

ROBINT

ROBOTIZACIÓN INTELIGENTE

ENTREGABLE E3.1

“Prototipos de sistemas de análisis de entorno”

CENTROS DE INVESTIGACIÓN:



1 Contenido

2	Introducción	3
3	Análisis de sistemas de visión.	3
3.1	Análisis de sistemas de visión comerciales	3
3.2	Sistema de visión desarrollado a medida.	4
4	Selección/desarrollo de sistema de visión.	5
4.1	Aplicaciones sector del juguete	6
4.2	Aplicaciones sector del calzado	11
4.3	Aplicaciones comunes al sector del calzado y juguete.....	12
4.4	Integración de sistemas HRC	24

2 Introducción

Este documento resume el proceso y la toma de decisiones llevadas a cabo por el consorcio para el desarrollo de los elementos y técnicas, que permitan la implementación eficiente de sistemas robotizados en colaboración con humanos de modo que sea viable y segura esta colaboración: viable en la medida que sea posible garantizar los tiempos operativos y segura en la medida que no implique riesgo para los operadores humanos.

3 Análisis de sistemas de visión.

Resulta imprescindible analizar exhaustivamente diferentes métodos de captación del entorno, ya sea mediante sistemas comerciales de visión dotados de inteligencia aplicando software de reconocimiento, así como sistemas de visión sensorizados que capten el entorno mediante sensores aportando soluciones que dependan mas del hardware.

3.1 Análisis de sistemas de visión comerciales

Las características principales de los sistemas de visión comerciales son:

- ✓ Sistemas poco flexibles.
- ✓ Costes elevados.
- ✓ Alta fiabilidad.
- ✓ Robustos.



Figura 1: Sistemas de visión comerciales.

A pesar de que estos sistemas son muy robustos y ofrecen una alta fiabilidad dentro del ámbito de aplicación para el que están orientados, nos encontramos con que los equipos de visión comerciales son sistemas cerrados orientados a aplicaciones concretas que tienen una demanda elevada en la industria. Por lo que a la hora de seleccionar un equipo de

visión para la implementación de aplicaciones que no son habituales en la industria es complicado obtener soluciones que se adapten al 100%. Además, estos sistemas no permiten realizar modificaciones que permitan adaptar estos sensores a procesos para los que no se han considerado en su fase de desarrollo, ya que son sistemas que dependen mucho del hardware. En cuanto al software, estos sistemas incluyen software propietario orientado a la configuración del equipo, y no al manejo a bajo nivel. Y no se facilita el acceder al código fuente para realizar adaptaciones que permitan integrar estos sistemas en aplicaciones desarrolladas a medida.

3.2 Sistema de visión desarrollado a medida.

En cuanto a los sistemas de visión desarrollados a medida, nos encontramos con las siguientes características:

- ✓ Sistema flexible.
- ✓ Costes de sistema inferior.
- ✓ Uso de librerías Open Source.
- ✓ Basado en PC o PLC.



Figura 2: Componentes utilizados para desarrollo de sistemas de visión.

Los sistemas de visión desarrollados a medida tienen como principal característica la capacidad de adaptación al proceso para el cual se diseñan. Esta flexibilidad en la adaptación se consigue partiendo de sensores o cámaras de visión alrededor de los cuales se desarrolla la programación necesaria para procesar las imágenes captadas y extraer la información deseada. En este caso se consigue reducir el coste inicial de los equipos, pero se debe asumir el coste de desarrollo para obtener un prototipo inicial. Además, se debe tener en cuenta que estos sistemas inicialmente no ofrecen la robustez que puede ofrecer un equipo comercial. A pesar de esto, en este caso es necesario desarrollar sistemas de visión orientados a aplicaciones que no están cubiertas 100% por los dispositivos comerciales por lo que se ha optado por el desarrollo de sistemas de visión basados en software libre y entornos PC.

Para este fin existen herramientas gratuitas o de pago que pueden ser utilizadas para el desarrollo de sistemas de visión a medida. Es necesario el uso de un entorno de programación que permite implementar los algoritmos de control y gestión de información y librerías de visión que permiten manejar y extraer la información obtenida de los sensores de visión.

A la hora de seleccionar el entorno de programación se debe tener en cuenta el lenguaje de programación que se utilizará para desarrollar el sistema. En este caso, la aplicación se desarrollará empleando el lenguaje de programación C# que es un lenguaje de programación orientado a objetos, desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. El entorno de programación para este lenguaje es el Visual Studio, también desarrollado por Microsoft y del cual se utilizará la versión 2015.

En cuanto a las librerías de visión, son muchas las que se pueden utilizar para la implementación de sistemas de visión artificial, como por ejemplo Torch3vision, VLX, OpenCV, etc. En este caso se ha seleccionado una adaptación de OpenCV que permite su uso en el entorno de programación seleccionado (Visual Studio 2015) y están compiladas para el lenguaje de programación C#, **Emgu CV**. Estas librerías están distribuidas bajo el concepto de Open Source lo que permite su uso de forma gratuita.

4 Selección/desarrollo de sistema de visión.

Una vez analizados los tipos de sistemas de visión disponibles y tras seleccionar el desarrollo propio como solución a la implementación de un sistema que permita integrar robot en entornos seguros de colaboración humano-robot, se ha procedido a la selección de los elementos hardware necesarios para el desarrollo. En este caso, los elementos fundamentales a tener en cuenta son el sistema de control y el sensor de visión artificial utilizado.

Para la selección de los sistemas de control, se plantea dos posibilidades:

- Uso de PLC con capacidad de procesamiento avanzado ya que integran funciones de tiempo real y opciones de comunicación adecuados para conectar con robots industriales. Estos sistemas además permiten el control en tiempo real y son dispositivos fiables y robustos. Aunque no integran entorno gráfico.
- Uso de PCs convencionales, ya que el coste de estos dispositivos es bastante más asequible que la opción anterior y son menos rígidos a la hora de realizar la implementación. El inconveniente de estos sistemas es que no trabajan en tiempo real, por lo que se deben implementar algoritmos o sistemas de comunicaciones que gestionen la pérdida o retraso en la entrega de información al sistema de control del robot.

Tras evaluar las opciones anteriores, se opta por el diseño utilizando PCs convencionales debido a que esta decisión permite abaratar los costes del sistema final. Otro aspecto que se ha considerado fundamental es disponer de un entorno gráfico, que permita la visualización del entorno de trabajo y verificar que el sistema trabaja de forma correcta en todo momento. Por último, la elección de desarrollo del sistema basado en PC permite implementar un sistema de visión flexible y fácilmente adaptable a distintos sistemas robóticos, por lo que se garantiza su integración con robots de distintas marcas realizando pequeñas adaptaciones en la implementación del protocolo de comunicaciones. Esto se

debe a que los fabricantes de robots pueden implementar protocolos de comunicaciones propietarios o protocolos estándar.

En cuanto a la selección de sensores de visión, se planteó el uso de sistemas basados en cámaras industriales y cámaras de bajo coste. Pero finalmente se llegó a la conclusión de que este no era el factor más restrictivo. En este caso los principales datos técnicos a tener en cuenta son la resolución y el formato de la imagen.

Dado que los modelos de cámaras utilizados por cada uno de los institutos varía debido a disponibilidad de cada centro, se plantea el desarrollo de un sistema de visión basado en la misma tecnología, pero con las adaptaciones necesarias para el correcto funcionamiento en cada una de las plantas. Además, en el caso de la planta piloto del sector del juguete se utiliza un robot de KUKA en el que el sistema de comunicaciones que se dispone se basa en comunicación Ethernet bajo el protocolo **UDP/IP**. Este sistema requiere de la implementación de un sistema de visión que se comunique con el mismo protocolo de comunicaciones. Mientras que en el sector del calzado se utiliza un robot de COMAU cuyo sistema de comunicaciones esta también basado en Ethernet, pero bajo el protocolo de comunicaciones **TCP/IP**.

