Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación de visión por computador para multipick robotizado en aplicaciones de empaquetado.

Proyecto Final de Carrera

Ingeniería Informática

Autor: Sergio Navarro Pérez

Director: Martín Mellado Arteche

5/7/2014

Resumen

Se trata de un proyecto de reconocimiento por visión de objetos para posteriormente ser manipulados por brazos robot.

Una de las tareas de empaquetado que mayor beneficio pueden traer a empresas es el multipick de alimentos, de forma que se agarren varios productos a la vez para su manipulación y empaquetado.

Típicamente en las empresas, a la hora de realizar en empaquetado de cualquier objeto, estos se desplazan por una cinta transportadora siendo su posición en la cinta totalmente aleatoria. En este caso se tratan de objetos pequeños.

Se pretende desarrollar una aplicación que mediante visión por computador reconozca los objetos que se desplazan por la cinta. Se obtenga su posición X e Y en base a un sistema de coordenadas. Se seleccione un número N de objetos que serán recogidos de la cinta todos de una vez por el brazo robot y colocados en su caja de manera ordenada (empaquetado). Una vez depositados los objetos en cajas, se vuelve a repetir todo el proceso.

En resumen, lo que se pretende es desarrollar un sistema que recoja objetos que vienen de una cinta transportadora totalmente desordenados para depositarlos en cajas para su empaquetado.

Palabras clave: Visión por computador, multipick, ordenación de objetos, manipulación, empaquetado, brazo robot, robótica.

A mis padres por haber creído siempre en mí, sin ellos no habría llegado tan lejos.

A Martín Mellado por haberme propuesto este proyecto, por la ayuda ofrecida en la realización del mismo, por aquellas cervezas en Reims, CHEERS!

A Adrián Ochoa por ser tan buen compañero. Sin él nuestros proyectos en paralelo no hubiesen llegado a buen puerto.

A Pep y Pablo por su inagotable paciencia ayudándonos con la puesta a punto del robot.

Y en general, gracias a todo el ai2 por dejarnos utilizar sus instalaciones y laboratorios.

Tabla de contenidos

1. Introducción

1.1. Problemática

En toda industria que se dedique a la producción de algún objeto en masa, son necesarios diversos ingenios mecánicos que faciliten la automatización de la producción, ya sean, cintas transportadoras, maquinaria de manipulación de objetos, sensores de cualquier tipo, cámaras de visión, brazos robotizados…

El problema que se trata en este proyecto es el de dar solución a un problema típico en todas las industrias. Los objetos producidos en la industria, al acabar su fabricación, son depositados sobre cintas transportadoras que los llevan hasta la zona de empaquetado. Una vez allí, es donde surge el problema sobre cómo podemos llevar los objetos que se disponen sobre la cinta de manera desordenada a sus cajas.

En definitiva, el problema es el de empaquetar objetos que se disponen de forma totalmente aleatoria sobre una cinta transportadora mediante brazos robot ayudados de un sistema de reconocimiento por visión.

1.2. Motivación

Como en cualquier industria, uno de los objetivos principales es minimizar costes y maximizar el rendimiento. Estos objetivos llevan alcanzándose en gran medida gracias a avances en la tecnología como es el caso de este proyecto.

Mediante la combinación de la programación de computadores, el uso de técnicas de visión artificial por computador y brazos robot se pretende dar una solución al problema planteado y además resolverlo de manera eficiente aumentando la productividad, y reduciendo los costes a largo plazo.

1.3. Objetivos

El objetivo principal de este proyecto es el de dar solución al problema planteado. Para ello se creará primeramente una aplicación software a modo de simulador. En dicha aplicación se pretende simular el comportamiento de la cinta paso a paso y evaluar el comportamiento de cada herramienta utilizada.

Posteriormente se seleccionará una herramienta de las utilizadas en el simulador y se llevará el caso a la práctica comprobando el comportamiento en la realidad.

2. Explicación del problema

Como se ha explicado anteriormente, se pretende recoger de una cinta transportadora objetos idénticos, distribuidos de manera totalmente aleatoria por la cinta. Un ejemplo de cinta transportadora puede ser como el de la imagen siguiente:

Figura 1: Ejemplo de cinta transportadora.

Mediante la ayuda de una cámara y técnicas de visión por computador se pretende obtener la posición de cada objeto sobre la cinta, típicamente sus coordenadas X e Y en base a un sistema de coordenadas.

A continuación se pretende mediante un brazo robot, al cual se le añade una herramienta especial, agarrar un número N de objetos al mismo tiempo. A esto denominamos Multipick. Una vez agarrados se depositan los objetos ordenadamente en su correspondiente caja o similar para ser empaquetados. Este proceso se repite continuamente

En cuanto a la herramienta global del robot, para poder conseguir realizar un Multipick de N objetos simultáneos, esta estará compuesta de N herramientas o módulos independientes entre sí.

Cada módulo o herramienta debe contener un TCP (Tool Center Point), es decir, situamos un sistema de coordenadas justo en la punta de la herramienta, por donde la herramienta agarra los objetos. Puesto que de momento vamos a realizar un simulador, podemos suponer que la punta de la herramienta estará compuesta por una ventosa que realizará una succión para agarrar dicho objeto.

Este TCP o punta de la herramienta o ventosa, debe ser capaz de realizar un desplazamiento en dos dimensiones, X e Y dentro de las dimensiones que forman su módulo. Gracias a este desplazamiento, la punta de la herramienta describe un área que denominaremos Área de trabajo (Work Area, WA)

El hecho de incluir este desplazamiento a cada módulo o herramienta nos da una gran versatilidad a la hora de realizar el multipick de los objetos.

Supongamos que tenemos 4 módulos dispuestos en una matriz 2x2 que conforman la herramienta global del robot. Sin este desplazamiento que hemos definido antes, sería muy difícil hacer encajar los 4 TCP de las 4 ventosas sobre 4 de los objetos que queremos agarrar. Gracias al desplazamiento de cada TCP, podemos hacer coincidir el TCP de cada módulo con cada objeto. De ahí que se obtenga gran versatilidad.

A continuación se muestra un esbozo de un módulo rectangular visto en alzado:

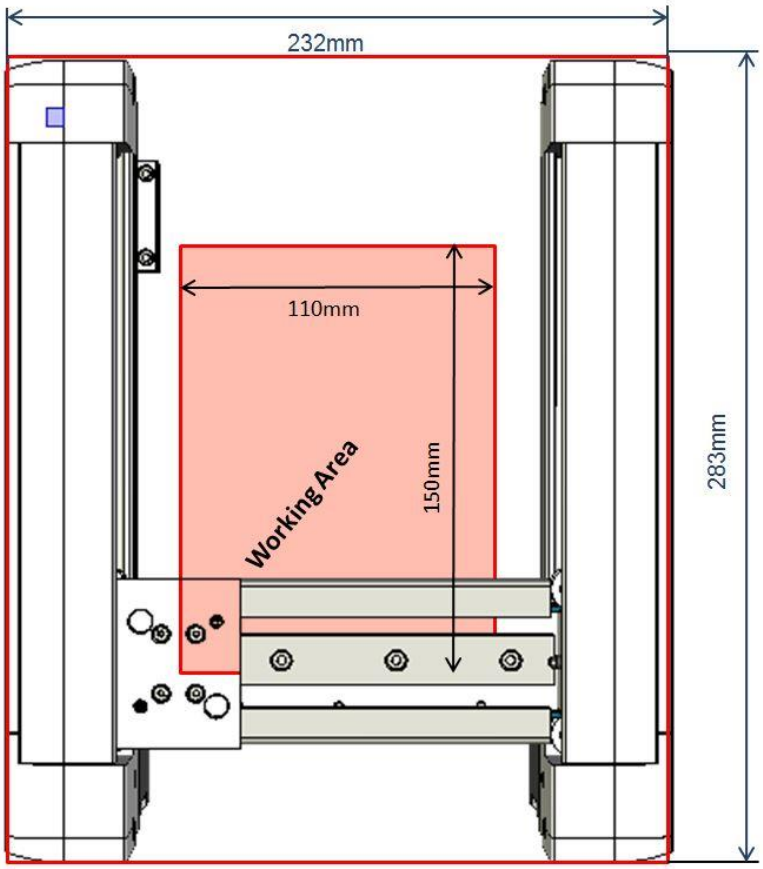

Figura 2: Esbozo de un módulo.

En la imagen se puede observar el cuadro de color salmón que corresponde al Área de trabajo donde la ventosa puede desplazarse.

Las medidas mostradas serían una aproximación a las medidas que podría tener el módulo una vez construido.

Para que el TCP de la herramienta pueda desplazarse son necesarios dos guías o carros (uno montado sobre otro). La primera guía doble sería la responsable de conseguir el desplazamiento en el eje Y (véase las dos guías de color blanco a los extremos izquierdo y derecho de la imagen que consiguen un desplazamiento en Y de 150mm)

La segunda guía simple, montada sobre la primera, consigue el desplazamiento en X (véase la guía de color grisáceo que consigue un desplazamiento en X de 110mm)

Sobre esta última guía va montado el soporte donde se ubica la ventosa, que en la imagen estaría situada en la esquina inferior izquierda del Área de trabajo.

