UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

ESTUDIO E IMPLEMENTACIÓN DE UN SISTEMA DE PICK AND PLACE DE PIEZAS EN CINTA TRANSPORTADORA USANDO VISIÓN POR COMPUTADOR Y ROBOTS INDUSTRIALES

Documento:
Memoria

Autor/Autora:
Lucas Pastor

TRABAJO DE FIN DE ESTUDIOS

Director/Directora - Codirector/Codirectora:
Rita Maria Planas Dangla

Titulación:
Máster universitario en Ingeniería de Sistemas Automáticos y Electrónica Industrial

Convocatoria:
Prórroga Otoño, 2023.

Resumen

El trabajo consiste en desarrollar una aplicación completa con dos robots para poder testear la mayor cantidad posible de las funcionalidades de éstos en un caso práctico.

Por un lado, tendremos una estación en la que llegarán piezas por una cinta transportadora y que se identificarán empleando visión artificial. Una vez identificadas, las piezas serán recogidas por un robot y depositadas en otra cinta de salida.

Dicha cinta será la entrada de la otra estación, en la que las piezas serán detectadas por un sensor fotoeléctrico, que se activará cada vez que pase una pieza por debajo. Esta señal se mapeará a otro robot que recogerá dicha pieza y la colocará en la cinta de entrada del robot que emplea visión para la detección. Esta pieza no se colocará siempre en la misma posición, sino que se generará un número aleatorio para decidir su coordenada horizontal.

Abstract

En este trabajo emplearemos sistemas de visión artificial integrados para la detección de piezas que pasan por una cinta transportadora. Veremos la importancia de los parámetros de una cámara en función del tipo de pieza que queremos identificar, y la importancia de factores externos como la iluminación, la velocidad de la cinta de entrada, etc…

También utilizaremos técnicas de seguimiento de cinta transportadora, utilizando la lectura de encoders incrementales. De esta forma las cintas pueden seguir en marcha sin tener que detenerse.

La finalidad es testear el software de los robots en pruebas de larga duración. Las pruebas de larga duración son necesarias para testear el software porque permiten simular situaciones de uso real a largo plazo y detectar problemas que no se pueden detectar en pruebas de corta duración. Estas pruebas pueden ayudar a los desarrolladores de software a encontrar problemas tales como: “memory leaks”, “segmentation fault”, etc…

In this work we will use integrated artificial vision systems for the detection of pieces that pass through a conveyor belt. We will see the importance of the parameters of a camera depending on the type of part that we want to identify, and the importance of external factors such as lighting, the speed of the conveyor belt, etc...

We will also use conveyor belt tracking techniques, using the reading of incremental encoders. In this way the conveyor belts can continue running without having to stop.

The purpose is to test the software of the robots in long-term tests. Long-term tests are necessary for testing software because they allow you to simulate real-world usage situations over the long term and detect problems that cannot be detected in short-term tests. These tests can help software developers to find problems such as: “memory leaks”, “segmentation fault”, etc…

Sumario

RESUMEN **1**

ABSTRACT **1**

SUMARIO **3**

ÍNDICE DE TABLAS **5**

ÍNDICE DE FIGURAS **6**

LISTA DE ABREVIATURAS/GLOSARIO **8**

1. INTRODUCCIÓN **9**

1.1 OBJETO 9
1.2 ALCANCE 9
1.3 REQUISITOS 9
1.4 JUSTIFICACIÓN 9

2 ANTECEDENTES Y/O REVISIÓN DEL ESTADO DE LA CUESTIÓN **10**

3 METODOLOGÍA **11**

4 PLANTEAMIENTO Y DECISIÓN SOBRE SOLUCIONES ALTERNATIVAS **12**

4.1 ROBOTS UTILIZADOS 12
4.1.1 SCARA *12*
4.1.2 Seis ejes *12*
4.1.3 Colaborativos *13*
4.1.4 Delta *13*
4.1.5 Conclusión *14*
4.2 ACTUADORES PARA MANIPULACIÓN DE PIEZAS 14
4.2.1 Conexión de los actuadores *15*
4.3 CÁMARAS DE VISIÓN 17
4.4 SENSOR FOTOELÉCTRICO 18
4.5 ENCODERS 19
4.6 PLC 19
4.6.1 Periféricos E/S para controlar las cintas desde el robot *19*
4.7 CINTAS TRANSPORTADORAS 20
4.7.1 Sistema mecánico de las cintas *20*
4.7.2 Motores de las cintas *21*

5 DESARROLLO DE LA SOLUCIÓN O SOLUCIONES ESCOGIDAS **22**

5.1 PROGRAMA DE PLC PARA EL CONTROL DE LAS CINTAS 22
5.2 CELDA DE FOTODETECTOR 24
5.2.1 Control del actuador de vacío *24*
5.2.2 Seguimiento de la cinta *24*
5.2.3 Detección de las piezas mediante eventos "latch" *26*
5.2.4 Mecanismo de seguimiento de piezas usando eventos de latch *29*
5.2.5 Programa V+ *32*
5.3 CELDA DE VISÓN 33
5.3.1 Calibración de la cámara *33*
5.3.2 Parámetros de la cámara *34*
5.3.2.1 Color 35
5.3.2.2 Exposure time 36
5.3.2.3 Gain 36
5.3.2.4 Balanceo de colores 36
5.3.3 Selección de piezas *37*

5.3.4 Entrenamiento del localizador de piezas 39
5.3.5 Concepto de VLATCHES en visión 42

6 VÍDEOS DE LA APLICACIÓN EN MARCHA 45

6.1 ESTACIÓN FOTOSENSOR 45
6.2 ESTACIÓN DE VISIÓN 45
6.3 TODAS LAS ESTACIONES EN MARCHA 45

7 RESUMEN DEL PRESUPUESTO Y/O ESTUDIO DE VIABILIDAD ECONÓMICA 46

8 ANÁLISIS Y VALORACIÓN DE LAS IMPLICACIONES AMBIENTALES Y SOCIALES 47

9 CONCLUSIONES 48

10 REFERENCIAS 49

11 ANEXOS 50

11.1 PROGRAMA COMPLETO DEL PLC 50
11.1.1 Homing 50
11.1.2 Power belts 50
11.1.3 Move Belts 51
11.2 PROGRAMA COMPLETO DE LA ESTACIÓN DE FOTO DETECTOR 51
11.3 PROGRAMA COMPLETO DE LA ESTACIÓN DE VISIÓN 56

Índice de tablas

Título y número de todas las tablas por orden de aparición en el texto.

TABLA 1. COMPARATIVA ENTRE DIFERENTES MODELOS 14
TABLA 2. COSTES DEL PROYECTO. 46

Índice de figuras

Título y número de todos los gráficos por orden de aparición en el texto.

Figura 1. Robot i4H (Fuente: https://industrial.omron.es/) 12
Figura 2. Robot Viper (Fuente: https://industrial.omron.es/) 13
Figura 3. Robot Viper (Fuente: https://industrial.omron.es/) 13
Figura 4. Robot iX3 (izquierda) y iX4 (derecha) (Fuente: https://industrial.omron.es/) 14
Figura 5. CobotPump Mini para la generación de vacío eléctrica (Fuente: https://www.schmalz.com/es-es/) 15
Figura 6. TIO cable pinout information (Fuente: https://industrial.omron.es/) 16
Figura 7. Asignación de clavijas del conector de brida de 6 polos (Fuente: https://www.schmalz.com/es-es/)... 17
Figura 8. Resultado final del nuevo conexionado del actuador con el robot. 17
Figura 9. Cámara Omron Sentech STC-MCS250. 18
Figura 10. Sensor fotoeléctrico de detección de piezas. 18
Figura 11. Encoder incremental para el seguimiento de la cinta. 19
Figura 12. PLC NJ-501 Robotics. 19
Figura 13. Cinta con sistema para acoplar el motor integrado. 20
Figura 14. Relación mm/rev del motor del belt en sysmac studio. 21
Figura 15. Conexionado del driver de los motores de las cintas. 21
Figura 16. En verde los elementos del PLC. En rojo la "celda de visión". En azul los componentes de la "celda de fotodetector". 22
Figura 17. Programa en Ladder para el encendido de los servos. 22
Figura 18. Programa en Ladder para el control de la velocidad y sentido de las cintas. Donde XIO son las entradas digitales del PLC conectadas a salidas digitales del robot. 23
Figura 19. Celda de fotodetector. 24
Figura 20. Ejemplo de programa V+ de definición de un "belt" 25
Figura 21. Desplazamiento de la ubicación del belt en función de la lectura de encoder. 26
Figura 3. Ejemplo de programa V+ de definición de una "window belt" 26
Figura 23. Ejemplo de programa V+ para saber si ha habido latches de una señal y encoder concreto. 27
Figura 24. Ejemplo de gestión de las colas de latches de cada encoder. En negro el valor al que apunta la keyword LATCHED, en rojo el valor añadido en ese instante. 28
Figura 25. Ejemplo de gestión de las colas de latches de cada encoder. En negro el valor al que apunta la keyword LATCHED, en rojo el valor añadido en ese instante. 29
Figura 26. Estado inicial de las ubicaciones. Belt y sensor loc están en el mismo lugar. La pieza 1 10 cm aguas arriba del sensor. La pieza 2 a 20 cm. 30
Figura 27. Pasado 1 segundo. Belt loc ha avanzado 10 cm. La pieza 1 también y se encuentra en la misma ubicación que el sensor. La pieza 2 está a 10 cm del sensor. 30
Figura 28. Pasados 2 segundos. Belt loc ha avanzado 20 cm. La pieza también y se encuentra a 10 cm del sensor. La pieza 2 se encuentra donde el sensor. 31
Figura 29. Desplazamiento de la variable belt en función del offset de SETBELT. 31
Figura 30. Programa en bucle para la detección y seguimiento de piezas 32
Figura 31. Esquema de montaje de los componentes 33
Figura 32. Imagen estándar para la calibración de una cámara. 33
Figura 33. Wizard de ACE para la calibración de una cámara. 34
Figura 34. Parámetros de la cámara disponibles en ACE. 35
Figura 35. Escala de grises vs RGB. 35
Figura 36. Variación del tiempo de exposición. 4000 (izquierda), 25000 (centro) y 50000 (derecha) 36
Figura 37. Variación de la ganancia. 4000 (izquierda), 25000 (centro) y 50000 (derecha). 36
Figura 38. Variación de la ganancia de cada color. azul (izquierda), rojo (centro) y verde (derecha). 36
Figura 39. Propiedades captadas por el algoritmo de entrenamiento en un grupo de fichas de póker. 37
Figura 40. Ejemplo de las malas propiedades de las cajas de grapas a la hora de entrenar el modelo. 38
Figura 41. Ejemplo piezas no identificadas por tener un modelo demasiado dependiente de la pieza y la ubicación. 38
Figura 42. Gomas de borrar y fichas de póker identificadas correctamente en una misma imagen. Las gomas tienen la ventaja de que es fácil identificar una orientación 39

FIGURA 43. FOTO ORIGINAL (IZQUIERDA), PROPIEDADES SUGERIDAS POR LA APLICACIÓN SELECCIONANDO EL CONJUNTO DE PIEZAS (CENTRO) Y PROPIEDADES SUGERIDAS SELECCIONANDO ÚNICAMENTE AL MEJOR CANDIDATO (DERECHA). 39
FIGURA 44. "WIZARD" PARA EL ENTRENAMIENTO DEL MODELO. TIENE DOS PESTAÑAS "OUTLINE" Y "DETAILS" CON DIFERENTES PROPIEDADES ENCONTRADAS 40
FIGURA 45. "OUTLINE" SUGERIDO POR LA APLICACIÓN (IZQUIERDA) VS EL QUE FINALMENTE ELEGIMOS (DERECHA). EN LILA SON LOS ATRIBUTOS SELECCIONADOS Y EN ROJO LOS DESCARTADOS 40
FIGURA 46. DETALLES IMPORTANTES DE LA PIEZA. SE IGNORAN (CELESTE) LOS DADOS DE LOS BORDES Y SE MANTIENE (VERDE) LO DEMÁS 41
FIGURA 47. EJECUCIÓN DEL ALGORITMO CON UNA IMAGEN DE MUESTRA. 42
FIGURA 48. CENTRO DE COORDENADAS DE LA CÁMARA DENTRO DE UNA IMAGEN (CONVERSIÓN PÍXEL/MM). 42
FIGURA 49. CENTRO DE COORDENADAS DE LA IMAGEN DE CÁMARA RESPECTO AL ROBOT. 43
FIGURA 50. CÓDIGO SIMPLIFICADO DE LA APLICACIÓN. LA FUNCIÓN "GETINSTANCE" DEVUELVE PULSOS DE ENCODER Y COORDENADAS Y ROTACIÓN DE LA PIEZA EN LA IMAGEN. 43
FIGURA 51. CÓDIGO SIMPLIFICADO DE LA APLICACIÓN. SEGUIMIENTO DE LA CINTA POR PARTE DEL ROBOT HASTA AGARRAR LA PIEZA. 44

Lista de abreviaturas/Glosario

ACE: Automation Control Environment. El software de Omron para interactuar con los robots.

Belt: Abreviación de “Conveyor belt”. Cinta transportadora, es un sistema de transporte continuo formado por una banda continua que se mueve entre dos tambores.

Encoder: dispositivo electromecánico usado para convertir la posición angular de un eje a un código digital.

Latch: Evento de software en el que se guarda el valor actual de los pulsos de un encoder.

Sysmac Studio: Software de Omron para interactuar con los PLC.

1. Introducción

1.1 Objeto

Se pretende conseguir una aplicación que utilice métodos de visión artificial y sensores fotoeléctricos para la manipulación de piezas por parte de robots, y que pueda funcionar de forma autónoma de forma indefinida.

1.2 Alcance

Desarrollar una estación independiente que detecte piezas empleando visión artificial.

Desarrollar una estación independiente que detecte piezas utilizando un sensor fotoeléctrico.

Para acabar, ambas estaciones deben depositar las piezas en la cinta de entrada de la otra estación.

1.3 Requisitos

Se debe llegar a una aplicación autónoma en la que todas las partes sean detectadas y se puedan ejecutar tests de larga duración. Tanto para los robots como para la visión artificial y las cámaras es necesario utilizar productos de la marca Omron.

1.4 Justificación

El objetivo de este trabajo es programar una aplicación de robots que involucre la mayoría de las herramientas disponibles en el entorno de Omron (incluyendo PLC, encoders, cámaras de visión) para simular las aplicaciones que un cliente podría darles.

La finalidad es desarrollar una aplicación para analizar la calidad del software del robot en cada nueva fase de desarrollo. De esta manera se puede tener una mejor visión de lo que es capaz de hacer este producto.

2 Antecedentes y/o revisión del estado de la cuestión

Actualmente en el mercado hay muchos fabricantes de robots industriales, entre los que se destaca: ABB, Fanuc, Kuka, Universal Robots, entre otros.

A día de hoy Omron ya no se encuentra entre los principales, sin embargo, es de las pocas marcas que es capaz de ofrecer una solución integral utilizando únicamente componentes suyos, al tener en su abanico todos los componentes necesarios: robots, PLCs, cámaras, sistemas de visión artificial.

La finalidad de este proyecto es analizar el estado de una solución integral Omron para encontrar sus problemas o puntos débiles y proponer mejoras para hacer de su solución un producto más competitivo.

3 Metodología

Para ello haremos uso de todas las herramientas posibles de las que dispone la empresa Omron para implementar soluciones robotizadas de procesos de producción.

En este caso haremos uso de:

- 2 Robots SCARA de la familia i4H para la manipulación de las piezas.
- Un PLC de la serie NJ conectado por red etherCat a 2 módulos de control de motores para controlar la velocidad de avance de cada cinta.
- Encoders para hacer un seguimiento de la velocidad de las cintas por las que aparecen las piezas.
- Una cámara de visión capaz de identificar las piezas que vendrán por una de las cintas.
- Un sensor fotoeléctrico para detectar el paso de las piezas de la otra de las cintas.
- Software ACE (Automatic Control Environment) para interactuar con el robot. Es decir:
    - Crear aplicaciones específicas del robot utilizando su lenguaje V+.
    - Crear aplicaciones de visión para procesar las imágenes de las cámaras "Sentech" e identificar los objetos que vienen por la cinta.
- Para el agarre de las piezas se utiliza un actuador de vacío de la marca Schmalz.

4 Planteamiento y decisión sobre soluciones alternativas

4.1 Robots utilizados

Omron dispone actualmente de un amplio portafolio de robots industriales y colaborativos de mediano y pequeño tamaño.

4.1.1 SCARA

Dentro de los SCARA se encuentran las siguientes familias de robot: cobra (legacy), i4H y i4L.

Los robots Cobra son la serie de robots originales de la compañía estadounidense Adept, fabricantes del legendario “AdeptOne” lanzado en 1985. Dicha compañía fue comprada por Omron en 2015 para hacerse con su portafolio de robots y crear nuevos modelos.

Luego tenemos las familias i4L y i4H, estos han sido desarrollados por Omron después de adquirir dicha compañía norteamericana.

Las diferencias radicales entre ambos modelos son:

- I4H tiene mayor capacidad de carga, pudiendo levantar una carga útil de hasta 15 Kg, mientras que i4L llega a 5 Kg.
- I4H Tiene la opción de utilizarse como un esclavo etherCat de un NJ controller (el conjunto se llama “Robot integrated controller”) y estar sincronizar hasta 8 robots simultáneos con un único PLC.
- Los i4L por el momento solo se pueden utilizar como “Stand Alone” controller, comunicándose por ethernet/IP con otros robots y con el controlador de visión.
- I4H tiene alcances de 650 mm, 750 mm y 850 mm con opciones de montaje sobre suelo, en pared y techo, mientras que i4L tiene de 350 mm, 450 mm y 550 mm.

Figura 1. Robot i4H (Fuente: https://industrial.omron.es/)

4.1.2 Seis ejes

En cuanto a los seis ejes, Omron dispone del modelo Viper, un “rebrand” de un robot denso. La pega de este modelo es que debido a su elevado precio es de los menos utilizados por los usuarios, con lo que no tiene interés usarlo como objeto de testeo.

Figura 2. Robot Viper (Fuente: https://industrial.omron.es/)

4.1.3 Colaborativos

A pesar de ser robots relativamente populares de Omron y de tener un buen alcance (ya que son robots de 6 ejes) no tiene sentido analizar estos robots ya que su software no está hecho por Omron, sino que es un "rebrand" de los robots colaborativos de la Marca taiwanesa TM. Así que no tiene sentido desarrollar la aplicación con este robot si no es un software desarrollado por Omron.

Figura 3. Robot Viper (Fuente: https://industrial.omron.es/)

4.1.4 Delta

Estos robots son populares y su software está hecho al 100% por Omron. Entre ellos tenemos el modelo iX3 de 3 ejes y el iX4 de 4 ejes, que sustenta el récord de hacer más ciclos de "pick and place" en un minuto.

El inconveniente es que son robots muy pesados y para su instalación requieren de una estructura muy cara y complicada, por lo cual no es posible llevar a cabo la aplicación con ellos.