4.1 Aplicaciones sector del juguete

Inicialmente se ha desarrollado un sistema de visión basado en las librerías de visión de OpenCV utilizando el lenguaje de programación C++. Se decidió utilizar esta versión de las librerías de visión debido a que está más extendida que la versión para C# (Emgu CV)

En esta versión inicial se implementó un entorno gráfico compuesto por:

- **Ventana de visualización de estado actual:** imagen de la izquierda de la **¡Error! No se encuentra el origen de la referencia.**Figura 3. Permite monitorizar el estado en tiempo de ejecución del entorno de trabajo del robot.
- **Ventana de visualización de entorno Inicial:** imagen de la derecha de la **¡Error! No se encuentra el origen de la referencia.**Figura 3. Muestra una imagen con la captura inicial del estado de la planta piloto. En este estado la planta piloto puede funcionar sin restricciones de velocidad.
- **Barras de configuración:** estas barras se muestran debajo de las imágenes y permiten configurar el tamaño y la localización de las zonas de detección.

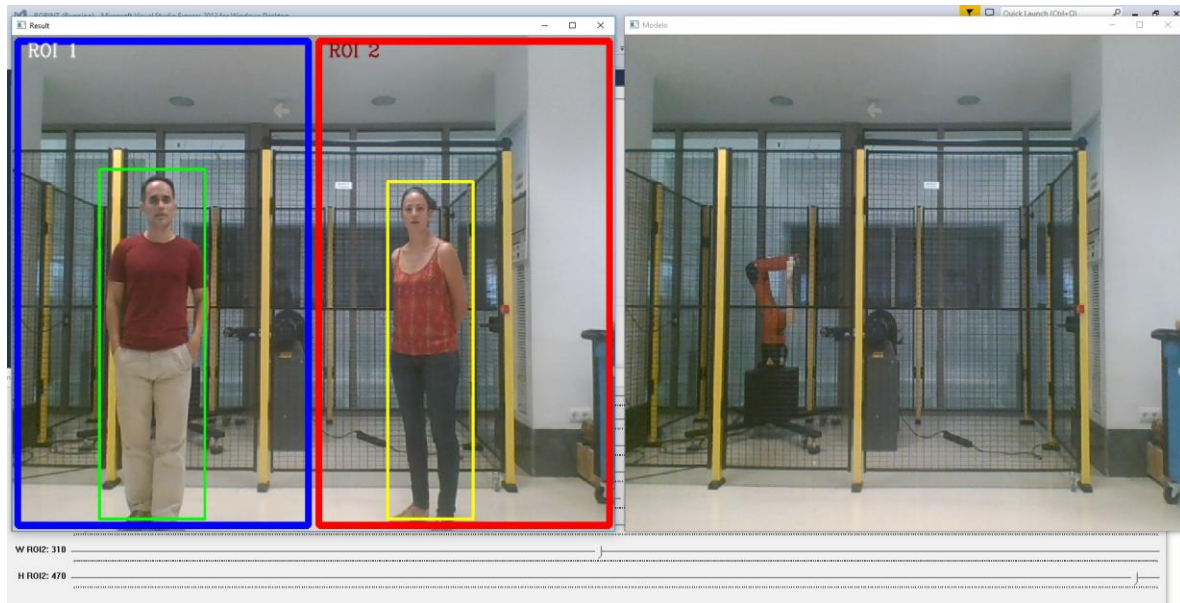


Figura 3: Versión inicial sistema de visión.

Los visualizadores permiten observar y comparar el estado del entorno de trabajo con respecto a un estado inicial deseado. Para el desarrollo de esta interfaz se implementaron los algoritmos de control necesarios para monitorizar la zona de trabajo mediante la adquisición de datos del sensor de imágenes utilizado. En primer lugar, se adquiere una imagen del estado de la planta en el que no hay objetos ajenos a esta y en el que el robot puede funcionar sin restricciones de velocidad. Esta imagen se guarda para compararla posteriormente con los datos suministrados en tiempo de ejecución por el sensor.

La comparación se realiza estableciendo regiones de interés (ROI 1 y ROI 2 en la imagen de la izquierda de la **¡Error! No se encuentra el origen de la referencia.**Figura 3), que delimitan dos zonas de trabajo. Cuando se detectan cambios en el entorno se marcan para identificar de forma visual que se ha introducido un objeto o una persona en la zona de trabajo. La implementación de detección se ha realizado mediante la implementación de distintos filtros que permiten detectar personas u objetos que no se encontraban inicialmente en el área de trabajo.

Tras validar la viabilidad técnica de la implementación se realizó la migración de código desarrollado al lenguaje de programación C#. Esta migración se realiza debido a que el entorno de Visual Studio para C++ no orientado al desarrollo de entornos gráficos, y la implementación de estas es más compleja que en C#.

Al migrar el código de la interfaz al lenguaje de programación C#, se ha creado una interfaz más compleja que permite configurar y modificar los parámetros de monitorización en función de los requisitos del sistema en el que se desee utilizar el desarrollo de detección. La interfaz está compuesta por:

- **Ventana de visualización de entorno Inicial:** imagen de la izquierda en la **¡Error! No se encuentra el origen de la referencia.**Figura 4. Muestra una imagen con la captura inicial del estado de la planta piloto.
- **Ventana de visualización de estado actual:** imagen de la derecha de la **¡Error! No se encuentra el origen de la referencia.**Figura 4. Permite monitorizar el estado en tiempo de ejecución del entorno de trabajo del robot.

- **Área de configuración:** permite configurar las áreas de detección e iniciar la detección. En esta área se encuentran los siguientes controles:
 - ✓ **Toma de captura:** al hacer clic en este control se realiza una captura del entorno de trabajo. Esta función debe ejecutarse inicialmente para establecer la situación inicial del entorno de trabajo.
 - ✓ **Adquirir zona:** Este botón se debe pulsar mediante un clic para configurar regiones de interés en la zona de detección. Una vez pulsado se deben marcar mediante clic dos puntos diagonalmente opuestos sobre la imagen de la ventana de visualización de estado actual para establecer la región. Se pueden establecer un máximo de tres regiones.
 - ✓ **Selector de tipo de región:** este selector permite seleccionar el tipo de región de interés antes de añadirla. Se puede seleccionar entre zona peligrosa (se envían comandos de parada al robot cuando se detectan personas en este tipo de zonas) y zonas de precaución (se envían comandos de reducción de velocidad).
 - ✓ **Añadir:** después de marcar el área de interés y seleccionar el tipo de área se debe pulsar este botón para añadir la zona de detección configurada.
 - ✓ **Visor de zonas:** en este visor se puede observar la lista de zonas configuradas en la interfaz y el tipo de zona.
 - ✓ **Eliminar:** Permite eliminar las regiones de interés. Se debe seleccionar la región a eliminar y hacer clic en el botón.
 - ✓ **Detener/Activar detección:** este botón permite poner en marcha o detener el sistema de detección tras realizar la configuración.