A continuación mostramos la herramienta global compuesta por 4 módulos dispuestos en una matriz 2x2, igual que en el ejemplo descrito anteriormente:

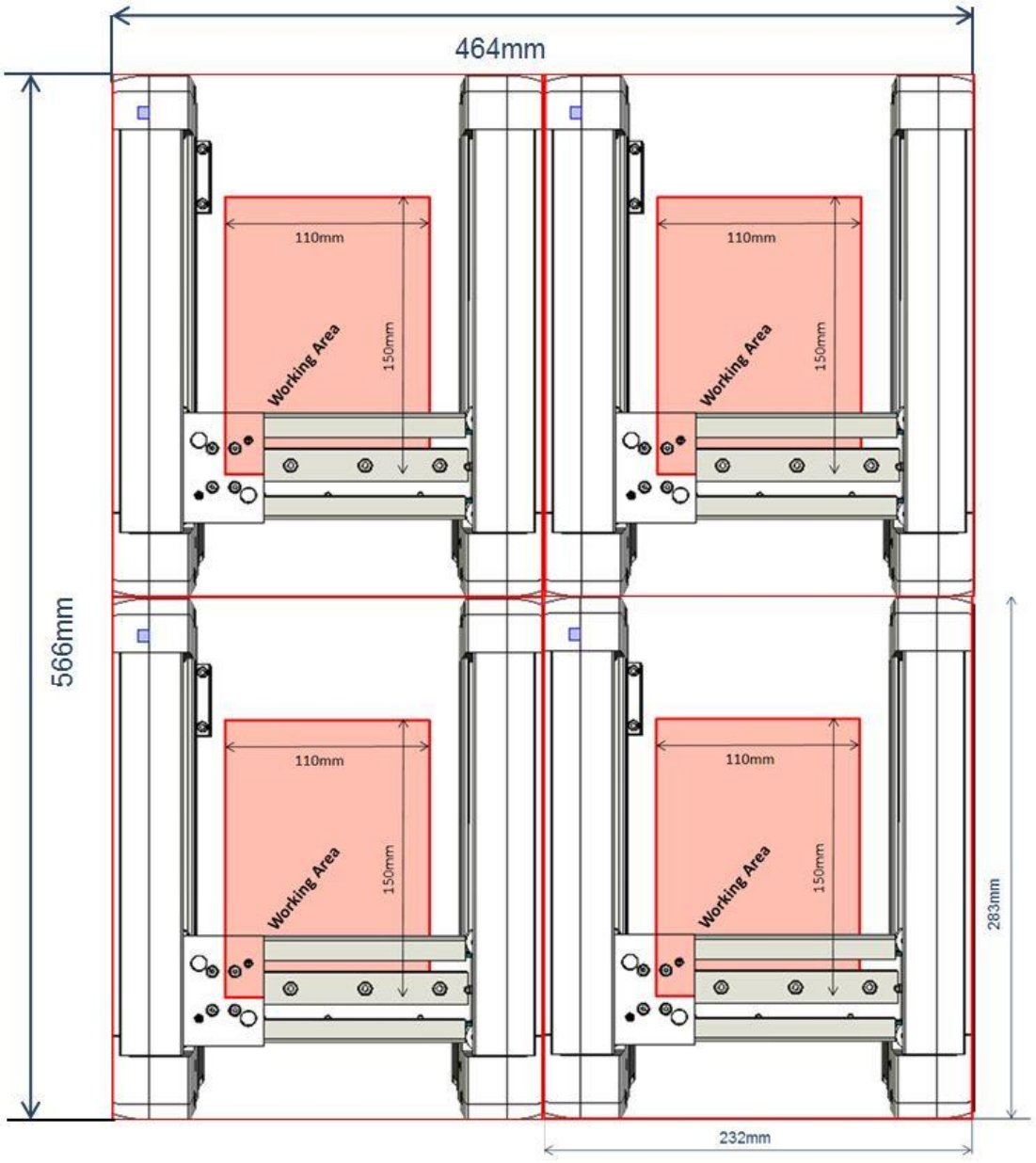

Figura 3: Esbozo de Herramienta compuesta por 4 módulos.

Se puede observar como obtenemos 4 zonas de trabajo, una por cada módulo. De esta manera, podemos seleccionar 4 objetos que encajen dentro de las 4 zonas de trabajo, mover los TCP para que coincidan con el centro de cada objeto y realizar un multipick de los 4 objetos simultáneamente.

Una vez agarrados los 4 objetos, el robot realizaría un movimiento de brazo para desplazarse hasta la posición donde se halle la caja. Para soltar los objetos agarrados, podemos volver a desplazar los TCP para conseguir que los objetos queden los más juntos posibles, o poder dejarlos en la posición que deseemos.

2.1. Situación de partida

La situación de partida fue simplemente la idea de mejorar el típico pick and place que se realiza con un brazo robot. En el cual, se agarran objetos de uno en uno.

Se pretende mejorar este pick and place típico de manera que se agarren varios objetos a la vez. Lo que se podría denominar Multipick and Multiplace.

Para conseguir este Multipick, es necesario diseñar básicamente la herramienta a utilizar y posteriormente desarrollar una serie de algoritmos para el control de la herramienta.

Con un diseño básico de la herramienta nos referimos a un primer diseño en el que solamente se tenga en cuenta aspectos básicos como número de módulos, forma de las áreas de trabajo, etc. Todas las dimensiones de los diferentes aspectos de la herramienta deben ser parametrizables. Para, de esta forma, cuando la herramienta haya sido construida físicamente, en base a este primer diseño, solo sea necesario especificar los tamaños reales de dicha herramienta.

Los algoritmos a desarrollar para el control de la herramienta deben ser capaces de funcionar sean cuales sean los valores de sus parámetros.

2.2. Alternativas de problemas y soluciones

2.2.1. Problemática principal. Herramienta 2x2

Como se ha comentado anteriormente, partimos de la idea de realizar una herramienta con 4 módulos independientes en matriz 2x2.

Al tener 4 módulos necesitamos fijar 4 objetos para agarrarlos. Dónde situar la herramienta para poder agarrar los 4 objetos supone el primer problema a resolver.

En un primer momento se pensó en situar la herramienta sobre el principio de la cinta y comprobar si bajo cada área de trabajo había al menos un objeto que fuese posible agarrarlo. Este ejemplo funciona muy bien cuando la cinta esté repleta de objetos muy juntos.

En el caso de que la cinta no contenga demasiados objetos, la posición de la herramienta sobre la cinta seguramente fallará con alta probabilidad a la hora de conseguir fijar 4 objetos como posibles a ser agarrados.

En tal caso, ¿qué podemos hacer? Una posible solución podría ser ir moviendo ligeramente la herramienta hasta conseguir fijar 4 objetos. Ahora nos planteamos hacia dónde y cuánto mover la herramienta. No resulta fácil proporcionar unos criterios válidos para resolver este problema.

Además, hay que resaltar que mover la herramienta de esta forma constituye una búsqueda totalmente ciega e ineficiente. Puesto que movemos la cinta sin tomar ningún criterio.

Descartada la opción anterior, se pensó en resolver de manera opuesta.

Fijamos el primer punto más avanzado en la cinta. Es decir, el punto que más cerca se encuentre del final de la cinta. Fijamos este punto y a continuación intentamos fijar los otros 3 restantes. Aquí vuelven a surgir nuevos problemas.

Situamos el área de trabajo del primer módulo sobre el objeto fijado. Pero, ¿Resulta adecuado situar la herramienta haciendo coincidir el centro del área de trabajo con el objeto? En este caso perdemos las prestaciones que nos da el área de trabajo puesto que no hacemos uso de ella. Además perdemos combinaciones de 4 objetos que podríamos considerar si usásemos adecuadamente esta área de trabajo (WA)

Situar el módulo haciendo coincidir el centro del WA con el objeto resulta poco apropiado. La solución adoptada ha sido la de considerar para el primer objeto fijado todas las posibles combinaciones de situar el WA sobre el objeto, creando lo que denominamos un Marco Frontera. A continuación fijaríamos un segundo punto, lo cual reduciría el tamaño del Marco Frontera que contiene todas las posibles posiciones de colocar la herramienta. De esta manera procederíamos hasta fijar los 4 puntos. Si por algún motivo no fuese posible fijar 4 puntos, simplemente saltaríamos al siguiente

punto y volveríamos a empezar el proceso de fijado de puntos mediante el marco Frontera.

Nótese que de este modo, en el caso de tener poca densidad de objetos sobre la cinta, no andamos a ciegas en la búsqueda. Sino que vamos directamente buscando de punto en punto. Lo cual si resulta eficiente y adecuado.

