ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT

PROYECTO FINAL DE GRADO

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño | ETSID

Grado en Ingeniería Electrónica Industrial y Automática

10 de Julio de 2017

Autor: Villalba Duarte, Kevin Daniel

Tutor: Ricolfe Viala, Carlos

CONTENIDOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT

MEMORIA DEL PROYECTO

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño | ETSID

Grado en Ingeniería Electrónica Industrial y Automática

10 de Julio de 2017

Contenido

1. OBJETIVO

El objetivo del proyecto es el desarrollo de un sistema automatizado que permita el ensamblaje de dos piezas de un motor con la ayuda de dos autómatas. La aplicación a realizar será la siguiente:

El primer autómata recogerá las piezas que vayan por la cintra transportadora y las colocará en sus respectivas plataformas de ensamblaje. Una vez colocadas las dos piezas necesarias, estarán listas para ser unidas por medio de una última pieza. A continuación, un segundo autómata agarrará la tercera pieza de un plato giratorio y procederá a realizar el ensamblaje final.

Para conseguir y poder realizar correctamente la aplicación, tenemos que lograr una serie de tareas concretas. La primera de ellas es hacer una correcta calibración entre el sistema de visión y el primer robot. La segunda es desarrollar un algoritmo que permita realizar una buena identificación de piezas para su posterior análisis y envío de datos. Y por último realizar una programación adecuada del autómata.

2. INTRODUCCIÓN

A lo largo del proyecto se planteará un posible método de automatización de un ensamblaje de dos piezas de un motor. Las piezas que hemos sugerido para realizar el trabajo son una biela, un pistón y un bulón.

2.1 ANTECEDENTES

En las últimas décadas la robótica ha sido uno de los adelantos tecnológicos más conocidos y más comentados. Esto se debe a las mejoras tecnológicas tales como: la aparición del microprocesador, mejorías en la informática, métodos de transferencia de datos, etc.

En el ámbito industrial, los robots han ido mejorando cada vez más con respecto a sus predecesores. A pesar de no notar las pequeñas diferencias a simple vista, sí que las podemos encontrar dentro del robot. La precisión y la rapidez son otros de los muchos aspectos que se han mejorado de forma significativa.

En la industria actual podemos observar robots que cumplen infinidad de tareas como cortar, fresar, soldar, pintar, embalar, ensamblar, etc. En casi la totalidad de trabajos que realizan, tanto la precisión, la habilidad como la eficiencia y la eficacia son aspectos fundamentales para su introducción en el medio industrial.

La automoción es el sector donde se puede observar más robots en sus cadenas de montaje. A pesar de que este sector ya cuenta con un nivel elevado de automatización, sigue habiendo elementos o tareas que se siguen realizando a mano. Esto ocurre en la

tarea de ensamblaje de pistones, con lo cual sería interesante considerar su posible y viable automatización.

2.2 MOTIVACIÓN

Hoy en día no es extraño que las empresas inviertan en robótica industrial o en la automatización en general, pues gracias a esto, se pueden conseguir resultados más eficientes desde el punto de vista de la producción, además de aportar mayores beneficios a la empresa.

En un futuro no muy lejano , y gracias a al desarrollo tecnológico, podremos ver a robots trabajando mano a mano con una persona sin el menor peligro de accidentes. Estos podrán encontrarse desde la barra de un bar o un restaurante hasta en los hospitales ayudando a los profesionales sanitarios en tareas de precisión, como una cirugía,. Además en el mercado actual de robots industriales ya podemos encontrar robots colaboradores, como el ur3 y sus versiones superiores de *Universal Robots* o *YUMI* de la marca comercial ABB .

El poder trabajar con robots industriales permite acercarse al ámbito de la automatización. Durante el Grado en Ingeniería Electrónica Industrial y Automática se ha trabajado de manera teórica sobre el tema de la robótica, ya sea resolviendo problemas de cinemática en las articulaciones de los brazos robot o realizando simulaciones, que en cierto modo te permite poner en práctica lo aprendido. El poder trabajar de manera real con los autómatas te ayuda a no solo plasmar los aprendido sino a ampliar conocimientos, además de la satisfacción de saber cómo se sentiría trabajar con ellos en una industria en el mundo real.

3. FACTORES A CONSIDERAR

3.1 NORMATIVA

La legislación que puede ser aplicada en el proyecto es la siguiente:

- UNE-EN ISO 10218-1:2012 : Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 1: Robots. (ISO 10218-1:2011)
- UNE-EN ISO 10218-2:2011 : Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 2: Sistemas robot e integración. (ISO 10218-2:2011).
- IEC 60870-5-104 : Red TCP/IP para establecer comunicaciones.
- ISO 8373:2012 : Define los términos de uso in relación con robots y dispositivos robóticos que operan en entornos industriales y no industriales.
- ISO 14539:2000 : Manipulación de robots industriales - Manejo de objetos con pinzas de sujeción - Vocabulario y presentación de características.

3.2 ESPECIFICACIONES DEL ENCARGO

Las especificaciones y requerimientos que tiene que reunir el proyecto vienen impuestos por:

- Las necesidades del cliente.
- Las características del entorno del sistema de proceso.
- La normativa vigente.
- La programación de los autómatas.

Los puntos mínimos que cubrirá el proyecto son los siguientes:

- Comunicación entre el sistema de visión y autómatas.
- Comunicación entre autómatas.
- Transferencia de datos mediante una conexión TCP/IP.
- Distribución de las piezas a los robots mediante una cinta transportadora.
- Posibilidad de ampliar las funciones al sistema, añadiendo más componentes como una cámara adicional o una mejor herramienta de agarre.
- Conseguir un coste bajo.
- Un control del proceso sencillo de realizar incluso para usuarios no especializados.

4. SOLUCIONES ALTERNATIVAS

4.1 ELECCIÓN DE ROBOTS

Para el desarrollo del proyecto hay que tener en cuenta que se necesita mucha precisión para poder llevarlo a cabo con éxito.

Dentro del mercado podemos encontrar mucha variedad de robots capaces de realizar un *pick and place*. De todas las opciones posibles, tenemos que elegir a la opción más eficiente.

Comparando varios modelos de brazos robots, los que mejor se adaptan a la tarea de "*pick and place*" ,que se va a desarrollar en este proyecto, son el robot KR 3 AGILUS del fabricante KUKA y el robot IRB 140 de la marca comercial ABB.

KR 3 AGILUS	IRB 140
Carga máxima de 3 Kg	Carga máxima de 6 Kg
Repetividad de posición de 0.02 mm	Repetividad de posición de 0.03 mm
Lenguaje de programación KRL	Lenguaje de programación RAPID
Radio de acción de 541 mm	Radio de acción de 810 mm

Tabla 1 Comparación de brazos robot.

En la tabla 1 podemos observar las características que diferencian a los robots. Teniendo en cuenta que las piezas no superan los 2 Kg, ambos son capaces de desarrollar el proyecto pues su carga máxima supera el peso de las piezas. Respecto al

lenguaje de programación, el lenguaje de programación de ABB (RAPID) es mucho más estructurado, además permite usar estructuras predefinidas para especificar la configuración del robot y de su herramienta, por contrario, el lenguaje de programación de KUKA (KRL) es menos estructurado y es un modelo basado en el lenguaje IRL, definido por la norma DIN 66312.

El aspecto para hacer la selección del robot recae en el radio de acción, debido a que si tenemos en cuenta las posibles características del entorno de trabajo, el robot AGILUS no podría alcanzar las posiciones a las que tendría que ir para realizar las operaciones correctamente. Por su mayor rango de trabajo y por su lenguaje de programación más sencillo, el robot seleccionado es el IRB 140 de la marca comercial ABB.

Para realizar el ensamblaje final y poder retirar las piezas, necesitamos un robot colaborativo. Dentro del ámbito de robots colaborativos, los seleccionados con el UR3 del fabricante *Universal robots* y LBR iiwa de KUKA .

LBR iiwa	UR3
Carga máxima de 7 Kg	Carga máxima de 3 Kg
Radio de acción de 800 mm	Radio de acción de 500 mm
Precio elevado	Precio económico

Tabla 2 Comparación de brazos robot 2.

Cabe destacar que el robot LBR iiwa está propuesto a realizar aplicaciones de investigación y no tanto relacionado con la industria. Además el lenguaje de programación de ambos robots es fácil e intuitivo para realizar un programa eficiente y no hace falta tener muchos conocimientos ni ser un experto. Teniendo en cuenta la desigualdad en el precio y a la facilidad de poder trabajar con él, elegiremos el ur3

4.2 ELECCIÓN DE IPD

Los computadores para el sistema de visión son muy diversos y dependen mucho de la aplicación y el entorno de trabajo al que van a estar sometidos. Para llevar a cabo una identificación de objetos con un IPD de clase media-baja es suficiente. El IPD va40 y va30 son los mejores de su clase.

VA40	VA30
Relative Speed 4	*Relative Speed* 2
Color Support YES	*Color Support No*
Program Memory 256MB	*Program Memory* 512MB
Storage Memory 80 GB	*Storage Memory* 128 MB Flash
Application Software : iNspect, Sherlock7	*Application Software : iNspect*

Tabla 3 Comparación IPD.

El IPD VA40 en general tiene mejores especificaciones técnicas, además nos permite trabajar con distintos software y tiene un soporte de color lo que nos permite realizar varias operación adicionales a diferencia del VA30. A pesar de contar con una menor memoria de programa, el IPD seleccionado es el VA40 de DALSA.

4.3 SOFTWARE DE VISIÓN

Los *software* de visión de permite el IPD seleccionado son *iNspect* y *Sherlock7.* Los dos programas ofrecen una multitud de opciones y herramientas que permiten desarrollar tareas como la identificación de piezas, guiado de robots, etc.

Ambos *software* no requieren programación para desarrollar una aplicación. iNspect trabaja con una lista a la hora de aplicar diferentes algoritmos o herramientas de trabajo, mientras que en *Sherlock7* si aplicamos un algoritmo este aparece en una ventana con la estructura del programa.

Respecto a la comunicación con elementos externos, los dos programas cuentan con una variedad de métodos para conexión, como pueden ser: E/S, Ethernet/IP, etc.

Los dos software pueden realizar las mismas aplicaciones como: la localización de objetos, verificar etiquetas, etc. Pero para hacer la elección, hemos tenido en cuenta la comodidad para poder observar de forma clara el programa realizado. La forma estructural que tiene Sherlock ayuda a seguir con facilidad el desarrollo del programa, por tanto, este es el *software* elegido.

4.4 CÁMARAS

Las cámaras son la parte fundamental del sistema de visión, son las que permiten observar los objetos de estudio a través de una lente y las que envían dicha imagen al sistema de procesado, que en nuestro caso es el *software Sherlock7*. Las cámaras de visión industriales tienen unas características técnicas superiores a las convencionales y éstas dependen del ambiente al que están sometidas. Asimismo, proporcionar una buena adquisición de datos para su posterior tratamiento es importante, puesto que si la recogida de datos es mala su tratamiento será más complicado.. Hay múltiples tipos de cámaras pero nosotros nos centraremos en la CV-M77 y la CV-M9GE ambas de la marca comercial JAI.

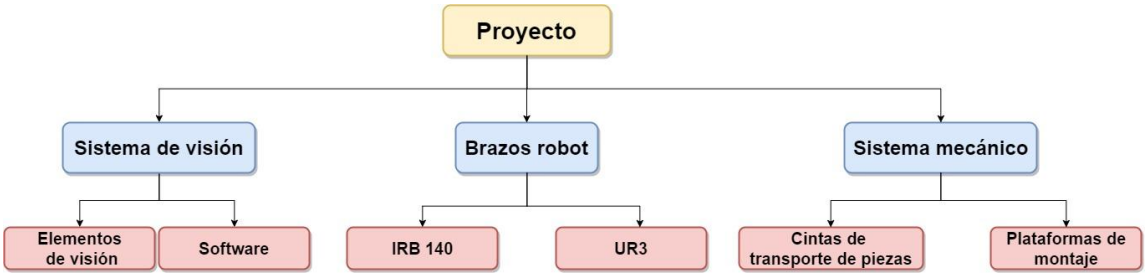
CV-M77	**CV-M10SX**
Resolution: 1024(h) 768(v)	*Resolution:* 659(h) 494(v)
Total Pixels: 786,432	*Total Pixels:* 325,546
Frame Rate(fps): 25	*Frame Rate(fps):* 30
Interface: Analog	*Interface:* Analog

Tabla 4 Comparación de cámaras.

En el proyecto, las cámaras van a tener que capturar la imagen cuando el objeto se encuentre quieto a una distancia de aproximadamente dos metros. A esa distancia la resolución de la imagen que capturan es importante, pues si la imagen no es nítida, después para su posterior análisis va a dificultar la identificación de piezas. Esto puede causar una complicación a la hora de la identificación de la forma de la pieza. Debido a tener una mayor resolución, la cámara seleccionada es la CV-M77.

5. DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN

Para poder ver con claridad los distintos aspectos del proyecto se ha elaborado el siguiente organigrama:



5.1 SISTEMA DE VISIÓN

El sistema de visión está compuesto por los elementos de visión y el *software* que permite analizar y tratar la imagen.

Dentro de los elementos de visión encontramos al IPD y la cámara. El IPD elegido es el VA40 y se trata de un computador que gobierna el sistema de visión artificial en el que se encontraría el *software* para poder analizar las imágenes y las carpetas de los programas realizados con el mismo, que además de funciona como servidor para poder realizar conexiones con diversos dispositivos. Respecto a la cámara, la que se ha seleccionado es la JAI CV-M77, y va ser la encargada de realizar la adquisición de datos para su posterior tratamiento en *Sherlock7*.