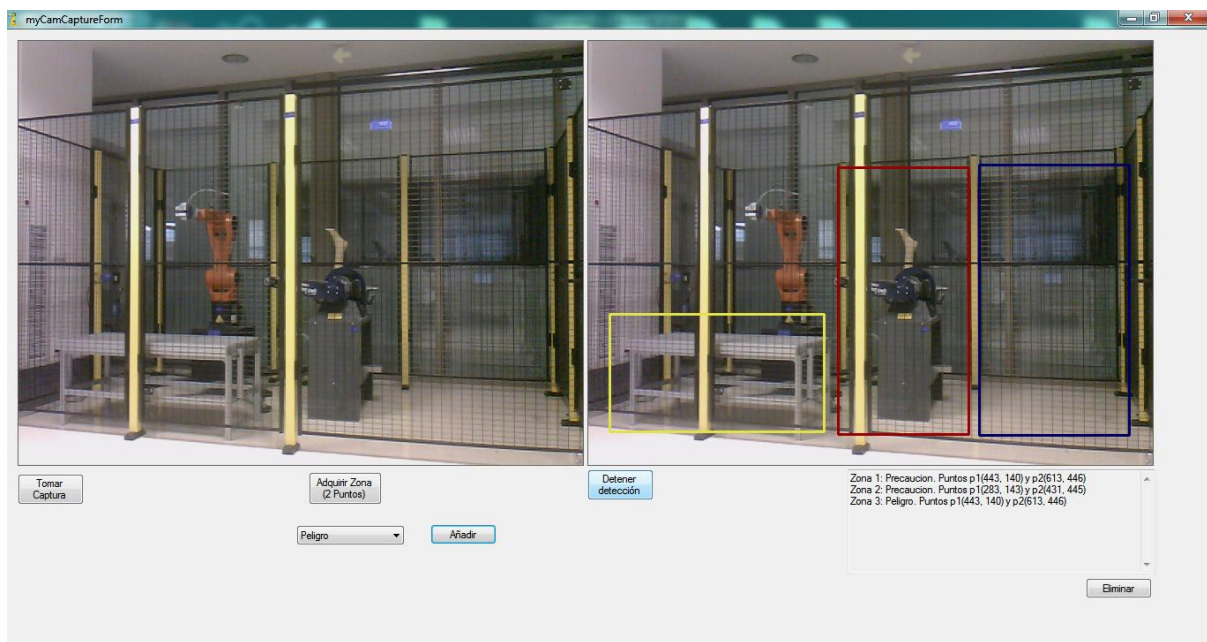


Figura 4: Interfaz sistema de visión para el sector del juguete.

Como ya se ha indicado anteriormente, esta interfaz parte del desarrollo inicial que se realizó en C++. Para realizar la adaptación se ha migrado el software de control a C# y se

han realizado los cambios oportunos para utilizar las librerías de Emgu CV. Estas librerías son una adaptación de OpenCV para su uso bajo el lenguaje de programación C#. Tras realizar la migración del código existente para la detección de objetos en el entorno de trabajo, se ha trabajado en implementar la interfaz descrita anteriormente y que se observa en la Figura 4 .

Además, a esta se le ha implementado el protocolo de comunicaciones con el robot KUKA para el envío de tramas con formato UDP. Los datos se envían en formato XML, y están compuestos de un campo de identificación del sensor que envía la trama, un código temporal para el control de envío de datos en tiempo real y los datos de detección en cada una de las zonas configuradas según el siguiente ejemplo.

```
<Sen Type="SistemaVisión">
<Tipo 01="0" 02="0" 03="1" />
<Zona 01="0" 02="0" 03="1" />
<IPOC>123645634563</IPOC>
</Sen>
```

Este mensaje del sistema de Visión se corresponde al envío de datos de una configuración en la que se han configurado tres zonas (las dos primeras de precaución y una tercera de peligro). En la zona de peligro se ha detectado la caja de herramientas que no debería estar en el entorno de trabajo (Figura 5).

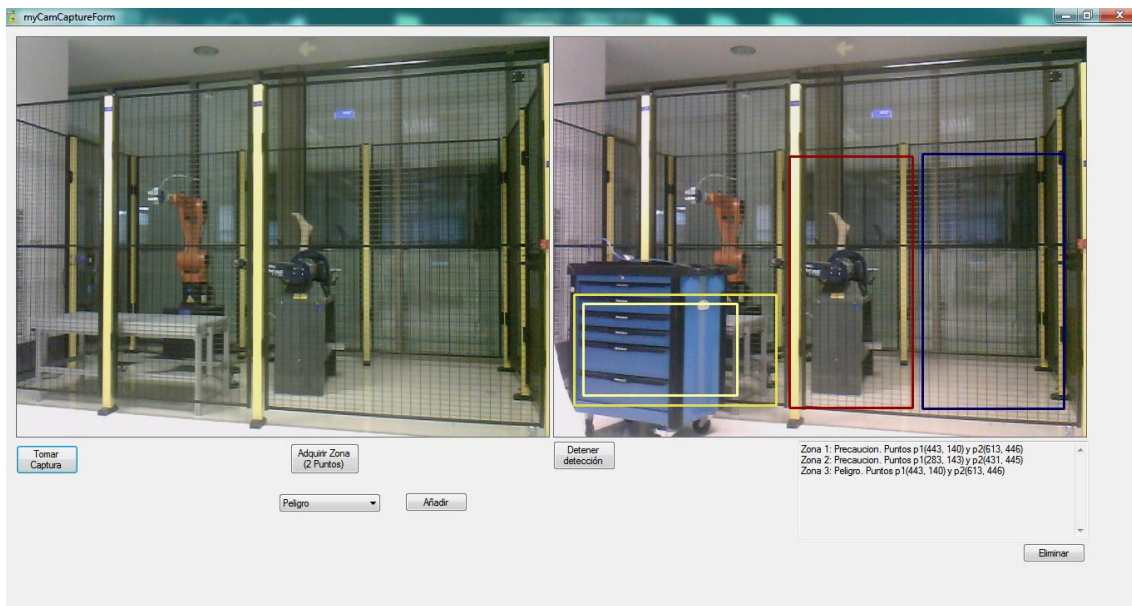


Figura 5: Detección de objeto en área de trabajo.

Una vez implementada la interfaz de detección para el sector del juguete y tras realizar las primeras pruebas de validación con el sensor seleccionado (cámara industrial Basler, Figura 6) se detectó que el área monitorizada por este sensor era inferior a la deseada. Esto se debe a que el ángulo de apertura de imagen del objetivo no permite captar toda el área de trabajo desde la distancia a la que se ha instalado. Esta distancia está limitada debido al espacio disponible en el laboratorio en el que se ha montado la planta.



Figura 6: Cámara industrial Basler.

Para solventar el problema ha sido necesario desarrollar un objetivo adecuado para la cámara que permita una apertura de ángulo de imagen superior. Esta solución permite monitorizar una región de trabajo más amplia con la cámara situada en el punto de instalación seleccionada inicialmente. En la Figura 7 se observa el objetivo utilizado finalmente.



Figura 7: Objetivo gran angular.

4.2 Aplicaciones sector del calzado

Para la creación de la interfaz se utilizó Visual Studio programando en lenguaje C#.

La interfaz cuenta con las siguientes características:

- **Ventana de imagen actual (1):** En esta ventana se muestra la secuencia de imágenes capturadas mediante la cámara a color.
- **Ventana de imagen procesada (2):** Aquí se presentan las imágenes que captura la cámara, junto con las marcas de zonas que establece el usuario. Las zonas que el sistema permite establecer son amarillas, que son zonas de precaución, en las que el robot debe reducir su velocidad y zonas rojas, que son zonas de parada, considerándose que la persona está demasiado cerca del robot.
- **Botón de inicio de captura (3):** Es el encargado de iniciar y parar la captura de imágenes mediante la cámara.
- **Sección para añadir zona de seguridad:** Esta sección está formada por diversos elementos cuya función es definir la posición y tamaño de la zona a crear, así como, el tipo de zona, es decir, si va a ser una zona de precaución o de parada.