Figura 4. Robot iX3 (izquierda) y iX4 (derecha) (Fuente: https://industrial.omron.es/)

4.1.5 Conclusión

Para saber cuál es el modelo adecuado tendremos en cuenta los siguientes factores:

- Alcance del robot, ya que tienen que compartir espacio de trabajo, es más cómodo si tienen un gran alcance.
- Popularidad del robot entre los clientes. Nos interesa que sea un robot utilizado, ya que queremos hacer el ensayo con
- Facilidad de la creación del setup.
- Versión del firmware del robot, ya que nuestro objetivo es testear un software que esté actualmente en desarrollo y no un producto Legacy que no va a evolucionar.

Cada familia de robots se va a puntuar con una nota del 1 al 5.

Tabla 1. Comparativa entre diferentes modelos

Tipo	Modelo	Alcance	Popularidad	Facilidad	Firmware	Puntuación
SCARA	Cobra	3	3	4	1	11
SCARA	i4H	5	4	4	5	18
SCARA	i4L	3	5	4	5	17
6 ejes	Viper	5	1	3	4	13
Colaborativos	TM	5	4	5	1	15
Delta	iX3	3	3	1	5	12
Delta	iX4	4	3	1	5	13

Viendo los resultados, la decisión está entre el i4H y el i4L. Nos decantamos por el i4H ya que necesitamos un espacio de trabajo amplio para mover piezas de una cinta a la otra y no queremos encontrarnos con limitaciones por el rango reducido de los i4L.

4.2 Actuadores para manipulación de piezas

Para la manipulación de piezas disponemos de varios tipos de actuadores cada uno basado en un mecanismo diferente: actuadores de vacío, magnéticos y pinzas.

Descartamos los magnéticos porque solo se pueden utilizar con piezas ferromagnéticas y porque tienen problemas para soltar las piezas una vez sujetadas, debido a que siempre se queda un campo magnético residual (histéresis). Es decir, las piezas ligeras se quedan pegadas al imán y se recomienda utilizarlo para piezas de 2 Kg en adelante.

Las pinzas suelen una buena opción cuando se trata de coger piezas con mucho volumen, no es el caso de nuestra aplicación, en la que utilizaremos piezas planas y con poco relieve para ser mejor detectadas por el sistema de visión. Además, las pinzas suelen tardar más que el resto de los actuadores en sujetar la pieza desde que se da la orden de agarre.

Por tanto, nos vamos a centrar en los actuadores de vacío.

Disponemos de dos opciones:

- Utilizar una bomba externa al robot para generar el vacío y simplemente tener un actuador para que modifique el circuito del aire, haciendo que la válvula pase de succionar a soplar y viceversa.
- Utilizar un actuador que tenga una bomba de baja capacidad integrada.

Valorando las dos opciones, nos decantamos por el actuador con el sistema de vacío integrado. Los motivos son la simplicidad, y el precio. Así nos evitamos tener una bomba externa ruidosa y cara que haya que tener en marcha durante toda la aplicación.

En cuanto a la desventaja del tiempo de sujeción de la pieza (el cambio de soplar a succionar es instantáneo con una bomba externa), se puede compensar dando la orden de succión cuando se está efectuando la aproximación vertical a la pieza, ahorrando así el segundo de retardo que tiene dicho mecanismo.

4.2.1 Conexión de los actuadores

El actuador elegido es el actuador de vacío ECBPMi 24V-DC FK de la marca Schmalz. La gran ventaja de este actuador es que no requiere de ninguna bomba externa para generar el vacío, sino que él mismo genera el vacío a partir de una tensión de 24V, simplificando considerablemente el montaje.

Figura 5. CobotPump Mini para la generación de vacío eléctrica (Fuente: https://www.schmalz.com/es-es/)

La secuencia típica para manejar una pieza es la siguiente:

Un ciclo de manejo típico se divide en tres fases: aspiración, soplado, reposo.

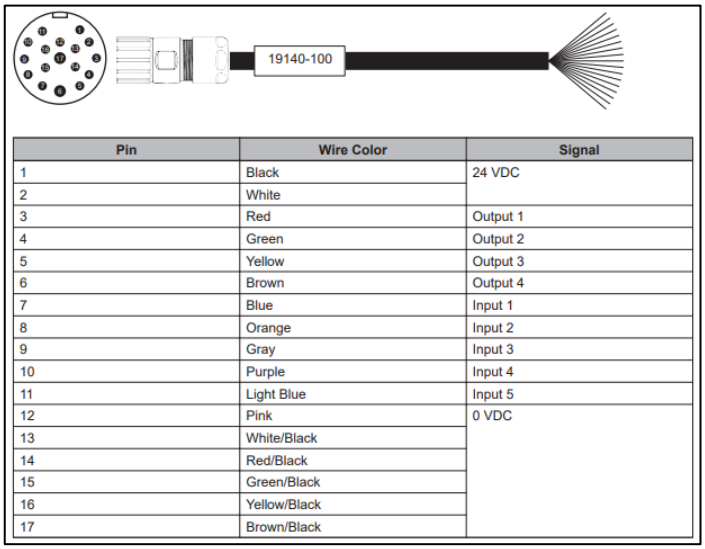
Para controlar si se ha generado suficiente vacío, un sensor de vacío integrado supervisa el valor límite H2 durante la aspiración y lo transmite al control de jerarquía superior a través de OUT2.

Fase	Paso de conmutación	ECBPMi		
		Señal		Estado
1	1	↑	IN1	Aspirar ON
	2	↑	OUT2	Vacío >H2
2	3	↓	IN1	Aspirar OFF
	4	↑	IN2	Soplar ON
3	5	↓	OUT2	Vacío < (H2-h2)
	6	↓	IN2	Soplar OFF

↑ Cambio de estado de la señal de inactivo a activo. ↓ Cambio de estado de la señal de activo a inactivo.

No utilizaremos el conector M-12 con el que viene, sino que conectaremos los cables del conector TIO del robot directamente al conector JST del actuador.

El conector TIO del robot, que dispone de señales de entrada-salida tiene el siguiente mapeado:



Pin	Wire Color	Signal
1	Black	24 VDC
2	White	
3	Red	Output 1
4	Green	Output 2
5	Yellow	Output 3
6	Brown	Output 4
7	Blue	Input 1
8	Orange	Input 2
9	Gray	Input 3
10	Purple	Input 4
11	Light Blue	Input 5
12	Pink	0 VDC
13	White/Black	
14	Red/Black	
15	Green/Black	
16	Yellow/Black	
17	Brown/Black	

Figura 6. TIO cable pinout information (Fuente: https://industrial.omron.es/)

Mientras que el JST del actuador tiene el siguiente:

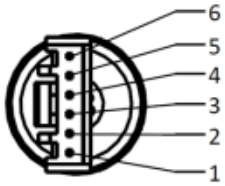
Conector de brida	PIN	Símbolo	Función
	1	U	Tensión de alimentación 24 V
	2	GND	Masa
	3	OUT2	Salida de señal "Part Present" /IO-Link
	4	OUT3	Salida de señal opcional, p. ej., activar Freedrive
	5	IN1	Entrada de señal "Aspirar"
	6	IN2	Entrada de señal "Soplar"

Figura 7. Asignación de clavijas del conector de brida de 6 polos (Fuente: https://www.schmalz.com/es-es/)

Gracias a este montaje podemos pasar el cableado del sensor por el interior del brazo del robot, quedando de la siguiente manera:

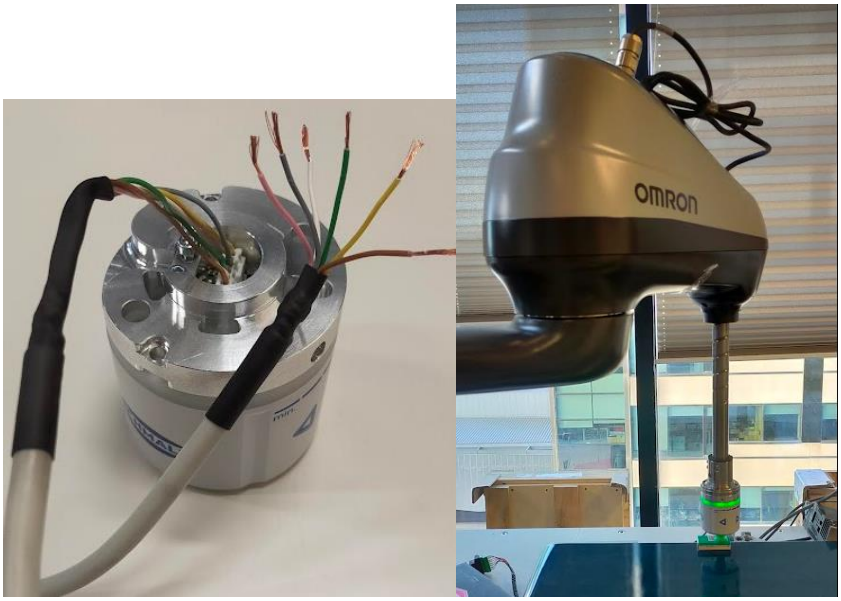


Figura 8. Resultado final del nuevo conexionado del actuador con el robot.

4.3 Cámaras de visión

Las cámaras de visión son las encargadas de detectar las piezas sobre la cinta y de pasar las coordenadas al robot. Por simplicidad y por emular las aplicaciones de los usuarios, vamos a valorar únicamente las marcas que están integradas en el sistema de visión de ACE, que son: Sentech y Basler.

Las cámaras Basler son desarrolladas por una empresa independiente y compatibles con el resto de las soluciones de sistemas de visión del resto de competidores.

Las Sentech, en cambio, son producidas por una empresa adquirida por Omron.

Aunque las cámaras Basler tienen unas mejores prestaciones y son más robustas, queremos testear y asegurarnos de que no haya problemas de integración con las Sentech (ya que son el producto desarrollado por Omron). Además, por pertenecer a la misma compañía tenemos mejores precios y tiempos de entrega.

Figura 9. Cámara Omron Sentech STC-MCS250.

4.4 Sensor fotoeléctrico

El sensor fotoeléctrico es el E3S-DCP24, que se alimenta a 24V y genera una señal de 24V cuando el haz de luz rebotado supera un umbral customizable.

Figura 10. Sensor fotoeléctrico de detección de piezas.

4.5 Encoders

Los encoders utilizados son de tipo incremental de 10.000 pulsos/revolución de la marca TM, modelo TR1-U2R6.

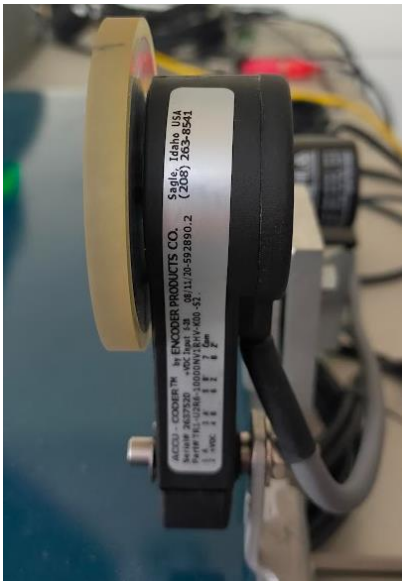


Figura 11. Encoder incremental para el seguimiento de la cinta.

4.6 PLC

El controlador elegido para esto es un NJ-501, con capacidad para conectar hasta 256 elementos en la red etherCAT.

Figura 12. PLC NJ-501 Robotics.

4.6.1 Periféricos E/S para controlar las cintas desde el robot

Para simplificar el testeo de la aplicación es más cómodo dar al robot la posibilidad de controlar la puesta en marcha de las cintas o el sentido. Para ello se conectan unas salidas digitales del robot directamente a las entradas digitales del PLC y se programa el PLC de forma que tenga en cuenta estas señales para el control de los motores.

La solución elegida consiste en conectar un módulo NX-EC203 en la red etherCAT del PLC y conectar sus entradas a las salidas 1 y 2. Dichas señales se mapean a las siguientes variables del PLC, que son tenidas en cuenta en el programa a la hora de controlar el servo, tal y como se ve en la siguiente captura del Ladder.

4.7 Cintas transportadoras

Para esta aplicación vamos a tener 2 cintas independientes.

- Una cinta con un eje al que acoplar un motor.
- Un Motor con su driver para controlar la velocidad y el sentido de cada cinta.
- PLC para controlar los drivers de los motores.

4.7.1 Sistema mecánico de las cintas

Se trata de un montaje bastante estándar y que en este caso se ha solicitado a una empresa externa. Lo único que debemos de considerar es que la relación entre el desplazamiento de la cinta (metros) y las vueltas del eje (grados/radianes).

Figura 13. Cinta con sistema para acoplar el motor integrado.

Lo más sencillo es conectarse con Sysmac Studio al PLC una vez ya tenemos configurado el proyecto (se explica más adelante):

1. Poner las cuentas del encoder del servomotor a 0.
2. Poner una marca en la cinta.
3. Mover el motor una distancia significativa hasta que la marca de la cinta se haya desplazado, por ejemplo, una distancia equivalente a la mitad de la longitud de la cinta.
4. Anotar la distancia desplazada de la cinta y el nuevo valor de grados del encoder.

$$n = \frac{grados\_servo}{distancia\_recorrida}$$

Esta relación es la que guardamos dentro del proyecto de Sysmac Studio.

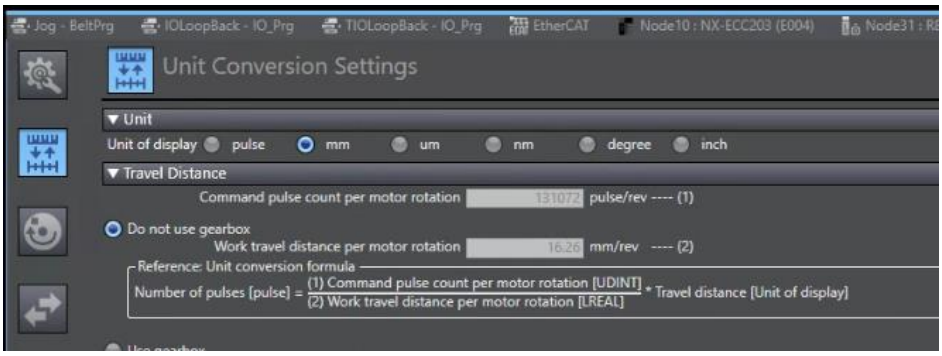


Figura 14. Relación mm/rev del motor del belt en sysmac studio.

4.7.2 Motores de las cintas

Para esto hemos elegido el modelo R88D-KN04H, controlado por EtherCAT por el PLC.

Dicho servo tiene toda una seria de conectores de entrada y salida que permiten configurar parámetros o comportamiento del servo en determinadas situaciones.

En el caso del modelo R88D-KN04H es importante fijarse en el conector BX, ya que por defecto tiene pines de entrada con lógica negada de para de emergencia para cablearle sistemas de seguridad. Como no disponemos de sistema de seguridad, es conveniente modificar este comportamiento por Sysmac para hacer que sean contactos de tipo "nomally open", y por tanto la parada de emergencia se active solo cuando la señal (que no tenemos conectada) tenga un voltaje de 24V.

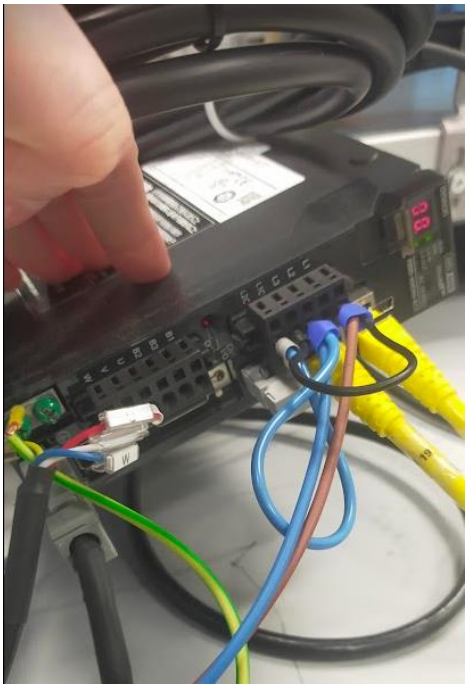


Figura 15. Conexionado del driver de los motores de las cintas.

5 Desarrollo de la solución o soluciones escogidas

La aplicación se puede programar como tres sistemas independientes. Por un lado, el sistema de control de las cintas gestionado por el PLC. Por otro, las celdas de robots, a las que vamos a llamar la “celda de fotodetector” y la “celda de visión” respectivamente.

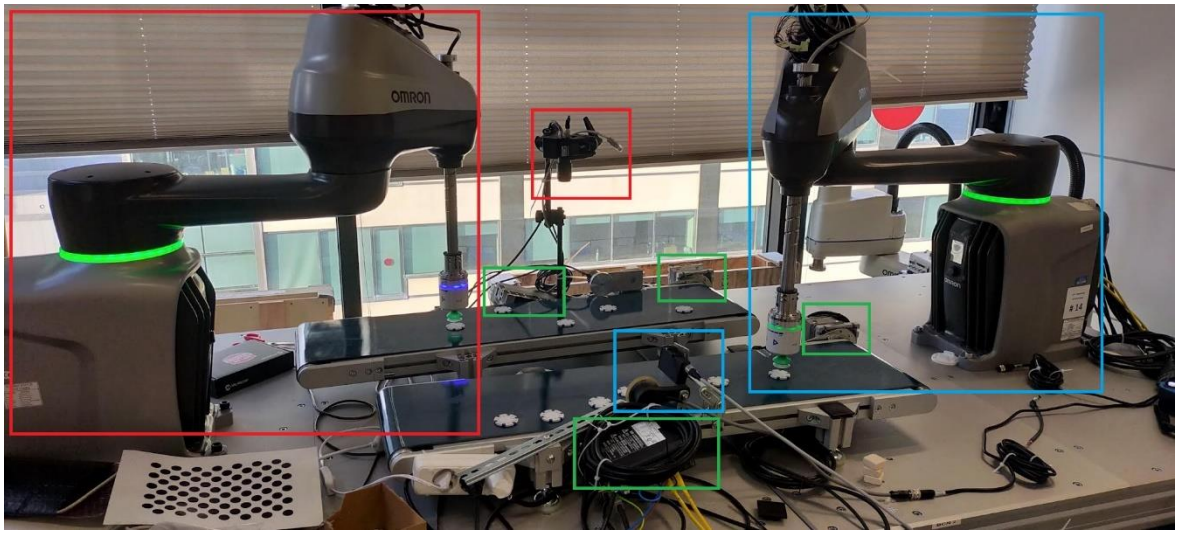


Figura 16. En verde los elementos del PLC. En rojo la “celda de visión”. En azul los componentes de la “celda de fotodetector”.

5.1 Programa de PLC para el control de las cintas

Finalmente tenemos el programa en Ladder para controlar las cintas. Hay una parta para poner en marcha los drivers de los servos y otra para decidir la velocidad y sentido de giro.

El encendido de los servos se hace mediante variables virtuales que se setean desde Sysmac Studio.

