IF Data2Process{2}=0 THEN

                        WaitDI DI10_14,1;
                        SetDO DO10_15,0;


                    ELSEIF Data2Process{2}=1 THEN

                        WaitDI DI10_14,0;
                        SetDO DO10_15,1;

                    ELSEIF Data2Process{2} = 2 THEN

                        SetDO DO10_15, 0; !!cinta parada
                    ELSEIF Data2Process{2} = 3 THEN

                        SetDO DO10_15, 1; !!cinta en marcha
    ENDIF
```

6. Envío de la información y posición

6.1. Introducción al Socket de comunicación

Una vez se han detectado correctamente los objetos deseados, se procederán a enviar el tipo de objeto, su color y su colocación al brazo robótico.

6.2. Datagrama de posición y orientación.

Los movimientos finales del robot se llevarán a cabo a través del datagrama de Posición y Orientación creado por Marek Frydrysiak.

Este datagrama permite enviar la posición y orientación a través de varios datos de 1 byte.

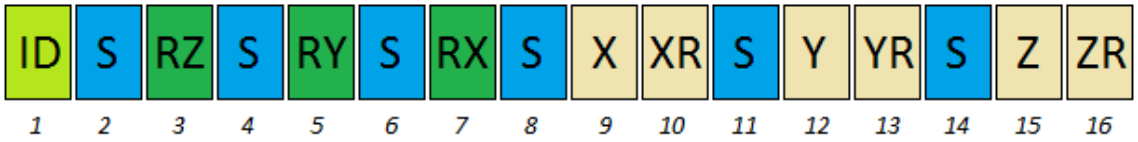


Figura 6.1. Datagrama de posición.

ID	**Nombre:** Tipo **de dato:** **Valor:** **Información:**	Identificador del datagrama Byte 4 Contiene el identificador del datagrama de posición y orientación.
S	**Nombre:** Tipo **de dato:** **Valor:** **Información:**	Signo del valor Byte 0 ó 1 $sgn(PosRot_{n+1}) = \begin{matrix} -PosRot_{n+1}\ si\ S_n=0 \\ PosRot_{n+1}\ si\ S_n=1 \end{matrix}$ Implica el signo del siguiente valor (posición o ángulo).
R_{zyx}	**Nombre:** Tipo **de dato:** **Valor:** **Información:**	Valor absoluto de rotación. Byte 0 – 255 Valor absoluto de la rotación EulerZYX.

X/Y/Z **XR** **YR** **ZR**	**Nombre: Tipo de dato:** **Valor:** **Información:**	Valor absoluto de posición. Byte X/Y/Z contiene valor en el rango 0 – 255 XR/YR/ZR contiene valores en el rango 0 – 99 Debido a la limitación del tipo de dato (byte) a la hora de enviar el datagrama es necesario una separación entre centenas y decenas y unidades. La fórmula matemática para la coordenada X es: $$Xreal = 100 \cdot Xframe + XR$$ Donde: Xreal – coordenada X en el Sistema de referencia delrobot; Xframe – cantidad de centenas en Xreal XR – el resto del valor (< 100)

Por ejemplo, para el acercamiento a una caja de **70mm de alto** que se encuentre en la posición:

X=373 **Y= - 241** **Ángulo de giro= 30º.**

El datagrama para acercarse a su posición seria:

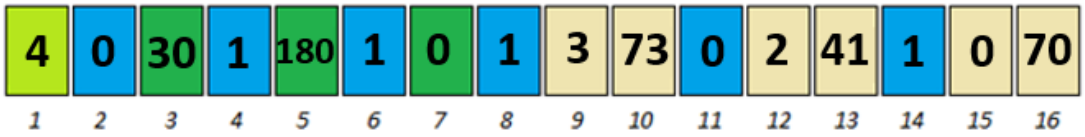


Figura 6.2. Ejemplo de datagrama de posición.

7.Manual de usuario

En este capítulo se detallará un manual para el usuario, con el cual se podrá acceder a cualquier funcionalidad de la aplicación. Se explicarán detalladamente los paneles de cada una de los Forms y su función dentro del programa. También se hará una guía de botones con los que se ayudará al usuario a entender mejor el funcionamiento del sistema.

7.1. Estructura de los paneles

7.1.1. Panel de control de posición y detección

Se trata del panel principal del programa. Contiene la información sobre los objetos y referencias a detectar, botones para comenzar el programa, calibración, conexión con el brazo robótico. También desde este panel se dan las órdenes de interactuación con los objetos.

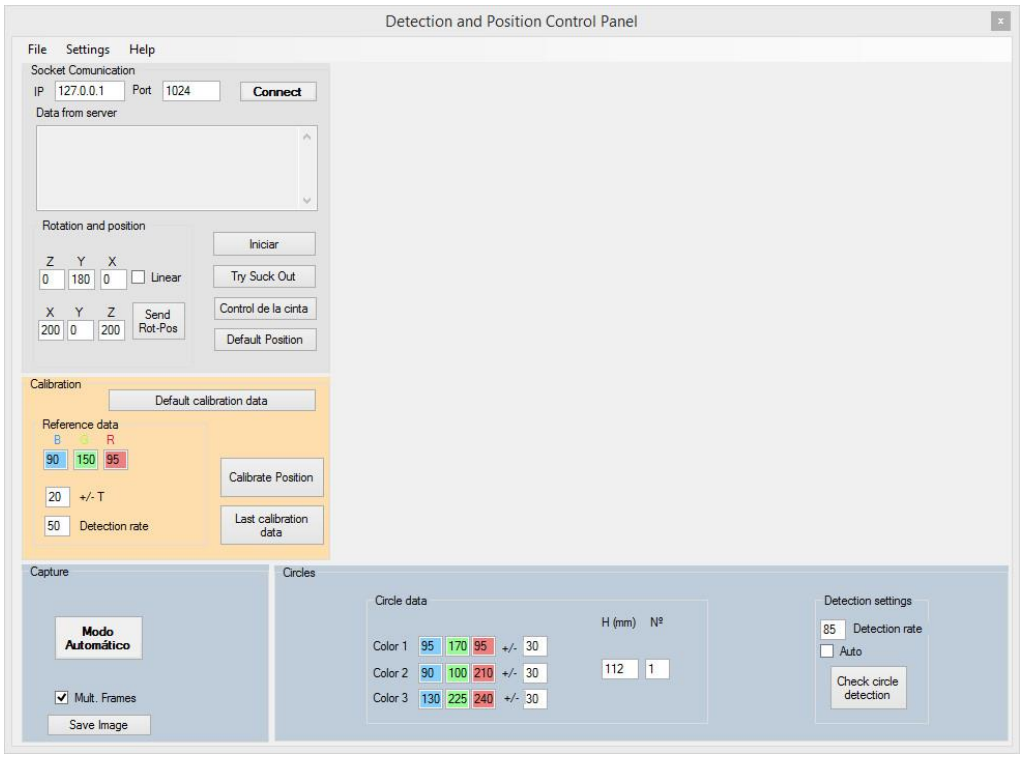


Figura 7.1. Panel de control de posición y detección

7.1.2 Calibración y detección de círculos

Se trata de un panel de ayuda al panel principal para la detección de los círculos. Se usa para ajustar los valores RGB de los diferentes círculos a detectar, así como aumentar o disminuir la detectabilidad. Contiene 3 cuadros de imagen que permiten ver como si se umbraliza correctamente la imagen para los diferentes colores.

Los datos RGB, tolerancias y detectabilidad que se cambien en este panel pueden actualizarse y llevarse directamente hacia el programa principal.

La estructura de acceso a los paneles es por tanto la siguiente:

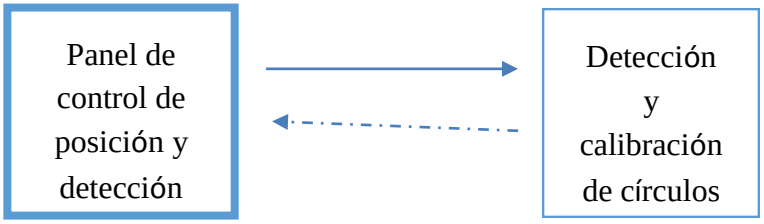


Figura 7.2. Esquema de paneles

7.2. Guía de Botones

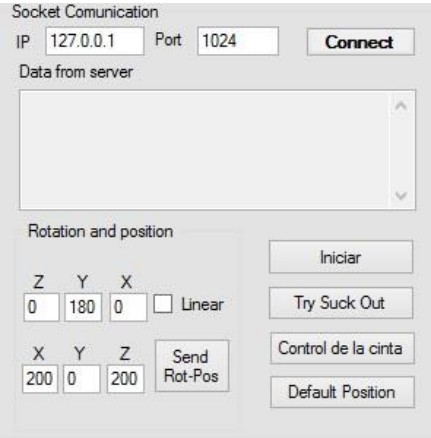
En este apartado se verán los botones, cuadros y cajas de texto dentro de los diferentes paneles para especificar su función.

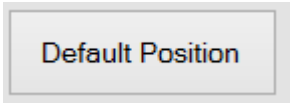
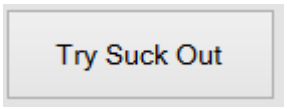
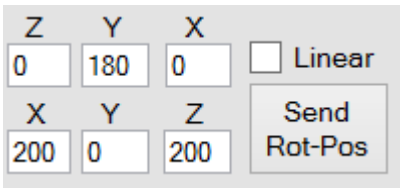
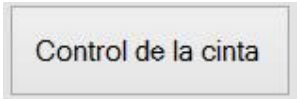
7.2.1. Botones y herramientas del panel de control de posición y detección

7.2.1.1. Comunicación del Socket