Estos elementos son los siguientes:

- **Botón de adquisición de zona (4):** Al presionar este botón, entramos en la fase de adquisición de zona. Esto nos permite, haciendo 'click' en 2 zonas de la ventana de la derecha, establecer una zona provisional, color azul, que podrá ser añadida o no, dependiendo de si estamos de acuerdo con la selección.
- **Selector del tipo de zona (5):** Este elemento desplegable nos permite elegir el tipo de zona, precaución o parada, a establecer para la zona previamente seleccionada.
- **Botón de incorporación de zona (6):** Este botón otorga la confirmación para la creación de la zona seleccionada en la imagen, con el tipo de zona escogido en el desplegable.
- **Sección para eliminar zona de seguridad:** Esta sección está formada por diversos elementos cuya función es eliminar una zona de seguridad, que es errónea o innecesaria. Estos elementos son los siguientes:
 - **Listado de las zonas creadas (7):** Este elemento presenta todas las zonas creadas, indicado la posición en píxeles de la imagen de cada una de ellas. Seleccionando una, podremos eliminarla posteriormente.
 - **Selector de visualización (8):** Al marcar este elemento obtenemos una representación en color azul de la zona seleccionada en el listado. De esta manera, no necesitamos acordarnos de las posiciones de las zonas creadas. Así, nos aseguramos de que la zona a borrar es la deseada.
 - **Botón de eliminado de zona (9):** Este botón otorga la confirmación para eliminar la zona seleccionada en el listado.

- **Botón de inicio de detección (10):** Es el encargado de iniciar el proceso de reconocimiento de elementos en la escena. Una vez hemos creado las zonas a controlar, presionamos el botón y el sistema empieza a evaluar el entorno.

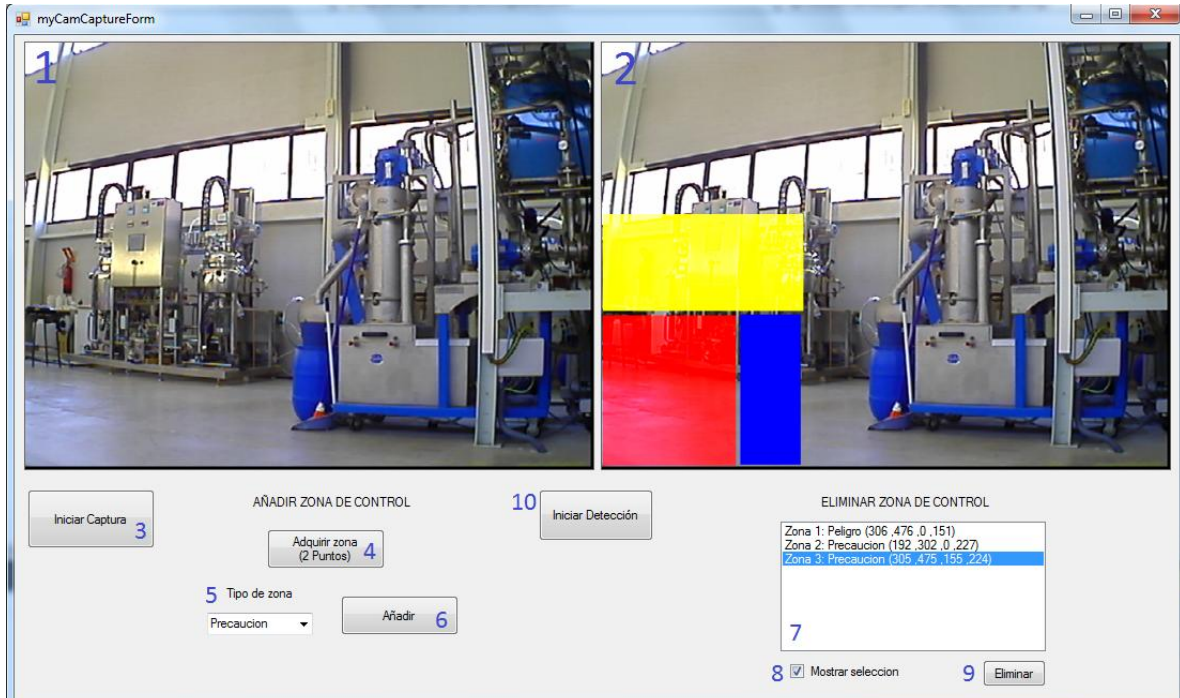


Figura 8: Interfaz de visión para el sector del calzado.

4.3 Aplicaciones comunes al sector del calzado y juguete.

Prototipo entorno colaborativo humano-robot (HRC)

Se ha desarrollado un prototipo de detección de entornos colaborativos hombre-máquina, compuesto por 3 láseres y una cámara a color que mediante un software controla la potencia de los láseres y los parámetros de la cámara para determinar variaciones en los láseres y detectar cuando se cortan por parte del operario, este software se conecta al robot y permite variar la velocidad del robot en función del haz del láser cortado.

Hardware bajo coste:

- ✓ Cámara desarrollo propio con tarjeta de control ftdi.
- ✓ Conjunto de 3 láseres R-G-B.
- ✓ Regulación independiente.