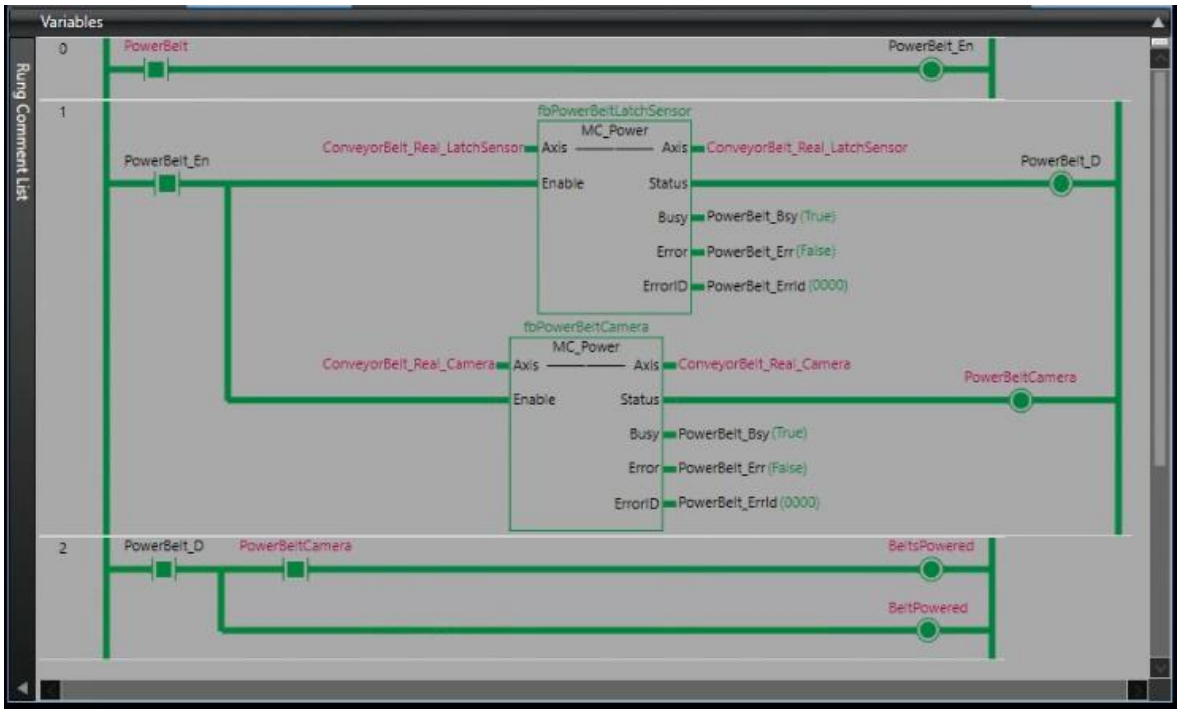


Figura 17. Programa en Ladder para el encendido de los servos.

La velocidad y la dirección de las cintas se puede controlar tanto mediante variables virtuales como mediante señales digitales del robot conectadas al módulo NX.

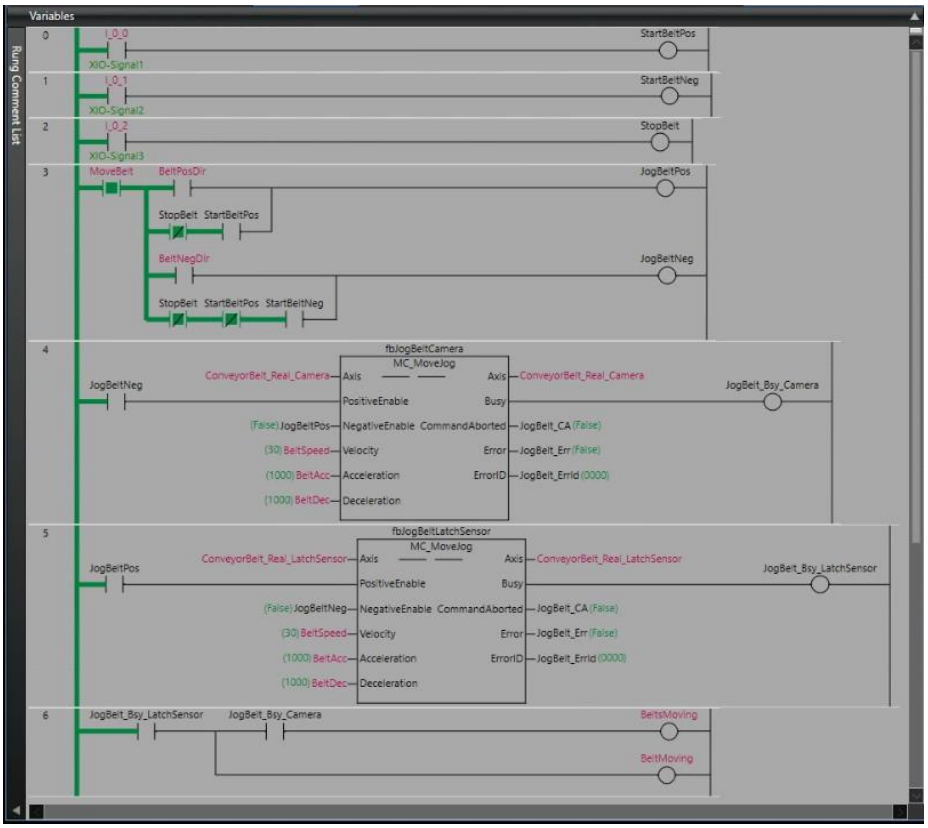


Figura 18. Programa en Ladder para el control de la velocidad y sentido de las cintas. Donde XIO son las entradas digitales del PLC conectadas a salidas digitales del robot.

5.2 Celda de fotodetector

Vamos a empezar con esta celda al tratarse de la más sencilla. Está compuesta por los siguientes elementos:

- Un robot i4H-850-210.
- Un actuador de vacío accionado por las señales digitales del robot.
- Una cinta controlada por el PLC.
- Un encoder conectado al robot.
- Un sensor fotoeléctrico conectado a las entradas digitales del robot.

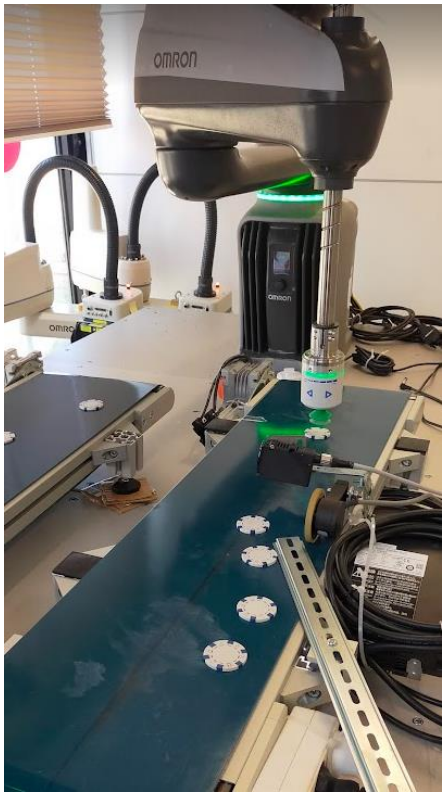


Figura 19. Celda de fotodetector.

Las piezas serán transportadas por el robot desde la cinta del fotodetector hasta la cinta de visión.

5.2.1 Control del actuador de vacío

Para agarrar las piezas simplemente hay que activar la salida de robot número 17 que está conectada a la señal de succión del actuador. Para soltar la pieza quitamos tensión de la señal 17 y activamos la 18 que está conectada a la señal de soplar del sensor.

5.2.2 Seguimiento de la cinta

Para sincronizar el robot con la cinta por la que llegan las piezas tenemos un encoder conectado directamente al conector de encoders del robot. No es necesario tener uno en la cinta en la que se depositan las piezas, ya que las podemos dejar caer a una distancia corta y no dará problemas.

Para definir una cinta en V+ necesitamos la siguiente información:

- Una coordenada del origen de la cinta. Puede ser cualquiera, pero yo prefiero ubicarla donde se encuentra el sensor de detección de piezas, así sabemos que cuando se detecta una pieza, se encuentra en el origen le belt. Es importante la orientación de la coordenada, ya que eso define el sentido del movimiento de la cinta y la inclinación que pueda tener.

- La relación entre los pulsos de encoder y el desplazamiento lineal de la cinta.
- El conector de robot al que hemos conectado el encoder (se puede hacer tracking de 2 cintas a la vez).
- La señal digital a la que está conectado el detector fotoeléctrico, para ir creando eventos de tipo "latch" cada vez que haya una detección.

Una vez definido esto, podemos crear un programa sencillo para hacer tracking de un belt:

```
; Define Belt
enc_radius = 31.83
pulses_rev = 10000
quad_factor = 4
scale_factor = 2*PI*enc_radius/(pulses_rev*quad_factor)  ; mm per revolution = 2*PI*radius

SET sensor_loc = TRANS(315,647,151,0,180,-90) ; belt movement direction along positive x axis
belt_num = 1

DEFBELT %belt1 = sensor_loc, belt_num, 0, scale_factor    ; apply scale factor for belt tracking

MOVES %belt1:NULL
```

Figura 20. Ejemplo de programa V+ de definición de un "belt".

Ahora se pueden ejecutar instrucciones de movimiento lineal respecto a una variable %belt. Esto hace que el robot se mueva a la ubicación de la coordenada de belt + un offset en el eje x del valor actual del encoder multiplicado por el factor definido en la variable belt en modo continuo. Es decir, si el valor del encoder cambia el robot recalculará la posición y se desplazará a la nueva coordenada. Si el encoder está cambiando continuamente porque el belt está en movimiento lo que veremos será que el robot se está desplazando de forma lineal a la misma velocidad del belt.

Un ejemplo para entender mejor, si tenemos que:

$$Belt_{origin} = TRANS(300, 100, 200, 0, 180, 0)$$
$$scale\ factor = 1\frac{mm}{pulse}$$
$$encoder\ counts = 50\ pulses$$

Una variable te tipo "TRANS" se define como:

$$T_1 = coordenada\ x\ [mm]$$
$$T_2 = coordenada\ y\ [mm]$$
$$T_3 = coordenada\ z\ [mm]$$
$$T_4 = primera\ rotación\ en\ eje\ z\ [degrees]$$
$$T_5 = rotación\ en\ eje\ x\ [degrees]$$
$$T_6 = segunda\ rotación\ en\ eje\ z\ [degrees]$$

Por tanto, la variable del origen de la cinta tiene su eje X alineado con el sentido positivo del eje Y, lo que implica que el desplazamiento del "belt" lo vamos a sumar el desplazamiento a la coordenada Y.

$$desplazamiento_y = encoder\ counts \cdot scale\ factor = 1\frac{mm}{pulse} \cdot 50\ pulses = 50\ mm$$

$$Belt_{current\ location} = Belt_{origin} + desplazamiento_y = TRANS(300, \mathbf{150}, 200, 0, 180, 0)$$

En la siguiente figura podemos verlo de una forma gráfica:

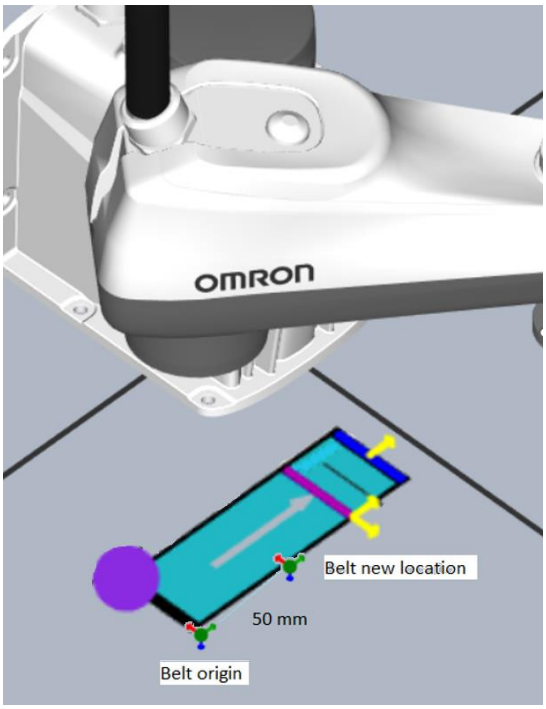


Figura 21. Desplazamiento de la ubicación del belt en función de la lectura de encoder.

Adicionalmente podemos definir una ventana de trabajo de la cinta especificando dos puntos que corresponden al "upper limit" y "lower limit". Esto nos permite definir el comportamiento del robot cuando tiene que ir a buscar una pieza que está aguas arriba de nuestro rango de trabajo, o una que en cambio ya ha salido de nuestro entorno por no haberla agarrado a tiempo.

En nuestro caso programamos el comportamiento de: esperar si la pieza está aguas arriba del "upper limit", y lanzar un error de programa si la pieza se nos ha escapado aguas abajo.

Para definir una ventana la instrucción correspondiente es así:

```
SET upper_limit = TRANS(315,447,151,0,180,-90)
SET lower_limit = TRANS(315,447,151,0,180,-90)
WINDOW %belt1 = upper_limit, lower_limit
```

Figura 22. Ejemplo de programa V+ de definición de una "window belt".

5.2.3 Detección de las piezas mediante eventos "latch"

Para la detección de las piezas vamos a utilizar un sensor fotoeléctrico que activará su salida de 24V cada vez que pase una pieza por debajo. Aprovechamos el contraste de color entre la cinta de color azul oscuro y las piezas de color blanco, para hacer que se active la salida cuando haya más luz del sensor reflejada.

La salida del sensor se conecta a la entrada digital del robot correspondiente a la señal 1010.

Lo siguiente es configurar en evento de tipo "belt latch" asociado a esta señal, de forma que cada vez que se active la señal, se guarde el valor actual de los pulsos de encoder del belt.

Con esto se consigue hacer el tracking de las piezas según van llegando de una forma más eficiente. El controlador automáticamente va guardando los latches de forma interna y el programa de robot simplemente debe ir comprobando si hay o no algún latch guardado.

Para ello utilizaremos las keywords LATCHED y ENCLATCH.

LATCHED devuelve la señal digital para la que se ha registrado un evento para un determinado encoder.

Por otro lado, la keyword ENCLATCH devuelve el valor registrado de pulsos para el último evento para un determinado encoder.

Por ejemplo, si queremos saber si ha habido latches para el encoder 1 del robot, utilizaremos esta sintaxis. Es necesario ver si el evento ha sido ocasionado por la señal que esperábamos, ya que podríamos tener más de una señal asociada a eventos de tipo latch utilizando un mismo encoder (por ejemplo tuviéramos 2 sensores en la misma cinta en ubicaciones diferentes y haciendo trigger de señales diferentes).

```
desired_signal = 1010
encoder_num = 1
signal_latched = LATCHED(encoder_num)

IF (signal_latched == desired_signal) THEN
    latched_puls_mm = enclatch(encoder_num)
END
```

Figura 23. Ejemplo de programa V+ para saber si ha habido latches de una señal y encoder concreto.

Para comprender mejor el funcionamiento, vamos a poner un ejemplo en el que tenemos 2 encoders a la vez, conectados a 2 cintas moviéndose a diferentes velocidades constantes (el número de pulsos de cada encoder va aumentando de forma lineal), y tenemos 3 señales (1001, 1002 y 1005) asociadas a los siguientes eventos de latch:

- Flanco de subida de la señal 1001: guarda la posición de encoder 1.
- Flanco de subida de la señal 1002: guarda la posición de encoder 2.
- Flanco de subida de la señal 1005: guarda la posición de encoder 1 y 2.

Cada encoder tiene una cola asociada y cada vez que se genera un evento de latch, se guarda el valor actual de los pulsos y la señal que ha hecho trigger.

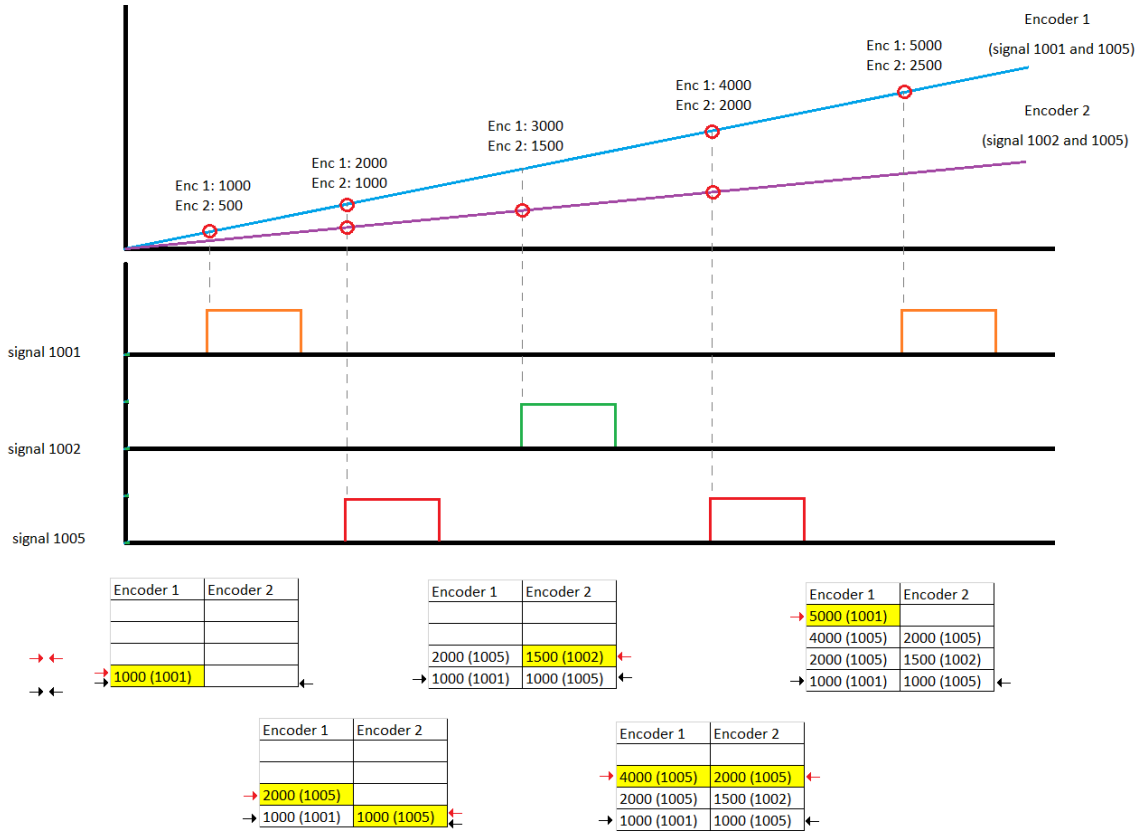


Figura 24. Ejemplo de gestión de las colas de latches de cada encoder. En negro el valor al que apunta la keyword LATCHED, en rojo el valor añadido en ese instante.

Cada vez que utilizamos la keyword LATCHED() sobre un encoder, estamos haciendo un "pop" de la cola de este. Cada vez que ejecutamos la instrucción estamos desechando el último valor para leer el nuevo.

Es por eso, que cuando consultamos si ha habido eventos de latch para un encoder en concreto, debemos consultar la señal que ha generado el evento.