Esta parte del panel permite la especificar y efectuar la conexión con el robot, así como la posibilidad de moverlo a una posición y orientación específicas y dar comienzo a la ejecución del trabajo. También permite probar la succión en la herramienta ventosa y el funcionamiento de la cinta transportadora.

Esta parte de la aplicación en su mayoría pertenece al proyecto "*Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station*" de Marek Frydrysiak, por lo que solo se dará una breve descripción de la misma.

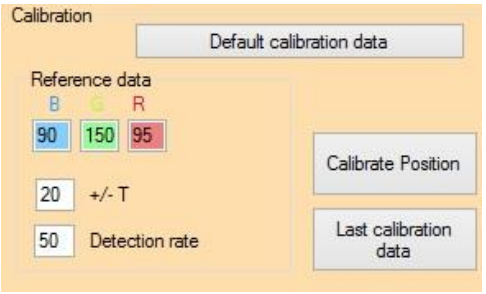


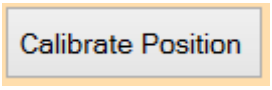
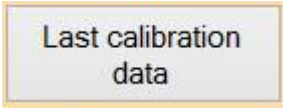
Botón **Connect** 	Permite establecer la conexión entre el programa y el brazo robótico IRB-120.
Botón **Default Position** 	Mueve el brazo robótico a una posición inicial, despejando una posible obstrucción del camino óptico entre la cámara y la zona de trabajo.
Botón **Try Suck Out** 	Permite comprobar la succión en la herramienta ventosa. Si se pulsa una vez, actuará la succión. Para dejar de succionar basta con pulsar de nuevo.
Botón **Send Rot-Pos** 	Mueve el brazo robótico a la orientación y posición especificada en los cuadros de textos próximos al botón.
Botón **Control de la cinta** 	Permite controlar el movimiento de la cinta transportadora. Si se pulsa una vez, activará el motor. Para desactivar el motor basta con pulsar de nuevo.

Botón **Iniciar** 	Mueve el brazo robótico a la posición inicial y activa el movimiento de la cinta. Esta seguirá en funcionamiento hasta que el sensor no detecte un objeto en sus proximidades.

7.2.1.2. Calibración

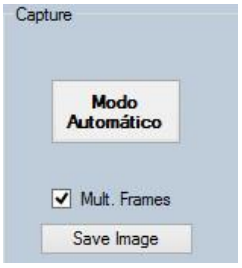
Esta pequeña parte contiene la información y la orden de calibrar la posición de la zona de trabajo.

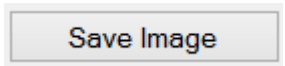


Botón **Calibrate Position** 	Permite obtener imágenes continuas de la cámara en las que se buscará la ubicación de las referencias a través de los datos incluidos en los cuadros de texto inferiores (coordenadas RGB, tolerancia y detectabilidad). En el cuadro de imagen se dibujará sobre las referencias siempre que se hayan detectado.
Botón **Default calibration data** 	Carga unas referencias ya creadas por defecto y se dibuja un rectángulo negro para poder ver con claridad el espacio de trabajo. Estas referencias están en un fichero llamado datosdefecto incluido en la carpeta de trabajo.
Botón **Last calibration data** 	Carga los últimos datos obtenidos al realizar la calibración pulsando Calibrate Position. Estos datos se guardan en un archivo llamado datoscalibracion incluido en la carpeta de trabajo.

7.2.1.3. Modo Automático

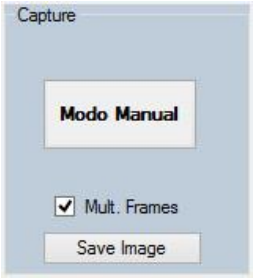
Esta parte del panel contiene los botones relacionados con el movimiento del brazo robot cuando hay un objeto en las proximidades del sensor y la posibilidad de guardar las imágenes obtenidas.



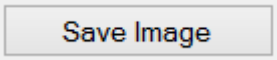
Botón **Modo Automático** 	Permite comenzar el movimiento del brazo robot. Este no se parará hasta que pulsemos el mismo botón con el texto Stop.
Casilla **Mult. Frames** 	Si está marcada, la adquisición de las imágenes será continua. En caso de no estar marcada, solo se adquirirá una imagen.
Botón **Save Image** 	Guarda la imagen procesada actual del cuadro de imagen, así como la original de la que se partió para procesarla.

7.2.1.4. Modo Manual

Esta parte del panel contiene los botones relacionados con el movimiento del brazo robot cuando hay un objeto en las proximidades del sensor y la posibilidad de guardar las imágenes obtenidas.

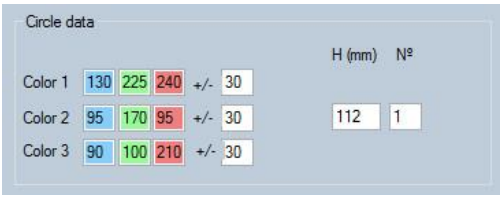


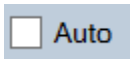
Botón **Modo Manual** 	Permite adquirir una imagen y comenzar el movimiento del brazo robot en caso de que el objeto detectado haya llegado al final de la cinta.
Casilla **Mult. Frames** 	Si está marcada, la adquisición de las imágenes será continua. En caso de no estar marcada, solo se adquirirá una imagen.

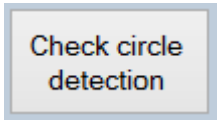
Botón **Save Image** 	Guarda la imagen procesada actual del cuadro de imagen, así como la original de la que se partió para procesarla.

7.2.1.5. Círculos

En esta zona del panel principal se podrán configurar los parámetros de los círculos a detectar (RGB, tolerancias, detectabilidad), modo de detección automático de los círculos y el envío de la altura de estos al robot.

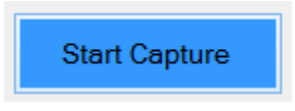
Cuadros de texto con la información de los círculos a detectar. 	Contienen la información de hasta 3 tipos de círculos a detectar: Coordenadas BGR de los 3 colores. (+/-T) Tolerancia de los valores BGR (para evitar problemas con texturas poco uniformes). Altura (H) en mm. Número de círculos de cada color.