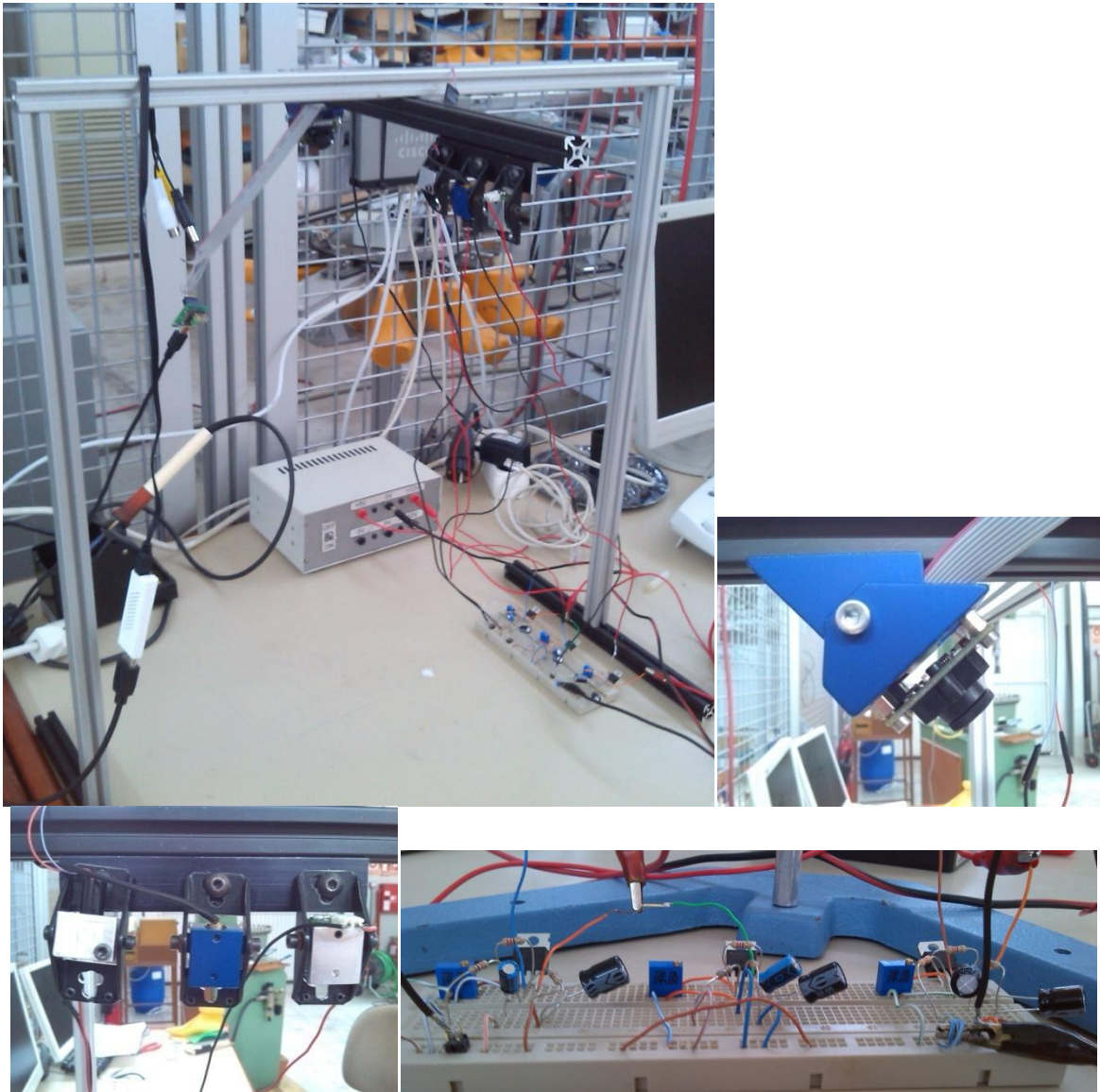


Figura 9: Prototipo 1 cámara y 3 láseres

Software:

- ✓ Control de cámara a bajo nivel.
- ✓ Control de capturadora.
- ✓ Conexión a robot.
- ✓ Control de modos de velocidad.

A continuación se describe el funcionamiento del software referente a la captura de imágenes, al tratamiento digital de la imagen necesario para la detección de los láseres y la generación de las variables necesarias para el cambio de velocidad del robot.

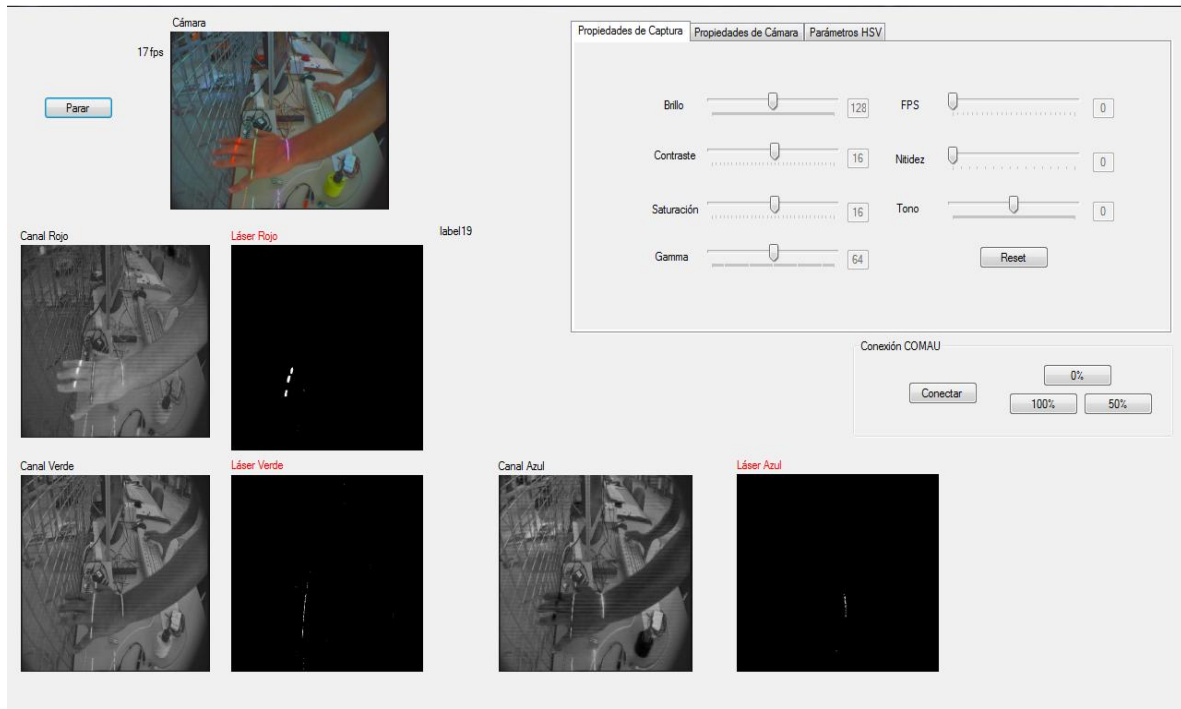


Figura 10: Interfaz usuario primer prototipo

Como se puede observar en la figura 10, en la parte superior de la interfaz de la aplicación, se encuentran un total de siete **ImageBox**, donde se muestran la imagen original capturada por la cámara, sus tres componentes de color y tres imágenes binarizadas, correspondientes a cada uno de los láseres detectados. Además, se incluye un botón de captura/parada y un contador de **fps** implementado haciendo uso de un **timer**. La función de estos **ImageBox** es solamente visual, con el objetivo de comprobar los resultados obtenidos, por lo que parte de ellos podrían ser eliminados para mejorar el rendimiento de la aplicación.

Control de parámetros, conexión y selección de modo

En la parte inferior izquierda de la interfaz, se encuentran una serie de controles para modificar tanto las propiedades de la capturadora de vídeo como las propiedades de la propia cámara a bajo nivel mediante el dispositivo FTDI.

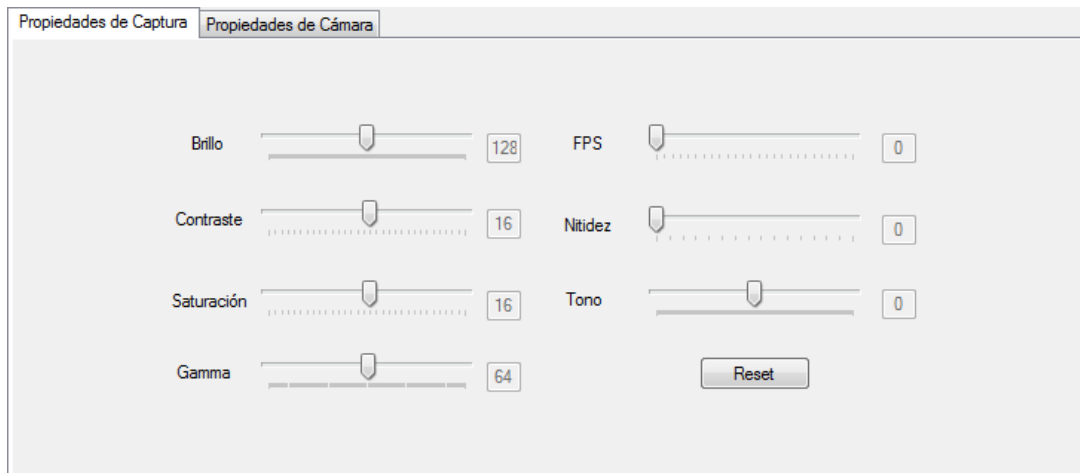


Figura 11: Control de propiedades de captura.

Las propiedades de captura son establecidas a sus valores por defecto tras el reinicio del equipo o al desconectar la capturadora de vídeo. Hasta el momento no se ha implementado una funcionalidad de guardado de los parámetros, por lo que hay que configurarlos nuevamente.