Si tomamos como ejemplo el estado de las colas al acabar el ejemplo anterior, vemos qué valores toman las keywords LATCHED y ENCLATCH.

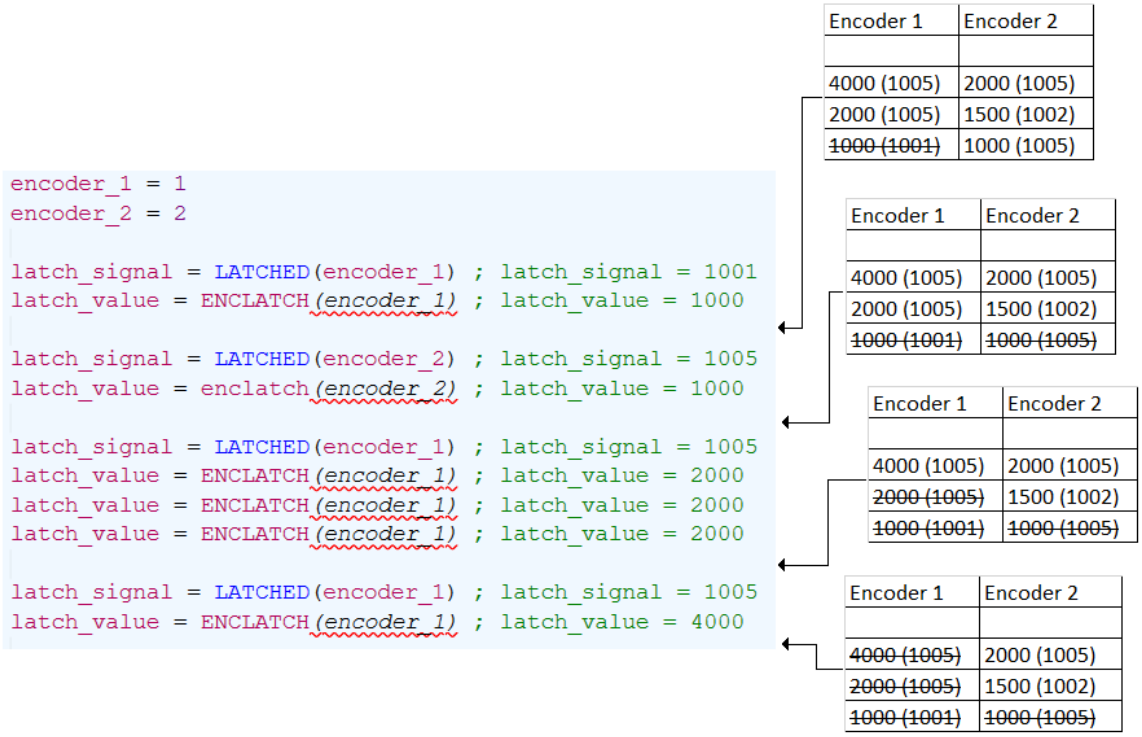


Figura 25. Ejemplo de gestión de las colas de latches de cada encoder. En negro el valor al que apunta la keyword LATCHED, en rojo el valor añadido en ese instante.

5.2.4 Mecanismo de seguimiento de piezas usando eventos de latch

Para entender cómo funciona el mecanismo latch, asumamos que:

- El programa apenas empieza y el encoder se reinicia a cero pulsos.
- Hemos definido la posición del belt donde se encuentra el sensor de latch.
- El belt se mueve a una velocidad constante de 10 cm/segundo.
- El encoder tiene una relación de 1 pulso/cm (10 pulso 10 cm).

Si ejecutáramos la instrucción "MOVES %belt" el robot se movería a la posición del sensor y haría seguimiento de la cinta en función de cómo varíe la lectura del encoder.

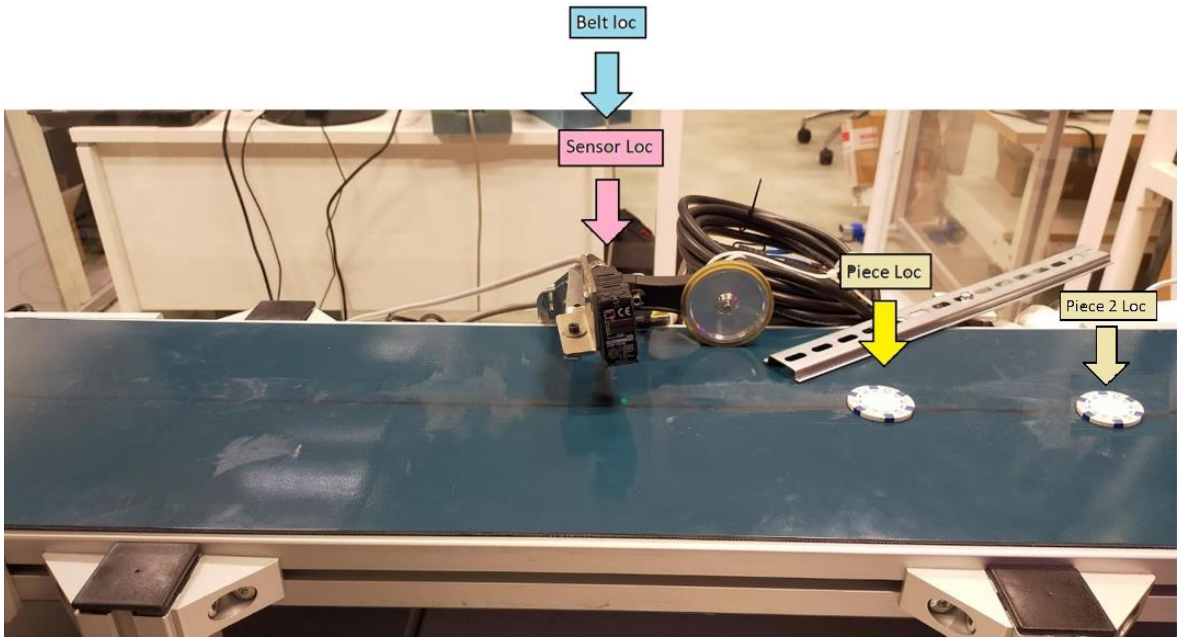


Figura 26. Estado inicial de las ubicaciones. Belt y sensor loc están en el mismo lugar. La pieza 1 10 cm aguas arriba del sensor. La pieza 2 a 20 cm.

Si transcurrido 1 segundo, pasa una pieza por debajo del sensor, la cinta habría avanzado 10 cm con lo que el encoder estaría marcando 10 pulsos. Se activaría la señal del sensor fotoeléctrico y se guardaría automáticamente un evento de latch marcando 10 pulsos.

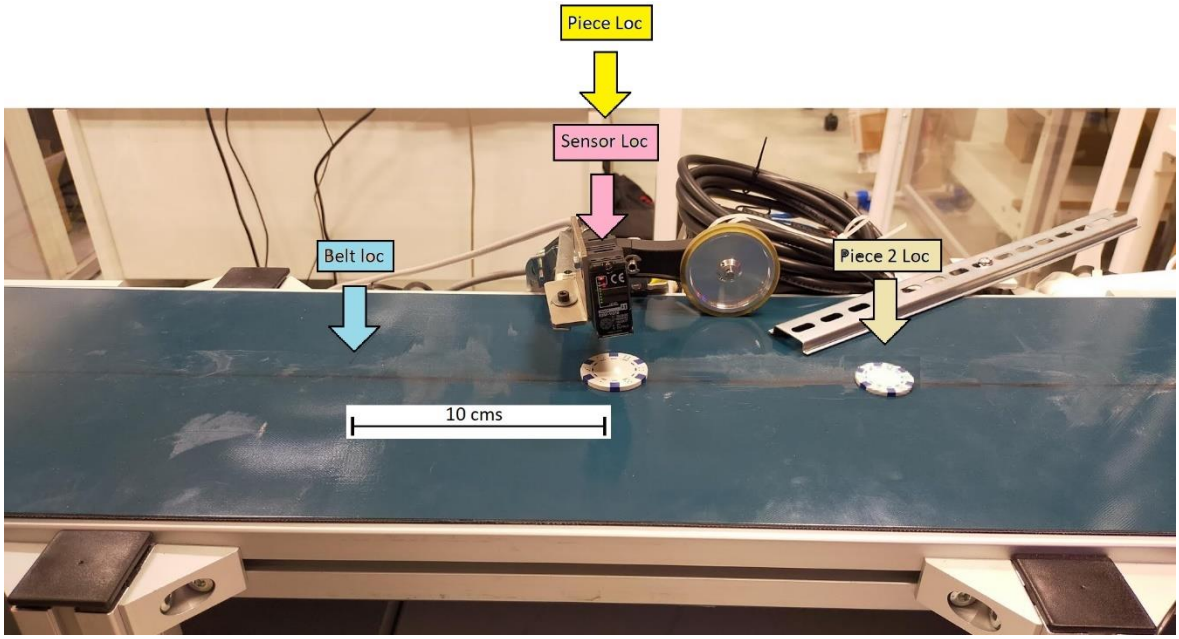


Figura 27. Pasado 1 segundo. Belt loc ha avanzado 10 cm. La pieza 1 también y se encuentra en la misma ubicación que el sensor. La pieza 2 está a 10 cm del sensor.

Si transcurrieran otro segundo, la cinta habría avanzado 10 m más, encontrándose a 20 cm del sensor, lo mismo estaría haciendo la posición del belt. La pieza, en cambio, se encontraría a 10 cm de distancia del sensor.

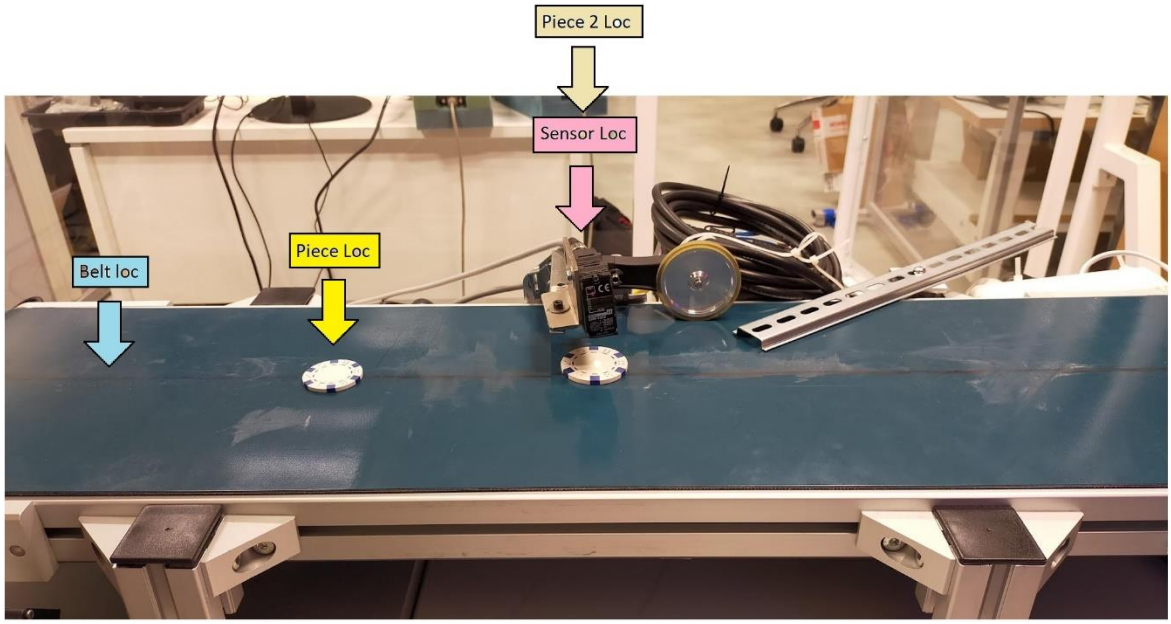


Figura 28. Pasados 2 segundos. Belt loc ha avanzado 20 cm. La pieza también y se encuentra a 10 cm del sensor. La pieza 2 se encuentra donde el sensor.

Aquí es donde podemos entender el funcionamiento de ENLATCH y SETBELT, con la keyword SETBELT podemos añadir un offset al contador de pulsos, para así poder desplazar la variable pos belt a donde queramos.

Si utilizamos la keyword SETBELT, estamos añadiendo un offset a la posición actual del belt en el eje de las X:

- Sin SETBELT: mantiene la ubicación del belt donde está, a 20 cm del sensor.
- SETBELT(10): traslada la ubicación del belt 10 cm aguas arriba, coincidiendo con la ubicación actual de la pieza.
- SETBELT(20): traslada la ubicación del belt 20 cm agua arriba, haciendo que coincida con la ubicación actual del sensor.

De forma gráfica:

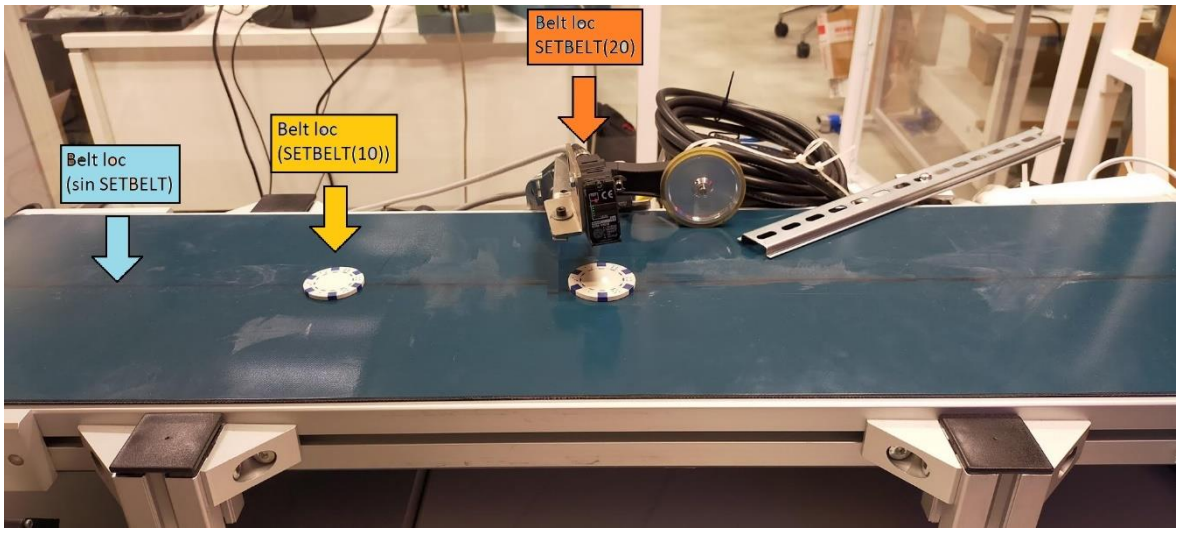


Figura 29. Desplazamiento de la variable belt en función del offset de SETBELT.

Si nos damos cuenta:

- SETBELT(ENCLATCH1): traslada la ubicación del belt a la ubicación de la pieza independientemente de cuándo se haga.
- SETBELT(ENCLATCH2) traslada la ubicación del belt a la ubicación del sensor de la segunda pieza.

5.2.5 Programa V+

A continuación, tenemos un programa sencillo de V+ para recoger una pieza a una coordenada específica de una cinta en movimiento:

```
; Initialize variables
CLEAR.LATCHES (-belt_num)

; Infinite loop: place all pieces in Belt
WHILE do_pick_place == TRUE DO
    last_latch = LATCHED(-belt_num) ; Check if we have seen any LATCH signal

    IF (last_latch == 1011) THEN
        ; Syncronize Belt with latched piece position
        travel_dist = ENCLATCH(belt_num) ; returns mm read when the last latch took place
        SETBELT %belt1 = travel_dist/scale_factor ; apply offset in encoder counts

        ; Pick the piece when in window
        pick_z_offset = 50
        APPROS %belt1:NULL, pick_z_offset
        BREAK
        MOVES %belt1:NULL
        CALL suck_piece()
        BREAK

        ; Stop tracking and move piece to place location
        DEPARTS pick_z_offset
        BREAK
        MOVES HERE ; this is to detach from belt tracking
        BREAK
        LEFTY
        APPRO place_loc, pick_z_offset
        MOVES place_loc
        BREAK
        CALL release_piece()

        DEPARTS pick_z_offset
        RIGHTY
    ELSE
        ; Come back to waiting position
        MOVE wait_loc
    END
END
```

Figura 30. Programa en bucle para la detección y seguimiento de piezas.

Para ver el programa completo, mejor irse al anexo correspondiente. Aquí solo se encuentra la lógica principal.

5.3 Celda de visón

La celda de visión, que está compuesta por:

- Una cámara de visión conectada por ethernet/IP al PC.
- Un PC en el que se ejecuta ACE Vision Server.
- Un robot i4H-650-210 conectado al PC por ethernet/IP.
- Una cinta controlada por el PLC.
- Un encoder conectado al robot.

Figura 31. Esquema de montaje de los componentes

La detección de las piezas se realiza mediante la aplicación de visión de ACE. Esta va analizando continuamente las imágenes de la cámara y cada vez que detecte una pieza nueva va a guardar un evento de tipo "latch" que contiene la información

5.3.1 Calibración de la cámara

El primer paso para tener la cámara correctamente es calcular la relación píxeles/mm que hay en cada fotografía. Esto se hace utilizando una imagen estándar de puntos separados homogéneamente, que se puede buscar en internet.

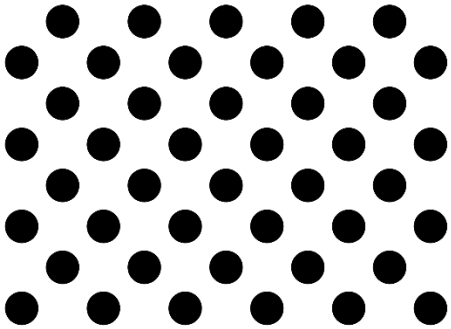


Figura 32. Imagen estándar para la calibración de una cámara.

Esta misma imagen se imprime y se coloca debajo de la cámara de forma que los puntos ocupen el máximo del campo de visión. Luego se le dice a ACE la distancia de punto a punto.