Este método de fijado de puntos es el que utilizaremos para todas las herramientas a desarrollar en este proyecto y será explicado en todo detalle más adelante en el apartado 4 de Programación e implementación de algoritmos.

La herramienta de 4 módulos la denominaremos Herramienta2x2.

2.2.2. Simplificación del modelo. Herramienta 2x1

Una vez planteado cómo se va a resolver el problema, pasamos a la fase de la fabricación de la herramienta.

Cabe aclarar en este momento, que la fabricación física de la herramienta no entra dentro de este proyecto. Todo el proceso de diseño detallado, construcción y montaje de la herramienta entra dentro de otro proyecto, la lleva a cabo otro estudiante de Master compañero mío.

Aunque no entra dentro de este proyecto la fabricación de la herramienta, si toca muy de cerca el diseño básico que hablábamos anteriormente sobre ella.

Para la fabricación de un primer prototipo de herramienta se pensó en simplificar el diseño lo máximo posible. Por lo que se decidió en utilizar el mínimo número de módulos en la herramienta. Es decir, en principio, la herramienta que diseñaremos ahora tiene 2 módulos dispuestos en matriz 2x1. Esto es uno a la izquierda y otro a la derecha. A esta herramienta la denominamos, Herramienta2x1.

2.2.3. Mejora del modelo. Herramienta 2x1 Circular

Posteriormente se planteó una mejora en el diseño que afecta a cada módulo por separado. En concreto se pensó en cambiar el diseño general de un módulo. Antes describía un Área de Trabajo rectangular mediante dos guías para obtener

desplazamientos en X e Y. Ahora se plantea cambiar el diseño por otro que describa un área de trabajo circular.

Anteriormente teníamos dos guías lineales y dos motores, uno para cada guía. Ahora lo que planteamos es eliminar una de las guías manteniendo los dos motores.

Por lo tanto el diseño es el siguiente: El primer motor lo atornillamos a la muñeca del brazo robot, esta es la base de la herramienta. Sobre el eje del primer motor atornillamos la guía (en principio atornillada sobre un extremo de la guía lineal). El segundo motor es el que se encarga de mover la plataforma móvil de la guía lineal mediante mecanismo de tornillo sin fin. Sobre esta plataforma móvil se acoplará la ventosa o el utensilio de agarre adecuado.

De esta manera conseguimos mediante la rotación de la guía y el desplazamiento lineal sobre la misma guía un área de trabajo circular. A continuación mostramos una imagen de la guía lineal ya que diseñó mi compañero en su proyecto.

Figura 4: Fotografía de la Guía Lineal ya diseñada

En la imagen se puede apreciar el tamaño de la guía aproximadamente.

A continuación se muestra un par de imágenes de la guía realizadas por mí hechas con SketchUp, un programa de diseño 3D muy sencillo e intuitivo. Válido para hacer bocetos sencillos en 3D.

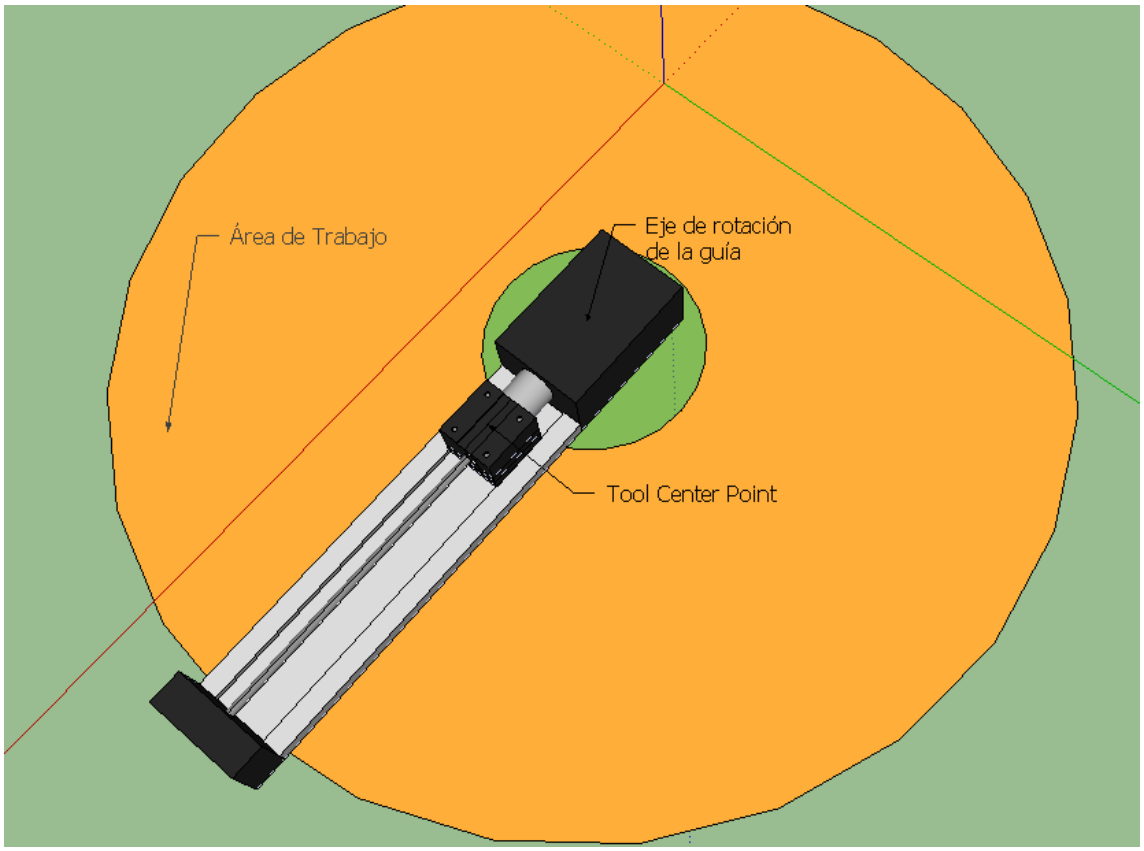

Figura 5: Diseño 3D de la guía mediante SketchUp

En la imagen se pretende mostrar el Área de trabajo en color naranja, sobre la que podríamos agarrar objetos. Se puede observar que se trata de un área circular tipo "donuts", puesto que en el centro del área tenemos una zona donde no alcanzamos con la guía. El eje de rotación de la guía. Y el TCP donde irá anclada la ventosa o artefacto usado para agarrar los objetos.

A continuación se muestra otra imagen de la guía:

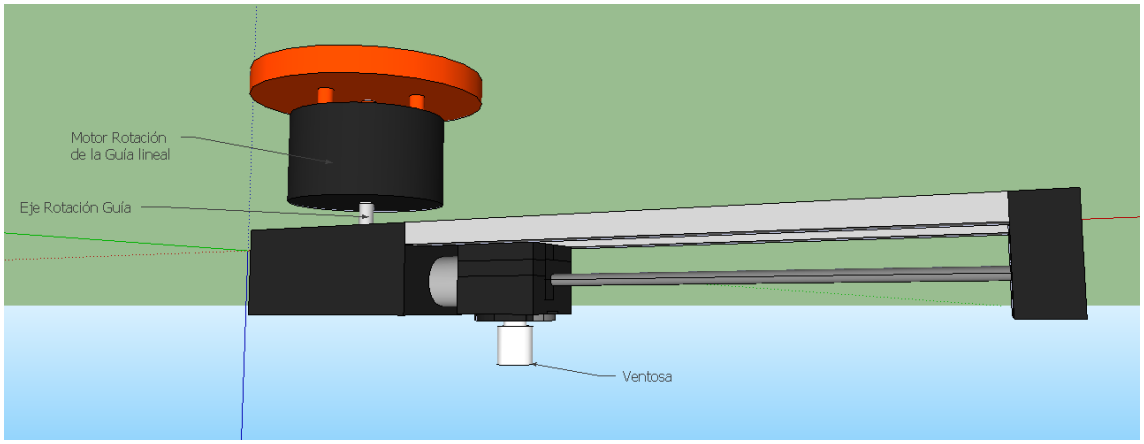

Figura 6: Diseño 3D Guía lineal completa.

En esta imagen se muestra la guía completa. Se puede observar dónde va la ventosa anclada. También se puede observar el motor que permite a la guía rotar.

La imagen número 6 muestra el diseño de uno de los módulos que formarían el grupo que denominamos herramientas circulares.

Este tipo de módulos circulares presentan ventajas e inconvenientes.

La ventaja más clara que presentan los módulos circulares frente a los rectangulares de dos ejes perpendiculares mostrados anteriormente. Es que los circulares presentan un menor coste de fabricación porque tenemos un eje lineal con su motor, y otro motor de rotación. En el caso de los módulos rectangulares teníamos dos ejes lineales con sus dos motores. Es decir, nos ahorramos una guía lineal con su correspondiente mecanismo.