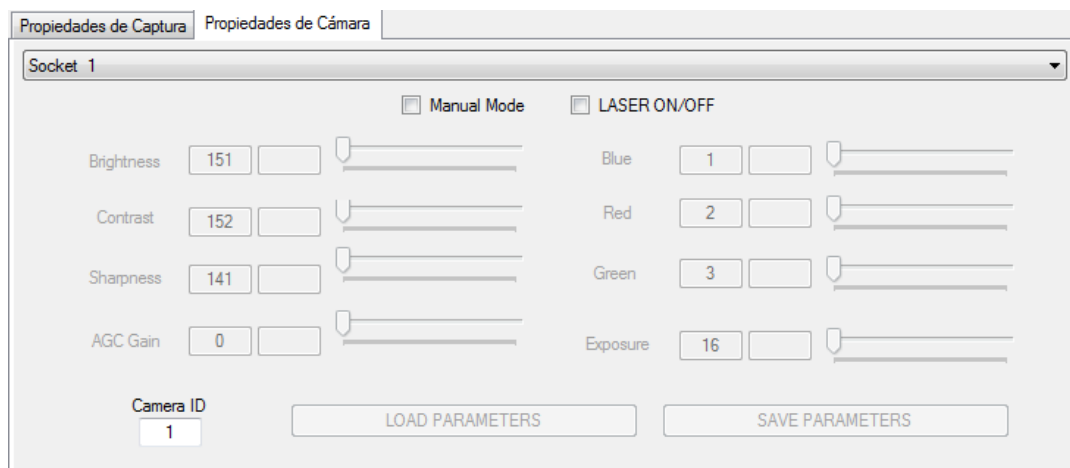


Figura 12: Control de propiedades de la cámara.

Al igual que las propiedades de captura, las propiedades de la cámara son establecidas a sus valores por defecto tras el reinicio del equipo o la desconexión del dispositivo FTDI. Sin embargo, los valores de ajuste pueden ser guardados usando el botón **SAVE PARAMETERS**. Para la carga de los parámetros es necesario desactivar y reactivar la opción **Manual Mode**, ya que el botón **LOAD PARAMETERS** no realiza la carga de forma adecuada

Si no se establece la conexión no se puede efectuar el control de velocidad. Además, se ha añadido una función de cambio de velocidad manual mediante el uso de tres botones (100% de velocidad, 50% y 1%).

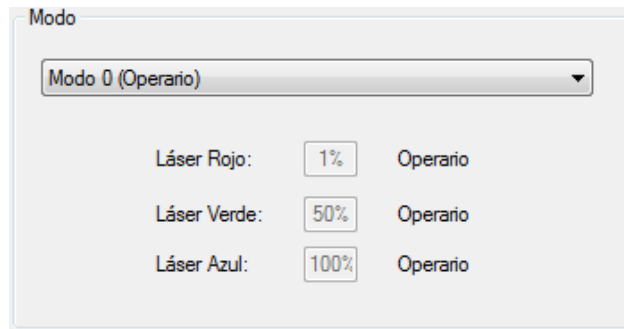


Figura 13: Selección de Modo (Modo 0).

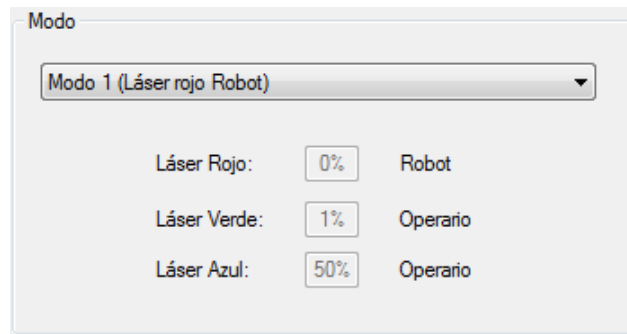


Figura 14: Selección de Modo (Modo 1).

El panel de selección de modo está compuesto por un **ComboBox** para la selección y un panel de información donde se muestra el cambio de velocidad del robot en relación al láser cortado. El modo 0 (figura 13) está pensado para que los tres láseres estén colocados en el lado del operario, mientras que el modo 1 (figura 14) proporciona la opción de colocar el láser rojo en la zona del robot.

Código y funcionamiento

En primer lugar es necesario ajustar las propiedades de captura y de la cámara con el fin de oscurecer lo máximo posible el fondo de la escena, de forma que solo se aprecie la luz de los láseres. Una vez ajustada, se realiza el procesamiento de la imagen, de la siguiente manera:

Procesado de la imagen

El procesamiento de la imagen se lleva a cabo en la función **ProcessFrame**. En primer lugar, se almacena la imagen captada por la cámara en la matriz frame:

```
Mat frame = newMat();
_capture.Retrieve(frame, 0);
```

A continuación se convierte dicha matriz a una imagen a color tipo byte:

```
Original = frame.ToImage<Bgr, Byte>();
```

Se transforma la imagen al formato de color **HSV** para el filtrado del color y se establecen los rangos de filtrado. El tono (Hue) en Emgu CV tiene un rango de 0° a 180°, por lo que el rango de color real debe ser reducido a la mitad. El filtrado de la imagen se lleva a cabo mediante el uso de la función ***inRange***, cuyo resultado será una imagen binarizada:

```
// Conversión a HSV
Image<Hsv, Byte> Org_Hsv = Original.Convert<Hsv, Byte>();
// Colores mínimos y máximos en HSV para el filtro por color
Hsv min_red   = newHsv(0, 160, 100);
Hsv max_red   = newHsv(5, 255, 255);
Hsv min_green = newHsv(40, 200, 100);
Hsv max_green = newHsv(85, 255, 255);
Hsv min_blue  = newHsv(110, 100, 100);
Hsv max_blue  = newHsv(180, 255, 255);
// Filtrado de la imagen
Image<Gray, Byte> FiltroRojo = Org_Hsv.InRange(min_red, max_red);
Image<Gray, Byte> FiltroVerde = Org_Hsv.InRange(min_green, max_green);
Image<Gray, Byte> FiltroAzul  = Org_Hsv.InRange(min_blue, max_blue);
```

A continuación se dilatan cada una de las imágenes obtenidas (con la función ***Dilate*** con 2 iteraciones) para facilitar el cálculo de líneas mediante la transformada de **Hough**, mediante el uso de la función ***HoughLinesBinary***, cuyo resultado será un vector de tipo ***LineSegment2D*** que se usará para la detección del láser:

```
FiltroRojo._Dilate(2);
FiltroVerde._Dilate(2);
FiltroAzul._Dilate(2);

LineSegment2D[] redlines = FiltroRojo.HoughLinesBinary(1, Math.PI / 180, 0, 20, 10)[0];
LineSegment2D[] greenlines = FiltroVerde.HoughLinesBinary(1, Math.PI / 180, 20, 10, 10)[0];
LineSegment2D[] bluelines = FiltroAzul.HoughLinesBinary(1, Math.PI / 180, 0, 20, 10)[0];
```

Además, se añade un umbral mínimo para la detección del láser en cuanto al tamaño de la imagen. Si la imagen binarizada no muestra un número suficiente de píxeles blancos, serán considerados como ruido. Para ello se calcula el número de píxeles totales de la imagen y se cuenta el número de elementos distintos de cero mediante la función ***CountNonZero*** y se calcula el porcentaje de píxeles blancos:

```
float pixel_Rojo = CvInvoke.CountNonZero(FiltroRojo);
float pixel_Verde = CvInvoke.CountNonZero(FiltroVerde);
float pixel_Azul = CvInvoke.CountNonZero(FiltroAzul);

float dim = (Original.Width) * (Original.Height);
float percent_r = (pixel_Rojo / dim) * 100;
float percent_g = (pixel_Verde / dim) * 100;
float percent_b = (pixel_Azul / dim) * 100;
```