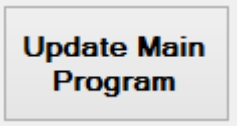
Casilla **Auto** 	Si existen problemas de detectabilidad temporal (solo en algunos frames se logran detectar todos los círculos que se quieren detectar) y se quieren detectar un cierto número de círculos es conveniente activar esta casilla. Si es activada detendrá la recepción de imágenes desde la cámara USB tan pronto se hayan detectado el número de círculos especificado.

Cuadro de texto **Detection Rate** 	Este valor tiene que ver directamente con la detectabilidad de los círculos. A mayor valor, más difícil será detectar círculos (tanto falsos positivos como verdaderos). A menos valor, más frecuencia de círculos detectados.
Botón **Check Circle Detection** 	Abre el panel **Circle Detection Calibration.**

7.2.2. Botones y herramientas Calibración y detección de círculos

Este panel contiene exactamente los mismos controles de detección de círculos que el programa principal, porque lo que no se hablará de ellos. Existen además los siguientes botones:

Botón **Start Capture** 	Permite comenzar a obtener imágenes desde la cámara USB. El resultado de las imágenes, las umbralizaciones y los objetos detectados serán visibles en los diferentes cuadros de imagen.
Botón **1 Frame** 	Si no está marcada, la adquisición de las imágenes será continua si se pulsa el botón **Start Capture**. En caso de marcarse, solo se adquirirá una imagen.

Botón **Update Main Program** 	Actualiza los valores de los cuadros de texto BGR, tolerancia y detectabilidad del panel principal.

7.3. Flujo del programa

Una vez detallados los botones, herramientas y paneles se puede pasar a explicar el flujo de trabajo de la aplicación.

7.3.1. Reconocimiento del espacio de trabajo

A) En esta fase se deberá activar la cámara USB a través del botón **Calibrate Position**. En el cuadro de imagen aparecerá la imagen de la cámara, y se verá si el programa es capaz de detectar las dos referencias de la zona de trabajo. Si no es capaz de detectarse, conviene ajustar los parámetros RGB, tolerancia y detectabilidad de las referencias.

1. Clic **Calibrate Position**

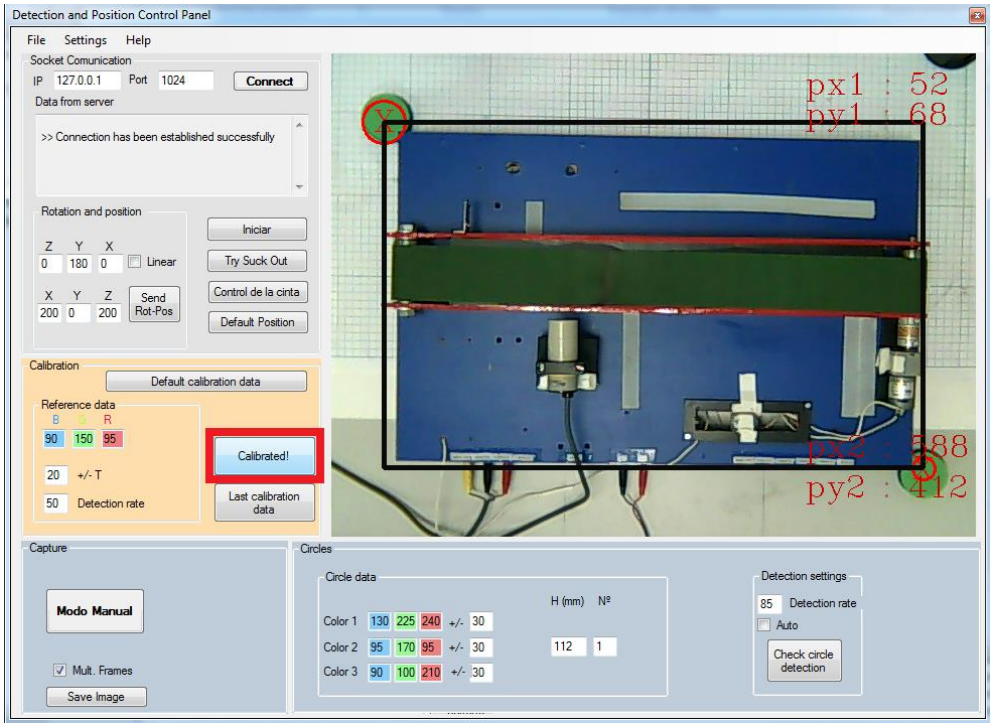


Figura 7.3. Paso calibrar posición

2. En caso de comprobar que se detectan bien las referencias, clic de nuevo **Calibrate Position.**
3. Si no se detectan bien, modificar los valores RGB (con ayuda de del software

PixelColor), tolerancia y detectabilidad y comenzar de nuevo con el paso 1.

4. Estas referencias se guardarán en el fichero datoscalibracion para ser usados posteriormente si el usuario lo desea.

B) Clic **Default Position**

1. En esta fase se deberá activar la cámara a través del botón **Default Position** o simplemente a través del botón **Iniciar**. En el cuadro de la imagen aparecerá la imagen de la cámara y un rectángulo de color negro que identifica el espacio de trabajo dado por las referencias por defecto.

2.

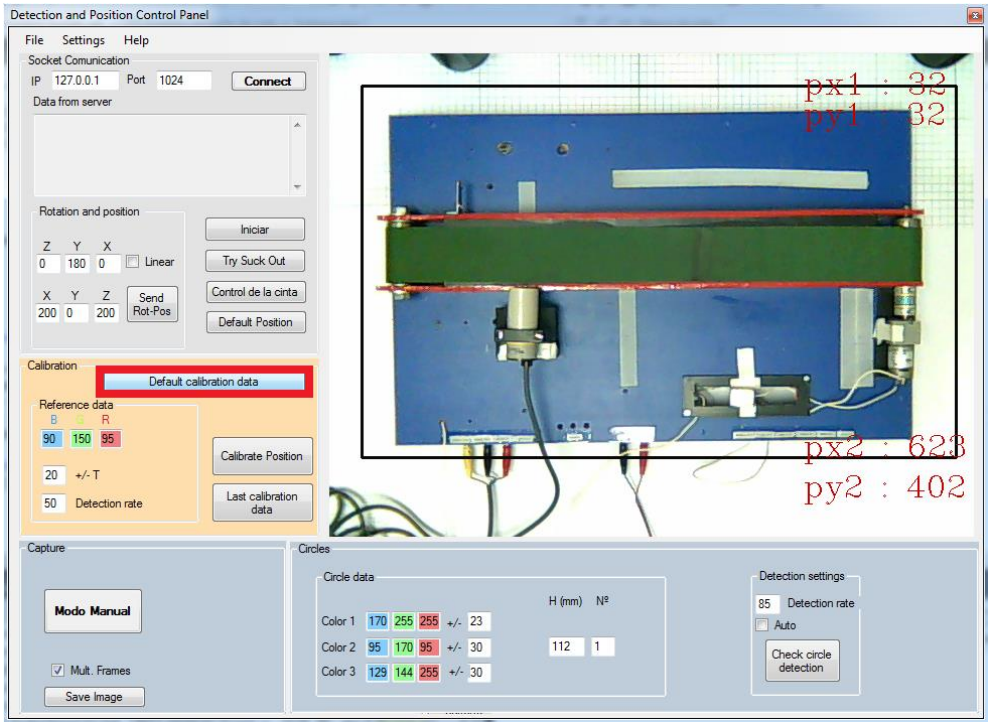


Figura 7.4. Calibración por defecto

3. Las referencias se adquieren del fichero llamado datosdefecto. Siempre utilizaremos esas referencias a no ser que pulsemos **Calibrate Position**.

Esta parte relacionada con el reconocimiento del espacio de trabajo es opcional tanto en su apartado A) como B) al estar implementada la calibración por defecto al iniciarse el programa.

7.3.2. Caracterización de círculos

Una vez calibrada la detección de cuadrados y círculos es necesaria la caracterización de círculos, en la que se añadirán datos sobre los objetos detectados o el modo de colocarlos.

1. **Rellenar los cuadros de texto** de los datos de altura de los círculos. Rellenar también el número de círculos de cada color si se prevé necesaria una detección automática.

Se ha de tener en cuenta que según cada color el círculo se depositará en una posición prefijada, por tanto, se puede elegir a que posición se quiere que vaya el círculo de un color determinado dependiendo de si se ponen sus datos en Color 1, Color 2 o Color 3.

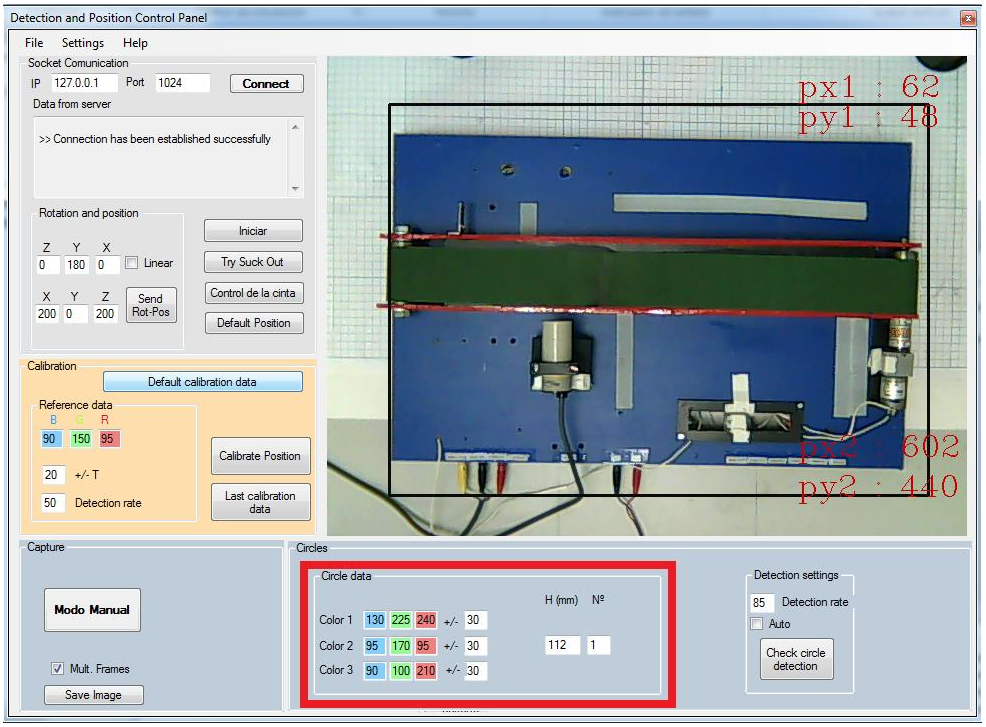


Figura 7.5. Paso información de círculos

7.3.3. Modo Manual

Hasta ahora los pasos son los mismo estemos en la interfaz manual o automática. A partir de ahora se separan los pasos a realizar.

1. Clic en **Connect**.

2. Clic en **Iniciar**.

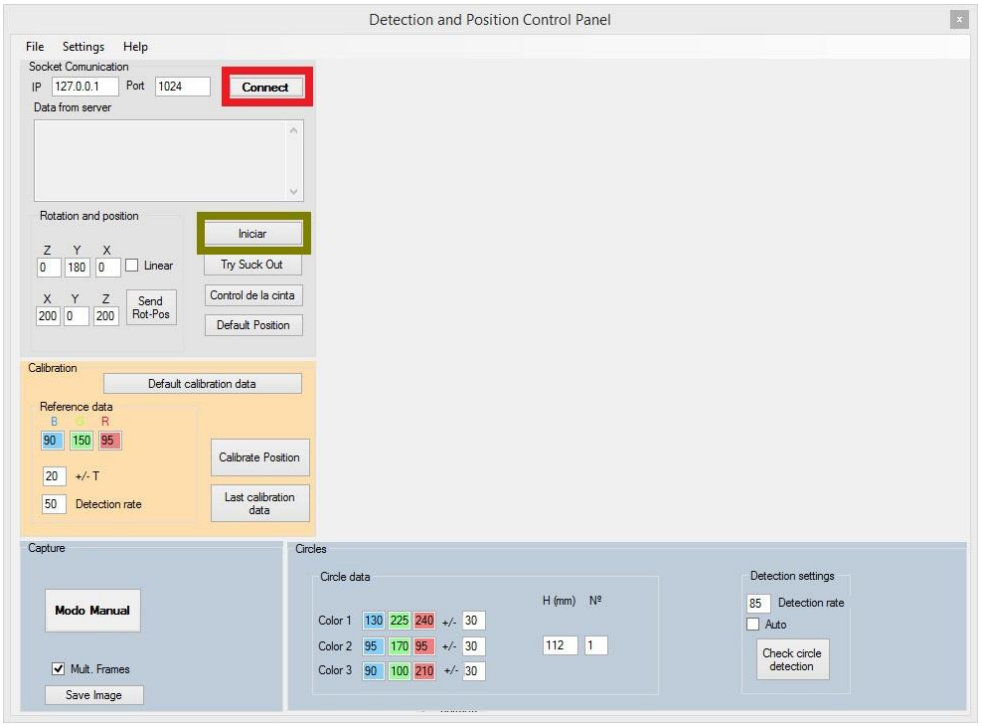


Figura 7.6. Pantalla inicial