Otra ventaja respecto a los módulos rectangulares es que, en principio, el movimiento de rotación podría llegar a ser mucho más rápido que el movimiento de tornillo sin fin necesario para moverse a lo largo de una guía lineal.

Una posible desventaja de los módulos circulares frente a los rectangulares, es la posible pérdida de precisión. En los módulos circulares, conforme nos alejamos del centro de rotación vamos perdiendo precisión. Esto es debido a que pequeñas variaciones de posición en el eje de rotación provocan grandes variaciones de posición en el extremo de la guía lineal. Por lo tanto el motor que se encargue de la rotación de la guía deberá tener muy buena precisión.

2.2.3.1. Herramienta circular 2x1

Todo esto en cuanto al módulo circular. Para integrar el módulo circular dentro de la herramienta circular que utilizaremos en el caso práctico se ha pensado en simplificar al máximo la herramienta pensando en el coste de fabricación.

En concreto se ha pensado en suprimir uno de los módulos circulares haciéndolo fijo. De este modo tenemos, a un lado tenemos un módulo circular como hemos descrito anteriormente y al otro lado tenemos una especie de módulo fijo que solo consta del elemento de agarre sin posibilidad de cambiar su posición.

A continuación se muestra una imagen con la herramienta 2x1 Circular:

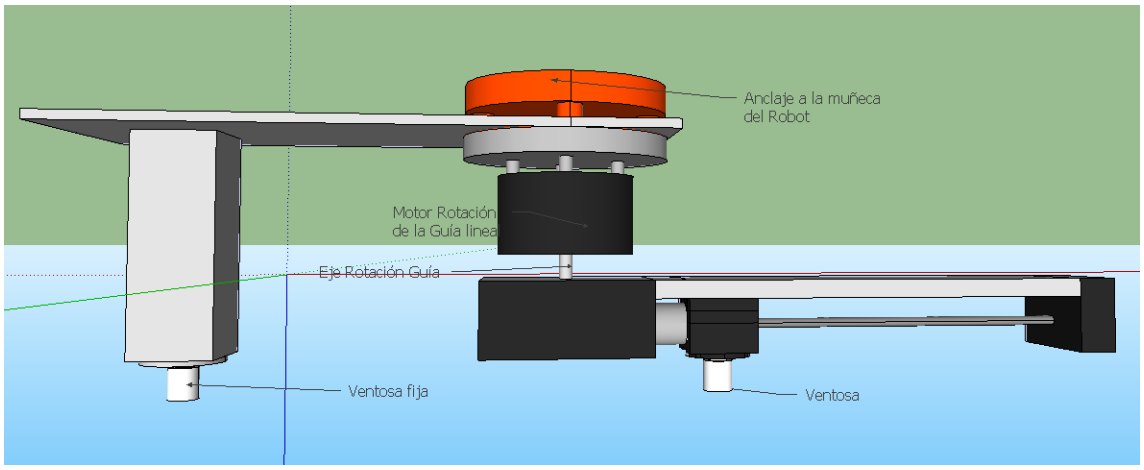

Figura 7: Diseño 3D de la Herramienta 2x1 Circular

Como se puede observar, a la izquierda tenemos el módulo fijo, y a la derecha tenemos el módulo circular. Ambos elementos de agarre tienen asignado un TCP para poder hacer coincidir la ventosa en este caso, con el objeto a agarrar. Ambos elementos se encuentran a la misma altura.

En la parte superior se puede ver la zona de la herramienta donde iría acoplada a la muñeca del robot.

También se puede observar que se ha limitado la rotación de la guía lineal a 180º. De este modo ahora el área de trabajo del área móvil queda reducida a una forma de "D"

Cabe decir que con esta herramienta será necesario trabajar en coordenadas polares donde usaremos como parámetros: el Ángulo entre un eje fijo que definiremos y la guía lineal, y una distancia que se corresponde con la distancia entre el centro de rotación y el TCP móvil situado en la ventosa anclada a la guía.

2.2.3.2. Interfaz Software con el Robot ABB

En cuanto a la interfaz software que se usará con el robot hay que comentar un par de cosas.

Se va a utilizar un robot de ABB IRB 360 FlexPicker. El robot está montado sobre una cinta transportadora. Sobre la cinta también hay instalada una cámara de visión. Todo ello está controlado por una aplicación llamada PickMaster de la compañía ABB. Esta aplicación se encarga del reconocimiento por visión de los objetos para obtener sus coordenadas. Por lo tanto la parte de visión que he desarrollado en este proyecto solo será utilizada en la parte del simulador.

Para comunicar las funcionalidades de mi aplicación con PickMaster se ha optado por crear una librería dinámica DLL en C++ pero con posibilidad de usarse en otros lenguajes de programación. Esto es así porque para comunicar con PickMaster, hace falta crear un proyecto en el lenguaje de programación C# e incluir en él todas las funcionalidades creadas en mi aplicación. El cómo se ha hecho esto se explicará más detalladamente en el apartado 3.5 Creación de librería DLL.

2.3. Herramientas sw/hw utilizadas

Las herramientas software utilizadas han sido Visual Studio 2008 para programar sobre C++ principalmente, y puntualmente sobre C# para comunicar con el Robot ABB.

Se ha utilizado la librería gráfica OpenCv sobre C++ para el desarrollo de la parte de reconocimiento por visión del simulador.

Se ha usado el programa CAD SketchUp para el diseño de los módulos y herramientas mostrado en imágenes en este proyecto.

Todo esto ha estado funcionando sobre Microsoft Windows 7 Ultimate.

3. Desarrollo de Algoritmos

Ahora explicaremos la idea que siguen los algoritmos y los explicaremos de manera intuitiva.

Recordamos mediante un diagrama de flujo cómo procede la aplicación:

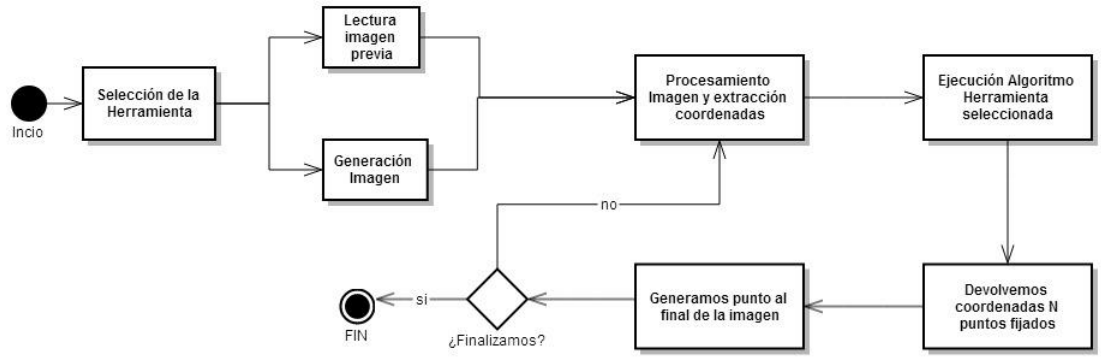

Figura 8: Diagrama flujo.

Todo lo explicado a continuación se corresponde con el simulador. El ejemplo práctico es un caso concreto del simulador al que se le han aplicado unos pequeños cambios. El ejemplo práctico corresponde al último punto de este apartado.

3.1. Generación de imágenes y objetos

Primeramente se empezó a trabajar con una imagen muy sencilla generada manualmente.

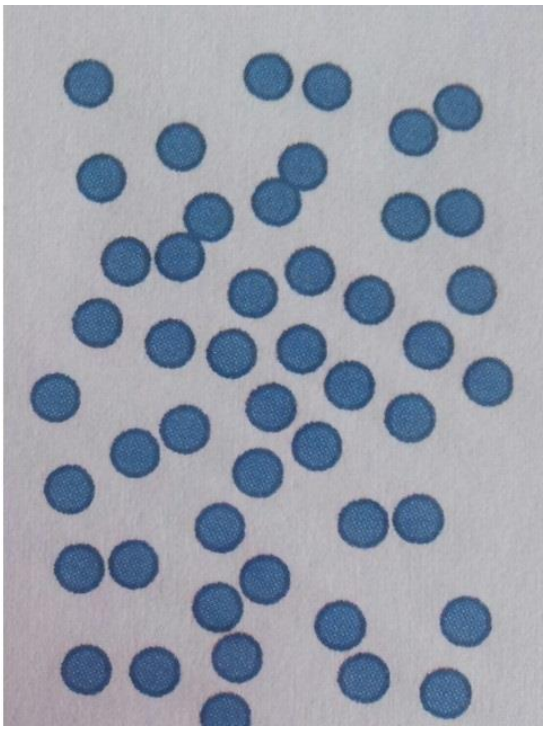

Figura 9: Simulación de objetos a agarrar, primera imagen con puntos.

Los puntos azules simulan los objetos depositados sobre la cinta. La cinta consideramos que se desplaza de abajo a arriba, es decir, los primeros objetos a agarrar deberían ser los puntos situados más arriba en la imagen.