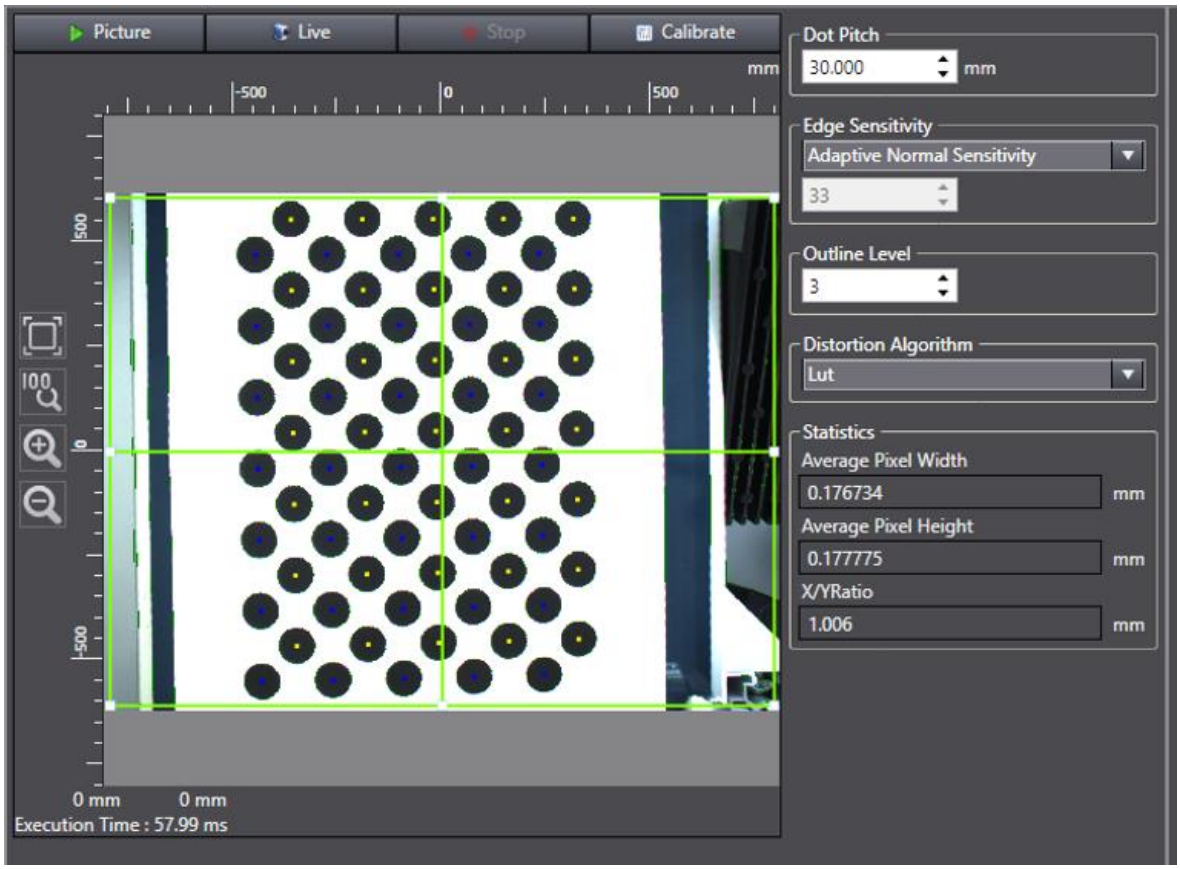


Figura 33. Wizard de ACE para la calibración de una cámara.

5.3.2 Parámetros de la cámara

Para una detección más eficiente podemos customizar una serie de parámetros de la cámara.

ACE permite modificar los siguientes parámetros, que son suficientes para una calibración de la imagen. En caso de que quisiéramos modificar más propiedades de la cámara, tendríamos que conectarnos a ella a través del software de Sentech, pero no es recomendable cambiarlos a menos que tengamos mucho conocimiento.

Esta es la parte más compleja, ya que no existe una única solución ideal, sino que hay una mejor combinación de parámetros dependiendo de factores como: la pieza a identificar, la luz ambiente, la velocidad de la cinta…

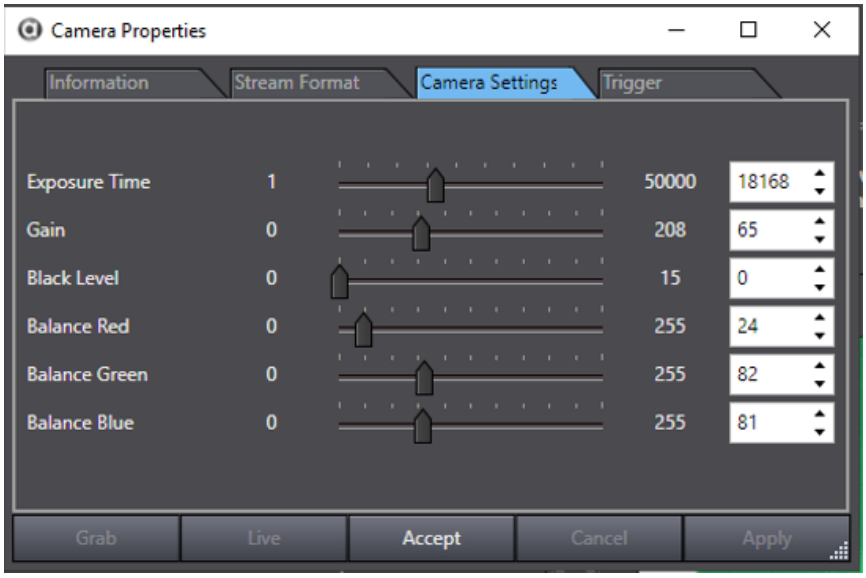


Figura 34. Parámetros de la cámara disponibles en ACE.

5.3.2.1 Color

Si el color de la pieza no es importante, es recomendable utilizar imágenes en blanco y negro para gastar menos CPU en el procesado de la imagen. Si tenemos piezas blancas sobre una cinta de color oscuro o viceversa, es recomendable utilizar escala de grises.

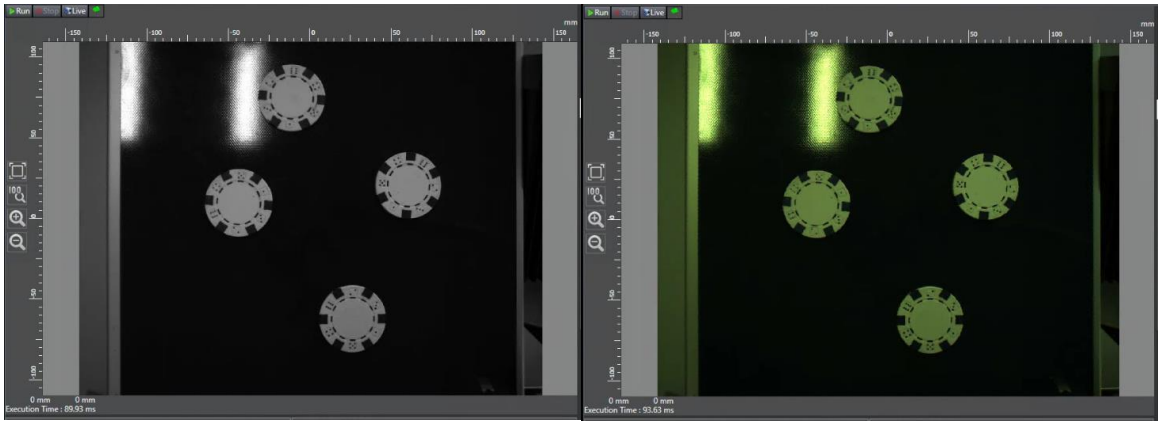


Figura 35. Escala de grises vs RGB.

5.3.2.2 Exposure time

Nos permite calibrar el tiempo que está abierto el objetivo de la cámara para tomar una foto. Cuánto más tiempo, más iluminada estará la foto. Sin embargo, tener un tiempo muy elevado es desaconsejable, ya que las imágenes quedan borrosas cuando tenemos objetos en movimiento. Como en nuestro caso queremos identificar piezas en movimiento en una cinta, lo mejor es asignar un valor entre 3000 y 4000 (valor obtenido a base de prueba y error y consultado el rango típico con ingenieros de campo).

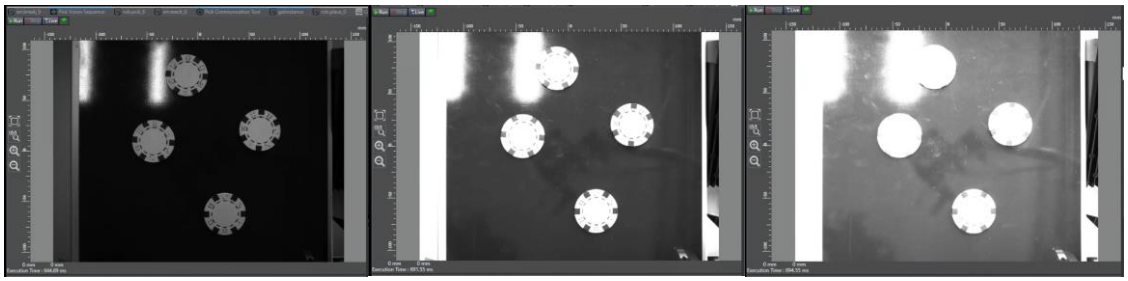


Figura 36. Variación del tiempo de exposición. 4000 (izquierda), 25000 (centro) y 50000 (derecha).

5.3.2.3 Gain

Este parámetro amplifica la luz captada por el objetivo. Un valor nulo nos dará una imagen muy negra, mientras que el máximo nos dará una imagen casi completamente blanca y los contornos mal definidos.

A continuación, vemos cómo cambia una misma foto en función de este parámetro.

Este factor es bueno para resaltar los detalles de nuestra pieza a identificar.

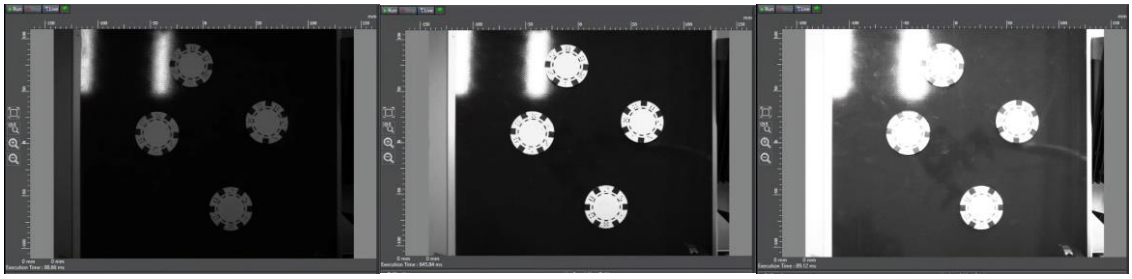


Figura 37. Variación de la ganancia. 4000 (izquierda), 25000 (centro) y 50000 (derecha).

5.3.2.4 Balanceo de colores

Este atributo es útil cuando tanto la pieza como la cinta tienen colores diferentes pero que cuestan de distinguir en una escala de grises. Sin embargo, esto no aplica a nuestro ejemplo, ya que solo vamos a utilizar una escala de grises y las piezas son blancas mientras que la cinta es oscura.

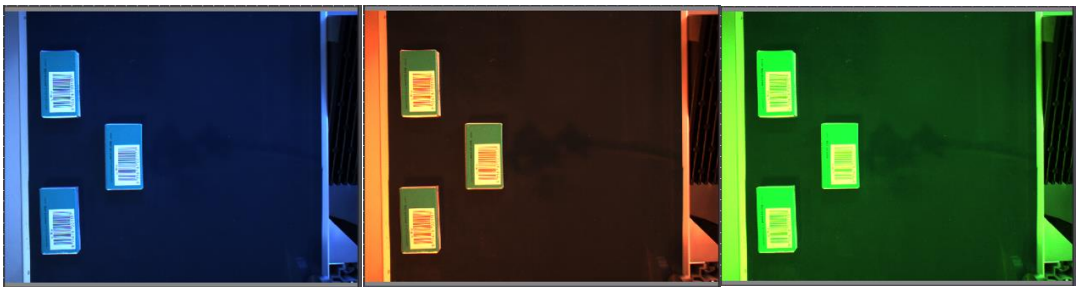


Figura 38. Variación de la ganancia de cada color. azul (izquierda), rojo (centro) y verde (derecha).

5.3.3 Selección de piezas

La selección de piezas va muy de la mano con los parámetros de la cámara. De hecho, a lo largo de la aplicación hemos estado probando varios tipos de piezas, cada una con sus ventajas y desventajas.

De entre todos los tipos, hemos probado:

- Fichas de póker.
- Gomas de borrar.
- Cajas de grapas.

La elección de utilizar material de oficina como piezas, se debe a que es lo que teníamos más a mano.

Las fichas de póker eran la pieza predeterminada. Tienen la ventaja de que es la pieza más plana, con lo cual la cámara no detecta relieves ni sombras cuando la ficha se encuentra en los bordes de la imagen. El inconveniente que presentan es que los bordes de color azul que tienen son de un color parecido al de la cinta, tanto es así que el modelo entrenado ha acabado por considerarlos parte de la cinta y no de la pieza en sí.

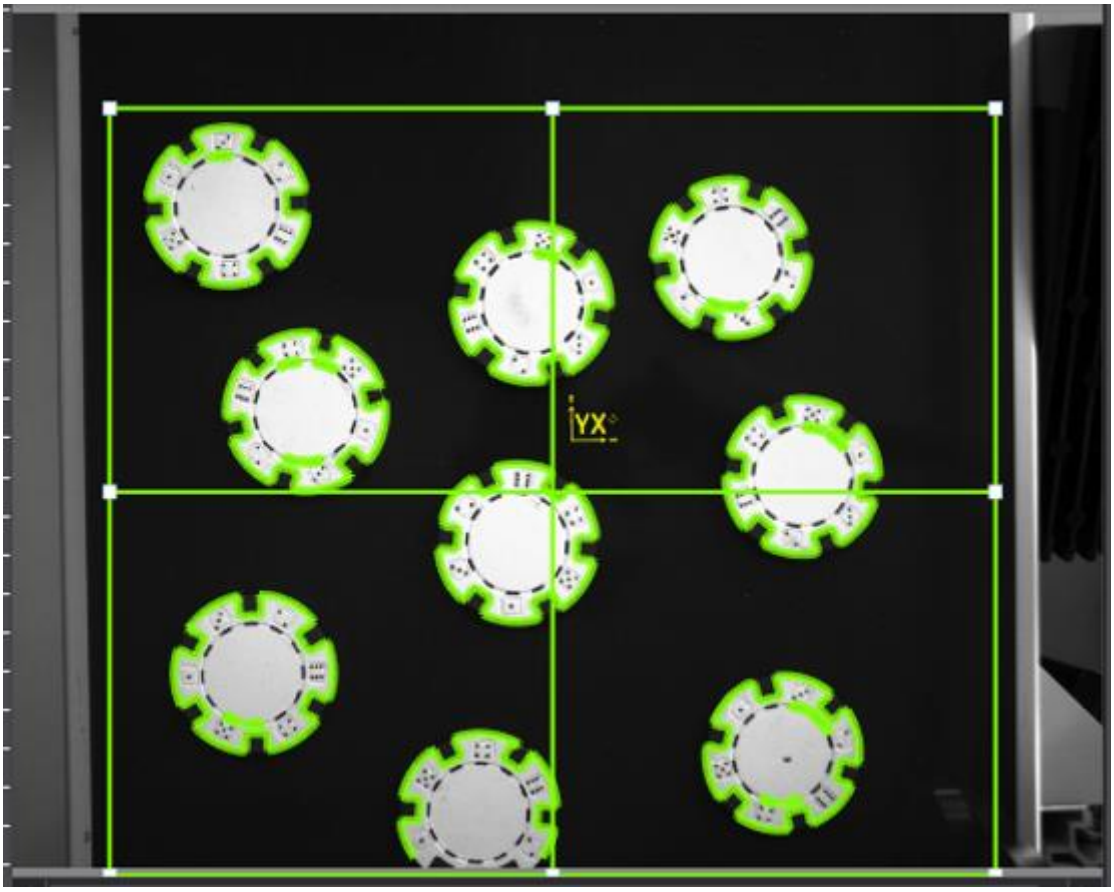


Figura 39. Propiedades captadas por el algoritmo de entrenamiento en un grupo de fichas de póker.

Las cajas de grapas obviamente han alternativa rápida vistos los problemas que presentaban las fichas de póker. Han sido descartadas debido a que: presentan reflejos, su color verde no destaca del azul verdoso de la cinta, son muy heterogéneas (en ocasiones el modelo se entrenaba de forma demasiado concreta y solo reconocía la pieza

con la que había sido entrenada como válida), tienen demasiado relieve (con lo que solo se reconocían las piezas que se encontraban en la región donde había sido entrenada la pieza), y los códigos de barras no se ven claramente en la cámara, con lo que no son una cualidad que se pueda tener en cuenta a la hora de entrenar el modelo.

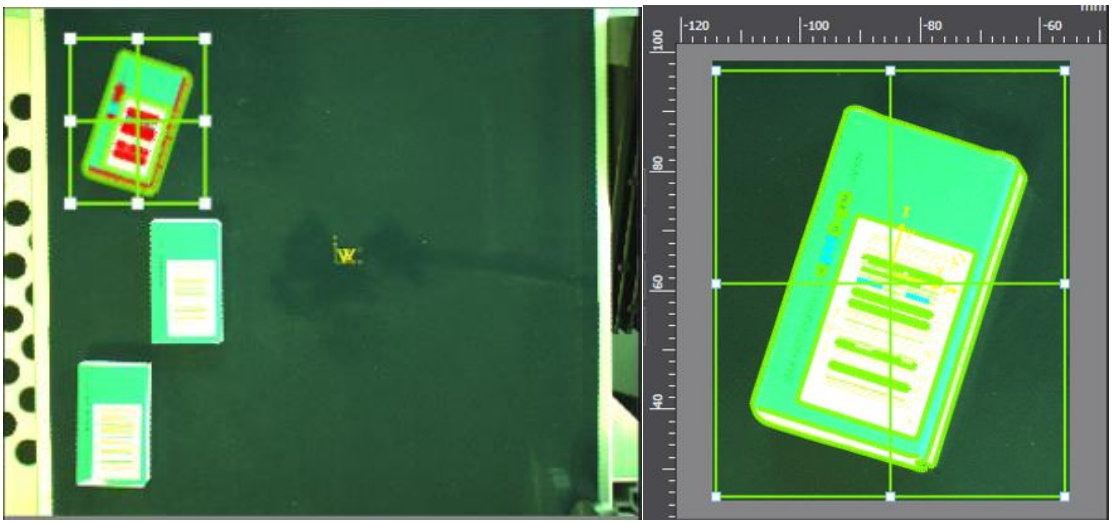


Figura 40. Ejemplo de las malas propiedades de las cajas de grapas a la hora de entrenar el modelo.

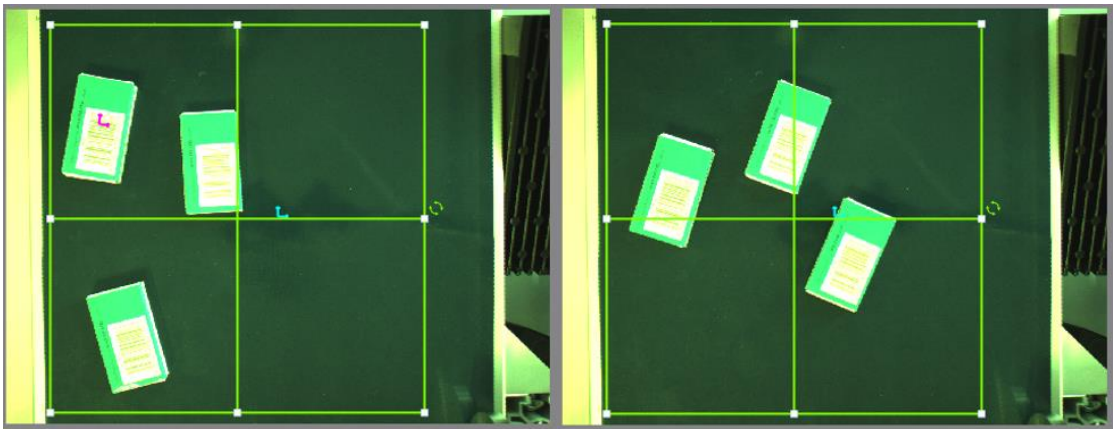


Figura 41. Ejemplo piezas no identificadas por tener un modelo demasiado dependiente de la pieza y la ubicación.