Ilustración 1 IPD.

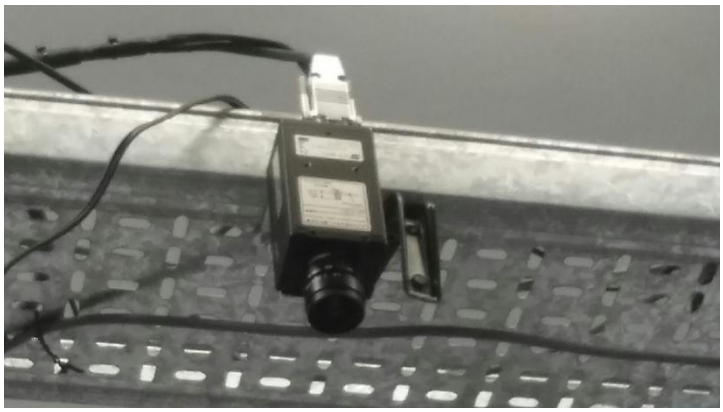


Ilustración 2 Cámara de visión.

El *software* que realizará la tarea del procesamiento de imágenes y la conexión con el primer robot es *Sherlock7*. Este programa nos permite trabajar con facilidad sobre la imagen captada por la cámara y modificar ciertos aspectos de la identificación de piezas de manera sencilla, esto es debido la baja complejidad de manejo del programa. Gracias a esto, se puede realizar una conexión vía TCP/IP con el primer autómata de manera rápida y eficiente.

5.2 BRAZOS ROBOT

Los brazos robot son los elementos principales para poder desarrollar el proyecto. El encargado de realizar la tarea de recoger el pistón y la biela una vez las detecte el sensor de posición de la cinta, y llevarlas a su respectiva plataforma va a ser el robot IRB 140 de la marca comercial ABB. Para poder realizar dicha tarea, el autómata va a estar conectado vía TCP/IP con el software de procesado de imágenes, el cual le pasará las posiciones de la piezas respecto el sistema de coordenadas del robot.

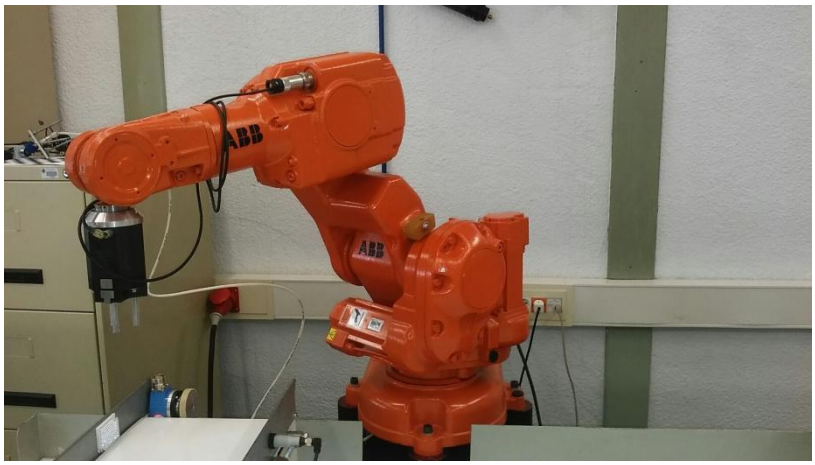


Ilustración 3 Brazo robot IRB 140.

El segundo autómata esperará a que el sensor de posición del plato giratorio detecte el elemento de unión de las piezas, que será el bulón, para llevarlo junto las resto de componentes y llevar a cabo el ensamblaje.

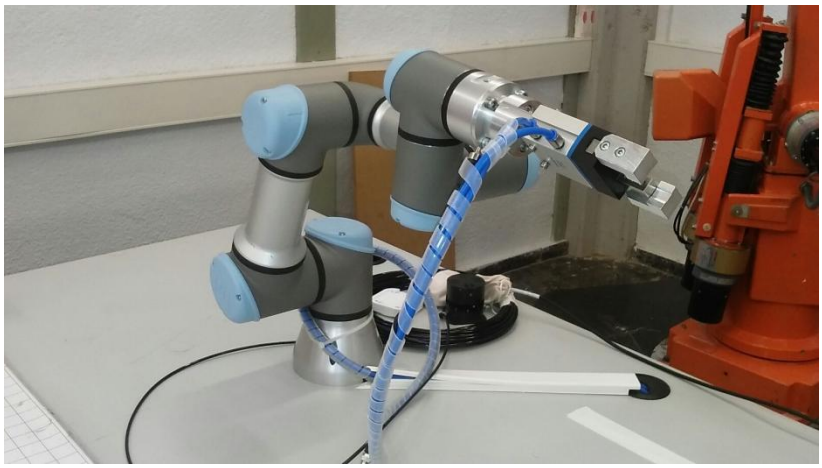


Ilustración 4 Brazo robot UR3.

5.3 SISTEMA MECÁNICO

El sistema mecánico es el conjunto de elementos que tienen un papel fundamental para realizar el proyecto. Dentro de este sistema se encuentran: sensores de posición, cinta transportadora, plato giratorio y también las plataformas de ensamblaje.

Los sensores son los encargados de detectar la pieza y enviar una señal cuando ésta llegue para que tanto la cinta como el plato giratorio se detengan. La cinta transportadora empezará a desplazarse al iniciar el proceso y al terminar de colocar una pieza en la plataforma de ensamblaje. Sólo se detiene cuando el sensor de posición detecta una pieza, y no reanuda su movimiento hasta que el robot deja la pieza. Este proceso se asemeja al movimiento del plato giratorio; este se pone en marcha cuando las dos piezas están listas para ser ensambladas y se detiene cuando el sensor de posición detecta la pieza de unión.

Ilustración 5 Cinta transportadora y su sensor de posición.

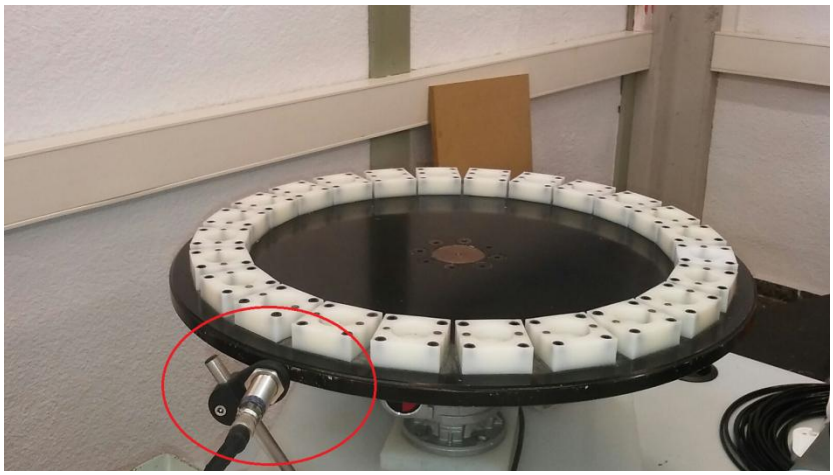


Ilustración 6 Plato giratorio y su sensor.

Los sensores de posición son los que se encuentran dentro del círculo rojo.

Por último tenemos las plataformas de ensamblaje. Contamos con una plataforma para el pistón y otra para la biela, además, las plataformas están alineadas y separadas cierta distancia para posteriormente juntarse y que las piezas estén listas para ser ensambladas mediante el bulón. También tenemos una plataforma donde va el bulón a lo largo del plato giratorio.

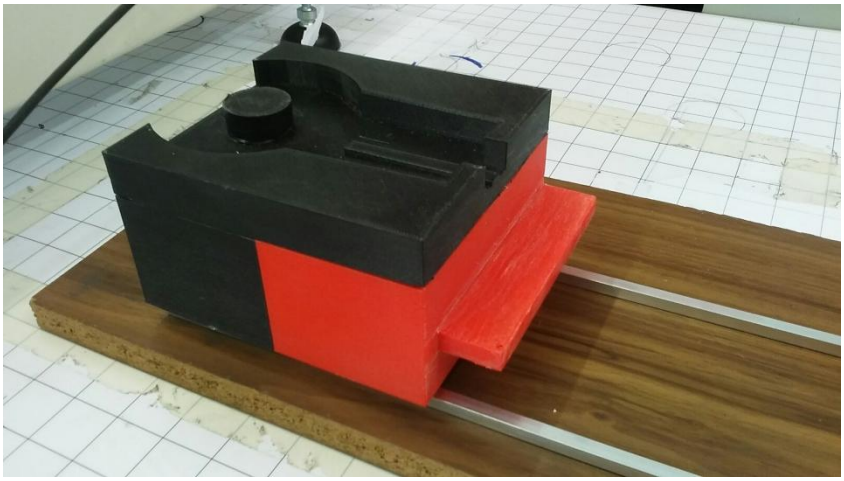


Ilustración 7 Plataforma de la biela.

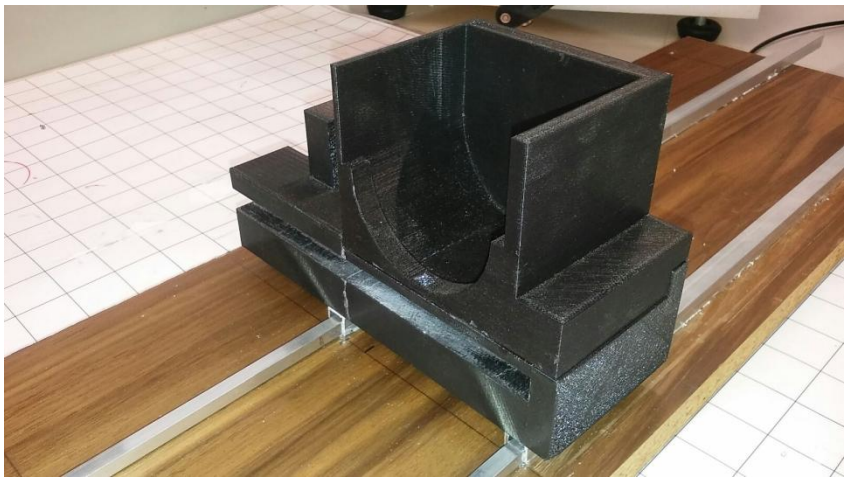


Ilustración 8 Plataforma del pistón.

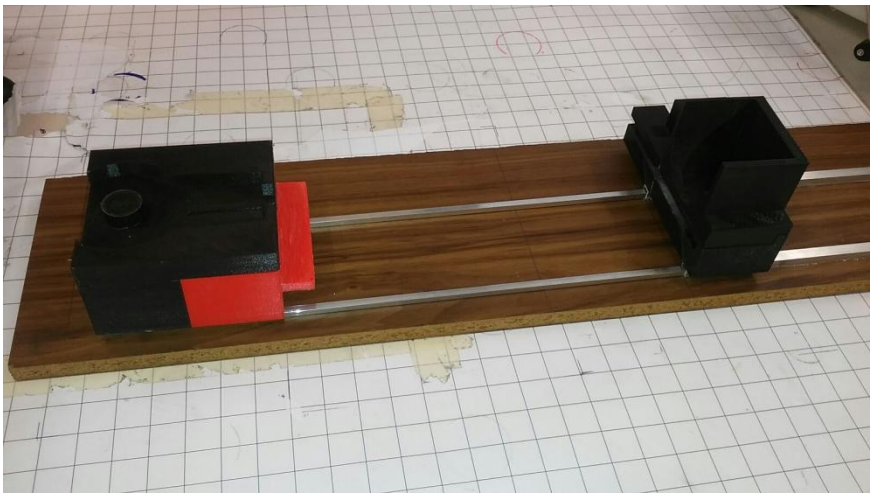


Ilustración 9 Plataforma de ensamblaje final.

6. DESARROLLO DE LA SOLUCIÓN

A continuación explicaremos de forma explícita el desarrollo del proyecto y los pasos que hemos realizado para completarlo.

6.1 CALIBRACIÓN

La calibración es el primer paso que realizamos y sirve para que el sistema de visión y el robot compartan el mismo sistema de coordenadas. Se llevará a cabo mediante una matriz de trasformación entre las coordenadas del sistema de referencia del robot y del sistema de visión.

El primer paso que realizamos es la toma datos de las coordenadas del sistema de visión y del robot. En primer lugar colocamos el pistón en diversas posiciones alrededor del sensor de posición de la cinta transportadora, después llevamos el robot hacia la pieza y apuntamos en un *excel* las coordenadas y los cuaterniones del primer punto. En las ilustraciones 10 y 11 podemos ver un ejemplo de la calibración del robot y los datos que tenemos que guardar.

Ilustración 10 Posicionamiento del brazo robot para realizar la calibración.