Primeramente se empezó a trabajar con esta imagen. Como se consideró que era necesario trabajar con más distribuciones de puntos para comprobar el correcto funcionamiento del simulador se decidió crear un generador de imágenes.

El generador tiene dos partes: La primera parte es la generación propiamente dicha de la imagen. Para generarla necesitamos indicar el ancho y alto de la imagen en pixeles y el número de objetos que deseamos generar.

A continuación se crea una imagen vacía y se van añadiendo los puntos. Los puntos son generados aleatoriamente. Si se genera un objeto encima de otro, este último se descarta, volviendo a intentar generar una posición válida para este. Se permite que dos objetos se “toquen”, es decir, que estén juntos, pero no que estén uno sobre otro.

La segunda parte del generador es la simulación del movimiento de la cinta. Tras cada iteración (cada pick and place) la cinta (la imagen) se desplaza un cierto número de pixeles hacia arriba. A continuación se intenta generar un objeto en la parte baja de la imagen que ha quedado libre de objetos por el desplazamiento. Si tras un cierto

número de intentos de generar un objeto no se ha conseguido, se toma por imposible y no se genera ningún objeto.

A continuación mostramos un ejemplo de imagen generada:

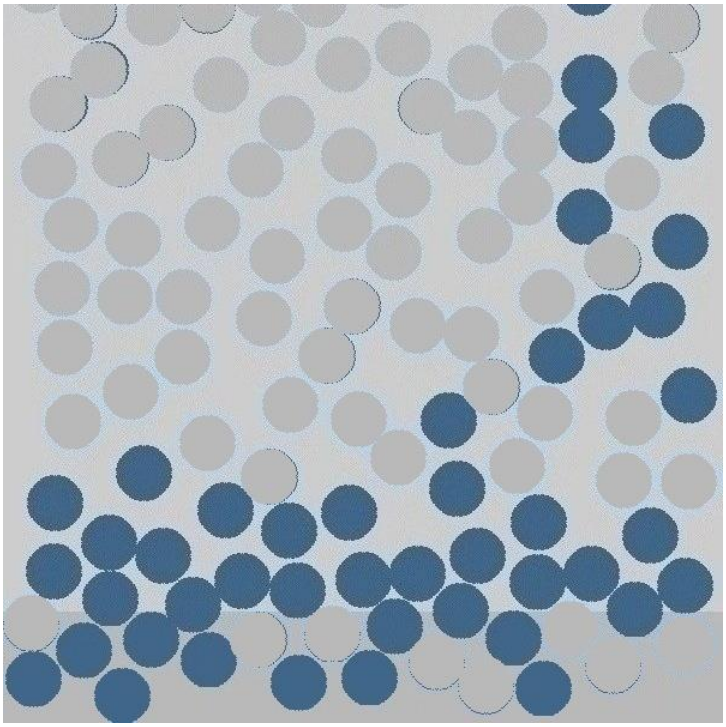

Figura 10: Ejemplo de imagen simulada aleatoriamente

En la imagen se puede apreciar el resultado de la generación de puntos tras bastantes iteraciones. Los puntos grises son objetos que ya han sido recogidos. La zona gris oscura de la parte baja de la imagen representa el trozo de cinta que ha avanzado. Los puntos azules en la parte gris oscura son los objetos que han sido generados tras el avance de la cinta.

La imagen está generada con muchos puntos para comprobar si la generación era correcta cuando la cinta se encontraba saturada de objetos.

3.2. Reconocimiento por visión

El objetivo principal del reconocimiento por visión es extraer de la imagen las coordenadas X e Y correspondientes al centro de cada objeto que se encuentra sobre la cinta.

Para ello se trata la imagen aplicando una umbralización. Con esta técnica obtenemos una imagen binaria en blanco y negro puro. A continuación aplicamos una serie

erosiones. En el proceso de erosión se recorre la imagen entera pixel a pixel. Cuando encontramos un cambio de pixel blanco a negro quiere decir que nos encontramos en la frontera o borde del objeto. En ese momento cambiamos el pixel negro por blanco. Por lo que el efecto final, es similar a una erosión del objeto.

Esta erosión la realizamos varias veces para separar posibles objetos que hayan quedado muy juntos y puedan considerarse como un único objeto dando lugar a errores en el reconocimiento de objetos.

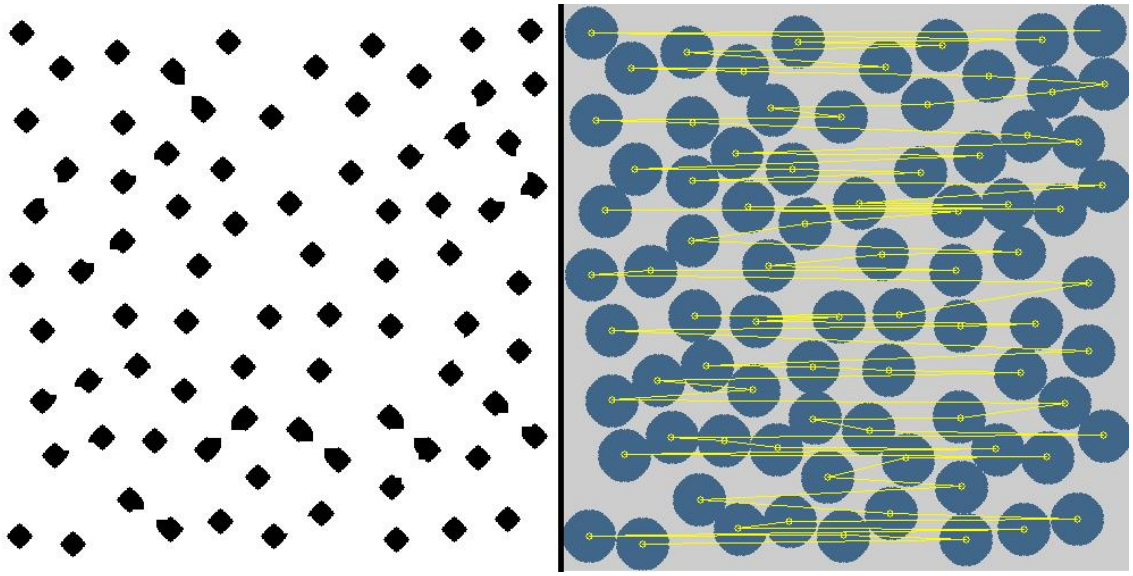

Figura 11: Binarización + Representación centroides

En la imagen de la izquierda podemos ver el resultado de procesar una imagen original de 500 x 500 pixeles y 80 objetos.

Una vez procesada la imagen obtenemos las coordenadas de los objetos gracias a una función contenida en OpenCv.

Y representamos las coordenadas X e Y sobre la imagen original, (imagen nº 10 derecha). Podemos comprobar como todos los centroides de los objetos, representados como un círculo pequeño amarillo, encajan perfectamente sobre cada objeto. Adicionalmente se ha añadido una línea amarilla entre dos puntos para ver el orden que llevan puntos. Se aprecia, que llevan una ordenación ascendente en la coordenada Y.

Cabe decir que el origen de coordenadas de la imagen está situado en la esquina superior izquierda. La coordenada X aumenta si nos desplazamos hacia la derecha en la imagen. La coordenada Y aumenta si nos desplazamos hacia abajo en la imagen.

En los siguientes apartados, detallaremos las ideas que siguen los algoritmos diseñados para cada herramienta. Estos trabajan sobre las coordenadas de los objetos que hemos extraído de la imagen.

3.3. Algoritmo para Herramienta 2x2

Aquí explicaremos de manera intuitiva las ideas que sigue el algoritmo para la herramienta de 4 módulos rectangulares dispuestos en matriz 2x2.

3.3.1. Marco Frontera

Anteriormente habíamos introducido el concepto de Marco Frontera y la forma en la que vamos a ir fijando puntos hasta conseguir 4 que puedan ser agarrados.

No resulta adecuado fijar un primer punto y situarlo sobre el centro de uno de los módulos, porque directamente no hacemos uso del módulo. Frente a esto, se ha pensado en calcular todas las posibles posiciones en las que podemos situar uno de los módulos sobre el punto que acabamos de fijar. De esta manera integramos dentro de un área rectangular a la que llamamos Marco Frontera la suma de todas las combinaciones posibles de colocar el área de trabajo sobre el punto.