El brazo robot se moverá a una posición inicial dada con antelación y comenzará la detección de círculos. Al mismo tiempo el motor que mueve la cinta transportadora se pone en marcha y el sistema espera el nivel alto de la entrada 14 para realizar la siguiente acción.

3. Clic en **Modo Manual**.

Una vez el círculo ha llegado a las proximidades del sensor y la cinta se ha detenido pulsamos **Modo Manual** y el brazo robot se desplazará a la posición donde está el círculo y mueve el círculo del color que haya detectado a la posición que corresponda. Tras esta acción el brazo vuelve a la posición inicial, la cinta se vuelve a poner en marcha y el sistema de visión vuelve a detectar nuevos círculos.
Estos nuevos círculos pueden pararse en la misma posición que el anterior o en otra distinta siendo indiferente para el desarrollo del programa.

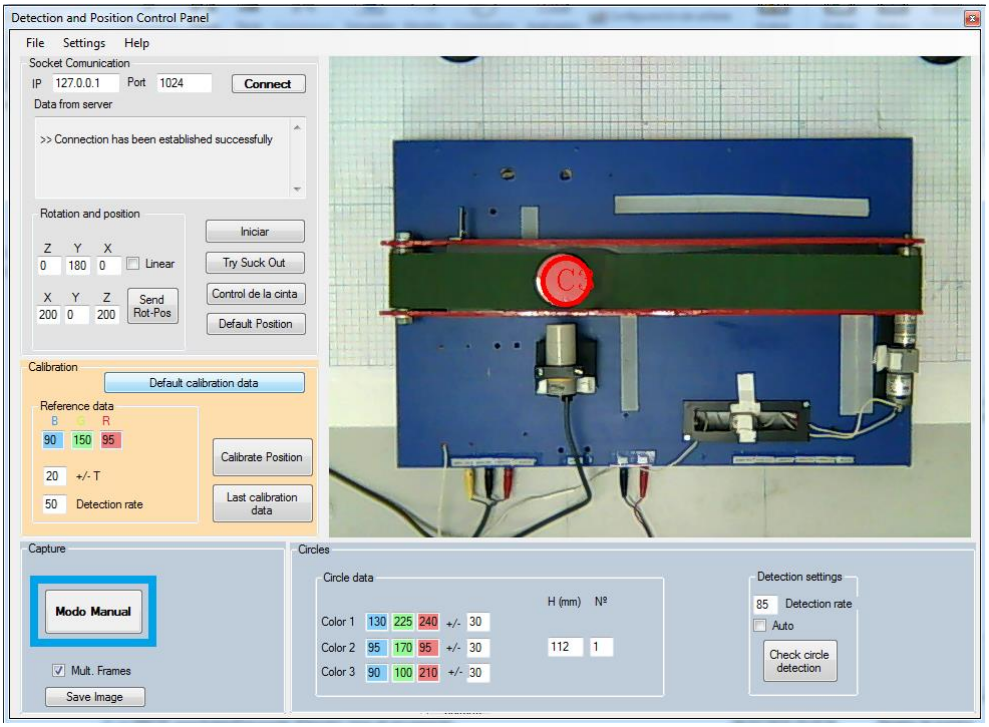


Figura 7.7. Llega el primer círculo

4. Cuando el nuevo círculo llega a las proximidades del sensor volvemos a pulsar **Stop.** Este paso se repite hasta que no lleguen más círculos al sensor.

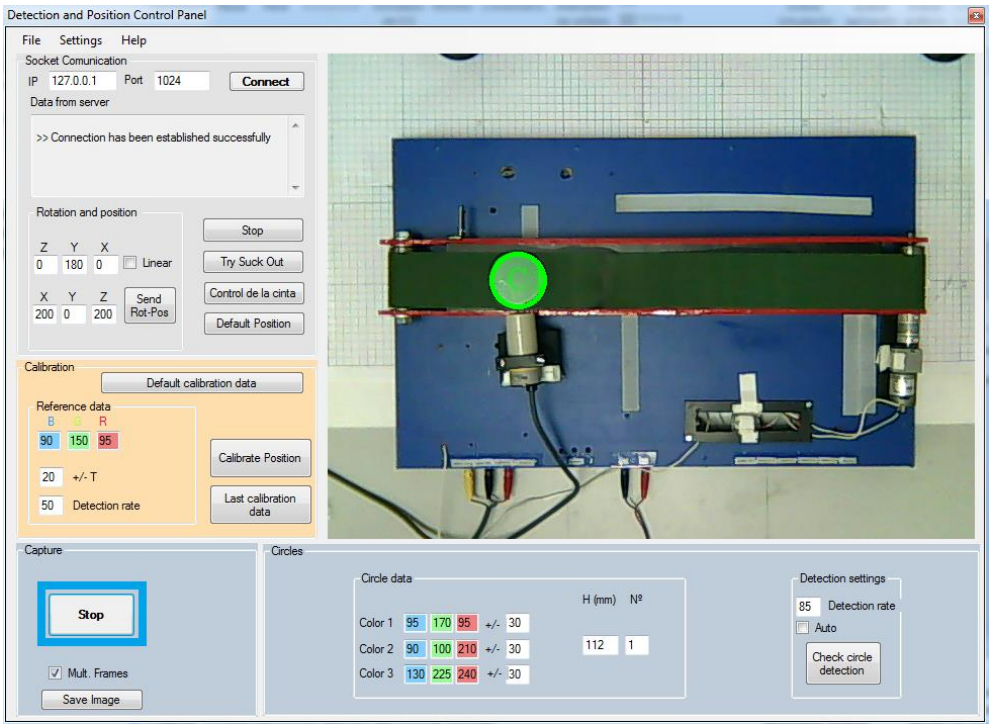


Figura 7.8. Aparición de otro color

7.3.4. Modo Automático

1. Clic en **Connect**.

2. Clic en **Iniciar**.

El brazo robot se moverá a una posición inicial dada con antelación y comenzará la detección de círculos. Al mismo tiempo el motor que mueve la cinta transportadora se pone en marcha y el sistema espera el nivel alto de la entrada 14 para realizar la siguiente acción.

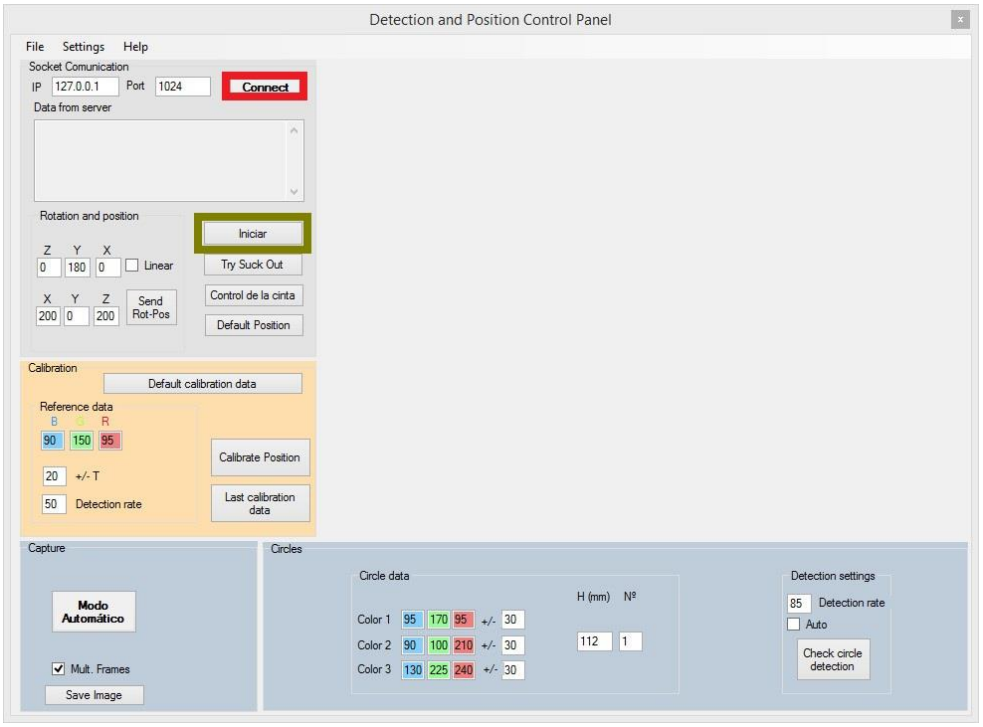


Figura 7.9. Pantalla inicial

3. Clic en **Modo Automático**.

El sistema espera 250 frames para captar la imagen. Si tenemos un nivel alto en la entrada 14 el brazo robot se mueve hacia la posición donde se encentre el círculo según las instrucciones determinadas para el color del círculo detectado. Una vez estas acaben, la cinta vuelve a ponerse en marcha y el sistema comienza la detección de círculos tardando de nuevo 250 frames en obtener la captura.
Se podrá mover la posición del sensor entre la detección de un color y otro dado que la posición de parada del círculo no afecta al programa.

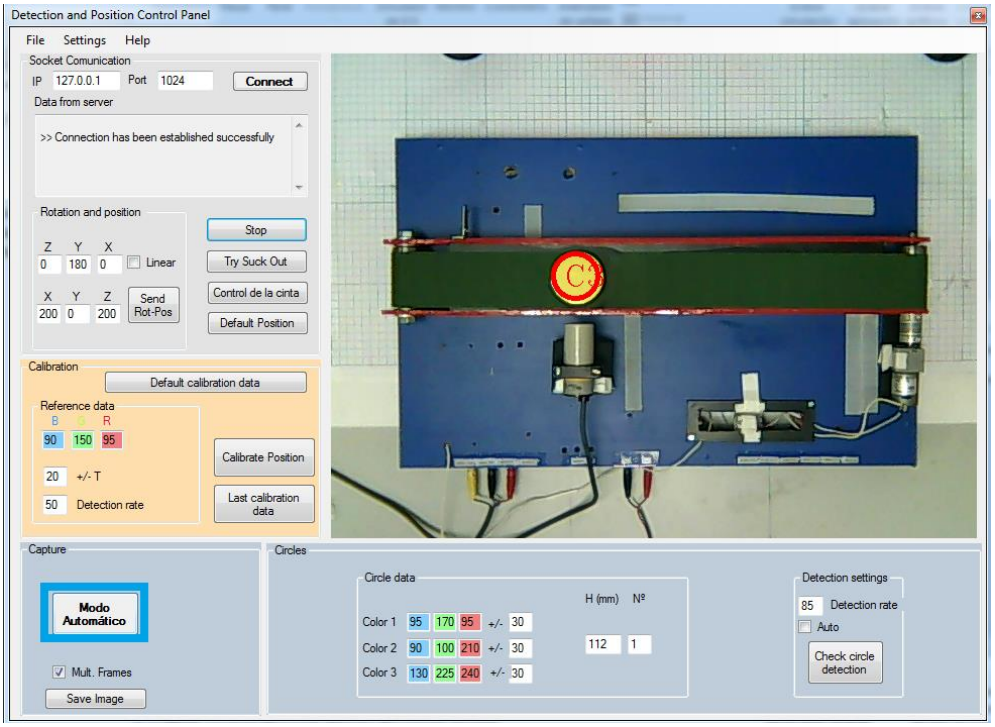


Figura 7.10. Primer color

Esto se repetirá de forma secuencial hasta que el sistema no detecte ningún círculo. Entonces mandaremos un nivel alto en la entrada 14 de forma manual y pulsaremos **Stop** donde antes estaba **Modo Automático** y **Stop** donde antes estaba **Iniciar** para que el robot vuelva a la posición inicial.

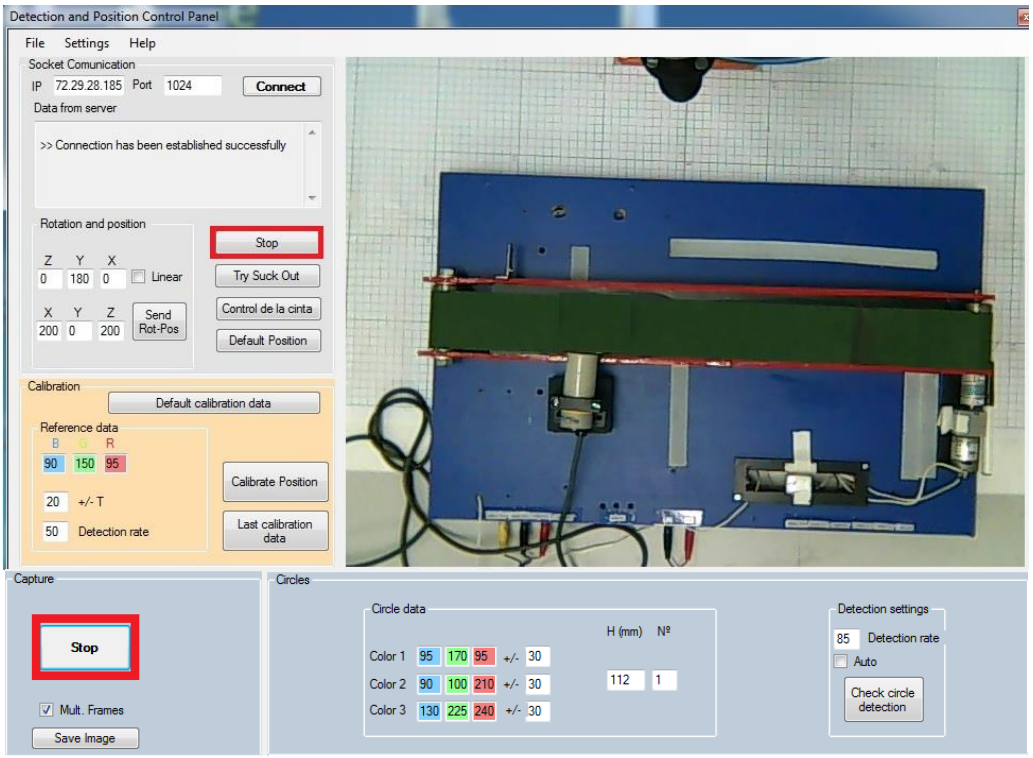


Figura 7.11. Finalizar

7.4. Cargar programa en armario IRC5

En este apartado se detallará como cargar el programa en RAPID en el armario de control IRC5 a través del FlexPendant.

Para comenzar, se debe actuar sobre estas llaves. La primera llave a ON, para encender la estación. Y la segunda llave indica el modo de funcionamiento, ya sea manual o automático. Es recomendable dejarlo en modo manual tal y como viene representado en la imagen puesto que ofrece un mayor grado de seguridad ante cualquier problema.

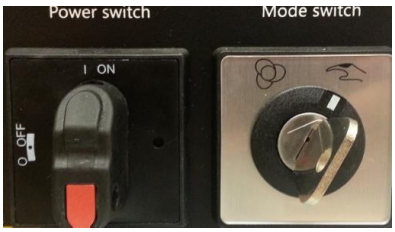


Figura 7.12. Encendido y Modos automático/manual

Una vez conectada la estación del robot, se pasa a trabajar con el mando FlexPendant. Lo primero que se debe hacer, si no está hecho ya, es cargar el programa en la estación, para ello se puede conectar una memoria externa en el mando. Una vez conectado:

- FlexPendant Explorer→ Copiar y pegar el programa en la carpeta en la que se trabaje→ Datos de programa → Cargar el programa → Puntero a main.

Se debe cargar el programa con el siguiente nombre:

MAQUETA_SENSOR.pgf

Para acceder a los directorios indicados en el párrafo anterior basta con pulsar el icono de **ABB** de la esquina superior izquierda.

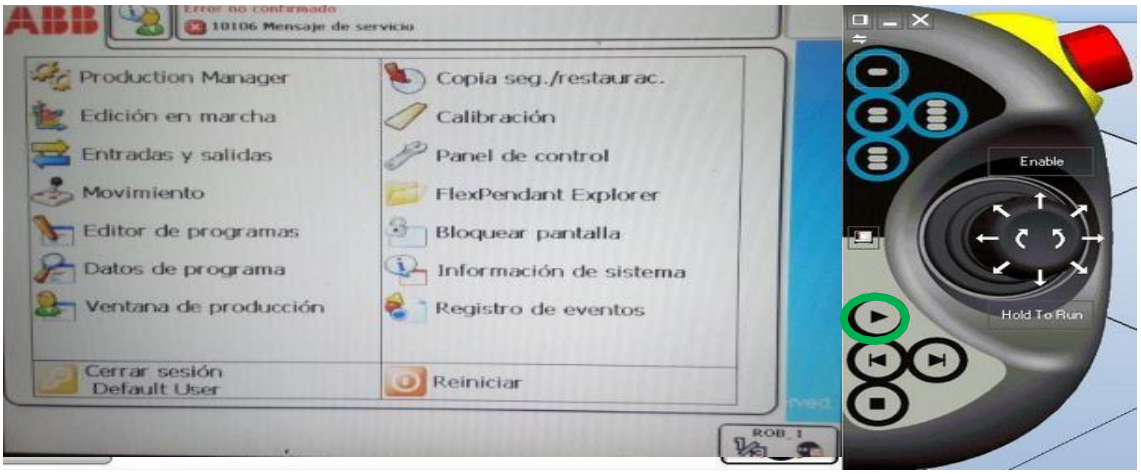


Figura 7.13. Ventana navegación FlexPendant

Una vez cargado el programa, con las instrucciones indicadas, se presiona el botón del "hombre muerto" y se pulsa el botón **Start.**

8.Resultados

8.1. Introducción

En esta sección se pasará a especificar los resultados obtenidos tanto de obtención de imágenes, detección de objetos, reconocimiento de espacio de trabajo e interactuación con los objetos.

De las numerosas situaciones que pueden darse en cuanto a los objetos detectados se van a proponer las siguientes:

1. Color 1= Amarillo, Color 2= Verde, Color 3= Rosa - Modo Manual.
2. Color 1= Verde, Color 2= Rosa, Color 3= Amarillo - Modo Automático.

Figura 8.1. Círculos de colores que usaremos en ambos modos

8.2. Colores asignados y modo manual