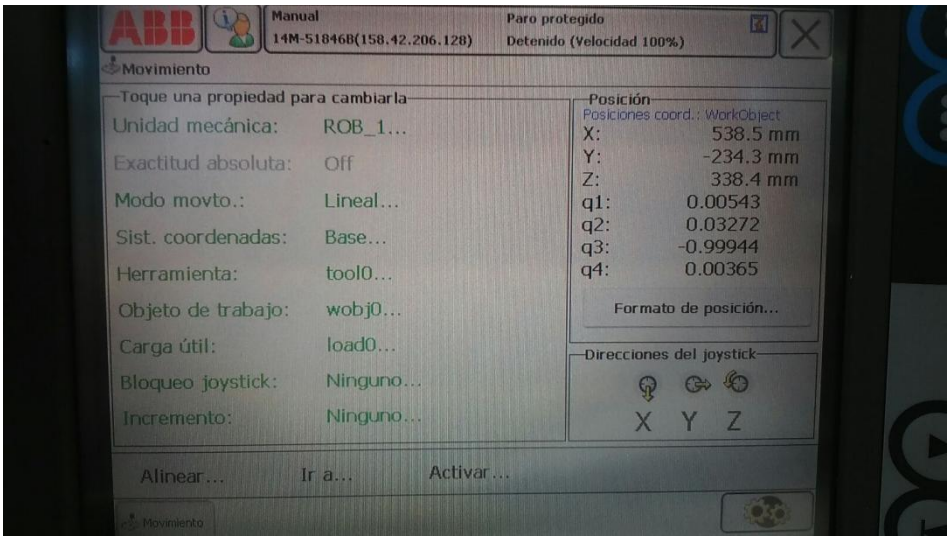


Ilustración 11 Lectura de las posiciones y los cuaterniones en la consola FlexPendant.

Por otro parte desarrollamos un programa con el algoritmo de *Conectivity - Binary* en Sherlock7 que nos permitará identificar el punto medio de la pieza. Con el pistón en la misma posición de calibración del robot, apuntamos las coordenadas del pistón en una hoja de cálculo.

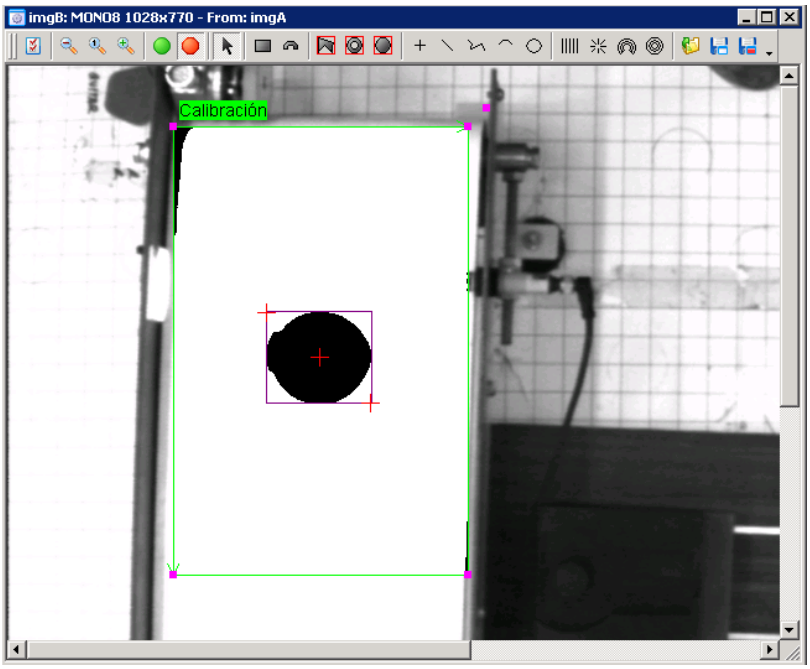


Ilustración 12 Aplicación del algoritmo Conectivity-Binary para realizar la calibración.

En la Ilustración 12 se puede observar la ROI[i] donde aplicamos el algoritmo *Conectivity - Binary.* Cabe destacar que para trabajar sobre la imagen y poder aplicar el algoritmo, hay que crear una nueva ventada de visualización. Para poder crearla seleccionamos el icono con apariencia de cámara justo debajo del código del programa. Una vez seleccionada, aparecerá la siguiente ventana:

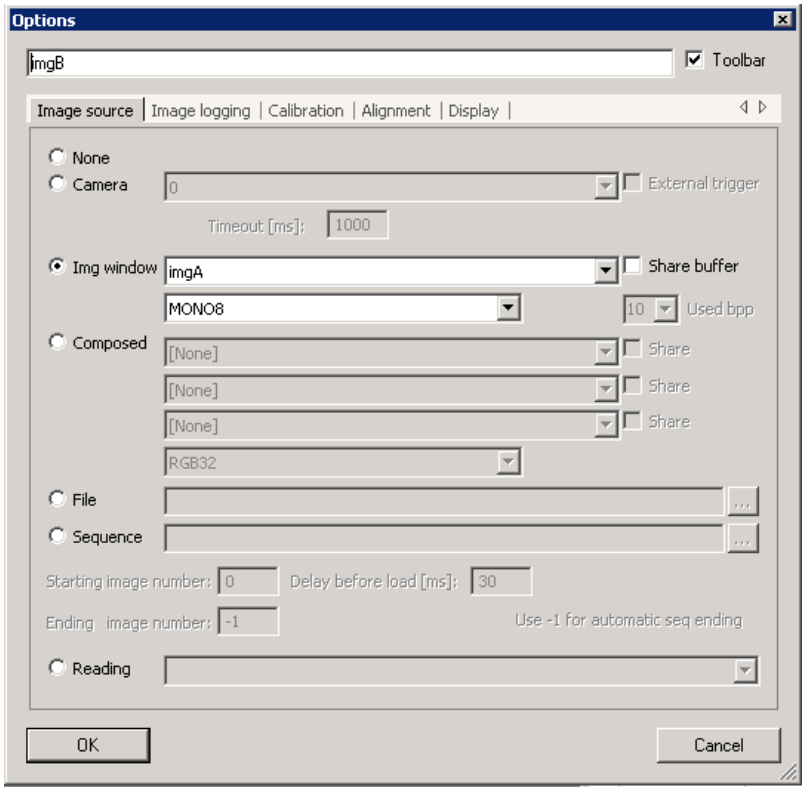


Ilustración 13 Selección de ventana de visualización.

Como se puede observar en la ilustración 13, en el apartado Img window seleccionamos la ventana de visualización principal y justo debajo, seleccionamos MONO8. De esta manera la nueva imagen estará en tonos monocromos y podremos aplicar los algoritmos necesarios. En la ilustración 14 podemos observar el programa de calibración realizado.

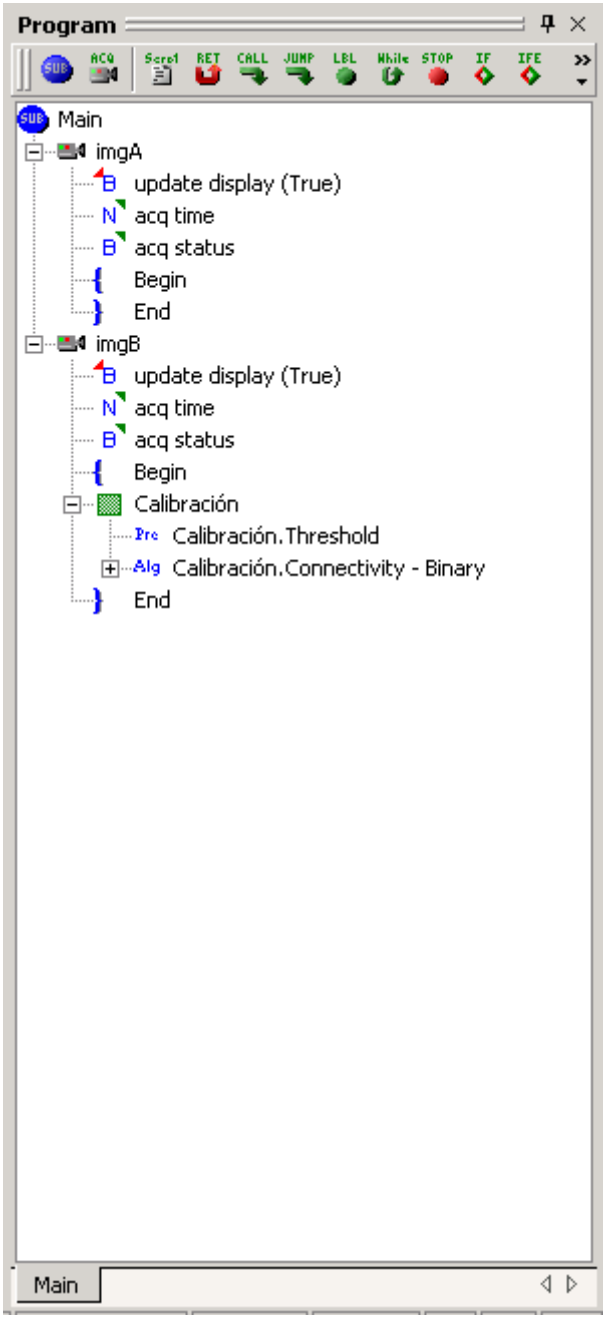


Ilustración 14 Código de calibración.

Repetimos el proceso de toma de datos cinco veces o más para tener una mejor exactitud. En el caso de tomar las coordenadas de *Sherlock7* no hará falta elaborar un

nuevo programa, sino que podemos reutilizar el mismo. Sólo tendremos que actualizar la imagen de la cámara para obtener la nueva posición.