Por último, si se cumple que ha habido una detección de líneas y si el número de píxeles supera el umbral mínimo establecido, las variables booleanas de detección cambiarán al estado true. Además, la etiqueta del ImageBox correspondiente al láser cortado cambiará a color rojo, a modo de señal visual:

```
if (redlines.Length != 0 && percent_r > 0.05){
    RedLaserLabel.ForeColor = Color.Red;
    red_detect = true;
}else{
    RedLaserLabel.ForeColor = Color.Black;
    red_detect = false;
}
if (greenlines.Length != 0 && percent_g > 0.05){
    GreenLaserLabel.ForeColor = Color.Red;
    green_detect = true;
}else
{
    GreenLaserLabel.ForeColor = Color.Black;
    green_detect = false;
}
if (bluelines.Length != 0 && percent_b > 0.05){
    BlueLaserLabel.ForeColor = Color.Red;
    blue_detect = true;
}else {
    BlueLaserLabel.ForeColor = Color.Black;
    blue_detect = false;
}
```

Una vez establecidas las variables de detección se procede al envío de los comandos necesarios para el cambio de velocidad del robot. El comando necesario será una variable tipo **string** con tres posibilidades (**D** para el 100% de velocidad, **E** para el 50% y **F** para el 1%, provisionalmente). El envío de comandos se realiza mediante llamadas a las funciones **Cambio_Velocidad_Modo0** y **Cambio_Velocidad_Modo1***, dependiendo del modo seleccionado por el usuario. Estas funciones envían el comando al robot mediante la función **Limpia_envia_comando**, atendiendo a todas las posibles combinaciones de detección de láseres:

```
private void Cambio_Velocidad_Modo0()
{
    // Velocidad por defecto
    if (red_detect == false & green_detect == false & blue_detect == false)
    {
        Limpia_envia_comando("D");
    }
    if (blue_detect == true)
    {
        if (green_detect == true)
        {
            if (red_detect == true)
            {
                // 3 Láseres detectados (R V A)
                Limpia_envia_comando("F");
            }
            else
            {
                // 2 Láseres detectados (V A)
                Limpia_envia_comando("E");
            }
        }
        elseif (green_detect == false & red_detect == true)
        {
            // 2 Láseres detectados (R A)
            Limpia_envia_comando("F");
        }
        else
        {
            // 1 Láser detectado (A)
            Limpia_envia_comando("D");
        }
    }
    elseif (blue_detect == false)
    {
        if (green_detect == true && red_detect == true)
        {
            // 2 Láseres detectados (R V)
            Limpia_envia_comando("F");
        }
        elseif (green_detect == true && red_detect == false)
        {
            // 1 Láser detectado (V)
            Limpia_envia_comando("E");
        }
        elseif (green_detect == false && red_detect == true)
        {
            // 1 Láser detectado (R)
            Limpia_envia_comando("F");
        }
    }
}
}
```

El envío de comandos al robot solo se produce siempre y cuando se haya establecido una conexión con el robot:

```
if (comunicacion == true)
{
  if (Modo == 0)
  {
    Cambio_Velocidad_Modo0();
  }
  elseif (Modo == 1)
  {
    Cambio_Velocidad_Modo1();
  }
}
```

Prototipo 3 cámaras y 3 láseres.

Se ha desarrollado un prototipo con 3 cámaras con filtros independientes para cada láser, con esto conseguimos un filtrado individual a nivel de hardware de cada uno de los láseres.

Se han modificado los algoritmos de detección de cada uno de los láseres para mejorar los problemas de detección ante cambios de color en la piel de los usuarios.

Con el nuevo sistema damos más robustez a la aplicación de captación del entorno.

Hardware bajo coste:

- ✓ 3 Cámaras desarrollo propio con filtros específicos.
- ✓ Tarjeta de control y conmutación de las tres cámaras.
- ✓ Conjunto de 3 láseres R-G-B con regulación independiente.

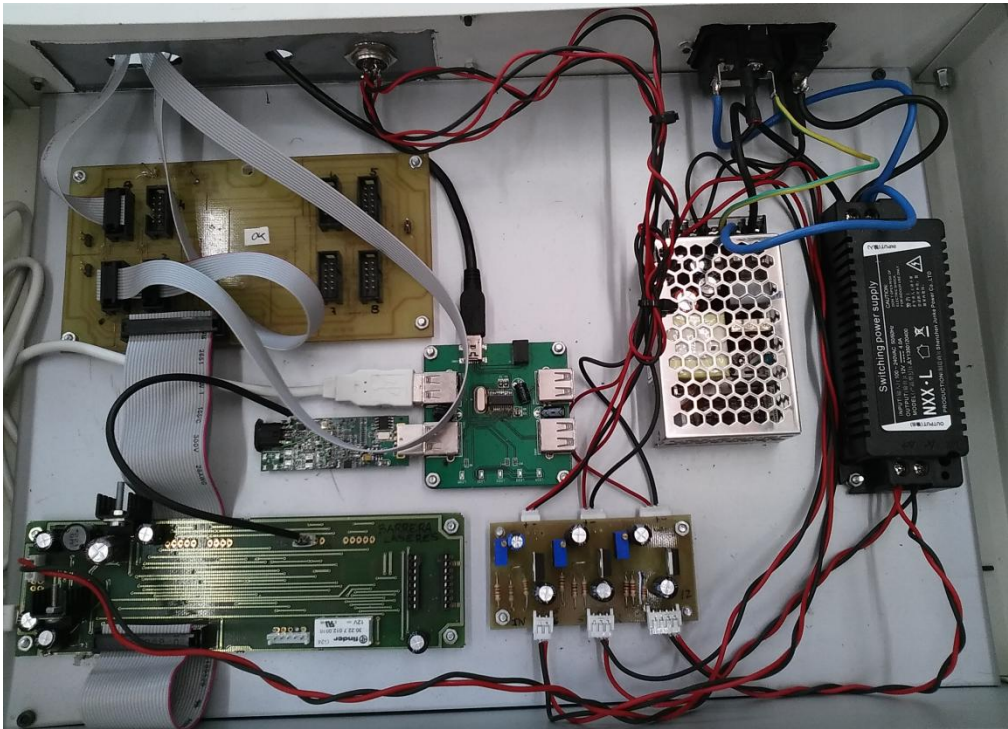


Figura 15: Electrónica de control.

Software:

- ✓ Control de cámaras a bajo nivel.
- ✓ Control de capturadora y sistema de conmutación de cámaras
- ✓ Conexión a robot.
- ✓ Control de modos de velocidad.

A continuación se describe el funcionamiento del software referente a la captura de imágenes, al tratamiento digital de la imagen necesario para la detección de los láseres y la generación de las variables necesarias para el cambio de velocidad del robot.