Se hará uso del programa para mover tres círculos de tres colores distintos que se dejarán en distinta posición.

Color	Posición paletizado
Amarillo	x= 100 y= -380
Verde	x= 100 y= -220
Rosa	x= 100 y= 380

8.2.1. Reconocimiento del espacio de trabajo e introducción de datos

Lo primero que ha de hacerse necesariamente es el reconocimiento del espacio del espacio de trabajo ya sea mediante la calibración manual a través de dos objetos de color verde situados en el extremo del espacio de trabajo o simplemente iniciando el programa sin más con lo que accedemos a la calibración por defecto. Se recomienda esta última opción al ser la más cómoda y la que menos pasos necesita al estar ya implantada.
Se introducen los datos de los colores en cada ranura. Los colores con los que se va a

trabajar serán, amarillo (140-255-255), verde (95-170-95) y rosa (129-144-255).

8.2.2. Iniciar movimiento del brazo robot y nuevo círculo

Una vez introducidos los datos de los colores amarillo, verde y rosa, se inicia el programa, este detectará el color de los anteriormente comentados que circule por la cinta y al llegar al sensor iniciará el movimiento hacia el lugar que le corresponda a ese color al pulsar el botón Modo Manual.

Una vez situados los tres círculos en sus lugares correspondientes se provoca una señal de nivel alto en el sensor de forma manual para que la cinta se pare y se da por terminado el programa.

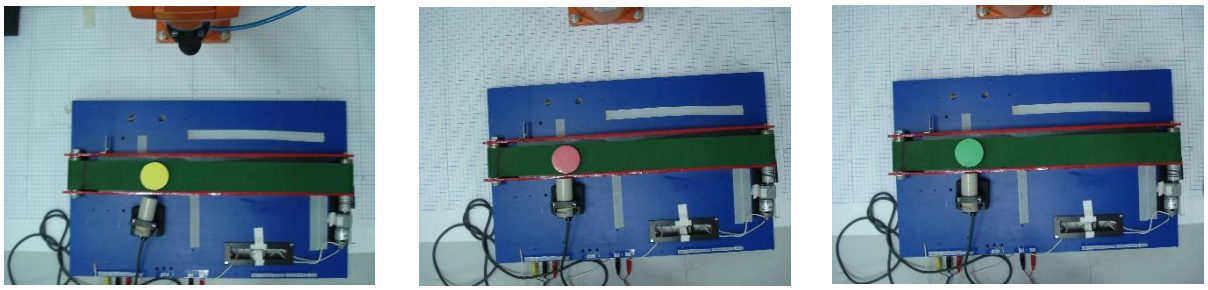


Figura 8.2.Círculos llegando al sensor

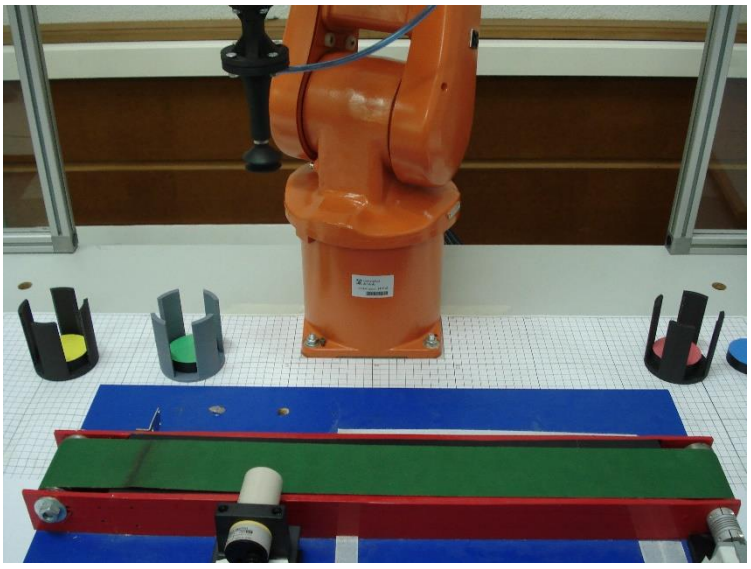


Figura 8.3. Círculos colocados

8.3. Colores asignados y modo automático

Se hará uso del programa para mover tres círculos de tres colores distintos que se dejarán en distinta posición.

Color	Posición paletizado
Verde	x= 100 y= -380
Rosa	x= 100 y= -220
Amarillo	x= 100 y= 380

8.3.1. Reconocimiento del espacio de trabajo e introducción de datos

Lo primero que ha de hacerse, como en la aplicación manual, es el reconocimiento del espacio de trabajo ya sea mediante la calibración manual a través de dos objetos de color verde situados en el extremo del espacio de trabajo o simplemente iniciando el programa sin más con lo que accedemos a la calibración por defecto. Se recomienda esta última opción al ser la más cómoda y la que menos pasos necesita al estar ya implantada.

Se introducen los datos de los colores en cada ranura. Los colores con los que se va a trabajar serán, verde (95-170-95), rosa (129-144-255) y amarillo (140-255-255).

8.3.2. Iniciar movimiento del brazo robot y modo automático de éste.

Una vez introducidos los datos de los colores verde, rosa y amarillo, se inicia el programa, este detectará el color de los anteriormente comentados que circule por la cinta y al llegar al sensor y cuando hayan transcurrido 250 frames iniciará el movimiento hacia el lugar que le corresponda a ese color.
Una vez acabado el movimiento un nuevo círculo se mueve por la cinta y se volverá a parar al llegar al sensor, iniciándose así de nuevo el paso anterior. Esto se repetirá con el círculo que queda.

Una vez situados los tres círculos en sus lugares correspondientes introducimos una señal de nivel alto en el sensor de forma manual para que la cinta se pare y se da por terminado el programa al pulsar en los dos Stop que aparecen.

Figura 8.4. Círculos llegando al sensor

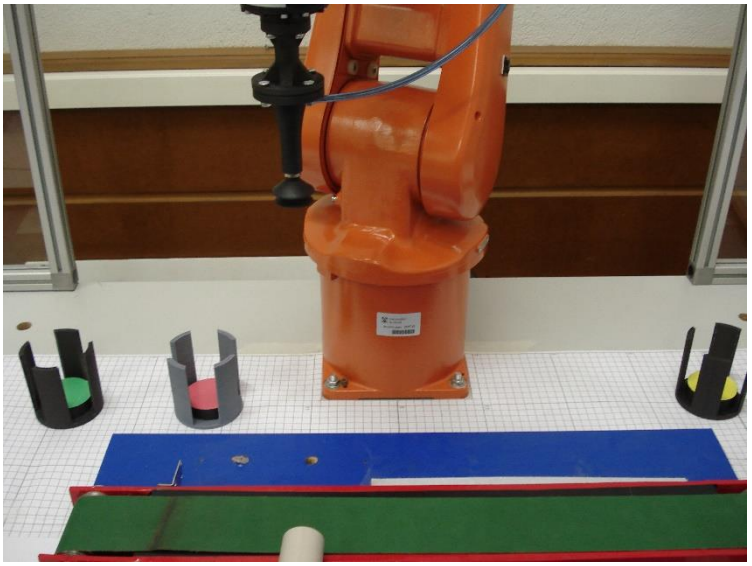


Figura 8.5. Círculos colocados

9. Pliego de condiciones

En este capítulo se detallan las características de las herramientas utilizadas.

9.1. Hardware

1. Ordenador de sobremesa AUS CM6330
    - Procesador: Intel® Core™ i7-3770S @ 3.10GHz
    - RAM: 8 GB
    - Tarjeta gráfica : NVIDIA GeForce 640M
2. Robot Industrial ABB-IRB120
    - Peso: 25kg
    - Altura: 580 mm
    - Carga soportada: 3kg (hasta 4 Kg con muñeca vertical)
    - Controlador: IRC5 Compact
3. Cámara USB Logitech C310
    - Calidad de imagen en programa 640x480 px
    - Enfoque y apertura automáticos.
4. Sistema neumático de succión.
5. Maqueta de un sistema de identificación y clasificación industrial.
6. Sensor E2K-C25MF1.

9.2. Software

1. Windows 8.1 © Microsoft Corporation.
2. ABB RobotStudio

3. Microsoft Visual C# Express

4. Microsoft Office Home and Student 2010

5. ColorPix

10. Planos

En este capítulo se detallarán las funciones más importantes implementadas en la aplicación y su flujo de trabajo. Las funciones no están completas, únicamente se detallan las instrucciones básicas.

10.1. Obtención de imágenes mediante cámara USB