Una vez obtenidas las coordenadas de los distintos sistemas de referencia, el segundo paso es conseguir la matriz de transformación a partir de ellas, la matriz resultante sería la de la ilustración 15 (Véase Anexo I, para el programa de calibración).

```
xee =

  557.4283  555.2398  498.0833  456.8709  453.9593  407.5964  367.2392  367.9826
 -190.9464 -286.6731 -237.2346 -281.4272 -186.1675 -236.1063 -277.8759 -183.5693
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

Ilustración 15 Matriz de transformación final.

Cada columna son las coordenadas respecto el sistema de referencia del robot de cada uno de los puntos que hemos utilizado para realizar la calibración. Es decir, son las coordenadas que le tendríamos que pasar al robot cuando la pieza se encuentre en uno de esos puntos. Como se puede observar, la última fila son todos unos y eso significa que la componente en z del sistema de coordenadas del robot va a ser siempre la misma para recoger las piezas. El valor de la coordenadas en z se pondrá en el código de programación del robot.

Como la pieza no va a estar siempre en uno esos puntos, sino que estará en un punto arbitrario de la cinta, tenemos que relacionar las coordenadas de *Sherlock7* y las coordenadas que hay que pasarle al robot. Para relacionar dichas coordenadas nos apoyamos en una hoja de cálculo y creamos una tabla de datos (Véase Tabla5).

Puntos Sherlock		Puntos robot	
x	y	x	y
296,76	293,01	557,4283	-190,9464
197,83	291,15	555,2393	-286,6731
249,37	233,52	498,0833	-237,2346
204,69	192,99	456,8709	-281,4272
302,49	188,87	453,9593	-186,1675
251,55	143,64	407,5964	-236,1063
209,61	104,84	367,2392	-277,8759
305,77	103,62	367,9826	-183,5693

Tabla 5 Datos de calibración.

En segundo lugar elaboramos dos gráficas, la primera, Ilustración 16, relaciona las coordenadas "x" de *Sherlock7* con las coordenadas "y" que tenemos que asignar al robot y la segunda, Ilustración 17, relaciona las coordenadas opuestas.

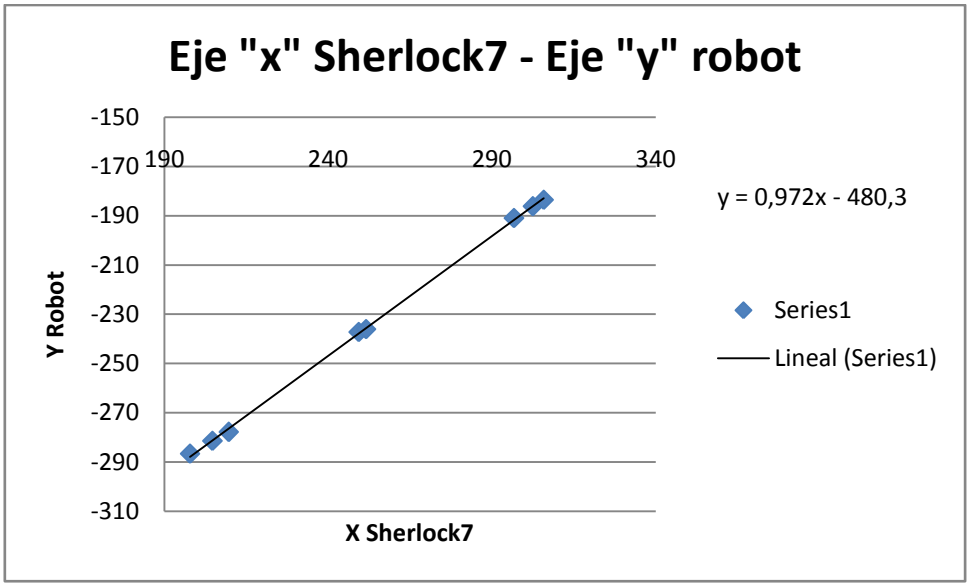


Ilustración 16 Gráfico de ecuación para el eje "y" del robot.

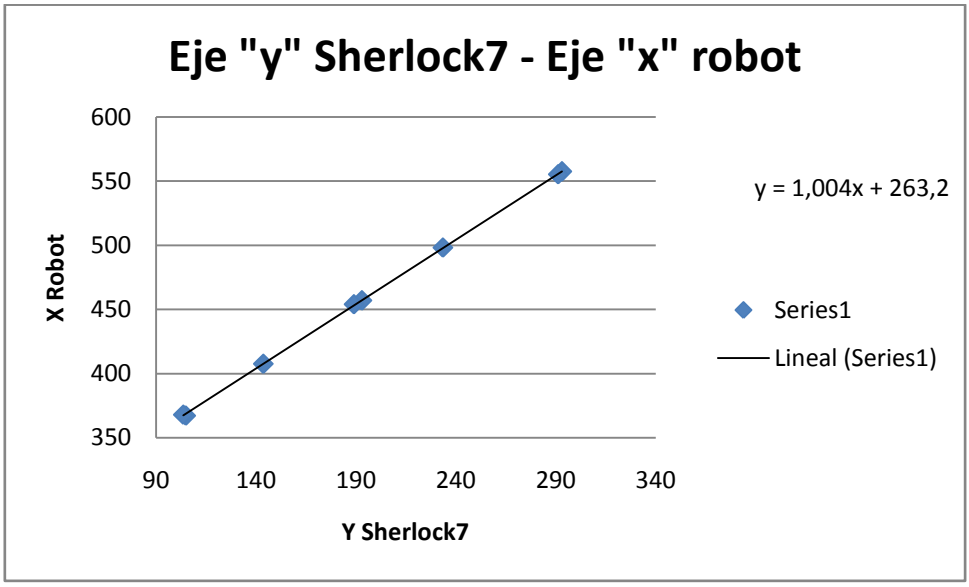


Ilustración 17 Gráfico de ecuación para el eje "x" del robot.

La ecuación de la línea de tendencia de los puntos de la calibración, es la relación entre sistemas de referencia del robot y del sistema de visión.

6.2 ALGORITMO DE SHERLOCK7

Para realizar el proyecto de una manera eficiente y eficaz, desarrollar un correcto algoritmo de visión es importante.

Nuestro programa de visión consta de varios apartados:

- Conexión.
- Inicializar variables.
- Identificar piezas.
- Programa principal o *Main*.

6.2.1 CONEXIÓN

La conexión que se va a realizar es vía TCP/IP entre el sistema de visión y el robot IRB140. Para poder establecer esta conexión, uno de los dos tiene que funcionar como servidor y el otro como cliente. En este caso el servidor será el sistema de visión, cuyo encargado para llevarla a cabo es *Sherlock7*.

Para crear un servidor seleccionamos *Options* en la barra de herramienta del programa. A continuación pulsamos IO-> Tcp/Ip y nos saldrá la pantalla de la Ilustración 18.

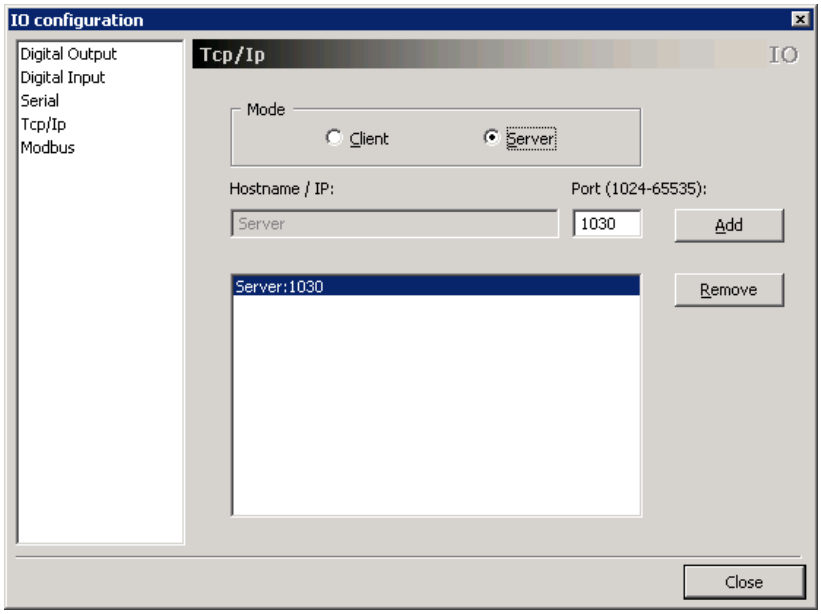


Ilustración 18 Ventana de conexión TCP/IP

Como queremos que *Sherlock7* actúe como servidor, seleccionamos el modo servidor y después elegimos el puerto de conexión. Para finalizar, añadimos el servidor con el puerto y pulsamos *Close*.

Una vez creado el servidor realizamos el programa de conexión. Para realizar una conexión desde del programa de visión tenemos dos bloques: *recv line* y *send lin.*

Recv line

La apariencia de este bloque es la siguiente:

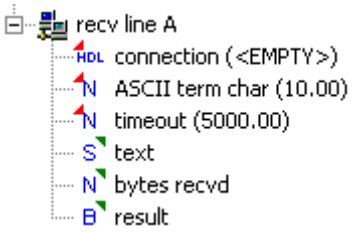


Ilustración 19 Bloque recv line de Sherlock7.

El primer término hace referencia al servidor por el que transmitir los datos. El segundo término hacer referencia al tipo de carácter según el código ASCII para finalizar la lectura de recepción de datos. El tercer término es el tiempo que espera sin que reciba una cadena de datos hasta saltar a la siguiente instrucción. El término "text" es la variable donde se guarda la cadena de datos recibida. Por último, *result* devuelve el valor verdadero o falso dependiendo si ha recibido o no un mensaje.

Send line

La apariencia de este bloque es la siguiente:

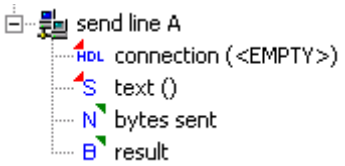


Ilustración 20 Bloque send line de Sherlock7

Sólo nos vamos a enfocar en los dos primeros términos de este bloque. El primero es el mismo que *revc line* y el segundo es la cadena de datos que enviamos desde el sistema de visión.

El programa de conexión estará formado por dos partes, una primera parte que será la toma de contacto entre *Sherlock7* y el robot y una segunda parte que confirmará la conexión.

Para poder rellenar todos los campos de ambos bloques hay que crear varías variables, elegir el carácter en código ASCII para finalizar la lectura de la cadena de datos recibida, conectarnos al servidor previamente creado y determinar el tiempo de espera antes de saltar a la siguiente instrucción. El carácter elegido es el número 33 y es "!", por lo tanto, recibirá toda la cadena de datos hasta que reciba "!". Respecto al tiempo de espera hemos establecido un minuto. En la ilustración 21 se puede observar el programa de conexión del sistema de visión.

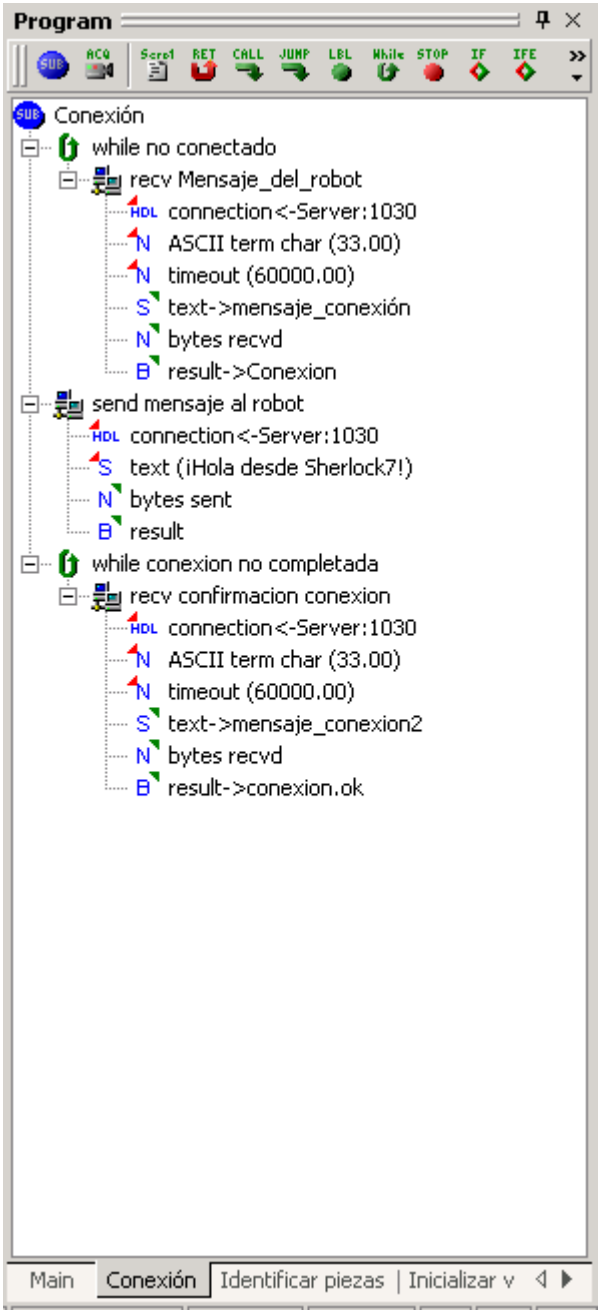


Ilustración 21 Programa de conexión con el robot IRB 140.

Para que el robot pueda comunicarse con el sistema de visión es necesario la utilización de canales de transmisión de datos. Estos canales son los *sockets* o zócalos, que permiten enviar y recibir información vía TCP/IP. Las instrucciones en código *Rapid* son las siguientes:

- *SocketCreate*: Crea un socket nuevo.
- *SocketConnect*: Realiza la conexión con el ordenador remoto deseado. En nuestro caso, será con el sistema de visión.
- *SocketSend*: Envía una cadena de datos al ordenador deseado.
- *SocketReceive*: Recibe datos y los almacena en una variable tipo *string*.
- *SocketClose*: : Cierra el socket creado.

La implementación en *Rapid* de nuestro código es la siguiente:

```
SocketCreate canal1;
SocketConnect canal1, "158.42.16.207", 1030;
SocketSend canal1\Str:="¡Hola desde el robot!";
SocketReceive canal1\Str:=datos_conexion;
SocketSend canal1\Str:="Conexion establecida!";
```

Ilustración 22 Secuencia de conexión con Sherlock7.

Como se puede observar en la Ilustración 22 en primer lugar creamos un canal de comunicación y nos conectamos a él con el sistema de visión al puerto establecido por *Sherlock7*. Acto seguido enviamos un mensaje que termina con el carácter seleccionado en el programa de visión (33, que hace referencia a "!"). Después guardamos en mensaje recibido en la variable tipo *string* llamada datos_conexion y para terminar de establecer la conexión enviamos un mensaje de confirmación. El comando *SocketClose* irá al final del código de programación

6.2.2 INICIALIZACIÓN DE VARIABLES

La inicialización de variables es importante porque evita que al iniciar el programa una variable tenga un valor que no le corresponde y ejecute una instrucción de forma errónea. Estas son algunas de las variables que iniciaremos a 0 o modo por defecto:

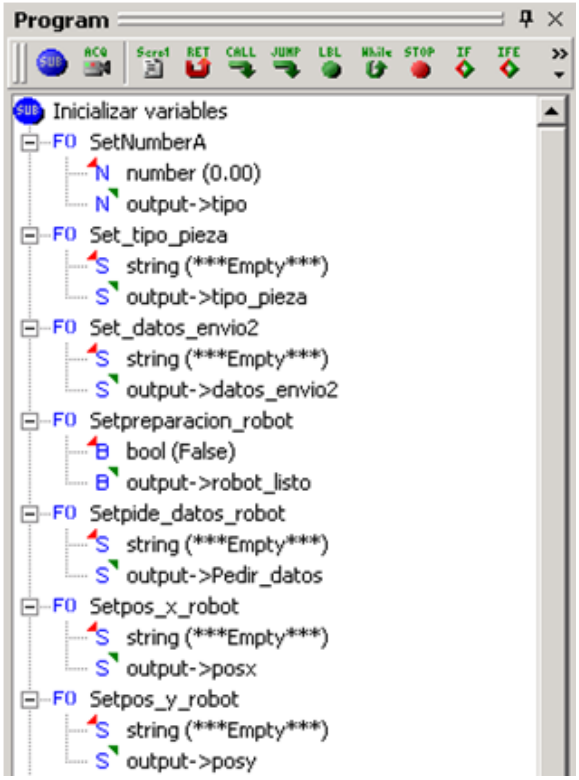


Ilustración 23 Programa de Inicialización de variables.

6.2.3 IDENTIFICACIÓN DE PIEZAS

Dentro del logaritmo de programación este apartado es el más importante, pues es la base del proyecto.

Para poder trabajar sobre la imagen y aplicar los logaritmos necesarios, hay seguir los mismos pasos de la calibración para poder trabajar sobre una imagen monocromo.

Lo primero que tenemos que hacer es crear dos ROI, una para cada elemento a identificar. Para identificar las piezas aplicaremos el algoritmo *Search - Geometric* pero para hacer una identificación más sencilla utilizaremos el proceso *Threshold*. Éste último nos permite poner un límite en la escala de grises, es decir, limitar la escala de grises a una escala binaria. El valor que hemos asignado dentro de los parámetros de este proceso es 165, por tanto, todos lo pixeles que tengan un valor inferior a este se pondrán a 0 , pasarán a ser de color negro, y todos los que tengan un valor igual o superior se pondrán a 255, pasarán a ser de color blanco.

Una vez tenemos hecha la diferenciación entre piezas, procedemos a analizar cada una. Lo que tenemos que hacer ahora es pasar las coordenadas más optimas de las piezas para que el robot las pueda agarrar.

PISTÓN

Como el pistón tiene una forma circular, las coordenadas más optimas son las de su centro. Para poder identificar las coordenadas del centro de la pieza utilizaremos el algoritmo *Conectivity - Binary*. Este algoritmo nos devuelve el valor de las coordenadas del centro de masa de las figuras que se encuentren dentro del ROI.

Para detectar sólo el pistón hemos tenido que variar los parámetros del algoritmo, a base de prueba y error, hasta conseguir detectar únicamente el pistón. Los parámetros finales son los de la Ilustración 24.

Constraints	
black blobs	True
8 way	True
min area	500
max area	1000000
min width	10
max width	1000000
min height	1
max height	1000000

Ilustración 24 Parámetros del algoritmo Conectivity-Binary.

Sólo tuvimos que modificar el *min area* y el *max area* que aparece por defecto al aplicar el algoritmo por primera vez.

Cuando el algoritmo detecta correctamente el pistón, elaboramos un script para la recogida de datos y para guardar la información en variables auxiliares. El script es el siguiente:

```
xp = Vars.Coordenadas_piston[0];
yp = Vars.Coordenadas_piston[1];

//Conversión a coordenadas del robot

Vars.xrobot = 1.004*yp + 263.2;

Vars.yrobot = 0.972*xp - 480.3;

Vars.tipo = 1;
```

La variable tipo hace referencia al tipo de pieza que es. En número 1 es el pistón y el número 2 es la biela. Además, *xrobot* e *yrobot* son las posiciones que le tenemos que pasar al robot para que pueda recoger la pieza correctamente.

BIELA

Al contrario que con el pistón, la biela no tiene una forma fácil de identificar. Las coordenadas más optimas para su agarre son la mitad de la pieza y no su centro de masas. Utilizaremos el mismo algoritmo de identificación que el pistón pero ésta vez tendremos que cambiar un parámetro adicional. Para poder identificar la posición de la mitad de la pieza vamos a detectar las coordenadas de los agujeros de la biela para así después poder calcular el punto medio entre los dos puntos, un punto por cada agujero.

Como ahora queremos detectar los *blobs* de color blanco dentro de la pieza, tenemos que marcar False el apartado de *black blobs*; además de cambiar su tamaño. Los parámetros finales de biela aparecen en la Ilustración 25.

Constraints	
black blobs	False
8 way	True
min area	1
max area	1000
min width	1
max width	1000000
min height	1
max height	1000000

Ilustración 25 Parámetros del algoritmo Conectivity-Binary 2.

Cuando tenemos completado el algoritmo y funciona correctamente, elaboramos un *sript* con para la recogida de datos y su almacenamiento en variables auxiliares.

El *script* de la biela es el siguiente:

```
puntos_biela1=Vars.circ_biela[0];
puntos_biela2=Vars.circ_biela[1];

x1 = puntos_biela1[0];
y1 = puntos_biela1[1];

x2 = puntos_biela2[0];
y2 = puntos_biela2[1];


//Ángulos del primer cuadrante

if ( x1 < x2 && y1<y2){
     x = x2 - x1;
     y = y2 - y1;
     ang = y / x;
     ang_rad_biela = Math.atan(ang);
     Vars.ang_biela = (ang_rad_biela* 180) / Math.PI;
}

if ( x1 < x2 && y1==y2){
     Vars.ang_biela = 0;
}

//Ángulos del segundo cuadrante

if ( x1> x2 && y1 < y2){
     x = x1 - x2;
     y = y2 - y1;
     ang = y / x;
     ang_rad_biela = Math.atan(ang);
     angbiela = (ang_rad_biela* 180) / Math.PI;
     Vars.ang_biela = 180 - angbiela;
}

if ( x1 == x2 && y1 < y2){
     Vars.ang_biela = 90;
}

//Ángulos del tercer cuadrante

if ( x1> x2 && y1 > y2){
     x = x1 - x2;
     y = y1 - y2;
     ang = x / y;
     ang_rad_biela = Math.atan(ang);
     angbiela = (ang_rad_biela* 180) / Math.PI;
     Vars.ang_biela = 270 - angbiela;
```

```
}

if ( x1 > x2 && y1==y2){
     Vars.ang_biela = 180;
}

//Ángulos del cuarto cuadrante

if ( x1< x2 && y1 > y2){
     x = x2 - x1;
     y = y1 - y2;
     ang = y / x;
     ang_rad_biela = Math.atan(ang);
     angbiela = (ang_rad_biela* 180) / Math.PI;
     Vars.ang_biela = 360 - angbiela;
}

if ( x1 == x2 && y1>y2){
     Vars.ang_biela = 270;
}

//Conversión a coordenadas del robot

xb = (x1+x2)/2
yb = (y1+y2)/2

Vars.xrobot = 1.004*yb + 263.2;

Vars.yrobot = 0.972*xb - 480.3;

Vars.tipo = 2;
```

Dentro del script analizamos posición de la biela así como el ángulo de inclinación que tiene la misma. El programa de identificación de piezas del sistema de visión es el de la Ilustración 26.

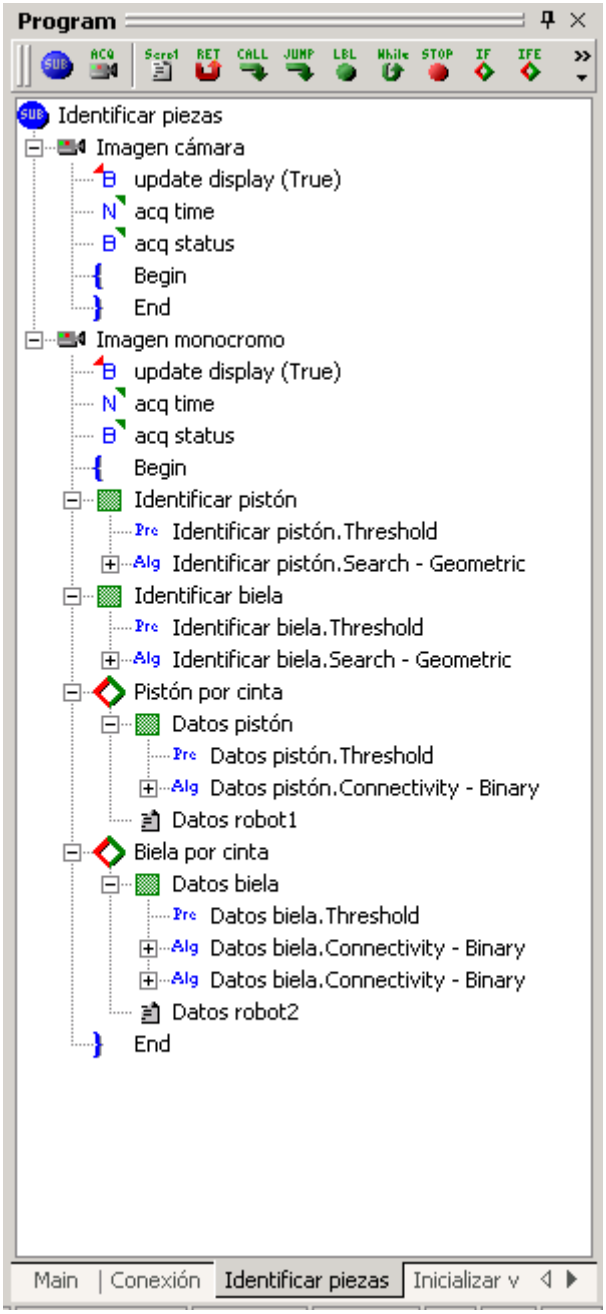


Ilustración 26 Programa de identificación de piezas.

6.2.4 PROGRAMA PRINCIPAL

Lo primer que hacemos es el programa principal es llamar al subprograma "Inicializar variables" y después llamar cal subprograma "Conexión".

Una vez la conexión esté hecha, creamos un bucle para la transmisión de datos con el robot. Dentro de este bucle esperamos que el robot nos pida los datos de la pieza cuando ésta sea detectada por el sensor de posición de la cinta transportadora. Cuando el robot haya solicitado los datos de la pieza, procedemos a identificar la pieza

y a obtener los datos necesarios para enviárselos. Pero antes de enviar los datos, hay que crear una *string* que junte todos los datos en una sola cadena de datos.

Tras crear la *string* con los datos finales la enviamos y volvemos al principio del bucle, es decir, volvemos a esperar a que el robot solicite nuevamente los datos de la pieza que detecte el sensor de posición de la cinta. El código principal del sistema de visión se puede observar en la Ilustración 27.

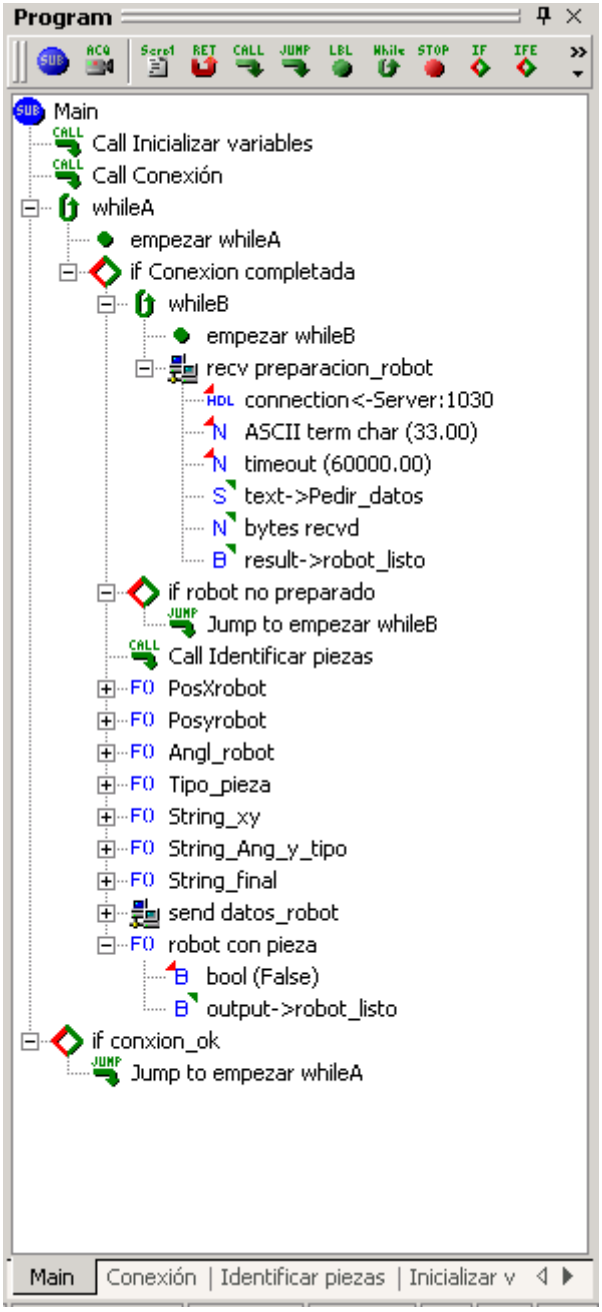


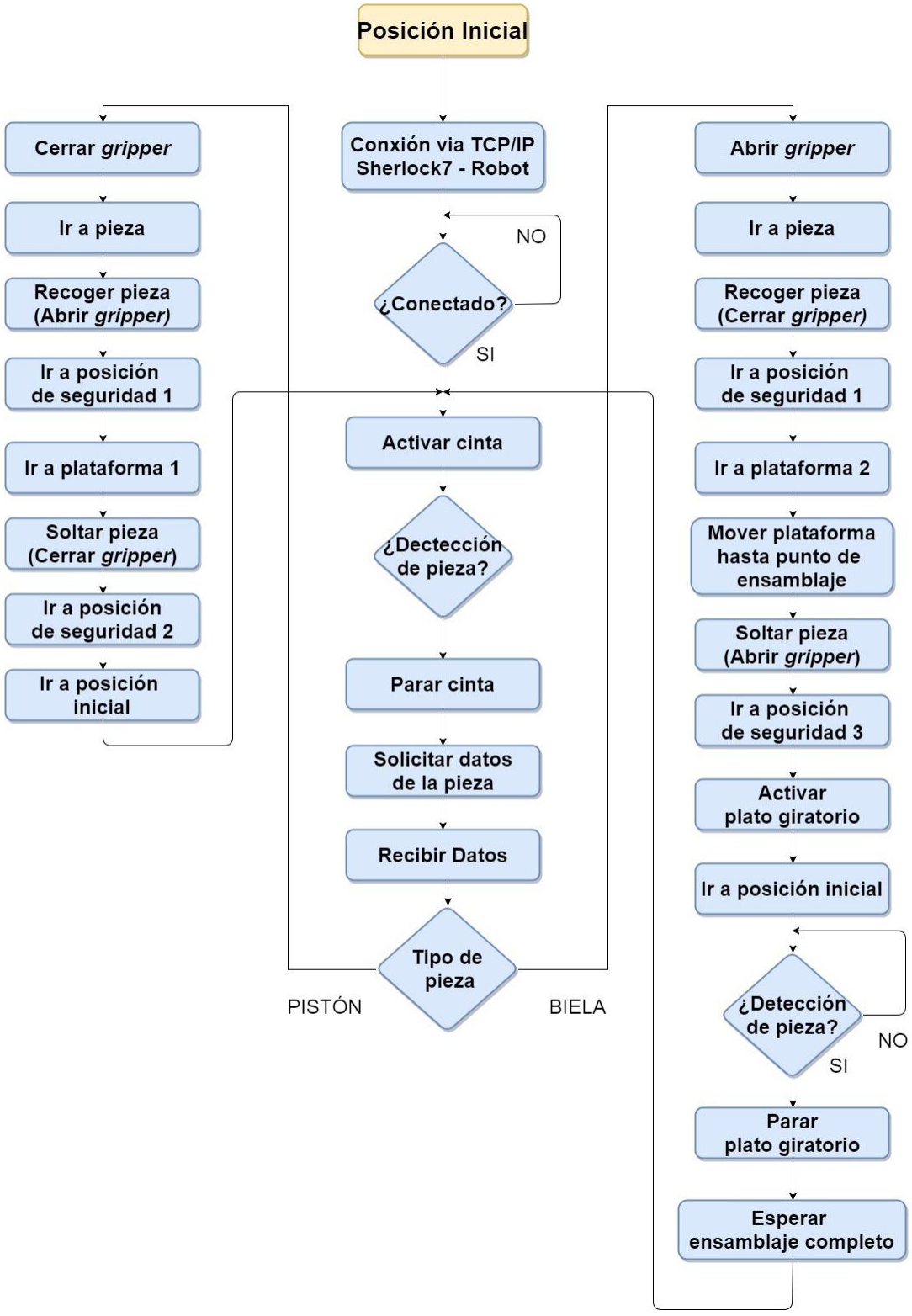
Ilustración 27 Programa principal.

En la realización del programa principal tuvimos diversos problemas. Cuando ejecutábamos el programa una vez no entraba dentro del bucle *while*, por lo que el programa no funcionaba. Para corregir este error direccionamos con un *Jump* desde una función *if* justo después del bucle. Cuando solucionamos este problema descubrimos otro problema. Cuando se ejecutaba el programa lo realizaba bien hasta que el sensor de posición detectaba la segunda pieza. Este problema era causado porque dentro del bucle enviaba datos de la pieza al robot todo el tiempo una vez la pieza era detectada por el sensor. Por lo tanto la cadena de datos se llenaba de información innecesaria hasta el punto de saturarse y cuando llegaba la segunda pieza, *Sherlock7* no enviaba los datos correctos. Para solucionar este problema, simplemente ponemos el valor a 0 de la variable que nos comunica que el robot está listo para recibir datos de la pieza y por tanto sólo ejecuta una sola vez la identificación de piezas cuando el sensor de posición detecta algo.

6.3 PROGRAMACIÓN ROBOTS

IRB 140

Para poder apreciar a simple vista las tareas que realizará el robot IRB 140, se ha elaborado el siguiente diagrama de flujo:



El programa del robot se ha realizado con el *software RobotStudio* que es una herramienta proporcionada por ABB y que permite trabajar fuera de línea y realizar simulaciones. El lenguaje de programación de este programa es *Rapid*, y es un leguaje propio para la programación de robots de la marca ABB.

El programa que se ha desarrollado se puede desglosar en distintas partes:

Fuera del "PROC main()" encontramos:

- Declaración de las variables.
- Declaración de los *robtarget.*

Dentro del "PROC main()" encontramos:

- Movimiento al punto inicial.
- Borrar *Flexpendant*.
- Comunicación con *Sherlock7*.
- Puesta a cero de cinta transportadora y sensores.
- Bucle infinito.
    - Lectura del sensor de la cinta transportadora.
    - Reconocimiento de los datos de la imagen enviados por *Sherlock7*.
    - Ejecución de movimientos según el tipo de pieza.

El único problema que ha surgido durante el desarrollo del código de programación es error de posición en el eje x del sistema de referencia del robot. Cuando *Sherlock7* le enviaba los datos de posición al robot, este no iba a la posición correcta de la pieza. Este error se solucionó observando las posiciones en la consola *Flexpendant*. En primer lugar se observó la posición errónea y después la posición correcta. En segundo lugar se calculó la diferencia de posiciones y se modificó la posición en el código.

La forma en la que se modificó la posición del punto de la pieza es la siguiente:

```
pieza1:=[posx-16.8,posy,380];
pieza.trans:=pieza1;
```

Ilustración 28 Corrección de la posición del robot.

Lo primero que hacemos cuando recibimos las coordenadas de la pieza desde *Sherlock7* es crear un variable tipo "pos" donde guardar las coordenadas "x,y" y además añadimos la coordenada z.

Como se puede observar en la Ilustración 28, la variable "pieza" es del tipo *robtarget* y es punto al que tiene que ir el robot para recoger la pieza. Lo que tenemos que hacer para que el robot recoja la pieza de forma correcta, es cambiar la posición de la variable "pieza" por las coordenadas recibidas vía TCP/IP y añadir la corrección del error de posición en el eje "x". Para cambiar la posición de una variable tipo *robtarget* tenemos que añadir ".trans" después del nombre de la variable e igualarlo a valor que queremos.

Por desgracia el error de posición no se corrige con el mismo valor para el pistón que para la biela por lo que tenemos que realizar el paso anterior dos veces, uno para cada pieza.

UR3

La programación del UR3 no se parece en nada a la programación del IRB 140. Para programar este robot simplemente tenemos que hacerlo mediante su consola de mando, similar a la *Flexpendant* del robot de ABB.

Los comandos para desarrollar el programa se encuentran dentro de la ventana de Estructura, tal como indica la Ilustración 29 .

Ilustración 29 Ventana estructura del UR3.

Una vez elegimos la instrucción que queremos aplicar vamos a la ventana Comando tal como indica la Ilustración 30, donde permite modificar los datos de la instrucción seleccionada

Ilustración 30 Ventana Comando del UR3.

La estructura a seguir el robot es relativamente sencilla.

- Abrir la pinza.
- Ir a la posición inicial.

- Esperar a que detecte la pieza el sensor de posición del plato giratorio

Una vez detecta la pieza:

- Ir a por a posición de seguridad del bulón.
- Agarrar el bulón.
- Sacar el bulón de la plataforma del plato giratorio.
- Ir a la posición de seguridad donde se realiza el ensamblaje.
- Ensamblar
- Ir a posición de seguridad.

El sensor de posición del plato giratorio se encuentra en la entrada digital número 5, y la herramienta para agarrar la pieza se encuentra en la salida digital que tiene el nombre "pinza".

7. CONCLUSIONES

El propósito principal de este proyecto, realizar un sistema automatizado que permita el ensamblaje de un pistón y una biela, se ha podido cumplir.

A lo largo del trabajo han surgido diversas complicaciones que no han impedido completar de forma correcta el proyecto.

En primer lugar, se tuvo que corregir en el código de programación del robot IRB 140 los errores provocados por la calibración; pero una vez resueltos, se consiguió una correcta calibración.

En segundo lugar, se corrigieron los problemas relacionados con el algoritmo del sistema de visión tales como: el error en el envío de datos, error con los comandos "*while*", entre otros, pero una vez resueltos se obtuvo un algoritmo del sistema de visión eficaz.

En tercer lugar, al no sufrir errores significativos en el código de programación de los robots, se pudieron realizar programas que funcionasen correctamente. En general, todas los objetivos que planteamos se han realizado con éxito.

Finalmente, el proyecto se ha desarrollado con éxito, pero esto no quiere decir que esté cerrado a posibles mejoras en un futuro que permitan su implantación en la industria de la automoción.

8. BIBLIOGRAFÍA

[1]
"Actualidad y perspectiva de la robótica." [Online]. Available: http://sisbib.unmsm.edu.pe/bibvirtual/publicaciones/indata/v04_n1/actualidad.htm. [Accessed: 27-Jun-2017].

[2]
"AENOR - Normas, buscador de normas." [Online]. Available: http://www.aenor.es/aenor/normas/buscadornormas/buscadornormas.asp. [Accessed: 27-Jun-2017].

[3]
"Anexos.pdf."[ii] .

[4]
"CB3_basic_training_3.1.ES.pdf."[iii].

[5]
"Dalsa IPD - Vision Appliances - Smart Camera | AES." [Online]. Available: http://www.adept.net.au/cameras/ipd/. [Accessed: 27-Jun-2017].

[6]
"infaimon-sistemas-de-vision-integrados-sistemas-de-vision-integrados-dalsa-ipd-500933.pdf." .

[7]
"ISO 8373:2012 - Robots and robotic devices -- Vocabulary." [Online]. Available: https://www.iso.org/standard/55890.html. [Accessed: 27-Jun-2017].

[8]
"ISO 14539:2000 - Manipulating industrial robots -- Object handling with grasp-type grippers -- Vocabulary and presentation of characteristics." [Online]. Available: https://www.iso.org/standard/24062.html. [Accessed: 27-Jun-2017].

[9]
"JAI CV-M9GE Color GigE camera." [Online]. Available: https://www.1stvision.com/cameras/JAI/CV-M9GE.html. [Accessed: 27-Jun-2017].

[10]
"JAI CV-M77 Color Analog camera." [Online]. Available: https://www.1stvision.com/cameras/JAI/CV-M77.html. [Accessed: 27-Jun-2017].

[11]
"KR 3 AGILUS | KUKA AG." [Online]. Available: https://www.kuka.com/es-mx/productos-servicios/sistemas-de-robot/robot-industrial/kr-3-agilus. [Accessed: 27-Jun-2017].

[12]
"Los robots se escapan de la cadena de montaje," *abc*, 30-Apr-2017. [Online]. Available: http://www.abc.es/sociedad/abci-robots-escapan-cadena-montaje-201704301953_noticia.html. [Accessed: 27-Jun-2017].

[13]
"manualv.pdf."[iv] .

[14]
"PR10031EN R15_En.pdf." [v].

[15]
"Universal Robots lanza el nuevo UR3." [Online]. Available: https://www.universal-robots.com/es/sobre-ur/news-centre/universal-robots-lanza-el-ur3/. [Accessed: 27-Jun-2017].

[16]
"Vision System Software - Teledyne DALSA Inc." [Online]. Available: https://www.teledynedalsa.com/imaging/products/vision-systems/software/. [Accessed: 27-Jun-2017].

[17]
M. Bélanger-Barrette, "What Are the Best Collaborative Robots?" [Online]. Available: http://blog.robotiq.com/what-are-the-best-collaborative-robots. [Accessed: 27-Jun-2017].

9. ANEXO I: Código de calibración

Script Matlab: Homografia.

```
function [H, A] = homografia(X,U)
% [Hest] = homomografia(X,U)
% Calcula la homografia entre dos imagenes de las cuales se tienen puntos que coinciden X y
U
% X - matriz con los puntos en la imagen 1 en coordenadas homogeneas organizados en
columnas
% U - matriz de los puntos en la imagen 2 en coordenadas homogeneas organizados en
columnas
% H - homografía estimada estimada


% Estimacion de la homografia
nPuntos = size(X,2);

A=[];
for (i=1:1:nPuntos)
   A=[A; X(:,i)' zeros(1,3) -U(1,i)*X(:,i)';
      zeros(1,3) X(:,i)' -U(2,i)*X(:,i)'];
end

% Homografia estimada
b = -A(:,9);
a = A(:,1:8);
C = inv(a'*a);
H = C*a'*b;

H=[H;1];
```

```
H=reshape(H,3,3)';
```

Script Matlab: Calibración.

```
%Calibración en el plano de la cinta
% Las tres primeras columnas corresponden a las coordenadas homogeneas de
% los puntos en el mundo respecto al sistema de coordenadas del robot.
% Las tres últimas columnas son las coordenadas homogeneas del mismo punto en la
% imagen respecto del sistema de coordenadas de la imagen

%Kevin camara 1
X=[557.2 -190.8 1 296.76 293.01 1;
   555.0 -287.3 1 197.83 291.15 1;
   498.4 -237.1 1 249.37 233.52 1;
   456.8 -281.4 1 204.69 192.99 1;
   454.4 -185.8 1 302.49 188.87 1;
   408.0 -235.2 1 251.55 143.64 1;
   367.2 -277.8 1 209.61 104.84 1;
   367.4 -184.6 1 305.77 103.62 1;]
x=X(:,1:3)';
u=X(:,4:6)';

H=homografia(x,u);
xe=inv(H)*u;

xee=[xe(1,:)./xe(3,:);xe(2,:)./xe(3,:);xe(3,:)./xe(3,:)];

error=xee-x

x=X(:,1:3)';
u=X(:,4:6)';
H1=homografia(x,u);
xe=inv(H1)*u;

xee=[xe(1,:)./xe(3,:);xe(2,:)./xe(3,:);xe(3,:)./xe(3,:)];
error1=xee-x
```

NOTA: Estos códigos han sido entregados por el tutor.

10. ANEXO II: Programa en RAPID

```
MODULE MainModule

!   Variables

   VAR socketdev canal1;
   VAR string datos_conexion:="";
   VAR string datos_imagen_sherlock:="";
   VAR string confirmacion_datos:="";
   VAR num longitud_trama:=0;
   VAR num posicion_string_inicial:=0;
   VAR num posicion_string_final:=0;
   VAR num vuelta;
   VAR string v:="";
   VAR num posx:=0;
   VAR num posy:=0;
   VAR num angulo:=0;
   VAR num angulo_rob:=0;
   VAR string trozo:="";
   VAR num Len2:=0;
   VAR bool ok;
   VAR num ChPos1:=0;
   VAR num ChPos2:=0;
   VAR Pose giro;
   VAR orient orient1;
   VAR orient orient2;
   VAR Pos posicion;
   VAR Pose posfinal;
   VAR Pose robot;
   VAR pos pieza1;
   VAR pos pieza2;
   VAR num tipo:=0;
   VAR num sensor;