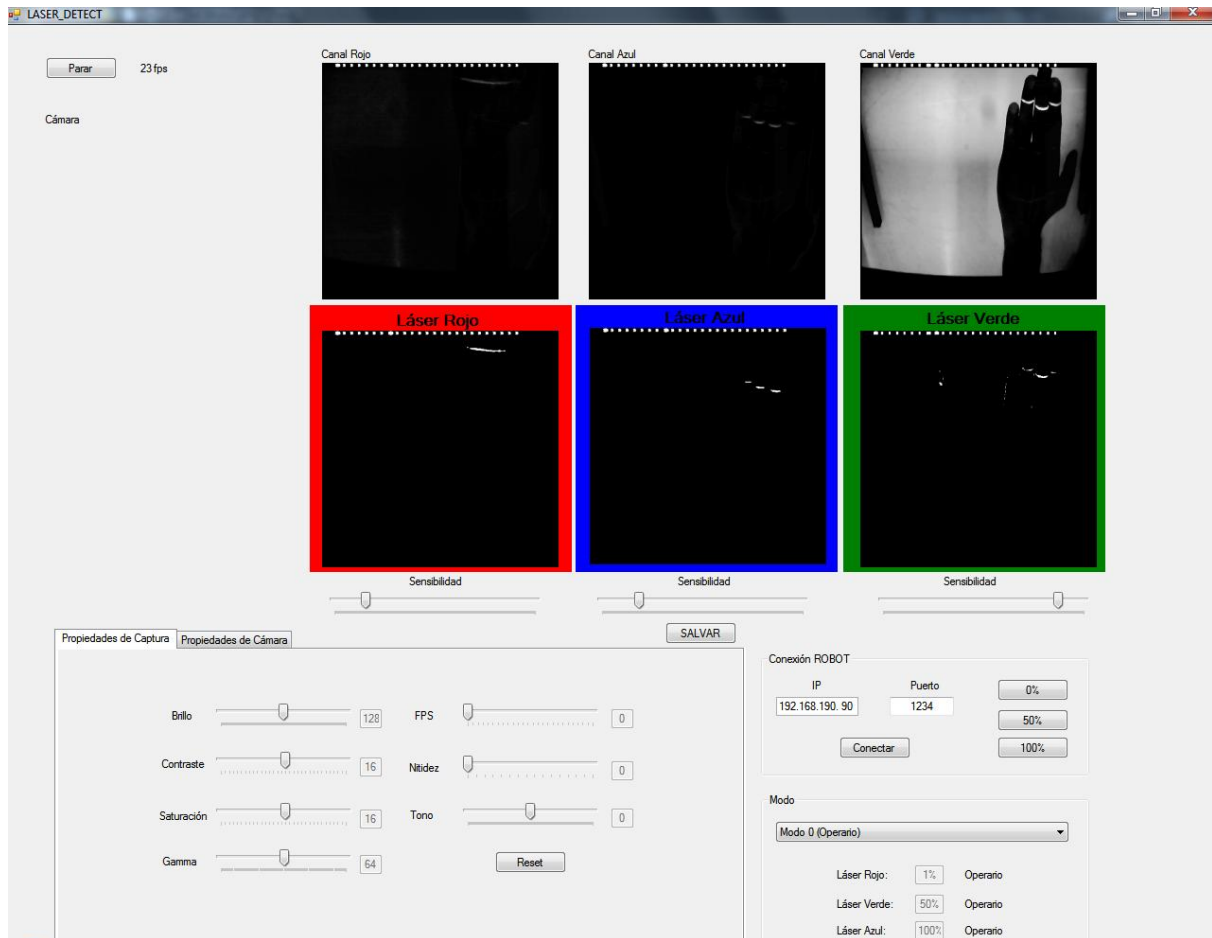


Figura 16: Interfaz sistema HRC.

Como se puede observar en la Figura 16, en la parte superior de la interfaz de la aplicación, se encuentran un total de seis **ImageBox**, donde se muestran la imagen original capturada por cada cámara, y las tres imágenes binarizadas, correspondientes a cada uno de los láseres detectados. Además, se incluye una barra de desplazamiento para controlar la sensibilidad de cada una de las cámaras. La función de estos **ImageBox** es solamente visual, con el objetivo de comprobar los resultados obtenidos, por lo que parte de ellos podrían ser eliminados para mejorar el rendimiento de la aplicación.

En primer lugar es necesario ajustar las propiedades de captura y de la cámara con el fin de oscurecer lo máximo posible el fondo de la escena, de forma que solo se aprecie la luz de los láseres. Una vez ajustadas, se realiza el procesamiento de la imagen, de la siguiente manera para cada una de las cámaras:

El procesamiento de la imagen se lleva a cabo en la función **ProcessFrame** que se realiza de forma independiente para cada cámara y laser, en la figura se puede ver el código para la cámara con filtro azul y el laser azul.

```

_capture.Retrieve(frame, 0);
fps_count++;
Original = frame.ToImage<Bgr, byte>();
Original_IM = frame.Bitmap;
/*Decodificar TTX*/
TTX.Decode(Original_IM);
if(TTX.CameraID==1)
{
    BlueBox.Image = frame;
    img = frame.ToImage<Gray, byte>();
    img._ThresholdBinary(new Gray(valor_BIN_BLUE), new Gray(255));
    BlueLaser.Image = img;
    /*Descartamos la zona del TTX*/
    Rectangle img_ROI = new Rectangle(0, 15, 640, 465);
    img.ROI = img_ROI;
    int pixel_Azul = CvInvoke.CountNonZero(img);
    float dim = frame.Width * frame.Height;
    float percent_b = (pixel_Azul / dim) * 100;
    if (percent_b > 0.05)
    {
        BlueLaserLabel.BackColor = Color.Blue;
        pictureBox_AZUL.BackColor = Color.Blue;
        blue_detect = true;
    }
    else
    {
        BlueLaserLabel.BackColor = Color.Transparent;
        pictureBox_AZUL.BackColor = Color.Transparent;
        blue_detect = false;
    }
}
}

```

Figura 17: Código detección de Laser.

4.4 Integración de sistemas HRC

Se ha realizado la integración de los sistemas HRC implementados con el sistema de control del robot. Durante las pruebas de funcionamiento se detectó que ante cambios en el color de la piel de los usuarios obteníamos algunos fallos de detección de los láseres.



Figura 18: -Integración del sistema en celda del sector del calzado.

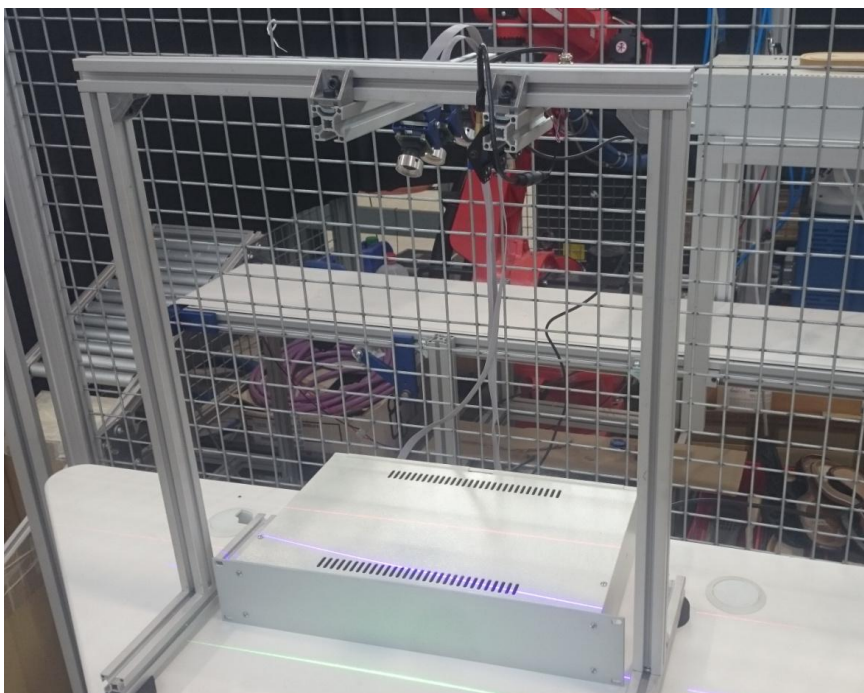


Figura 19: Integración del sistema en celda del sector del calzado.



Figura 20: -Integración del sistema en celda del sector del juguete.