A continuación mostramos unas imágenes que aclara el concepto de marco frontera:

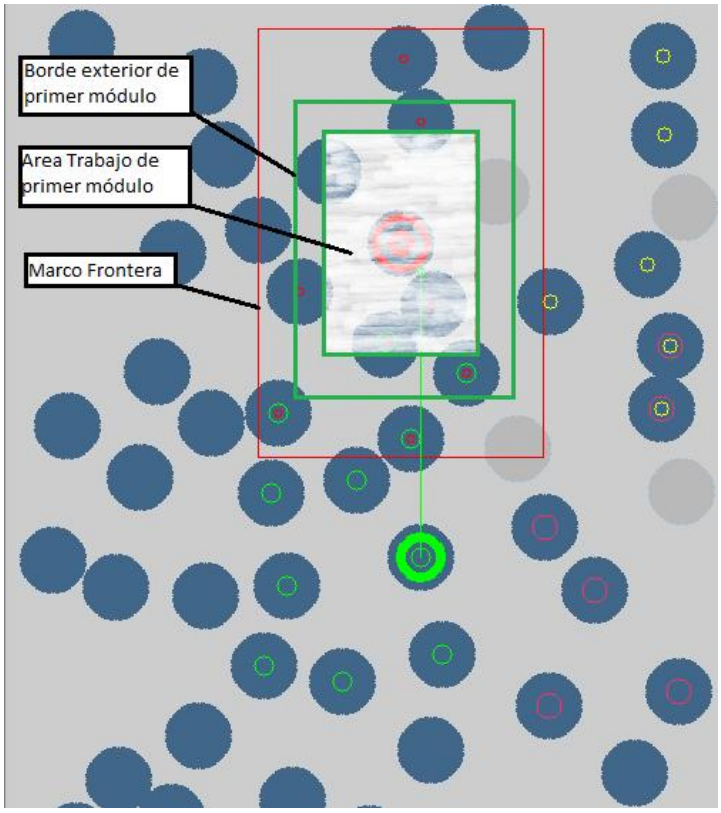

Figura 12: Representación gráfica de un módulo rectangular.

En la imagen se puede observar un módulo rectangular dibujado en verde. El borde exterior del módulo representa el tamaño total del módulo visto desde arriba. El borde verde interior, representa el área de trabajo donde sería posible agarrar un objeto. El rectángulo más exterior de color rojo representa el Marco Frontera que a continuación mediante imágenes pasamos a explicar.

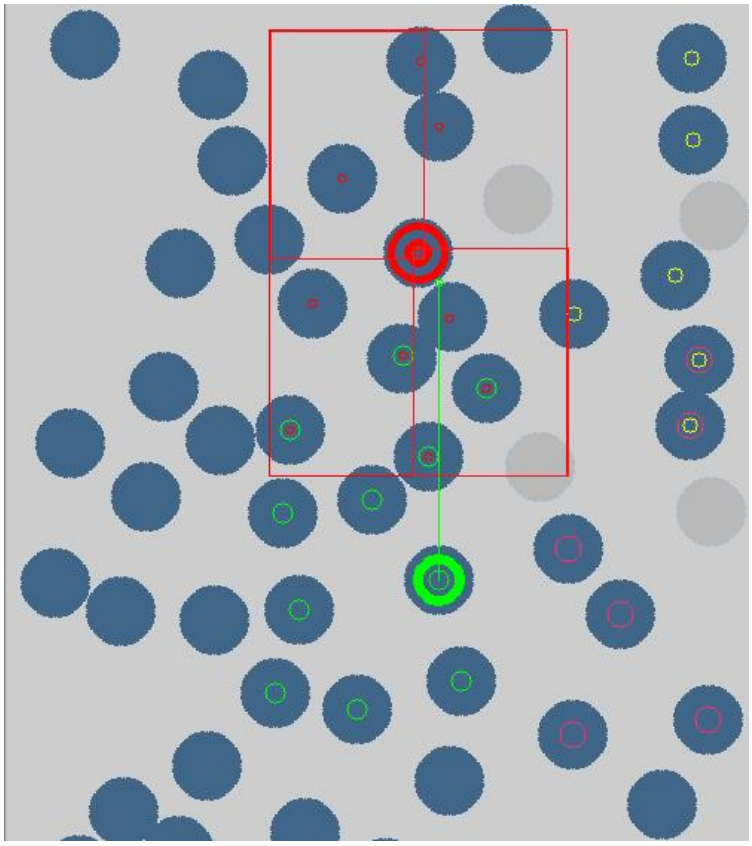

Figura 13: Obtención Marco Frontera para un objeto.

En la imagen se puede observar el marco frontera que está representado por el rectángulo rojo exterior. Los 4 rectángulos interiores de borde rojo representan cada una, cuatro posibles formas de colocar el área de trabajo del módulo rectangular para agarrar el objeto que se encuentra en el centro de color rojo.

Esas cuatro posiciones del área de trabajo del módulo están especialmente elegidas porque recogen las 4 posiciones extremas a las que podemos llevar el área de trabajo teniendo fijado únicamente el objeto central de color rojo. De esta forma, el rectángulo exterior rojo forma lo que llamamos el Marco Frontera. En otras palabras, los objetos que caigan dentro del marco frontera son susceptibles de ser agarrados.

El marco frontera calculado anteriormente es el correspondiente al Marco Frontera de tipo 1. MF tipo 1 significa que contiene 1 elemento fijado en su interior. Tipo 2 es aquel marco que contiene 2 objetos. Y Tipo 3 el que contiene 3 objetos. No será necesario calcular un Marco frontera tipo 4 porque no lo necesitamos. Con un marco tipo 3 que contiene 3 puntos fijados en su interior podemos calcular el cuarto punto a fijar. Si tuviésemos que fijar 5 puntos si necesitaríamos un marco tipo 4.

A medida que incrementamos el tipo del marco frontera, este se irá haciendo más pequeño. Esto es así porque dentro contiene más puntos fijados. Cuanto más alejados estén estos del primer punto fijado más pequeño será el siguiente Marco Frontera.

Se puede decir, que el Marco Frontera contiene todas las posibles formas de colocar el área de trabajo sobre los puntos fijados hasta el momento.

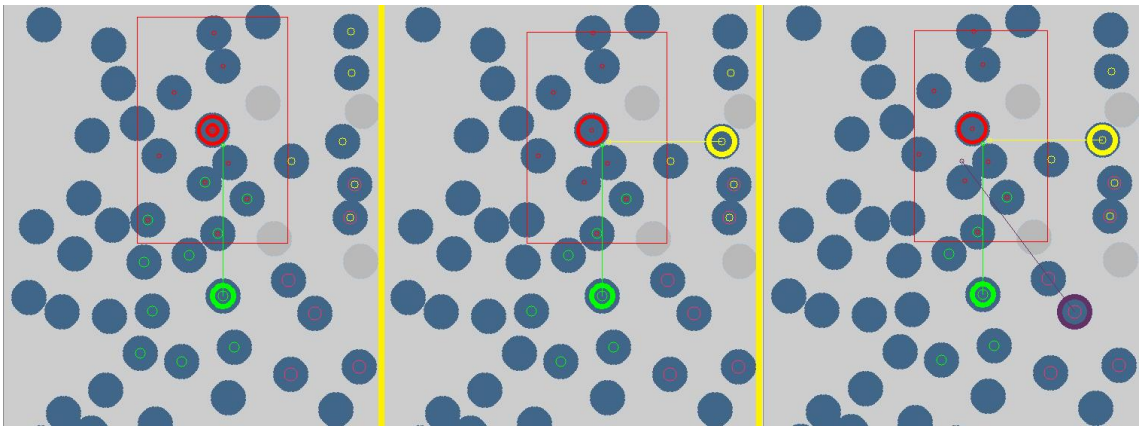

Figura 14: Marcos Frontera tipos 1, 2 y 3.

En la imagen anterior se puede ver el proceso que sufre el marco frontera tras pasar por los distintos tipos. De izquierda a derecha tenemos, MF tipo 1, MF tipo 2 y MF tipo 3.

Se puede apreciar un ligero cambio de tamaño en los bordes superior e izquierdo a través de las fases.

Los 4 puntos fijados son aquellos 4 que están marcados con un círculo de borde gordo. Se puede apreciar como tenemos 4 colores diferentes. Esto es debido a que tenemos 4 tipos de puntos.

3.3.2. Tipos de puntos

Ha sido necesario crear 4 tipos de puntos, uno correspondiente a cada uno de los 4 módulos.

Por lo tanto tenemos punto tipo 1, correspondiente al módulo superior izquierdo, marcado de color rojo.

Punto tipo 2, correspondiente al módulo superior derecho, marcado de color amarillo.

Punto tipo 3, correspondiente al módulo inferior izquierdo, marcado de color verde.

Y punto tipo 4, correspondiente al módulo inferior derecho, marcado de color morado.

Es necesario hacer esta distinción para no fijar dos puntos del mismo tipo.

3.3.3. Clasificación dentro de los 4 tipos de puntos y compatibilidad

El hecho de que un punto haya sido clasificado en alguno de los 4 tipos significa que es un punto compatible para al menos un tipo de punto. Un punto puede ser de varios tipos a la vez.