```
Private void ButtonCapture ()
[…]
Capture.start();


[…]

private void
ProcesarFrame(object sender,
EventArgs e)

 […]

imageBox1.Image =
captura.RetrieveBgrFrame();
return;

[…]
```

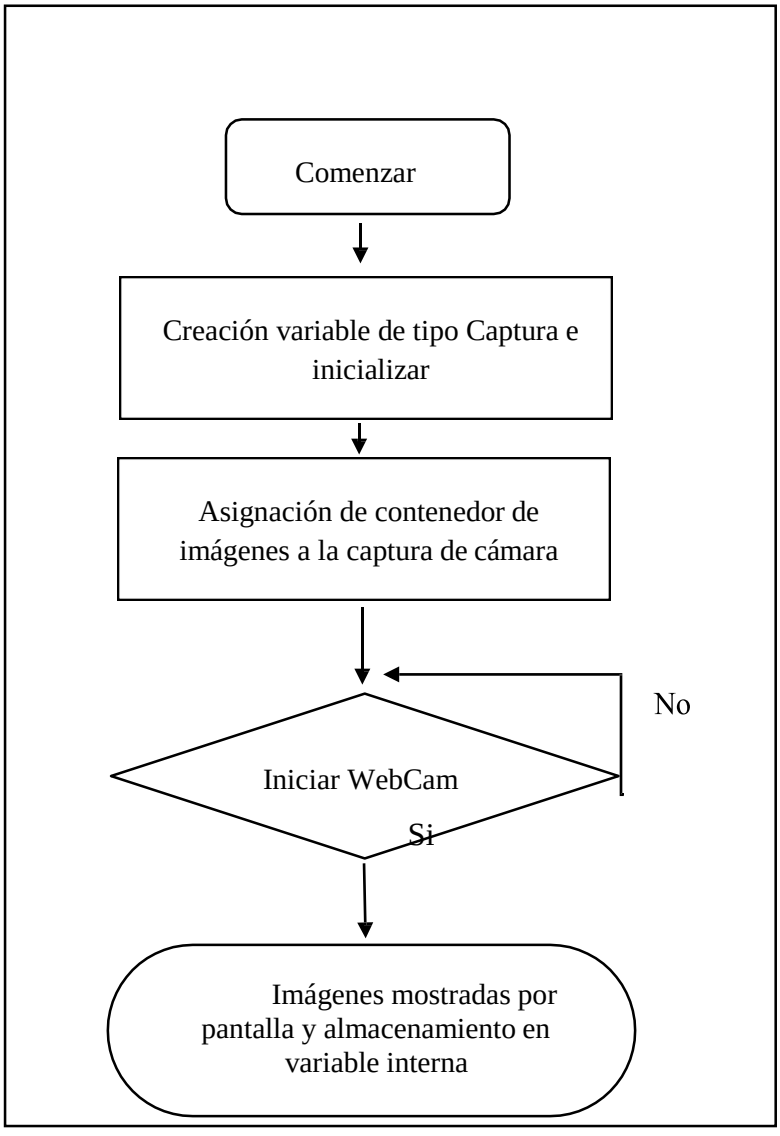


Figura 10.1. Diagrama de flujo – Iniciar captura Cámara USB

10.2. Detección de círculos en modo automático

```
if ((circle.Center.Y < py2 && circle.Center.Y > py1) && (circle.Center.X < px2
        && circle.Center.X > px1))
                        {
                                frame2.Draw(circle, new Bgr(0, 255, 0), 3);
                                frame2.Draw("C1", ref f, new
Point((int)circle.Center.X - 10, (int)circle.Center.Y + 10), new Bgr(0, 255,
0));
                                arrayCIRCLE[i_circles] = circle;
                                arrayHEIGHTS[i_circles] = color1H;
                                circulo_color1 = true;
                                circulo_detectado = true;


                                AuxDatagram3(0);
                                circles = 1;
                                if (Mover_Bolas == true &&
Procesamiento_Imagen>405)
                                {
                                int CX;
                                int CY;
                                int CZ;


                                byte i = 0;

                                for (i = 0; i < circles; i++)
                                {
                                    CZ = arrayHEIGHTS[i];

                              CX = (int)Math.Round((((arrayCIRCLE[i].Center.X-
        px1) * (my2 - my1) / (px2 - px1)) + my1) * (1000 - CZ) / 1000, 0);

                              CY = (int)Math.Round(((arrayCIRCLE[i].Center.Y -
py1) * (mx2 - mx1) / (py2 - py1)) + mx1, 0);

                              CY = (int)(mx2 - (mx2 - CY) * (1000 - CZ) /
1000);

                                    Robot2Tin(CX, CY, CZ, i);

                                      }
                                 Procesamiento_Imagen = 0;
                            }
                        }
```

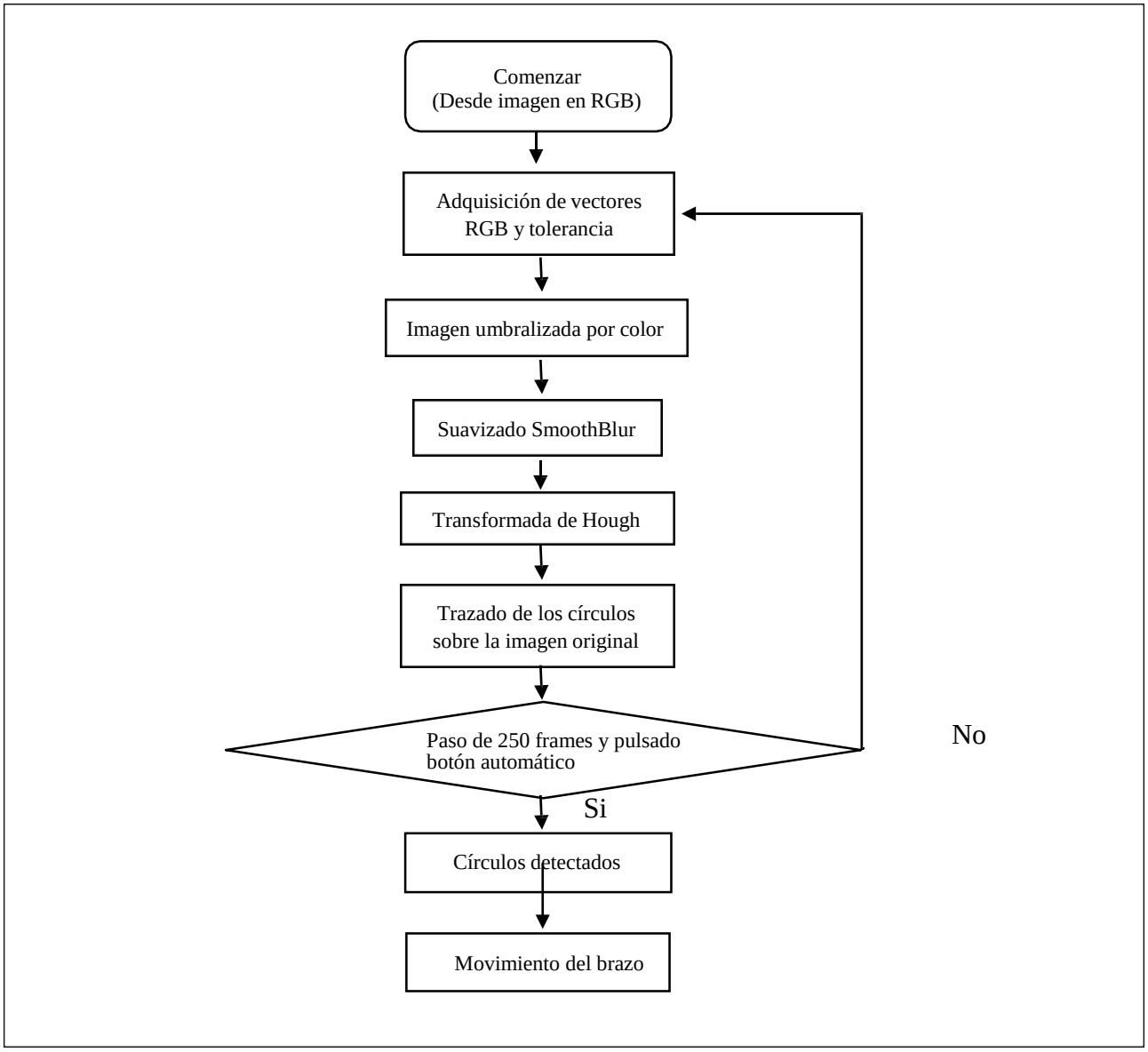


Figura 10.2. Diagrama de flujo – Detección de círculos en modo automático

10.3. Detección de círculos en modo manual

```
[…]
int color1B = Convert.ToInt32(Color1B.Text);
int color1G = Convert.ToInt32(Color1G.Text);
int color1R = Convert.ToInt32(Color1R.Text);
int color1T = Convert.ToInt32(Color1T.Text);
[…]
Image<Gray, Byte> Color1Image = frame.InRange(new Bgr(color1B - color1T,
color1G - color1T, color1R - color1T), new Bgr(color1B + color1T, color1G +
color1T, color1R + color1T));

Color1Image = Color1Image.SmoothBlur(5, 1);

CircleF[] circlesColor1 = Color1Image.HoughCircles(
                cannyThreshold,
                circleAccumulatorThreshold,
                3.0,
                50, //min distance
                10, //min radius
                100 //max radius
                )[0];