Las gomas de borrar, en cambio, han sido una buena pieza para entrenar: su material opaco no tiene reflejos, las letras del logo se distinguen claramente en la cámara y permiten dar una orientación. El problema es que tienen demasiado relieve, dando lugar a sombras, y son demasiado pequeñas para la ventosa del actuador, con lo que una mala identificación del centro hacía que no se enganchara correctamente la pieza.

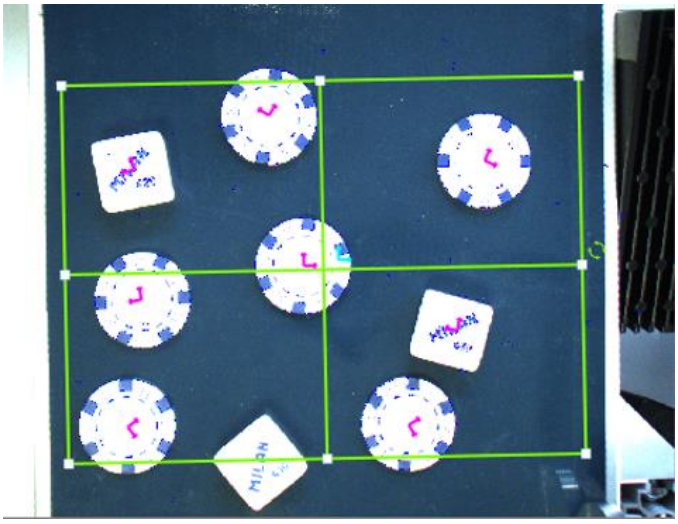


Figura 42. Gomas de borrar y fichas de póker identificadas correctamente en una misma imagen. Las gomas tienen la ventaja de que es fácil identificar una orientación.

Finalmente, valorando las tres opciones nos hemos quedado con las piezas de póker: hemos priorizado la ausencia de relieve y hemos comprobado que la identificación era correcta a pesar de no distinguir los cantos azules de la cinta.

5.3.4 Entrenamiento del localizador de piezas

Para entrenar el modelo es necesario tomar una foto de una pieza bajo la cámara y seleccionar el área en la que se encuentra el objeto de interés. No tenemos que dibujar ningún tipo de figura, sino que la aplicación ya sugiere propiedades que pueden ser de interés.

Es recomendable situar más de una pieza en la cámara y probar a entrenar el algoritmo con piezas diferentes para ver si encuentra detalles diferentes en función de cada individuo o de su ubicación en la imagen. En nuestro caso, vemos alguna diferencia en los detalles del interior de la pieza o alguna sombra, pero también vemos que se les detecta el mismo contorno a todas y se resaltan más o menos los mismos detalles.

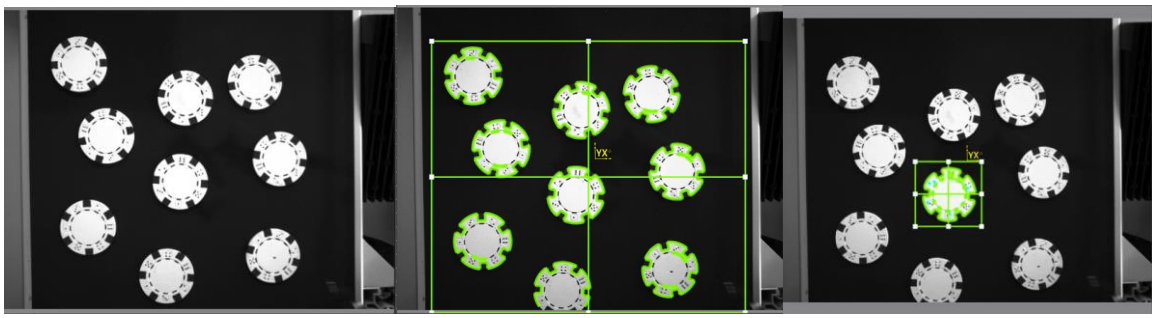


Figura 43. Foto original (izquierda), propiedades sugeridas por la aplicación seleccionando el conjunto de piezas (centro) y propiedades sugeridas seleccionando únicamente al mejor candidato (derecha).

Una vez comprobado la homogeneidad de los individuos, pasamos a elegir al mejor candidato para entrenar el modelo, seleccionando un área de interés en el que solo se encuentre éste. De nuevo, la aplicación identificará las propiedades válidas y nosotros elegiremos cuáles queremos que sean consideradas para definir la pieza.

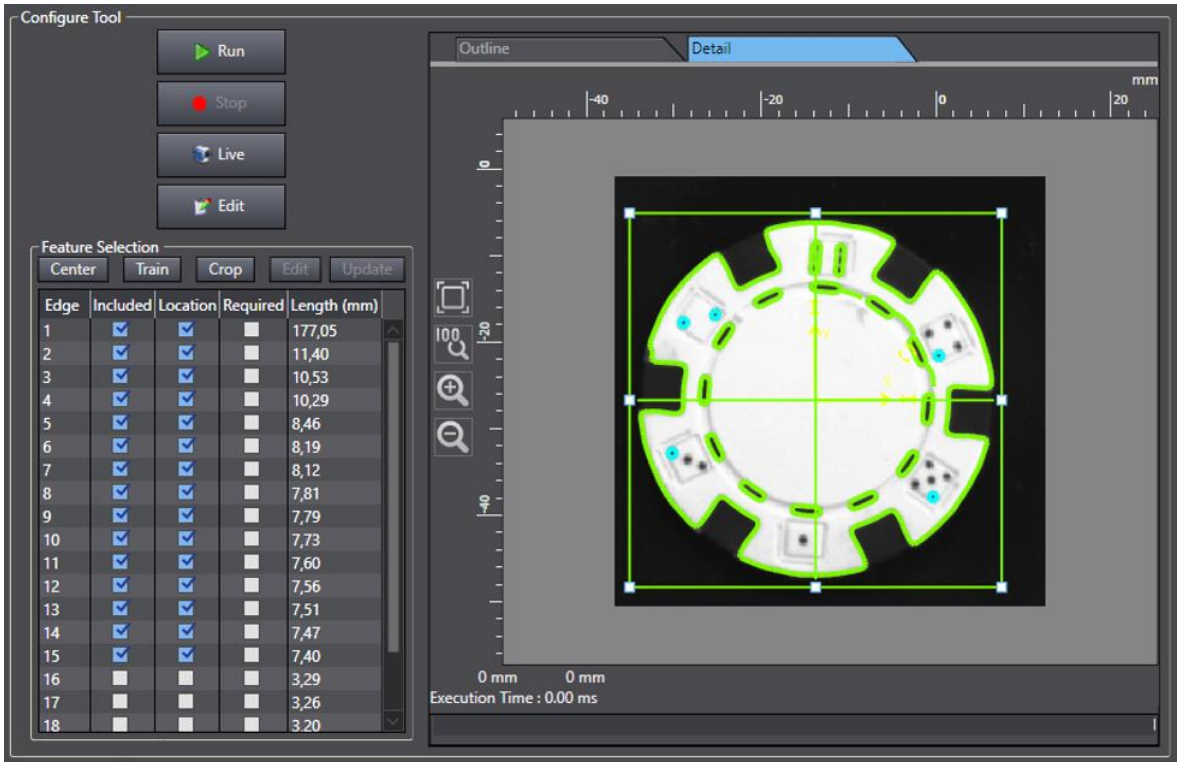


Figura 44. "Wizard" para el entrenamiento del modelo. Tiene dos pestañas "outline" y "details" con diferentes propiedades encontradas.

Empezamos por el "outline", que se refiere a la forma del objeto sin tener en cuenta los detalles. Dada la peculiaridad de la pieza, las marcas azules del centro las ha considerado como agujeros, como en realidad son detalles, vamos a desmarcarlas.

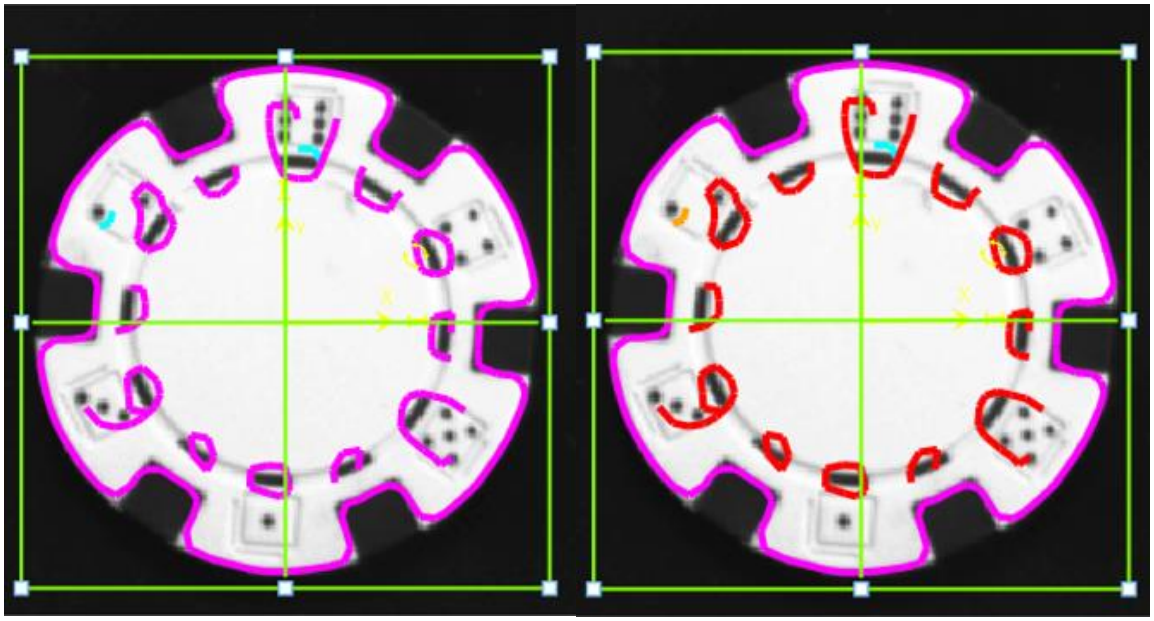


Figura 45. "Outline" sugerido por la aplicación (izquierda) vs el que finalmente elegimos (derecha). En lila son los atributos seleccionados y en rojo los descartados.

La siguiente ventana se refiere a los detalles de la figura. Son aquellas características con las que se puede ser más laxos a la hora de decidir si el objeto analizado es o no la pieza que estamos buscando.

Aquí vemos cómo los reflejos y las sombras pueden jugar una mala pasada con algunos detalles, por ejemplo, algunas de las líneas delgadas del cuadrante superior derecho se han mezclado con la sombra del relieve del círculo interior de la pieza. Por suerte, como son detalles y no se requiere de un "match" exacto, el algoritmo de detección será menos estricto con este detalle.

El detalle que sí que vamos a ignorar son los números de los dados, ya que, por la nitidez de la imagen y la calidad de estos, no es capaz de detectarlos de forma correcta y previsible. Esto hará que no podamos asignar una orientación a la pieza.

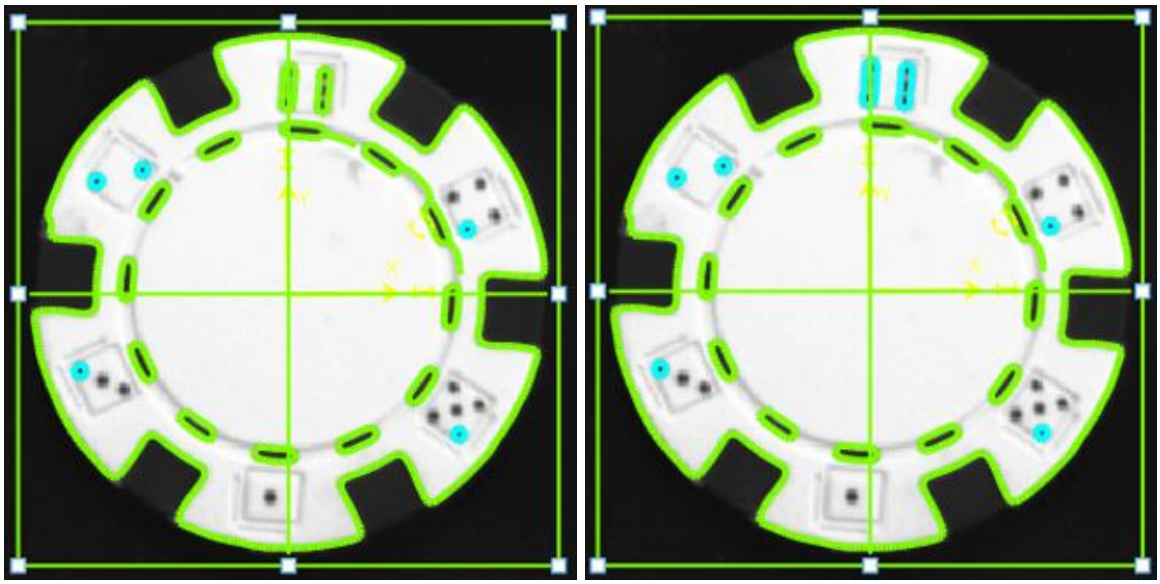


Figura 46. Detalles importantes de la pieza. Se ignoran (celeste) los dados de los bordes y se mantiene (verde) lo demás.

Una vez acabada la definición del modelo, podemos ejecutar el análisis sobre una imagen de prueba para comprobar el número de detecciones y la calidad de éstas.

Los resultados son buenos: con un factor de escala cercano al 1 y una calidad de entre el 95% y el 99.6%.

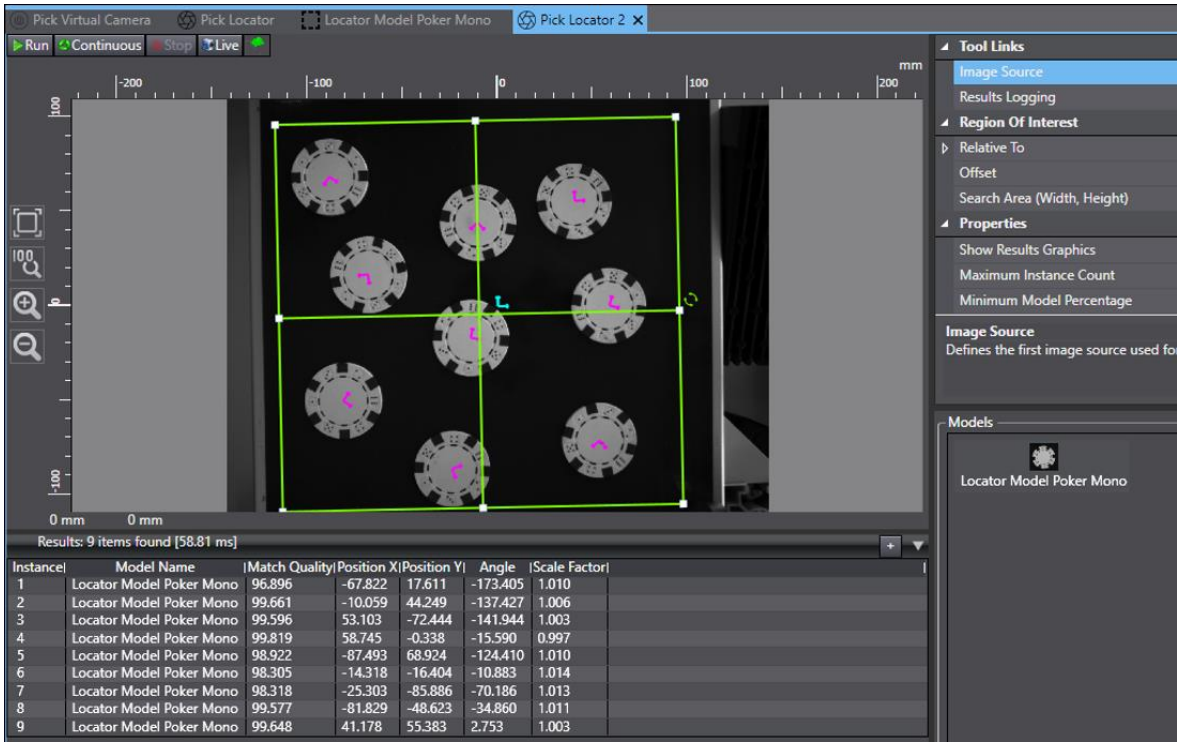


Figura 47. Ejecución del algoritmo con una imagen de muestra.

5.3.5 Concepto de VLATCHES en visión

Es un concepto análogo al de los encoder latches, la diferencia está en el trigger de los eventos, que en este caso se da cuando se reconoce una pieza nueva dentro de la imagen. Ahí se guarda un evento de VLATCH que contiene:

- Pulsos de encoder de la cinta asociada.
- Coordenadas X e Y de la pieza en ese instante dentro de la imagen.

Para simplificar, se pone la ubicación inicial del belt en el centro de la imagen tomada por la cámara, que también representa la coordenada (0,0) de los píxeles.

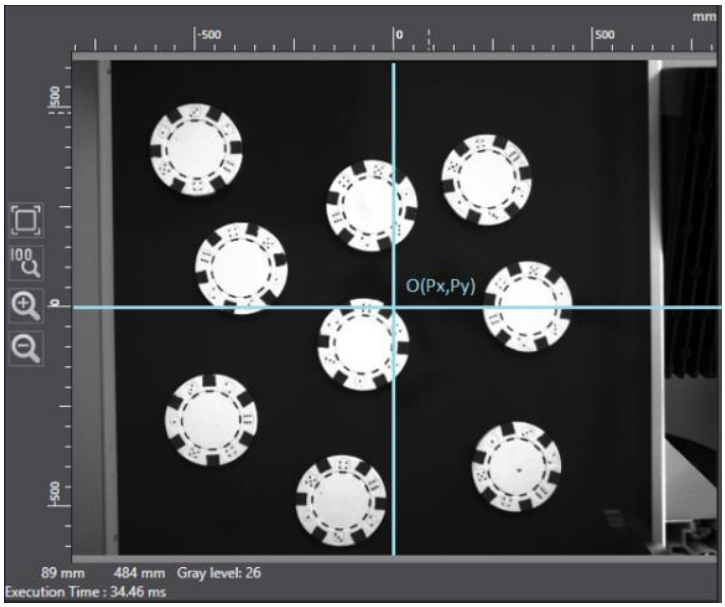


Figura 48. Centro de coordenadas de la cámara dentro de una imagen (conversión píxel/mm).