Que un punto sea compatible significa que es un punto válido para poder ser fijado. Estos puntos compatibles son marcados con un círculo pequeño y fino, como se puede observar en las imágenes anteriores. El color del círculo indica el tipo al que pertenece según la clasificación de puntos anteriormente descrita. En las imágenes anteriores se puede observar como algunos puntos tienen más de un círculo de distinto color, esto es porque se trata de círculos que pertenecen a más de un tipo a la vez.

Si un punto no está marcado con ningún círculo, significa que es un punto no clasificado. Por lo tanto se trata de un punto incompatible y no lo consideramos a la hora de fijar puntos.

A cada punto se le aplica un desplazamiento en X e Y según su tipo. El desplazamiento viene dado por el tipo de punto que tengamos. Este desplazamiento nos sirve para solamente tener que calcular un Marco Frontera válido para todo tipo de puntos, y no construir 4 MF, uno para cada tipo. Con el desplazamiento, se podría decir que solapamos todos los puntos como si fuesen del mismo tipo pero sin confundirlos de tipo.

Los tipos de desplazamiento se pueden observar en las imágenes anteriores como líneas rectas del color del tipo al que corresponda el punto.

Para calcular la compatibilidad de un punto se comprueba que las coordenadas X e Y del punto desplazado caigan dentro del Marco Frontera. Si esto es así lo marcamos con un círculo pequeño del tipo correspondiente. Para cada punto comprobamos si es compatible para todos los tipos.

Cabe decir que toda la información visual que estamos explicando aquí, la aplicación la mantiene guardada interiormente. Es decir, esta información visual de puntos, colores, y líneas, no es más que la representación gráfica y visual de la información contenida interiormente por la aplicación. Esta información visual es muy útil para ver la evolución del proceso de fijado de puntos y comprobar que todo funcione correctamente.

3.3.4. Elección del punto a fijar

Una vez clasificados todos los puntos procedemos a fijar uno de ellos. La elección del punto a fijar se basa en un parámetro que hemos considerado. Este parámetro es la distancia al centro del área de trabajo del módulo.

Es decir, de todos los puntos clasificados (compatibles) fijaremos un punto cuyo tipo aun no haya sido fijado, y su distancia al centro del Área de Trabajo sea mínima.

Esto lo hacemos para minimizar el tiempo que la herramienta debe dedicar a mover las guías lineales para hacer coincidir el TCP de la ventosa con el objeto en cuestión.

Una vez fijado el punto volvemos a calcular el Marco Frontera para los puntos fijados hasta ahora y procedemos a fijar un nuevo punto. Así hasta que hayamos fijado 4 puntos. Para esta herramienta procedemos de la siguiente manera:

Fijar 1er punto -> Crear MF tipo1 -> Fijar 2º pto -> Cear MF tipo2 -> Fijar 3er pto -> Crear MF tipo 3 -> Fijar 4º pto -> Devolver coordenadas de los 4 puntos.

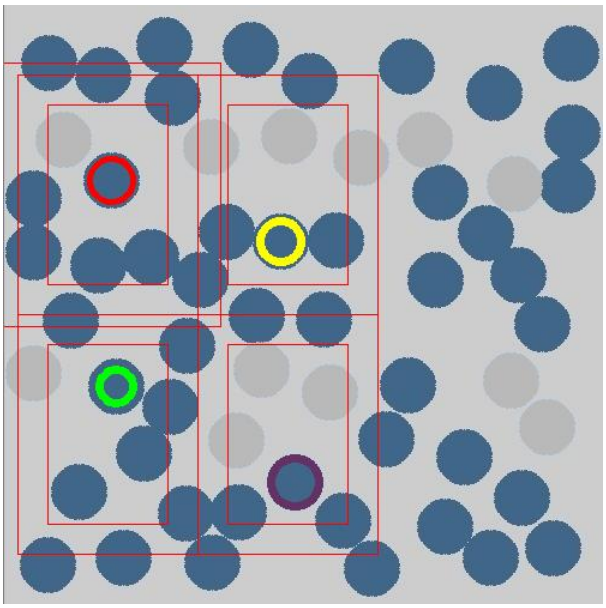

Figura 15: Resultado de fijar 4 ptos.

En la imagen se puede observar un ejemplo de cuál sería el resultado de fijar 4 puntos. Se puede observar los 4 puntos fijados con un círculo de color. Cada uno de un tipo. Y también la posición de la herramienta sobre la cinta transportadora. El cálculo de la posición de la herramienta será explicado en la parte de Programación e Implementación.

3.4. Algoritmo para Herramienta 2x1

Esta segunda herramienta se trata de una simplificación de la explicada en el apartado anterior.

Consta de dos módulos rectangulares iguales que la herramienta 2x2. En este caso tenemos un módulo rectangular en el lado izquierdo y otro en el derecho.

El algoritmo para esta herramienta es una simplificación del algoritmo para la herramienta anterior. Es decir se comporta igual solo que con dos módulos en lugar de cuatro. Haremos un pequeño resumen para ver las pequeñas diferencias:

3.4.1. Resumen

La manera de trabajar es igual. Fijamos un primer punto, punto líder. A partir de este punto calculamos el Marco Frontera del mismo modo que para la herramienta anterior.

Para esta herramienta solo necesitaremos un Marco Frontera de tipo 1. El MF de tipo 2 no será necesario calcularlo puesto que solo fijamos dos puntos.

Una vez calculado el MF, pasamos a calcular todos los puntos compatibles. Al igual que para la herramienta 2x2 teníamos 4 tipos de puntos compatibles. Para esta solo tenemos 2 tipos puesto que solo tenemos dos módulos. Tipo 1 puntos correspondientes al módulo izquierdo marcados en color rojo. Tipo 2, puntos correspondientes al módulo derecho marcados en color amarillo.

A los puntos compatibles también les aplicamos un desplazamiento. A tipo 1 les aplicamos el desplazamiento nulo. Y a los de tipo 2 les aplicamos un desplazamiento para solapar todos los puntos sobre los de tipo 1. Se podría decir que al desplazar los puntos hacemos una conversión de tipo. La idea es solapar todos los puntos como si fuesen del mismo tipo para así trabajar sobre el mismo Marco Frontera.

Una vez calculado todos los puntos compatibles. Calculamos aquel punto que obtenga menor error cuadrático en distancia respecto al objeto líder. Es decir, elegimos aquel que tenga menor distancia al objeto líder fijado en primer lugar.

El proceso sería el siguiente:

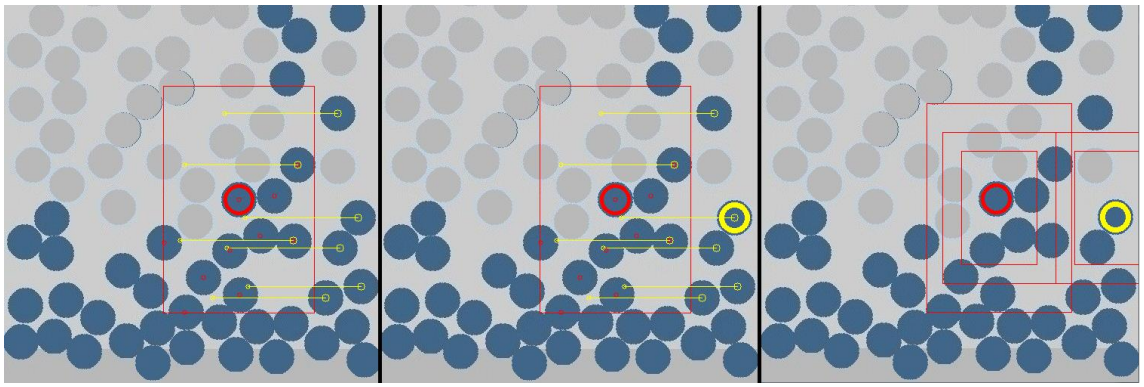

Figura 16: Proceso de fijación de puntos.

En la imagen de la izquierda se puede observar el MF tipo 1 calculado representado por un rectángulo rojo, además de los puntos compatibles de tipo 1 y tipo 2, en círculos pequeños rojos y amarillos respectivamente.

En la imagen central se puede observar cual ha sido el punto tipo 2 calculado para ser fijado. Se puede observar como es el más cercano (en coordenadas desplazadas) al punto líder en color rojo.

En la imagen de la derecha se puede ver cuál sería el resultado final. Se observa la posición de la herramienta sobre la cinta transportadora y los dos objetos fijados.

3.4.2. Nueva funcionalidad

Para esta Herramienta se ha introducido una nueva funcionalidad pensando en simplificar el diseño para abaratar costes de fabricación para el modelo que se construya posteriormente.

Esta nueva funcionalidad es la capacidad de hacer fijo uno de los dos módulos, cualquiera de ellos. El hecho de fijar uno de los módulos significa que la ventosa es fija, no puede desplazarse en el plano X, Y. Esto es útil si, a la hora de construir la herramienta, se decide por tener una ventosa fija a un lado y un módulo rectangular al otro lado. Para el primer prototipo construido esto es así porque se ha empezado por construir el diseño más simple posible, y así simplificar costes.