!   Variables tipo robtarget

   VAR robtarget posini:=[[562,100,432.1],[0.00252,0,-1,-0.00194],[0,0,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   VAR robtarget pieza:=[[0,0,0],[0.00252,0,-1,-0.00194],[0,0,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   VAR robtarget
pfinalpiston:=[[670.2,300.8,179.3],[0.01544,0.02086,0.71570,0.69792],[0,0,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   VAR robtarget pfinalbiela:=[[679.3,62.1,283.8],[0.00249,-0.02208,-0.99975,-
0.00185],[0,0,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


PROC main()
```

```
   MoveJ Offs(posini,-50,0,100),v300,z20,tool0;

!    Borrar Flexpendant

   TPErase;

!    Comunicacion con Sherlock7

   SocketCreate canal1;
   SocketConnect canal1, "158.42.16.207", 1030;
   SocketSend canal1\Str:="¡Hola desde el robot!";
   SocketReceive canal1\Str:=datos_conexion;
   SocketSend canal1\Str:="Conexion establecida!";
   WaitTime 0.25;
   TPWrite "----------------------------";
   TPWrite "Conexión correcta";
   TPWrite datos_conexion;
   TPWrite "----------------------------";

!    Codigo principal
!   Resetear la cinta y el sensor de la misma
   Reset CONVEYOR_FWD;
   Reset CONVEYOR_FWD_SENSOR_ENABLE;
   WHILE TRUE DO
      Inicio:
      Set CONVEYOR_FWD;
!   Agarrar la pieza si el sensor la detecta

      sensor := DOutput (CONVEYOR_FWD_SENSOR_ENABLE);
      IF sensor = 1 THEN

         Reset CONVEYOR_FWD;

         !Recococimiento de la string de datos de sherlock7
         SocketSend canal1\Str:="Dame datos de imagen!";
         SocketReceive canal1\Str:=datos_imagen_sherlock;
         TPWrite datos_imagen_sherlock;
         longitud_trama:=StrLen(datos_imagen_sherlock);
         IF longitud_trama>0 THEN
            posicion_string_inicial:=0;
            posicion_string_final:=1;
            vuelta:=0;
            Reset GRIPPER_ENABLE;
            Reset GRIPPER_OPEN;


         WHILE vuelta<4 DO
               ChPos1:=posicion_string_inicial+posicion_string_final;
               v:=StrPart(datos_imagen_sherlock,ChPos1,1);

               WHILE v<>"," DO
                  posicion_string_final:=posicion_string_final+1;
                  ChPos1:=ChPos1+1;
                  v:=StrPart(datos_imagen_sherlock,ChPos1,1);
```

```
        ENDWHILE
        Len2:=posicion_string_final-1;
        ChPos2:=posicion_string_inicial+1;
        trozo:=StrPart(datos_imagen_sherlock,ChPos2,Len2);

        IF vuelta=0 THEN
           ok := StrToVal(trozo,posx);

        ELSEIF vuelta=1 THEN
           ok := StrToVal(trozo,posy);

        ELSEIF vuelta=2 THEN
           ok := StrToVal(trozo,angulo);
        ELSEIF vuelta=3 THEN
           ok := StrToVal(trozo,tipo);
        ENDIF

        posicion_string_inicial:=posicion_string_inicial+posicion_string_final;
        posicion_string_final:=1;
        vuelta:=vuelta+1;
    ENDWHILE

    !Movimiento de las piezas a sus plataformas
    TPWrite "Coordenada X="\Num:=posx;
    TPWrite "Coordenada Y="\Num:=posy;
    TPWrite "Angulo de giro="\Num:=angulo;
    TPWrite "Tipo pieza(Piston->1,Biela->2)="\Num:=tipo;
    TPWrite "----------------------------";
    TPWrite "------Final------";
    TPWrite "----------------------------";
    pieza1:=[posx-16.8,posy,380];
    pieza.trans:=pieza1;
    MoveJ pieza,v200,z0,tool0;
    set GRIPPER_ENABLE;
    IF tipo=1 THEN
       !Mover piston
       MoveL Offs(pieza,0,0,-47),v10,z0,tool0;
       Set GRIPPER_OPEN;
       MoveJ Offs(pieza,0,0,100),v300,z20,tool0;
       MoveJ Offs(pfinalpiston,0,0,50),v200,z10,tool0;
       MoveL Offs(pfinalpiston,0,0,-40),v20,z0,tool0;
       MoveL Offs(pfinalpiston,0,0,-90.2),v5,z0,tool0;
       WaitTime 0.25;
       Reset GRIPPER_OPEN;
       WaitTime 0.25;
       MoveL Offs(pfinalpiston,0,-30,-90.2),v10,z0,tool0;
       MoveL Offs(pfinalpiston,0,-50.5,-90.2),v20,z20,tool0;
       MoveL Offs(pfinalpiston,0,-50.5,-70),v50,z20,tool0;
       MoveJ posini,v200,z100,tool0;
        GOTO inicio;
    ENDIF

    IF tipo=2 THEN
       !Mover biela
```

```
                MoveJ Offs(pieza,0,0,0),v200,z0,tool0;
                Set GRIPPER_OPEN;
                pieza2:=[posx-11,posy,380];
                 pieza.trans:=pieza2;
                MoveL Offs(pieza,0,0,-54),v10,z0,tool0;
                Reset GRIPPER_OPEN;
                MoveJ Offs(pieza,0,0,100),v300,z0,tool0;
                MoveL Offs(pfinalbiela,0,0,222),v200,z10,tool0;
                MoveL Offs(pfinalbiela,0,0,0),v50,z0,tool0;
                MoveL Offs(pfinalbiela,0,0,-28),v5,z0,tool0;
                MoveL Offs(pfinalbiela,-3,326,-28),v20,z0,tool0;
                Set GRIPPER_OPEN;
                MoveJ Offs(pfinalbiela,0,326,-9.1),v10,z0,tool0;
                Set GRIPPER_OPEN;
                MoveJ posini,v200,z200,tool0;
                WHILE TABLE_OBJ_SEN=0 DO
                   Set TABLE_FWD;
                   IF TABLE_OBJ_SEN = 1 THEN
                      Reset TABLE_FWD;
                   ENDIF
                ENDWHILE
                 GOTO inicio;
             ENDIF