foreach (CircleF circle in circlesColor1)
{
   frame2.Draw(circle, new Bgr(0, 255, 0), 3);
}
[…]
```

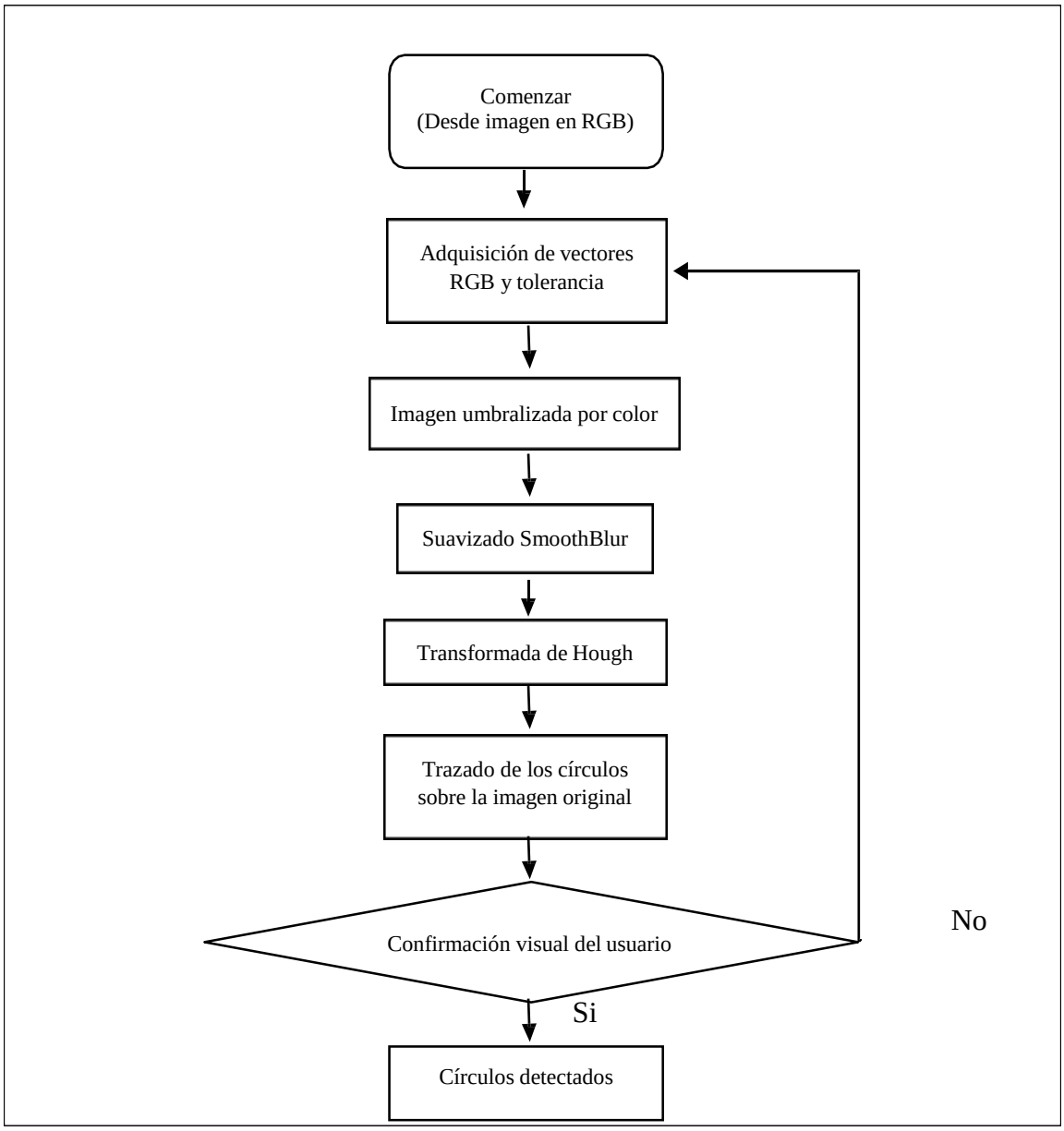


Figura 10.3. Diagrama de flujo – Detección de círculos modo manual.

10.4. Reconocimiento del espacio de trabajo

En este apartado se parte de una detección de círculos similar a la vista anteriormente. Se pasará ahora a describir como se reconoce el espacio de trabajo mediante las dos referencias (círculos) ya detectados previamente.

```
[…]
bool firstcalib=true;

foreach (CircleF circle in circlesColorx)
 {
   frame2.Draw(circle, new Bgr(0, 0, 255), 2);
   frame2.Draw("X", ref f, new Point((int)circle.Center.X - 10,
   (int)circle.Center.Y + 10), new Bgr(0, 0, 255));

    if (circle.Center.X < px1 || firstcalib == true)
    {
       px2=px1;
       py2=py1;

       px1 = (int)Math.Round(circle.Center.X, 0);
       py1 = (int)Math.Round(circle.Center.Y, 0);

       if (firstcalib == false)
       {
         calibrated = true;
         Rectangle zonework = new Rectangle(px1, py1, px2 - px1, py2 - py1);
         frame2.Draw(zonework, new Bgr(10, 10, 10), 3);
        }

        firstcalib = false;

      }

      else
      {
         px2 = (int)Math.Round(circle.Center.X, 0);
         py2 = (int)Math.Round(circle.Center.Y, 0);

         calibrated = true;
         Rectangle zonework = new Rectangle(px1, py1, px2 - px1, py2 - py1);
         frame2.Draw(zonework, new Bgr(0, 0, 255), 2);
       }
```

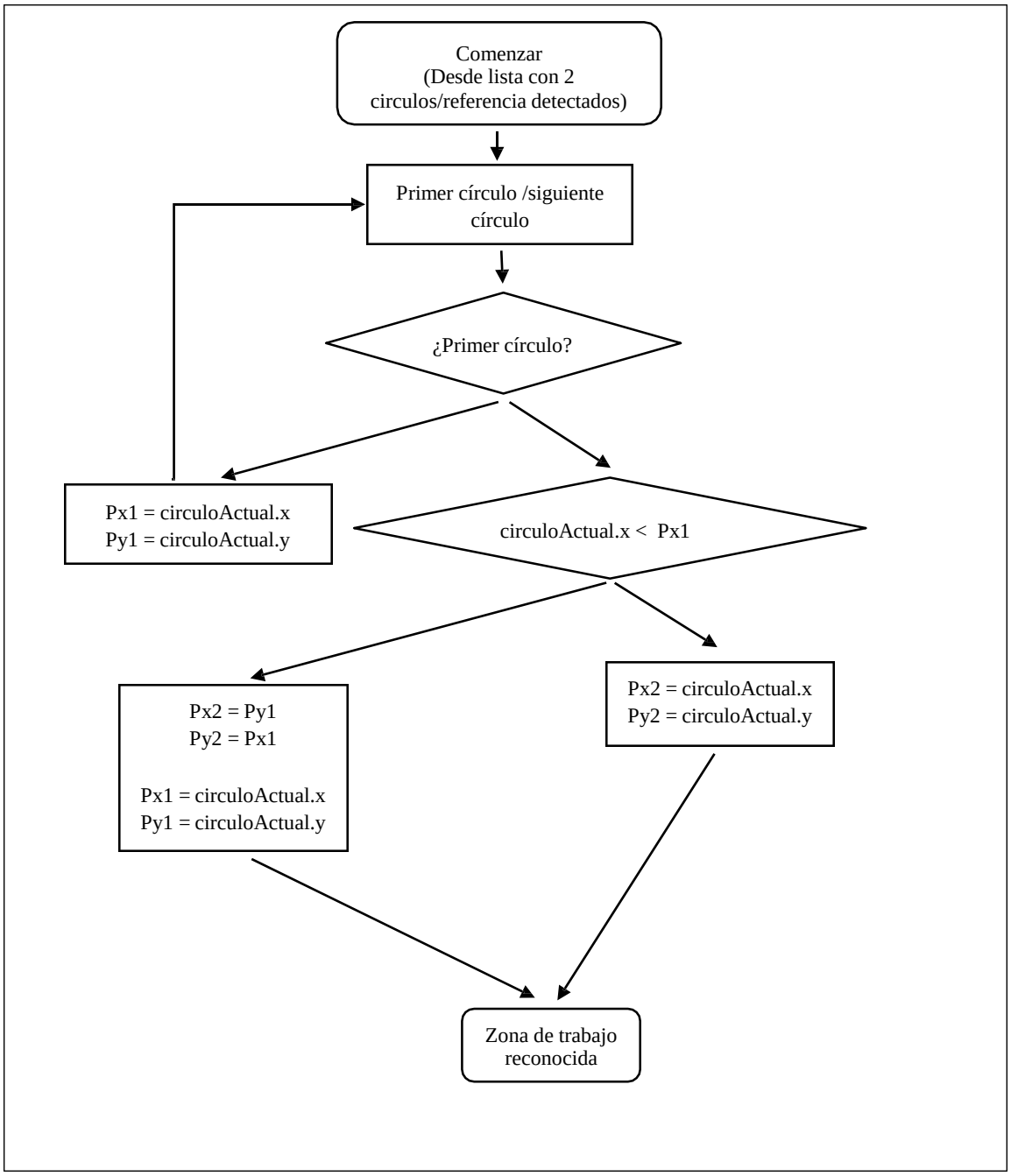


Figura 10.4. Diagrama de flujo – Reconocimiento de referencias

10.5. Reconocimiento del espacio de trabajo por defecto

En este apartado se implementan unas referencias incluidas en un fichero de texto llamado datosdefecto para ser usadas al iniciar el programa de forma predeterminada.

```
// ---------------------------------------- DATOS CALIBRACION POR DEFECTO
                string linea;
                int counter = 0;
                inicio = true;
                if (inicio == true && primeravez == true)
                    {
                    System.IO.StreamReader file = new
System.IO.StreamReader(@"C:\Users\W7\Desktop\AndresSenen\datosdefecto.txt");
                    while ((linea = file.ReadLine()) != null)
                    {
                        counter++;
                        if (counter == 1)
                        {
                           px1 = Convert.ToInt16(linea);
                        }
                        if (counter == 2)
                        {
                            px2 = Convert.ToInt16(linea);
                        }
                        if (counter == 3)
                        {
                            py1 = Convert.ToInt16(linea);
                        }
                        if (counter == 4)
                        {
                            py2 = Convert.ToInt16(linea);
                        }
                    }

                    file.Close();
                    inicio = false;
                    primeravez = false;

                    calibrated = true;
                    Rectangle zonework = new Rectangle(px1, py1, px2 - px1,
py2 - py1);
                    frame2.Draw(zonework, new Bgr(10, 10, 10), 2);

             frame2.Draw("px1 : " + px1.ToString(), ref f, new Point(470,
40), new Bgr(0, 0, 200));
             frame2.Draw("py1 : " + py1.ToString(), ref f, new Point(470,
70), new Bgr(0, 0, 200));
             frame2.Draw("px2 : " + px2.ToString(), ref f, new Point(470,
400), new Bgr(0, 0, 200));
             frame2.Draw("py2 : " + py2.ToString(), ref f, new Point(470,
440), new Bgr(0, 0, 200));
                    captureImageBox.Image = frame2;
                    photoMod = frame2.Clone();
                }
       //-----------------------------------FIN DATOS POR DEFECTO
```

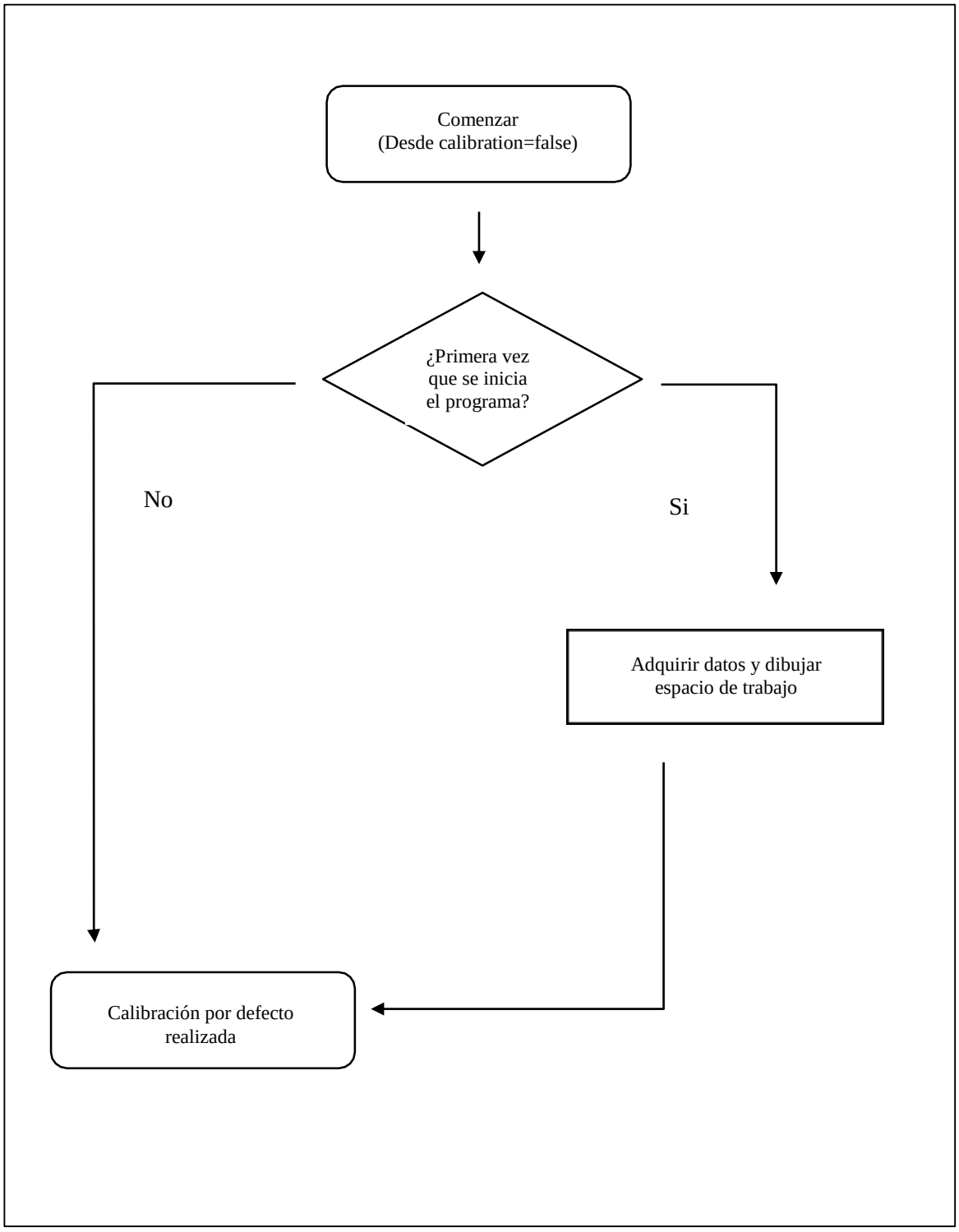


Figura 10.5. Diagrama de flujo – Calibración por defecto.

10.6. Guardar datos obtenidos en la calibración manual