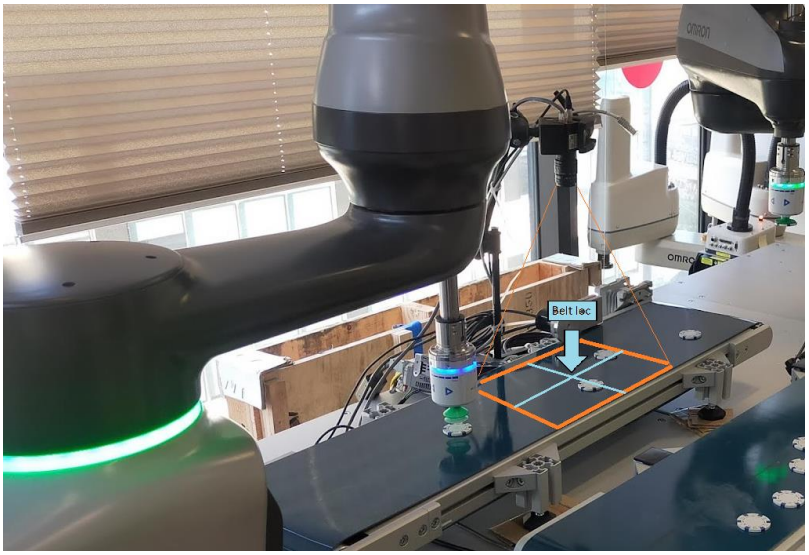


Figura 49. Centro de coordenadas de la imagen de cámara respecto al robot.

Por tanto, el programa queda bastante parecido al anterior con encoder latches:

- Consultamos de forma continua si ha habido algún evento de VLATCH.
- Cunado lo hay, cogemos los pulsos de encoder registrados en ese evento y lo aplicamos como offset a la variable Belt utilizando la keyword SETBELT. Esto haría que hiciéramos tracking del centro de la imagen de la cámara, ya que es ahí donde está definida la variable belt.
- Antes de hacer el seguimiento del belt, le aplicamos un offset de las coordenadas de la pieza dentro de la imagen, que viene dado por la misma keyword VLATCH.

```
30      DO
31          ; Peek in the queue and see if an instance is available.
32          ; If not, then move to the wait location
33          CALL getinstance(pick.queue[0], peek, inst.loc, model.idx, encoder.idx, vision.x, vision.y, vision.rot)
34          IF (model.idx == -1) AND (NOT at.wait) THEN
35              MOVES pick.wait[0]
36              BREAK
37          END
38
39          ; Do a blocking wait for the instance and configure the belt variablegg
40          CALL getinstance(pick.queue[0], block, inst.loc, model.idx, encoder.idx, vision.x, vision.y, vision.rot)
41          SETBELT %pick.belt[0] = encoder.idx
42          SET loc = inst.loc:pick.offset[0]
43          SET appro.loc = loc:TRANS(0,0,25)
44
45          ; Wait until the instance tracks in range of the robot belt window
46          ; and make sure it is far enough from the downstream limit to process.
47          DO
48              distance = WINDOW(%pick.belt[0]:appro.loc,0,1)
49              WAIT.EVENT , 0.016
50              IF (NOT at.wait) AND (distance < 0) THEN
51                  CALL rob.pk.align_0(inst.loc)
52                  at.wait = TRUE
53              END
54          UNTIL (distance >= 0)
55          distance = WINDOW(%pick.belt[0]:appro.loc,belt.pick.time,1)
56      UNTIL distance == 0
```

Figura 50. Código simplificado de la aplicación. La función "getinstance" devuelve pulsos de encoder y coordenadas y rotación de la pieza en la imagen.

La función "getinstance" la podemos considerar como una caja negra que nos devuelve la información comentada anteriormente. Para consultarla en detalle, dirigirse al anexo correspondiente.

Luego tenemos la lógica del robot cuando tiene que ir a buscar la pieza una vez ya está dentro del rango:

```
58          APPROS %pick.belt[0]:loc, 25
59          distance = WINDOW(%pick.belt[0]:appro.loc,0,1)
60          IF (distance > 0) GOTO 10
61
62  ; Move into position and actuate the gripper
63          MOVES %pick.belt[0]:loc
64          BREAK
65
66          distance = WINDOW(%pick.belt[0]:loc,0,1)
67          IF (distance > 0) GOTO 10
68
69          CALL rob.grip_0(FALSE)
70          WAIT.EVENT , 0.8
71
72  ; Depart from the position
73          APPROS %pick.belt[0]:loc, 25
```

Figura 51. Código simplificado de la aplicación. Seguimiento de la cinta por parte del robot hasta agarrar la pieza.

6 Vídeos de la aplicación en marcha

Aquí tenemos la lista de reproducción con todos los vídeos:

https://youtube.com/playlist?list=PLdBCI3JmZhS4nyfWcnu0gHWMjBrYpJhXI

6.1 Estación fotosensor

https://youtu.be/A1qIIQaeJhU

6.2 Estación de visión

https://www.youtube.com/watch?v=IiTqOdx8ETA

https://youtu.be/Hi7hAU4Oc3k

6.3 Todas las estaciones en marcha

https://youtu.be/AB3QQxhe_tc

https://youtu.be/qFSphLKHZ-o

7 Resumen del presupuesto y/o estudio de viabilidad económica

Tabla con el precio de cada componente utilizado. Algunos de ellos son orientativos ya que han sido cedidos por otros departamentos sin algún coste.

Tabla 2. Costes del proyecto.

Elemento	P/N	Cantidad	Precio	Total
Mesa antivibración	-	2	10,000 €	20,000 €
Cinta de transporte	-	2	900 €	1,800 €
PLC	NJ-501	1	7,600 €	7,600 €
Modulo E/S digitales	NX-EC203	1	673 €	673 €
Driver control servo + motor	R88D-KN04H	2	1,400 €	2,800 €
Robot SCARA	i4H-850	1	22,000 €	22,000 €
Robot SCARA	i4H-650	1	18,000 €	18,000 €
Cámara 2D	Omron FZ	1	230 €	230 €
Encoder 2000 pulsos	TM	2	180 €	360 €
Actuador válvula de vacío	Shmalz	2	900 €	1,800 €
				75,263 €

8 Análisis y valoración de las implicaciones ambientales y sociales

Supongo que las implicaciones ambientales y sociales que puede tener este trabajo son las mismas que pueda tener cualquier proyecto de automatización de un trabajo manual en el que se sustituye la mano de obra por robots.

Por un lado, tenemos que la tarea que se lleva a cabo por estos robots puede ser más eficiente si se hace con éstos, ya que, si la aplicación está bien programada y no hay fallos, pueden trabajar 24 horas sin cansarse y su inversión se puede amortizar rápidamente (un robot i4L de pequeñas dimensiones oscila los 7000€ mientras que un i4H es de 18000€).

Por otro lado, estaríamos a una persona de hacer una tarea manual tan repetitiva y desmotiva como la selección de piezas, y estaríamos cambiando trabajos poco motivadores por otros mejor remunerados y más estimulantes. Este tema va ligado a la desventaja de que no cualquiera puede acceder a trabajos relacionados con la programación de robots, ya que esto requiere de más estudios y conocimiento.

En cuanto a implicaciones ambientales, también hay ventajas y desventajas. Para la fabricación de un robot hacen falta muchos componentes electrónicos que luego pueden acabar convirtiéndose en residuos difícilmente reciclables, así como que un robot necesita de electricidad para funcionar, y esta puede que sea generada por fuentes no renovables. Sin embargo, tener a una persona también tiene su impacto ecológico, ya que tendrá que desplazarse cada día para ir al trabajo (a diferencia del robot) y podemos hasta considerar que para llevar a cabo su trabajo tendrá alimentarse, que a su vez deja una huella ecológica detrás de todo el proceso para la obtención de sus alimentos (transporte, etc…).

9 Conclusiones

En general, nos podemos dar por satisfechos con los resultados de la aplicación, ya que finalmente ha acabado funcionando correctamente.

Cabe destacar que nos hemos sorprendido de la facilidad con la que se puede crear un programa de visión utilizando el wizard de ACE, y las buenas recomendaciones que hace la herramienta de entreno de modelos de visión.

Por otro lado, también nos ha servido para encontrar múltiples errores o mejoras dentro del software del robot y la interfaz de usuario, como por ejemplo:

- Error lanzado por el software cuando se consultan demasiados eventos de tipo encoder latch.
- Detención de la aplicación ACE durante la configuración de la visión.
- Problemas con la obtención de las imágenes de la cámara si se conectaba al PC de visión utilizando un adaptador de Ethernet/usb.
- Sobrecalentamiento de la cámara al ser un componente de muestra.
- Falta de una keyword específica para detener el seguimiento de cinta.
- Interrupción de la secuencia de visión.

Con estos inputs, vamos a poder proponer mejoras que nos ayudarán a mejorar el producto para que sea más fácil de manejar para el usuario.

De cara a futuro, las mejoras a hacer son:

- Tener un mejor setup de visión: eliminar los reflejos y tener un soporte más estable para la cámara y que no se balancee cuando se ponen los robots a máxima velocidad.
- Utilizar un IPC de Omron en lugar de un PC convencional para ejecutar las tareas de visión.
- Tener una cámara oficial en lugar de utilizar una de muestra.
- Poner los robots a trabajar al 100% de la velocidad.
- Añadir un paso intermedio para cada robot, de forma que en lugar de depositar las piezas directamente en la cinta de salida, se depositen en un pallet y cuando se llene hacer el vaciado en la otra cinta.

10 Referencias

Conjunto de documentos consultados y citados a lo largo del trabajo.

https://industrial.omron.es/es/products/robotics

https://industrial.omron.es/es/products/programmable-logic-controllers

https://www.ia.omron.com/products/family/3521/

https://industrial.omron.es/es/products/sysmac-studio

https://industrial.omron.es/es/products/nx-series

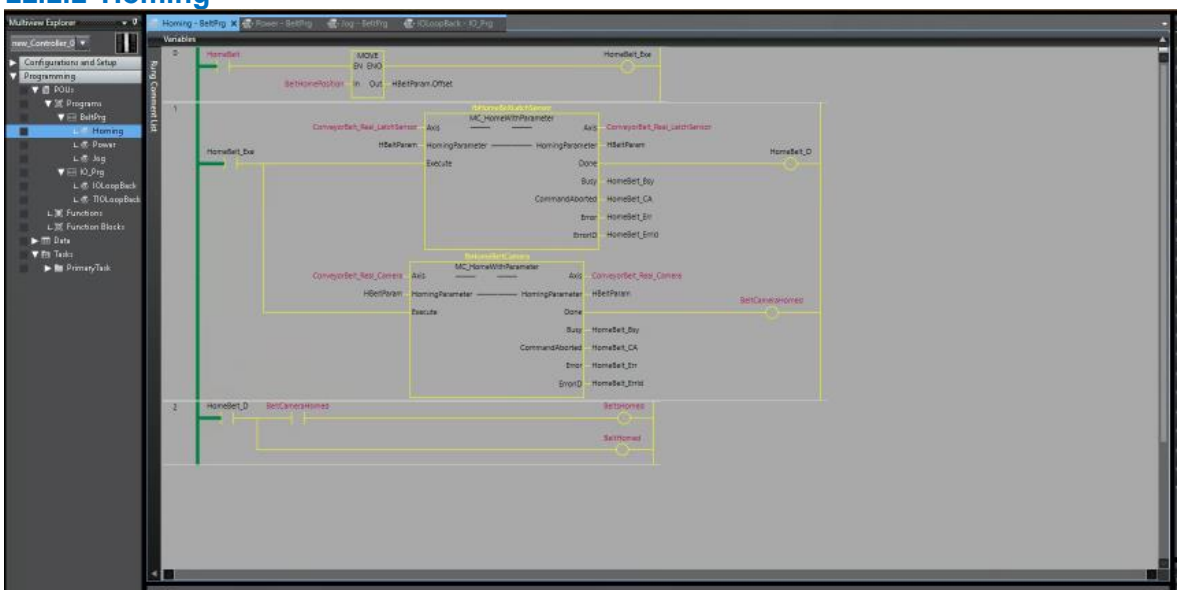
https://industrial.omron.es/es/products/R88D-1SN04H-ECT