          ENDIF
       ENDIF
    ENDWHILE
    SocketClose canal1;
ENDPROC
ENDMODULE
```

11. ANEXO III: Programa robot UR3

- Ajsutar pinza = Encender
> MoveJ
    • Pos_Inicial
- Esperar DI[5] = HI
> Move J
    • Punto_seguridad_1
    • Punto_bulón
    - Esperar 0.2
    - Ajustar pinza = Apagado
    - Esperar: 0.5
    • Punto_seguridad_2
    • Punto_seguridad_3
> MoveL
    • Punto_ensamblaje
    - Ajustar pinza = Encender
    - Esperar: 1
    • Punto_seguridad_4

[i] Es la zona rectangular con los bordes de color verde y es el lugar de trabajo en el que podemos ejecutar los algoritmos.
[ii] Documento de calibración entregado por el tutor.
[iii] Manual de usuario del robot UR3 entregado por el tutor.
[iv] Documento relacionado con el lenguaje de programación KRL y RAPID con dirección: http://www.etitudela.com/profesores/rpm/rpm/downloads/manualv.pdf
[v] Data sheet IRB 140: https://library.e.abb.com/public/98ba43a906331fec48257c6f00374818/PR10031EN%20R15_En.pdf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT

PLANOS DEL PROYECTO

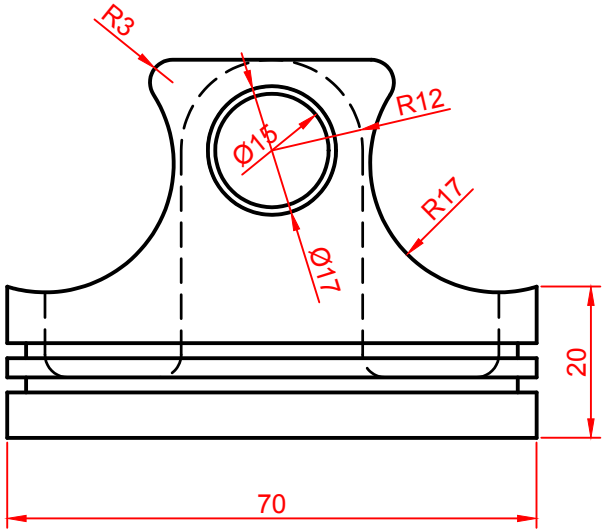
Universidad Politécnica de Valencia

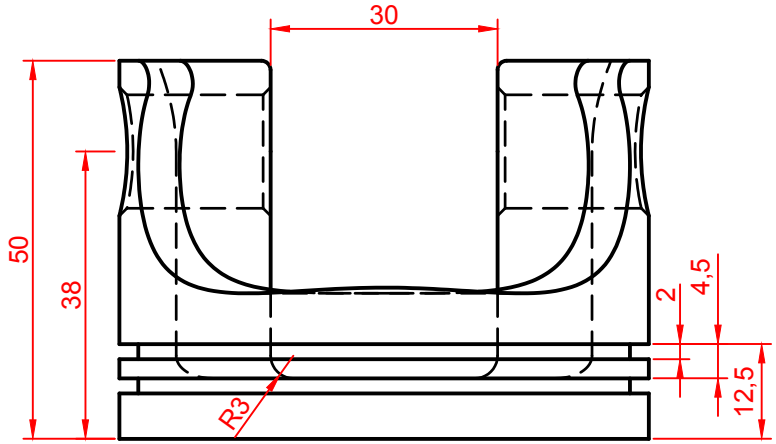
Escuela Técnica Superior de Ingeniería del Diseño | ETSID

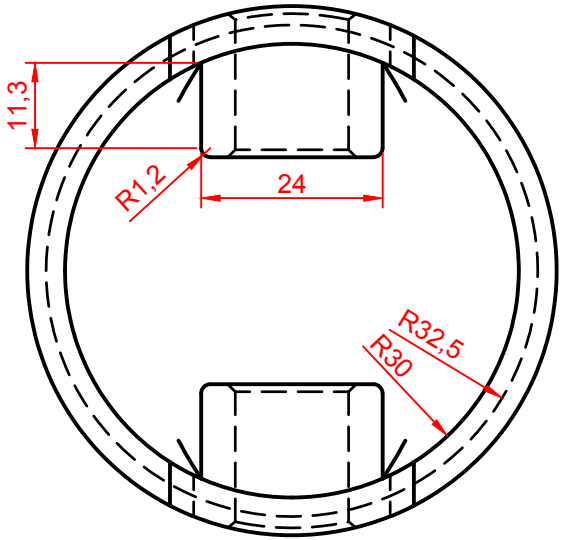
Grado en Ingeniería Electrónica Industrial y Automática

10 de Julio de 2017

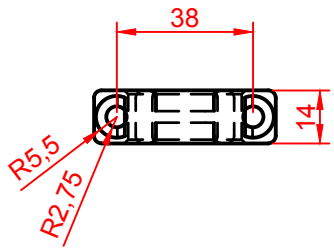
CONTENIDO

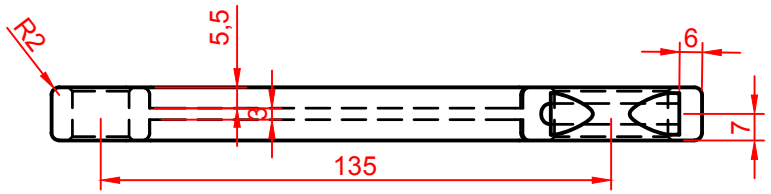


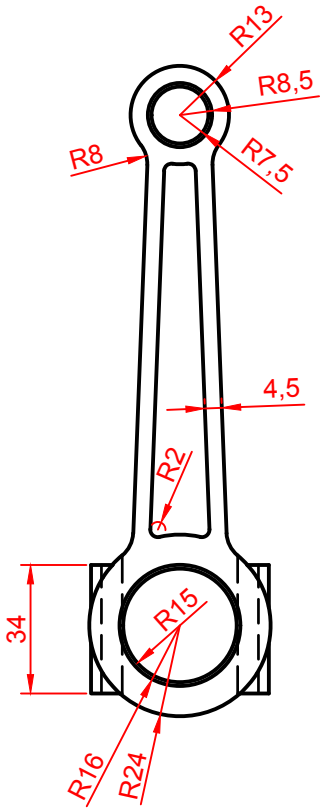




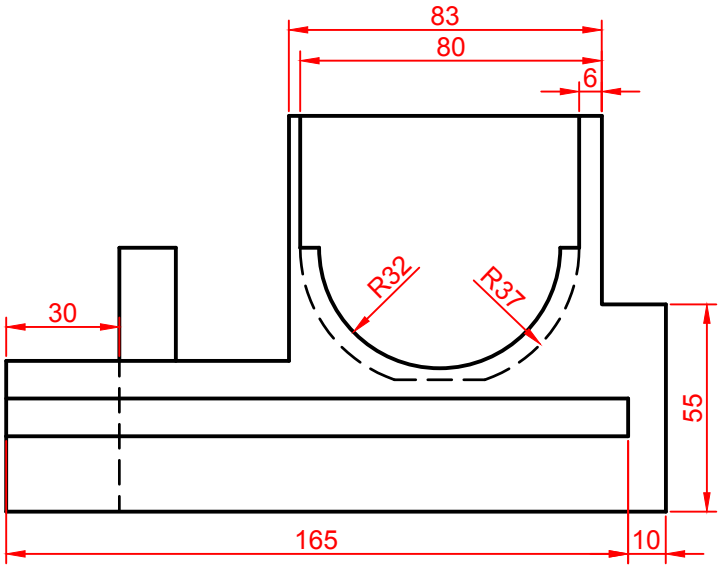
REFERENCIA	NOMBRE PLANO	FECHA
P-001	PISTÓN	01/07/2017
PROMOTOR: TRABAJO FINAL DE GRADO		ESCALA 1:1
AUTOR: VILLALBA DUARTE, KEVIN DANIEL		
TÍTULO DEL PROYECTO: ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT		Nº PLANO 001

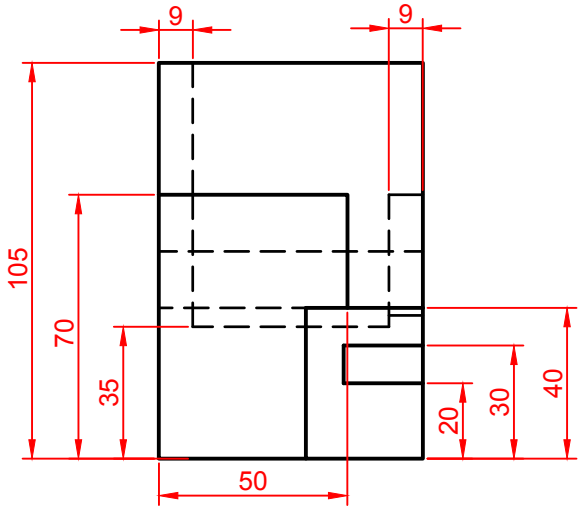


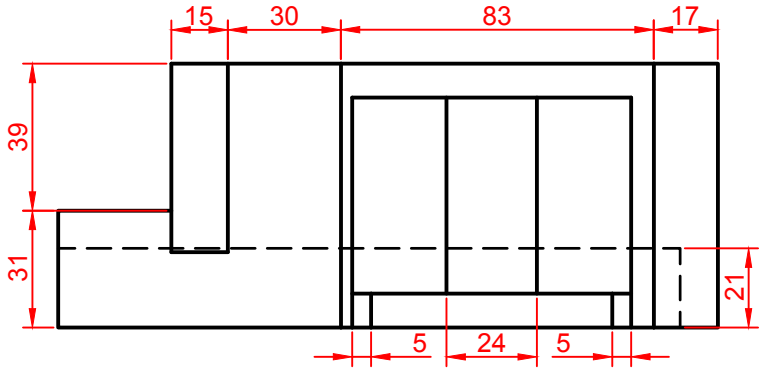




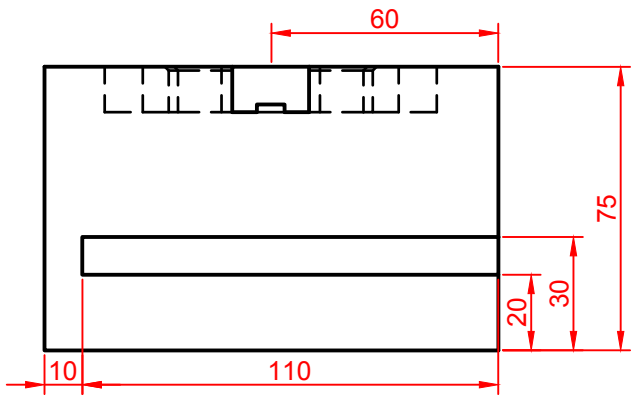
REFERENCIA	NOMBRE PLANO	FECHA
P-002	BIELA	01/07/2017
PROMOTOR: TRABAJO FINAL DE GRADO		ESCALA 1:2
AUTOR: VILLALBA DUARTE, KEVIN DANIEL		
TÍTULO DEL PROYECTO: ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT		Nº PLANO 002

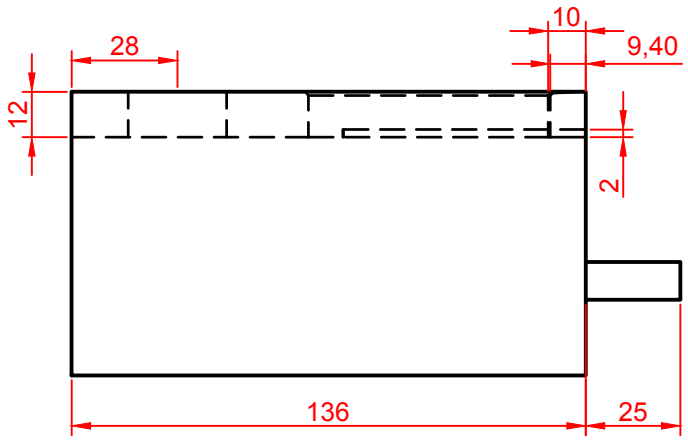


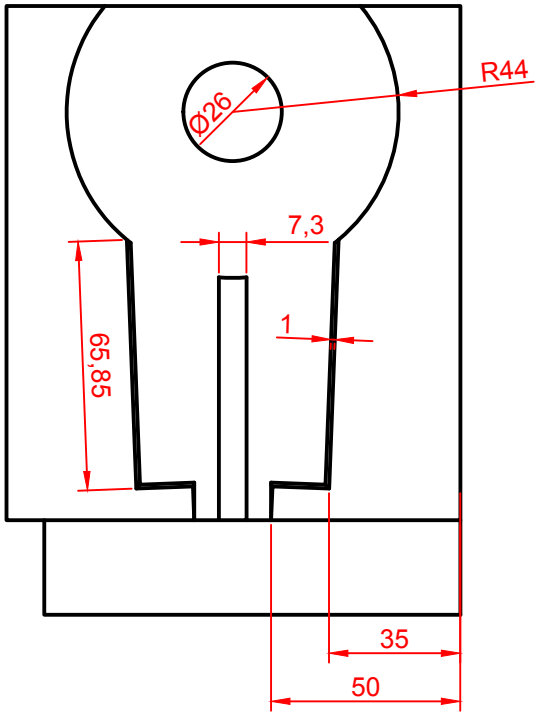




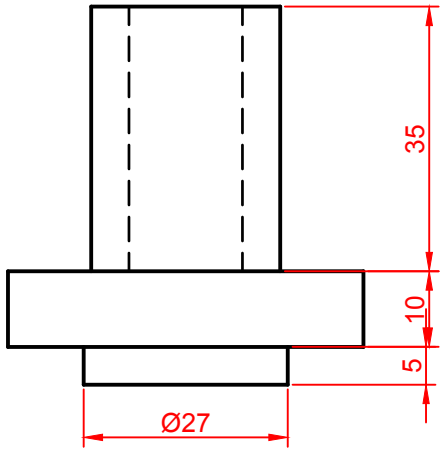
REFERENCIA	NOMBRE PLANO	FECHA
P-003	PLATAFORMA PISTÓN	01/07/2017
PROMOTOR: TRABAJO FINAL DE GRADO		ESCALA 1:2
AUTOR: VILLALBA DUARTE, KEVIN DANIEL		
TÍTULO DEL PROYECTO: ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT		Nº PLANO 003

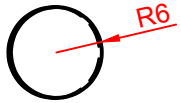


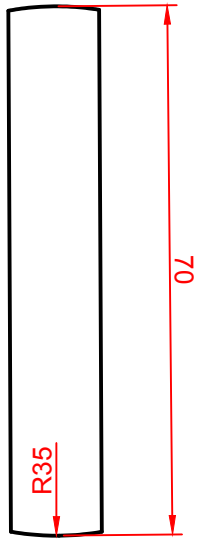


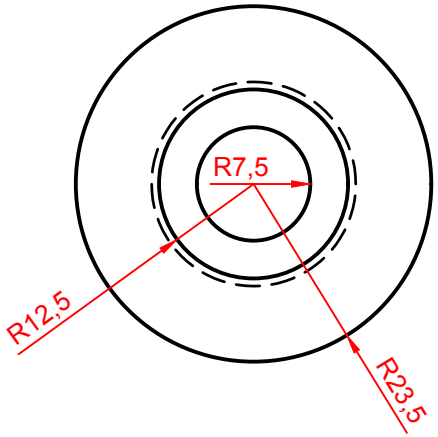


REFERENCIA	NOMBRE PLANO	FECHA
P-004	PLATAFORMA BIELA	01/07/2017
PROMOTOR: TRABAJO FINAL DE GRADO		ESCALA 1:2
AUTOR: VILLALBA DUARTE, KEVIN DANIEL		
TÍTULO DEL PROYECTO: ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT		Nº PLANO 004









REFERENCIA	NOMBRE PLANO	FECHA
P-005	PLATAFORMA BULÓN Y BULÓN	01/07/2017
PROMOTOR: TRABAJO FINAL DE GRADO		ESCALA 1:1
AUTOR: VILLALBA DUARTE, KEVIN DANIEL		
TÍTULO DEL PROYECTO: ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT		Nº PLANO 005

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

ENSAMBLAJE AUTOMATIZADO PISTÓN-BIELA MEDIANTE DOS BRAZOS ROBOT

PRESUPUESTO DEL PROYECTO

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño | ETSID

Grado en Ingeniería Electrónica Industrial y Automática

10 de Julio de 2017

Contenido

1. CUADRO DE PRECIOS ELEMENTALES

En el siguiente documento se va a elaborar el presupuesto del automatismo planteado donde se verá reflejado el coste total del proyecto.

En el Convenio para la Industria, la Tecnología y los Servicios del Sector del Metal de Valencia 2015-2016 y Estatal 2017 se encuentran las retribuciones salariales según la profesión. Como graduado en Ingeniería Electrónica Industrial y Automática me corresponde la categoría de Técnico.

Concepto	Importe
Salario base + Plus de convenio	1660.5 €
Gratificaciones especiales (Navidad y Vacaciones)	290.10 €
Seguridad Social (Contingencias comunes, formación profesional, accidentes de trabajo y enfermedad profesional, fondo garantía salarial)	902.83 €
TOTAL ANUAL	34359.96 €
TOTAL MENSUAL	2863.33 €
Por jornada	95.44 €
Por hora	11.93 €

Tabla 1 Gastos de la empresa por la contratación del ingeniero.

El tiempo que se ha tardado en completar el proyecto ha sido de dos meses. Pero el tiempo estimado de la dedicación del ingeniero en Electrónica Industrial y Automática ha sido de 270 horas. Como resultado, el precio de la mano de obra asciende a la cantidad que aparece en la tabla 2.

Trabajador	Precio (€/h)	Tiempo de trabajo (h)	Importe (€)
Ingeniero	11.93	270	3221.1 €
		TOTAL MANO DE OBRA	**3221.1 €**

Tabla 2 Costes de mano de obra.

A continuación se pasará a detallar los costes de la maquinaria y hardware que han sido utilizados para elaborar el proyecto.

Concepto	Cantidad	Precio (€)
Robot industrial IRB 140	1	20000 €
Robot industrial UR3	1	10500 €
IPD	1	3000 €
Cámara Jai CV-M77	1	545 €
TOTAL MAQUINARIA Y HARDWARE		**34045 €**

Tabla 3 Costes de maquinaria y hardware.

2. CUADRO DE PRECIOS DESCOMPUESTOS

En este apartado indicaremos las distintas tareas a realizar con sus respectivos gastos.

Unidad	Descripción	Cantidad	Precio	Parcial
	Diseño de la trayectoria de los robots.			
	Materiales			
Ud.	Robot industrial IRB 140	1	20000 €	20000 €
Ud.	Robot industrial UR3	1	10500 €	10500 €
	Mano de obra			
h.	Programación de los autómatas.	4	11.93 €	42.72 €
h.	Corrección de errores.	2	11.93 €	23.86 €
			PRECIO TOTAL DE LA EJECUCIÓN	**30566.58 €**

Tabla 4 Diseño de la trayectoria de los robots.

Unidad	Descripción	Cantidad	Precio	Parcial
	Acondicionamiento del sistema de visión.			
	Materiales			
Ud.	IPD	1	3000 €	3000 €
Ud.	Cámara Jai CV-M77	1	545 €	545 €
	Mano de obra			
h.	Diseño el algoritmo de visión.	3	11.93 €	35.79 €
h.	Corrección de errores.	1	11.93 €	11.93 €
			PRECIO TOTAL DE LA EJECUCIÓN	**3592.72 €**

Tabla 5 Acondicionamiento del sistema de visión.

Unidad	Descripción	Cantidad	Precio	Parcial
	Calibración del sistema.			
	Mano de obra			
h.	Recopilación de datos	0.5	11.93 €	5.965 €
h.	Desarrollar ecuaciones de calibración.	0.5	11.93 €	5.965 €
			PRECIO TOTAL DE LA EJECUCIÓN	**11.93 €**

Tabla 6 Calibración del sistema.

3. VALORACIÓN DEL PRESUPUESTOS

El coste total del proyecto es la suma de los importes de las distintas tareas que se van a realizar, tal como indica la tabla 7.

Descripción	Importe
Diseño de la trayectoria de los robots.	30566.58 €
Acondicionamiento del sistema de visión.	3592.72 €
Calibración del sistema.	11.93 €
TOTAL (Sin I.V.A)	34171.23 €
I.V.A (21 %)	7175.95 €
PRESUPUESTO TOTAL	**41347.18 €**

Tabla 7 Presupuesto total del proyecto.