```
// ---------------------------------------- DATOS CALIBRACION POR DEFECTO
                string linea;
                int counter = 0;
                inicio = true;
                if (inicio == true && primeravez == true)
                    {
                    System.IO.StreamReader file = new
System.IO.StreamReader(@"C:\Users\W7\Desktop\AndresSenen\datosdefecto.txt");
                    while ((linea = file.ReadLine()) != null)
                    {
                        counter++;
                        if (counter == 1)
                        {
                           px1 = Convert.ToInt16(linea);
                        }
                        if (counter == 2)
                        {
                            px2 = Convert.ToInt16(linea);
                        }
                        if (counter == 3)
                        {
                            py1 = Convert.ToInt16(linea);
                        }
                        if (counter == 4)
                        {
                            py2 = Convert.ToInt16(linea);
                        }
                    }

                    file.Close();
                    inicio = false;
                    primeravez = false;

                    calibrated = true;
                    Rectangle zonework = new Rectangle(px1, py1, px2 - px1,
py2 - py1);
                    frame2.Draw(zonework, new Bgr(10, 10, 10), 2);

             frame2.Draw("px1 : " + px1.ToString(), ref f, new Point(470,
40), new Bgr(0, 0, 200));
             frame2.Draw("py1 : " + py1.ToString(), ref f, new Point(470,
70), new Bgr(0, 0, 200));
             frame2.Draw("px2 : " + px2.ToString(), ref f, new Point(470,
400), new Bgr(0, 0, 200));
             frame2.Draw("py2 : " + py2.ToString(), ref f, new Point(470,
440), new Bgr(0, 0, 200));
                    captureImageBox.Image = frame2;
                    photoMod = frame2.Clone();
                }
       //-------------------------------------FIN DATOS POR DEFECTO
```

10.7.Envío de posiciones al brazo robótico

En este apartado se resumirá el envío de posiciones al brazo robótico así como el lugar de paletización de cada color.

```
      private void Robot2Tin(int cx, int cy, int cz, int palletPos)
              {
                  int raisedHeight;

                  if (defaultRaisedHeight == true) raisedHeight = cz + 200;
                  else raisedHeight = defaultRaisedHeightCircle;

                  if (defaultFinalObjectPos == true)
                  {
                      if (circulo_color1 == true)
                      {
                          AuxDatagram3(0);

                          PosAndRot(200, 0, 300, 0, 180, 0);   //POSICION INICIAL


                          PosAndRot(cy, cx, cz + 70, 0, 180, 0);   //APROXIMACION
Al BOTE

                          PosAndRot(cy, cx, cz, 0, 180, 0);     //POSICION BOTE


                          AuxDatagram(0);                              //attacher
      ventosa  >>>--
                          Thread.Sleep(500);

                          PosAndRot(200, -100, raisedHeight, 0, 180, 0);
//levantar bote


                          PosAndRot(100, -220, raisedHeight, 0, 180, 0);
                          PosAndRot(100, -380, 150, 0, 180, 0);
                          PosAndRot(100, -380, 100, 0, 180, 0);
                          AuxDatagram(1);                              //detacher
      ventosa    --<<<
                          Thread.Sleep(500);



                          PosAndRot(200, 0, raisedHeight, 0, 180, 0);   //POSICION
INICIAL
                          AuxDatagram3(1);
                          Thread.Sleep(2000);

                          circulo_color1 = false;


                      }
```

```
            else if (circulo_color2 == true)
               {
                   AuxDatagram3(0);

                   PosAndRot(200, 0, 300, 0, 180, 0);   //POSICION INICIAL


                   PosAndRot(cy, cx, cz + 70, 0, 180, 0);   //APROXIMACION Al
BOTE

                   PosAndRot(cy, cx, cz, 0, 180, 0);     //POSICION BOTE


                   AuxDatagram(0);           //attacher ventosa       >>>--
                   Thread.Sleep(500);

                   PosAndRot(200, -100, raisedHeight, 0, 180, 0);
//levantar bote


                   PosAndRot(100, -220, raisedHeight, 0, 180, 0);
                   PosAndRot(100, -220, 150, 0, 180, 0);
                   PosAndRot(100, -220, 100, 0, 180, 0);

                   AuxDatagram(1);             //detacher ventosa       --<<<

                   Thread.Sleep(500);


                   PosAndRot(200, 0, raisedHeight, 0, 180, 0);   //POSICION
INICIAL
                   AuxDatagram3(1);
                   Thread.Sleep(2000);

                   circulo_color2 = false;

               }
```

```
                else if (circulo_color3 == true)
                {
                   AuxDatagram3(0);

                    PosAndRot(200, 0, 300, 0, 180, 0);   //POSICION INICIAL


                    PosAndRot(cy, cx, cz + 70, 0, 180, 0);   //APROXIMACION Al
BOTE

                    PosAndRot(cy, cx, cz, 0, 180, 0);     //POSICION BOTE


                    AuxDatagram(0);                  //attacher ventosa  >>>--
                    Thread.Sleep(500);

                    PosAndRot(200, -100, raisedHeight, 0, 180, 0);
//levantar bote


                    PosAndRot(100, 220, raisedHeight, 0, 180, 0);
                    PosAndRot(100, 380, 150, 0, 180, 0);
                     PosAndRot(100, 380, 100, 0, 180, 0);
AuxDatagram(1);                            //detacher ventosa    --<<<

                    Thread.Sleep(500);



                    PosAndRot(200, 0, raisedHeight, 0, 180, 0);   //POSICION
INICIAL
                    AuxDatagram3(1);
                    Thread.Sleep(2000);

                    circulo_color3 = false;

                }




                   }
```

La función PosAndRot en el código anterior es una modificación de la función de envío de posición y rotación al robot de [9].

11.Presupuesto

Este capítulo proporciona información detallada acerca de los costes teóricos del desarrollo del proyecto, incluyendo los costes de materiales y las tasas profesionales.

11.1.Costes materiales

Objeto		***Cantidad***	***Precio unidad (€)***	***Precio total (€)***
Hardware	PC Asus	1	800	800
	ABB-IRB120	1	10954,00	10954,00
	Cámara USB Logitech C310	1	40,00	40,00
	Maqueta	1	0,00	0,00
	Sensor E2K-C25MF1	1	163,80	163,80
Software	Microsoft Windows 8.1	1	0,00	0,00
	RobotStudio	1	0,00	0,00
	Microsoft Visual C# Express	1	0,00	0,00
	ColorPix	1	0,00	0,00
	Microsoft Office Home and Student 2010	1	100,00	100,00
TOTAL				11817,80

Tabla 11.1 Costes materiales (hardware y software) sin IVA.

11.2. Tasas profesionales

Las tasas profesionales incluyen los costes de realización del proyecto, entendiéndose estos costes como el tiempo dedicado a su desarrollo.

Objeto	*Tiempo (meses)*	*Coste unidad (€)*	*Coste total (€)*
Ingeniería	4	1400 €	5600 €
Escritura	1	800 €	800 €
TOTAL			6400 €

Tabla 11.2. Costes profesionales sin IVA.

11.3. Costes totales

Los costes totales del Proyecto han sido obtenidos sumando los costes materiales y profesionales aplicándose el IVA. Además se ha de tener en cuenta los costes de impresión y encuadernación.

Objeto		*Costes totales*
Costes materiales		1187,80 €
Costes profesionales		6400,00 €
Proyecto	Impresión	40 €
	Encuadernación	100 €
Subtotal		7727,8 €
IVA (21%)		1622,84 €
TOTAL		9350,64 €

Tabla 11.3. Costes totales con IVA.

Los costes totales de la implementación del proyecto ascienden a nueve mil trescientos cincuenta con sesenta y cuatro céntimos.

12. Conclusiones

Este proyecto se ha basado en la comunicación entre una maqueta que hacía las veces de cinta transportadora y un sistema de visión acoplado al brazo robot IRB-120.
Se ha intentado obtener algo parecido a lo que se puede encontrar en una empresa de ingeniería o en una cadena de montaje donde el brazo robot interacciona con los elementos sin participación alguna de personas tal y como hace el Modo Automático de este proyecto.

Al establecer la comunicación entre la cinta, el sensor y el robot mediante RAPID se permite a otros alumnos interactuar con estos elementos en futuros proyectos para realizar otras aplicaciones, aplicaciones parecidas e incluso mejorar la comunicación o incluir otros elementos para hacer un sistema aún más complejo haciendo que este proyecto sea una especie de proyecto base para ellos.

13.Bibliografía

[1] Página web http://new.abb.com

[2] "Enciclopedia del lenguaje C#" Francisco Javier Ceballos

[3] "Implementación de un sistema de visión para control del brazo robot IRB120" de Álvaro Fernández Expósito

[4] Página web http://www.ia.omron.com/product/item/e2k_5019r/

[5] "Diseño, construcción y programación de una maqueta de un sistema de identificación y clasificación industrial" de Diego Sánchez Navarro.

[6] Página web http://www.infoplc.net/foro/showthread.php?17547-Entradas-Salidas-del-robot-ABB-IRC5-compact

[7] Página web http://www.youtube.com/watch?v=vdjoutNR2DQ

[8] Página web http://www.init.uji.es/index.php?option=com_content&view=article&id=81:ensambladodeteselas&catid=27:colaboracionempresas&Itemid=49&lang=es

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