https://encoder.co.uk/applications/by-function/conveying/

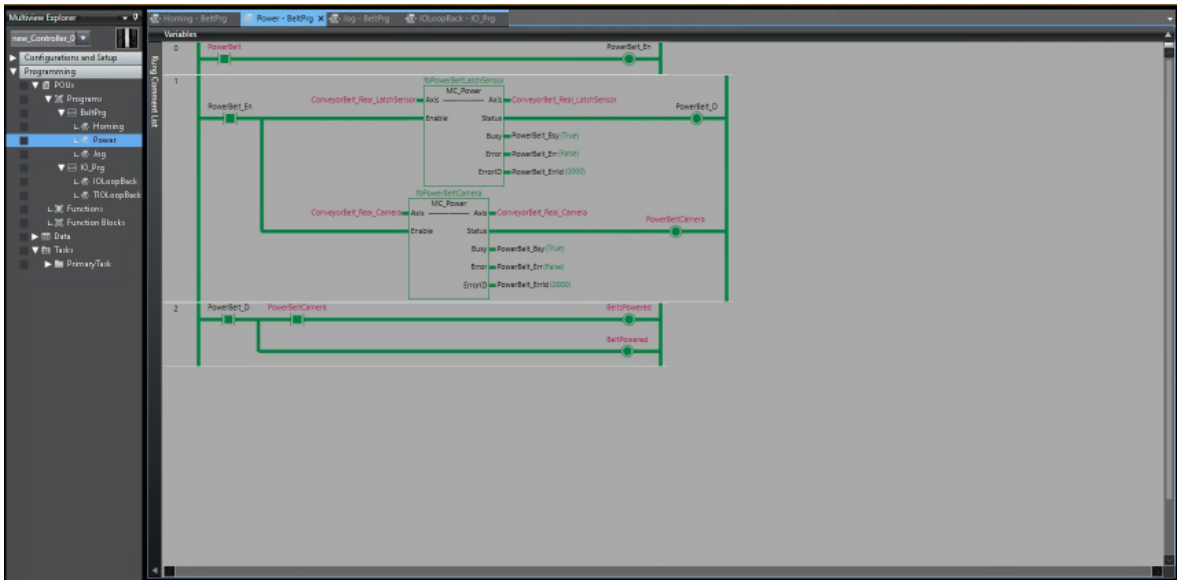
11 Anexos

11.1 Programa completo del PLC

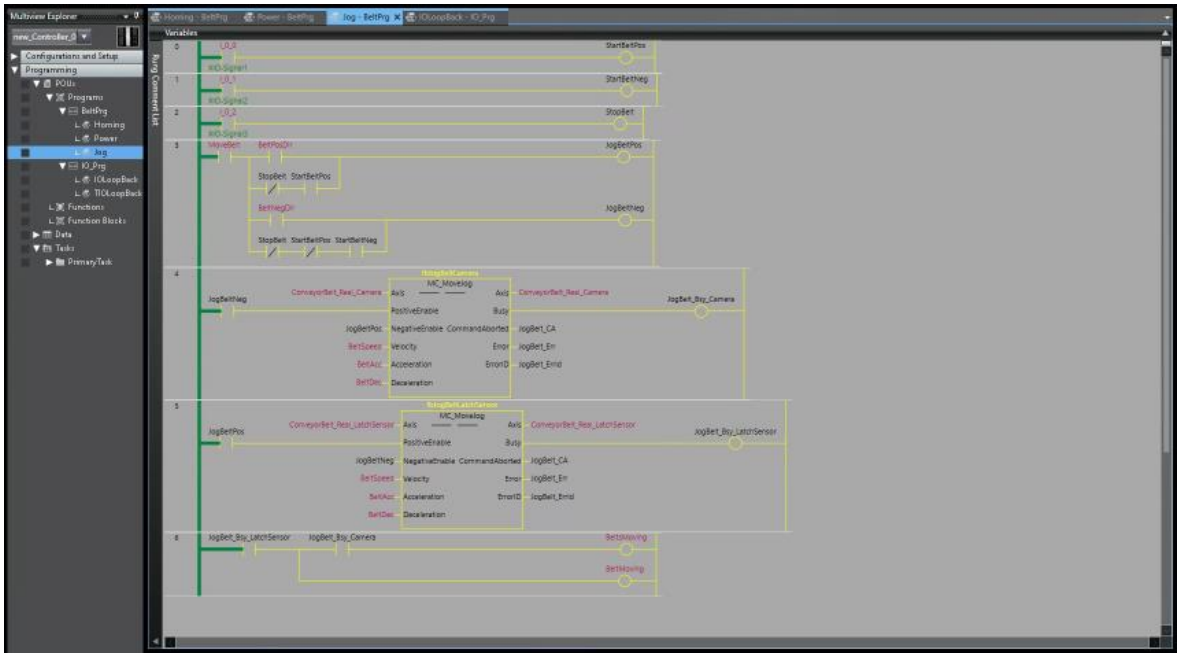
11.1.1 Homing



11.1.2 Power belts



11.1.3 Move Belts



11.2 Programa completo de la estación de foto detector

```
.PROGRAM main_850()

    GLOBAL suck_signal, drop_signal, gripper_time, move_belt_sig
    GLOBAL wait_loc, place_loc, belt_loc
    GLOBAL place_y_near, place_y_far, piece_radius
    GLOBAL %belt1

    move_belt_sig = 1
    suck_signal = 17
    drop_signal = 18
    gripper_time = 0.5
    piece_diameter = 40
    rad_piece = piece_diameter/2

    pick_z_small = 212
    pick_z_big = 217
    pick_y_coord = 271.038

    place_x = 600
    place_y_near = -135
    place_y_far = -280
    place_y_center = (place_y_near+place_y_far)/2
    place_z_adjust = 10

    ; Define all locations for BCN-5
    pick_z_offset = pick_z_small     ;Test=230/Normal:212.7/nomral con ventosa pequeña = 12
    wait_z_offset = 260
    belt_weith = 235

    ;usable belt weith its 23.5 cm (of the total belt) - 2 cm for radius of poker coin per side of the belt.
    SET max_reach_loc = TRANS(782,pick_y_coord,pick_z_offset,0,180,0)
    win_distance = 600
    SET min_reach_loc = SHIFT(max_reach_loc BY -win_distance)
    SET sensor_loc = SHIFT(max_reach_loc BY 160) ; location corresponding to latch sensor, outside robot limits
```

```
    SET wait_loc = SHIFT(max_reach_loc BY ,,wait_z_offset-pick_z_offset)    ;SET wait_loc =
TRANS(673,275,wait_z_offset,0,180,0)
    SET place_loc = TRANS(place_x,place_y_center,pick_z_offset,0,180,180)

    ; Define Belt
    enc_radius = 31.83
    pulses_rev = 10000
    quad_factor = 4
    scale_factor = 2*PI*enc_radius/(pulses_rev*quad_factor)  ; mm per revolution = 2*PI*radius
    belt_num = 1

    SET belt_loc = sensor_loc ; used also to define the belt position and orientation, positive -x in movement direction
    DEFBELT %belt1 = belt_loc, belt_num, 0, scale_factor    ; apply scale factor for belt tracking
    WINDOW %belt1 = max_reach_loc, min_reach_loc
    SETDEVICE (0, belt_num-1, , 8) scale_factor ; apply same scale factor for DEVICE functions (read encoder) as for belt
tracking

    SET belt_loc2 = place_loc
    DEFBELT %belt2 = belt_loc2, belt_num, 0, scale_factor
    SETDEVICE (0, belt_num-1, , 8) scale_factor ;

    ; Attach the robot and move to READY
    SELECT ROBOT = 1
    ENABLE POWER
    CALIBRATE

    ATTACH ()
    READY
    BREAK

    ; Do an infinite loop waiting for the sensor to catch pieces
    CALL pick_and_place()

.END

.PROGRAM check_enclatch(enclatch_value, is_valid)

    GLOBAL piece_diameter
    LOCAL last_enclatch
    AUTO difference

    IF DEFINED(last_enclatch) THEN
       difference = ABS(ABS(enclatch_value)-ABS(last_enclatch))
       IF (difference > piece_diameter) THEN
          TYPE "last enclatch is valid. last value: ", last_enclatch, ". current latch: ", enclatch_value, ". difference: ",
difference
          last_enclatch = enclatch_value
          is_valid = TRUE
       ELSE
          TYPE "last enclatch is not enough difference. Last latch: ", last_enclatch, ". current latch: ", enclatch_value, ".
difference: ", difference
          is_valid = FALSE
       END
    ELSE
       TYPE "last enclatch not defined. Current latch: ", enclatch_value
       last_enclatch = enclatch_value
       is_valid = TRUE
    END


.END
```

```
.PROGRAM drop_in_belt(appro_dist)

      GLOBAL place_y_near, place_y_far, place_loc_rand

      ; Stop tracking and move piece to place location
      rand_shift_y = 0
      CALL get_place_y(place_y_near, place_y_far, rand_shift_y)
      APPRO place_loc:TRANS(0,rand_shift_y,0,0,0,0), appro_dist
      SETBELT %belt2 = BELT(%belt2)
      MOVES %belt2:TRANS(0,rand_shift_y,-place_z_adjust,0,0,0)
      BREAK

      CALL drop_piece()

      ; Generate latch event and save coordinates to be passed to the other robot
      SET drop_loc = HERE
      d_x = DX(drop_loc)-DX(place_loc)
      d_y = rand_shift_y-place_y_near
      TYPE "Piece dropped at dx=", d_x, "; dy=", d_y

      DEPARTS appro_dist
      BREAK
      MOVES HERE

.END

.PROGRAM drop_piece()

      GLOBAL suck_signal, drop_signal, gripper_time
      AUTO belt2.drop.dist, max_track_time, track_time

      belt2.drop.dist = 15
      max_track_time = 1

      ;code to avoid errors due to too long drop times when belt speed is to fast.
      ;define drop time as a variable refered to te belt speed and
      belt.speed = BELT(%belt1,1)
      track_time = belt2.drop.dist/belt.speed
      drop_time = MIN(track_time,max_track_time)

      SIGNAL -suck_signal, drop_signal
      WAIT.EVENT , drop_time ;gripper_time
.END

.PROGRAM err.reacte()

      GLOBAL belt_num, wait_loc

      AUTO last.err
      AUTO current.loc

      ; Record the current error values.

      last.err = ERROR(-1,0)        ;Store most recent error
;     last.err.var = ERROR(-1,1)    ;Store variable part of error
;     last.err.rob = ERROR(-1,3)    ;Store robot associated with error

      err.code = 0 ;
      $err.msg = "" ;

      ; If an error occurred
```

```
    IF last.err < 0 THEN
       err.code = last.err
       $err.msg = $ERROR(last.err)
    END

    TYPE "Error occurred:", $err.msg

    ; Reattach the robot before leaving the REACTE
    ;   without failing and without waiting
    IF (SELECT(ROBOT, -1) > 0) THEN
       ATTACH (0, 2)
    END

    ; If robot is out of range, clear latches and go to ready
    SET current.loc = HERE
    is_in_range = INRANGE(current.loc)
    IF (is_in_range <> 0) THEN
       TYPE "Move robot to ready location and clear latches"
       MOVES ready.loc
       BREAK
       CLEAR.LATCHES (-belt_num)
    END

    ; Reset the REACTE.
    REACTE err.reacte

    ; RETURNE will return the program control to the next line of
    ;   execution of the routine that invoked the REACTE.
    RETURNE

.END

.PROGRAM get_place_y(place_near_y, place_far_y, result)

    AUTO width, centre
    width = MAX(place_near_y,place_far_y)-MIN(place_near_y,place_far_y)
    centre = (place_near_y+place_far_y)/2

    rand_sign = RANDOM
    IF (rand_sign > 0.5) THEN
       rand_sign = 1
    ELSE
       rand_sign = -1
    END

    rand_dist = RANDOM
    result = (rand_sign*rand_dist*width/2)

.END

.PROGRAM pick_and_place()
    AUTO appro_dist, curr_belt_loc

    ; Initialize variables
    appro_dist = 50
    latch_signal = 1010
    simulated_latch = 1006
    CLEAR.LATCHES (-belt_num)

    ; Set REACTE routine
    REACTE err.reacte
```

```
    ; Infinite loop: place all pieces in Belt
    do_pick_place = TRUE
    SIGNAL move_belt_sig
    WHILE do_pick_place == TRUE DO

      ; Check if we have seen any LATCH signal
      latched_signal = LATCHED(-belt_num)
      IF ((latched_signal == latch_signal) OR (latched_signal == simulated_latch)) THEN
        enclatch_value = ENCLATCH(belt_num)
        CALL check_enclatch(enclatch_value, is_valid_latch)

        ; Syncronize Belt with latched piece position if enclatch value is valid
        IF (is_valid_latch) THEN
          SETBELT %belt1 = (enclatch_value+rad_piece)/scale_factor ; apply offset in encoder counts

        ; Check if the piece is in range to move joint interpolated
          IF (INRANGE(%belt1:NULL) == 0) THEN
            SET curr_belt_loc = %belt1:NULL
            APPRO curr_belt_loc, appro_dist
            BREAK
          ELSE
            MOVE wait_loc
            BREAK
          END

        ; Pick the piece when in window
          APPROS %belt1:NULL, appro_dist
          MOVES %belt1:NULL
          CALL suck_piece()
          BREAK

        ; Stop tracking and move piece to place location
          DEPARTS appro_dist
          BREAK
          MOVES HERE
          CALL drop_in_belt(appro_dist)
        END

      ELSE
        ; Come back to waiting position
        MOVE wait_loc
      END
    END

    RESET
.END

.PROGRAM suck_piece()

    GLOBAL suck_signal, drop_signal, gripper_time

    SIGNAL suck_signal, -drop_signal
    WAIT.EVENT , gripper_time

.END

.PROGRAM trigger_latches()

    GLOBAL trig_latch_time
    AUTO signal_out, signal_in

    signal_in = simulated_latch
```

```
    signal_out = simulated_latch-1000
    trig_latch_time = 2.25


    WHILE (TRUE) DO
      SIGNAL signal_out
      WAIT SIG(signal_in) == TRUE
      WAIT.EVENT , trig_latch_time

      SIGNAL -signal_out
      WAIT SIG(signal_in) == FALSE
      WAIT.EVENT , trig_latch_time
    END

.END
```

11.3 Programa completo de la estación de visión

```
.PROGRAM rob.get.ip_0($ip)
    GLOBAL $ip.address[]
    $ip = $ip.address[0]

    IF DEFINED(sv.emulate.mode) THEN
      IF sv.emulate.mode THEN
        $ip = "127.0.0.1"
      END
    END

    RETURN
.END

.PROGRAM rob.grip_0(open)
    GLOBAL REAL o.open[], o.close[], i.open[], i.close[]
    AUTO REAL outputs[2], input
    AUTO REAL start.time, time.out, continue, code
    AUTO $text

    IF open THEN
      outputs[0] = o.open[0]
      outputs[1] = -o.close[0]
      input = i.open[0]
      $text = "Unable to open gripper"
    ELSE
      outputs[0] = -o.open[0]
      outputs[1] = o.close[0]
      input = i.close[0]
      $text = "Unable to close gripper"
    END

    start.time = TIMER(-3)
    time.out = 5

    SIGNAL outputs[0], outputs[1]
    IF (input <> 0) THEN
      WHILE NOT SIG(input) DO
        IF (TIMER(-3)-start.time) > time.out THEN
          CALL err.report_0(code, $text)
          start.time = TIMER(-3)
        END
        WAIT
      END
```

```
    END

    RETURN
.END

.PROGRAM rob.init.grip_0()

    GLOBAL REAL o.open[], o.close[], i.open[], i.close[]
    GLOBAL LOC grip.tool[]
    AUTO REAL tip, rob.num
    AUTO $ip
; Read the IO signals used to open/close the gripper
; from the gripper editor in the workspace.

    tip = 0
    rob.num = rob.number[0]-1
    CALL rob.get.ip_0($ip)

    o.open[0] = VPARAMETER($ip, -1, tip, 5511, rob.num)
    o.close[0] = VPARAMETER($ip, -1, tip, 5512, rob.num)
    i.open[0] = VPARAMETER($ip, -1, tip, 5514, rob.num)
    i.close[0] = VPARAMETER($ip, -1, tip, 5515, rob.num)

; Read the gripper offset from the gripper editor in the workspace.
    SET grip.tool[0] = VLOCATION($ip, -1, tip, rob.num, 11000)
    TOOL grip.tool[0]

; Set the payload mass
    PAYLOAD -1
    RETURN

.END

.PROGRAM rob.init_0()
    GLOBAL REAL rob.number[], rob.run[]
    AUTO REAL code, cal.state, bit

; Indicate the application sample is running
    rob.run[0] = TRUE

; Change the default vision timeout
    PARAMETER VTIMEOUT = 10

; Ensure emergency stop circuits are clear
    WHILE (STATE(4) BAND ^B100 == 4) DO
      WAIT.EVENT , 1
      TYPE "Please release all EMERGENCY STOP circuits"
    END

; Ensure power is enabled before attaching
    ENABLE POWER

; Check for calibration
    CALIBRATE

; Attach the robot so it can be moved
    DETACH ()
    SELECT ROBOT = rob.number[0]
    ATTACH (0, 1)
    code = IOSTAT(0)
    IF (code == -515) THEN
      code = 0
```

```
    END
    IF (code < 0) THEN
      TYPE "Unable to attach the robot: ", code
      rob.run[0] = FALSE
      HALT
    END

; Initialize the speed for all moves
    SPEED 100 ALWAYS
    RETURN

.END

.PROGRAM rob.main_0()
    GLOBAL REAL rob.run[]

; Initialize error reporting
    CALL err.init_0()

; Initialize robot and application variables
    CALL rob.init_0()
    CALL rob.init.grip_0()
    CALL rob.pk.init_0()

; Move the robot to a safe location
    CALL rob.move.safe_0()

; Main robot processing loop
    WHILE rob.run[0] DO
      CALL rob.pick_0()
      CALL rob.place_0()
    END

    CALL rob.move.safe_0()
    RETURN
.END

.PROGRAM rob.move.safe_0()
    GLOBAL LOC safe.loc[]
    AUTO REAL tmp[5], pos[5]
    AUTO REAL code, resp
    AUTO LOC z.up.loc

; Calculate the current robot location aligned with the
; z of the safe position and move the robot to that point.
    TOOL NULL
    DECOMPOSE tmp[] = HERE
    DECOMPOSE pos[] = safe.loc[0]
    SET z.up.loc = TRANS(tmp[0],tmp[1],pos[2],tmp[3],tmp[4],tmp[5])

; Move the robot
    MOVE z.up.loc
    BREAK

; Move to the safe position
    MOVE safe.loc[0]
    BREAK

; Make sure the gripper is opened
    TOOL grip.tool[0]
    CALL rob.grip_0(TRUE)
```

```
    RETURN
.END

.PROGRAM rob.pick_0()
    GLOBAL REAL pick.encoder[], pick.queue[]
    GLOBAL LOC pick.wait[], pick.offset[], %pick.belt[]
    AUTO REAL block, peek
    AUTO REAL model.idx, encoder.idx, vision.x, vision.y, vision.rot
    AUTO REAL distance, at.wait, belt.pick.time
    AUTO LOC loc, appro.loc, inst.loc

; Initialize constants/variables
    belt.pick.time = 1.2        ; Amount of time (in seconds) it will take the robot to pick the instance
    block = 1
    peek = 2

  10

; Access the queue and wait until an instance is available for processing

    at.wait = FALSE
    DO
      ; Peek in the queue and see if an instance is available.
      ; If not, then move to the wait location

      CALL getinstance(pick.queue[0], peek, inst.loc, model.idx, encoder.idx, vision.x, vision.y, vision.rot)
      IF (model.idx == -1) AND (NOT at.wait) THEN
        MOVES pick.wait[0]
        BREAK
      END

      ; Do a blocking wait for the instance and configure the belt variablegg
      CALL getinstance(pick.queue[0], block, inst.loc, model.idx, encoder.idx, vision.x, vision.y, vision.rot)
      SETBELT %pick.belt[0] = encoder.idx
      SET loc = inst.loc:pick.offset[0]
      SET appro.loc = loc:TRANS(0,0,25)

      ; Wait until the instance tracks in range of the robot belt window
      ; and make sure it is far enough from the downstream limit to process.

      DO
        distance = WINDOW(%pick.belt[0]:appro.loc,0,1)
        WAIT.EVENT , 0.016
        IF (NOT at.wait) AND (distance < 0) THEN
          CALL rob.pk.align_0(inst.loc)
          at.wait = TRUE
        END
      UNTIL (distance >= 0)
      distance = WINDOW(%pick.belt[0]:appro.loc,belt.pick.time,1)
    UNTIL distance == 0

; Move above the location first
    ;DECOMPOSE coords[] = %pick.belt[0]:loc
    ;SET loc_no_orient = TRANS(coords[0],coords[1],coords[2],0,180,180)
    DECOMPOSE belt_coords[] = %pick.belt[0]:loc

    TYPE "APPROS %pick.belt[0]:loc, 25; ", belt_coords[0], belt_coords[1], belt_coords[2], belt_coords[3],
belt_coords[4], belt_coords[5]
    APPROS %pick.belt[0]:loc, 25
    ;APPROS loc_no_orient, 25

    distance = WINDOW(%pick.belt[0]:appro.loc,0,1)
```

```
    IF (distance > 0) GOTO 10

; Move into position and actuate the gripper

    MOVES %pick.belt[0]:loc
    BREAK

    distance = WINDOW(%pick.belt[0]:loc,0,1)
    IF (distance > 0) GOTO 10

    CALL rob.grip_0(FALSE)
    WAIT.EVENT , 0.8

; Depart from the position
    APPROS %pick.belt[0]:loc, 25
    RETURN
.END

.PROGRAM rob.pk.align_0(loc)
    GLOBAL LOC %pick.belt[], pick.belt[], pick.wait[]
    AUTO REAL offset, vals[5]
    AUTO LOC inst.pos, appro.pos

; The distance from the usptream belt window edge to
; use for the waiting position.
    offset = 10       ; Positive = downstream

; Calculate the position in-line with the instance at the specified offset distance
    SET inst.pos = %pick.belt[0]:loc
    DECOMPOSE vals[] = INVERSE(pick.belt[0]):inst.pos
    SET appro.pos = pick.belt[0]:TRANS(offset,vals[1],vals[2],vals[3],vals[4],vals[5])

    ; Set the height to be the same as the waiting position
    SET appro.pos = appro.pos:TRANS(0,0,-(DZ(pick.wait[0])-DZ(appro.pos)))

; Move to the position
    MOVES appro.pos
    BREAK

    RETURN

.END

.PROGRAM rob.pk.init_0()
    GLOBAL REAL pick.seq[], pick.cam.num[], rob.number[], pick.encoder[], pick.queue[], sv.emulate.mode
    GLOBAL LOC %pick.belt[], pick.belt[]
    AUTO REAL beltcalframe, beltcaluplim, beltcaldownlim, beltcalscale, beltonoff
    AUTO REAL single_mode, continuous_mode
    AUTO REAL conv, scale, emul.belt.vel, stt
    AUTO LOC upstream, downstream
    AUTO $ip

; Initialize indexes used to access information over the AdeptSight protocol
; and general constants.
    beltcalframe = 10000
    beltcaluplim = 10001
    beltcaldownlim = 10002
    beltcalscale = 10004
    beltonoff = 10005
    single_mode = 0
    continuous_mode = 1
    emul.belt.vel = 30
```

```
    CALL rob.get.ip_0($ip)

; Put the AdeptSight Sequence in continuous mode and start it
    CALL set_as_exec_mod($ip, pick.seq[0], single_mode)          ; Stop AdeptSight if it is running
    CALL as.wait.seq(pick.seq[0], $ip)
    CALL reset_seq($ip, pick.seq[0])                             ; Reset AdeptSight and the local data queue
    CALL clear_queue(pick.queue[0])
    CALL set_as_exec_mod($ip, pick.seq[0], continuous_mode)      ; Setup AdeptSight to run in continuous mode
    VRUN $ip, pick.seq[0]                                        ; Start execution of AdeptSight

; Initialize the belt window
    scale = VPARAMETER($ip, -1, pick.cam.num[0], beltcalscale, , rob.number[0])
    SET upstream = VLOCATION($ip, -1, pick.cam.num[0], , beltcaluplim, rob.number[0])
    SET downstream = VLOCATION($ip, -1, pick.cam.num[0], , beltcaldownlim, rob.number[0])
    SET pick.belt[0] = VLOCATION($ip, -1, pick.cam.num[0], , beltcalframe, rob.number[0])

; Configure encoder
    SETDEVICE (0, pick.encoder[0]-1, , 0)

; Configure belt
    ENABLE BELT
    DEFBELT %pick.belt[0] = pick.belt[0], pick.encoder[0], 1, scale
    WINDOW %pick.belt[0] = upstream, downstream

; Start the belt
    conv = VPARAMETER($ip, -1, pick.cam.num[0], beltonoff, , rob.number[0])
    SIGNAL conv

    IF sv.emulate.mode THEN                                      ; Turn the conveyor on in emulation mode
      SETDEVICE (0, pick.encoder[0]-1, stt, 12) emul.belt.vel/scale
    END

    RETURN

.END

.PROGRAM rob.place_0(index)
;
; ABSTRACT:    Perform a place operation at a static location
;
; INPUTS:      None
;
; OUTPUTS:     None
;

    GLOBAL LOC place.loc[]
    LOCAL LOC place_loc
    SET place_loc = TRANS(500,-350,239,0,180,-133)

; Approach and move to the location
    ;APPROS place.loc[0], 25
    ;MOVES place.loc[0]
    APPROS place_loc, 25
    MOVES place_loc
    BREAK

; Actuate the gripper and depart
    CALL rob.grip_0(TRUE)
    APPROS place.loc[0], 25

; Clear any tool applied for the placement operation
```

```
    IF (NOT IDENTICAL(TOOL,grip.tool[0])) THEN
       TOOL grip.tool[0]
    END

    RETURN
.END
```