3.4.3. Resumen con módulo izquierdo fijo

El Marco Frontera para una herramienta que tenga el módulo izquierdo fijo resulta ser igual al Área de Trabajo del módulo en cuestión. Esto es así porque al no poder moverse

la ventosa del módulo izquierdo, obliga a la herramienta a posicionarse justo para que el TCP de la ventosa izquierda encaje con el objeto que se va a agarrar en ese módulo. Por lo tanto solo tenemos como área para agarrar otro objeto, el área de trabajo del segundo módulo.

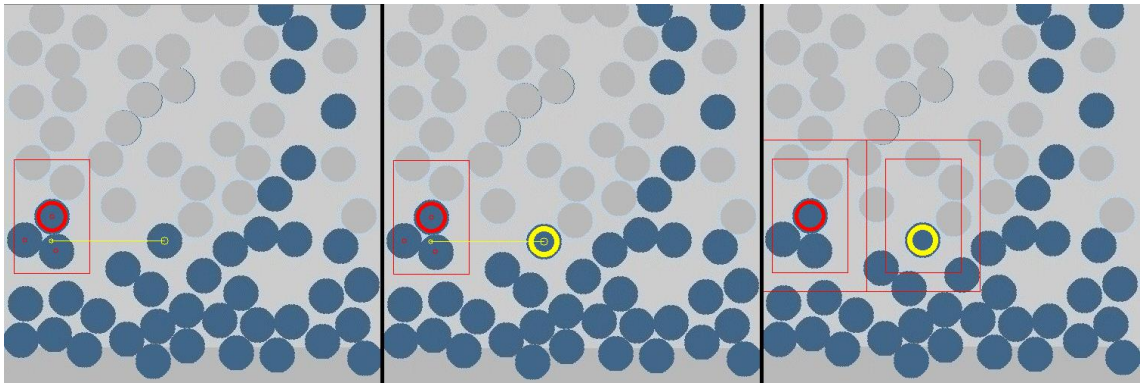

Figura 17: Proceso de fijado de puntos con módulo izquierdo fijo.

Se puede observar el proceso seguido para fijar dos puntos. A la izquierda el Marco frontera tipo 1, más pequeño que en el caso anterior que teníamos los dos módulos libres. En el centro, el punto seleccionado. Y a la derecha el resultado final, con la posición de la herramienta. Nótese que en la imagen derecha no se puede apreciar el Marco Frontera porque coincide exactamente con el Área de trabajo del módulo izquierdo.

3.5. Algoritmo para Herramienta 2x1 Circular

Anteriormente se ha explicado por qué vamos a usar módulos circulares para la construcción del prototipo. Es más, este prototipo tendrá el módulo izquierdo fijo, y el derecho será un módulo circular, pero con algunas restricciones.

Para esta herramienta no hay mucho que decir, puesto que solo hemos cambiado el área de trabajo. De ser rectangular, ahora pasa a ser circular. La forma de trabajar con módulos circulares es exactamente igual que en las herramientas anteriores.

También decir, que mantenemos la ampliación de poder elegir entre módulo izquierdo como fijo o libre.

3.5.1. Marco frontera circular

Para este caso, el Marco Frontera describe una forma circular. Solamente necesitaremos Marco Frontera tipo 1 puesto que como en el caso anterior, solo tenemos 2 puntos a fijar. Por lo tanto, el MF tipo 1 es simplemente un circulo. Nótese que si

tuviésemos que calcular un Marco Frontera de tipo mayor a 1, el marco no sería de forma circular, sino más bien de forma ovalada. No entra al caso, porque solamente tendremos que calcular MF tipo 1.

3.5.2. Objetos compatibles

Una vez calculado el MF tipo 1 circular. Pasamos a calcular los objetos compatibles de tipo 1 y tipo 2. Rojos y amarillos como en el caso anterior. Para este caso es un poco diferente. Un punto es considerado compatible si cae dentro del Marco Frontera, pero en este caso, el único cambio, es que el Marco Frontera es circular. Eso es todo.

3.5.3. Elección del punto a fijar

Una vez calculados todos los puntos compatibles, pasamos a elegir uno de ellos. Como anteriormente se ha explicado, el objeto elegido es aquel que menor error cuadrático presente en distancia al líder. Es decir, aquel objeto cuyas coordenadas desplazadas presenten menor distancia con el objeto líder.

A continuación ponemos dos ejemplos del proceso seguido para fijar dos puntos:

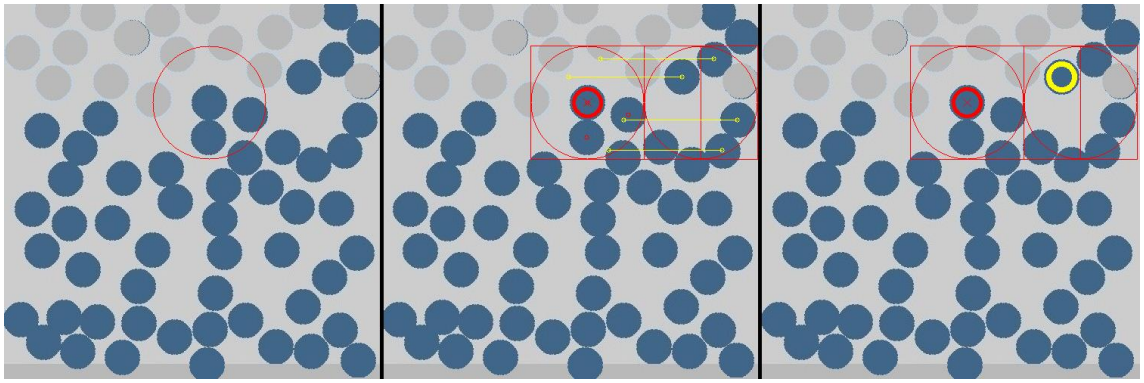

Figura 18: Proceso fijado, Herramienta circular con módulo fijo.

Se puede observar en el ejemplo anterior como el proceso seguido es igual a las herramientas anteriores. En este caso tenemos Módulos circulares. El módulo izquierdo es un módulo fijo, sin movimiento. Y el derecho es un módulo libre.

A la izquierda se puede observar el cálculo del Marco Frontera tipo 1. En el centro, se puede ver los objetos compatibles que han sido calculados. Y a la derecha, El resultado de fijar dos puntos. Nótese como el módulo izquierdo al ser fijo, fuerza a la herramienta a situar el centro del módulo izquierdo sobre el objeto fijado de tipo 1. Y también cómo el área del Marco frontera es exactamente como el área de trabajo del módulo derecho.

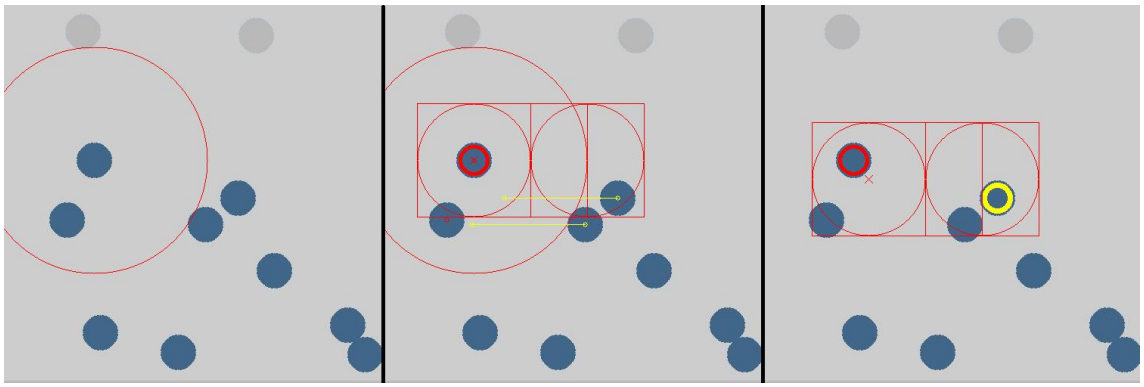

Figura 19: Proceso fijado, Herramienta circular con módulo fijo.

En este ejemplo utilizamos la herramienta circular, pero con sus dos módulos libres, es decir, son capaces de moverse dentro de la zona de trabajo.

Se puede observar a la izquierda el Marco Frontera tipo 1 como es más grande que en el ejemplo anterior, que usábamos uno de los módulos fijos. En el centro se puede observar los puntos compatibles calculados. Y a la derecha se puede observar el resultado. Nótese que en este caso, el objeto de tipo 1 fijado en el módulo izquierdo, no encaja con el centro de la herramienta. En el de la derecha tampoco. La herramienta se sitúa en un punto intermedio entre los dos objetos calculados. De esta manera minimizamos el desplazamiento que tiene que mover un solo módulo repartiéndolo entre ambos.

4. Programación e implementación

En este apartado vamos a explicar en detalle todo lo introducido de manera intuitiva en el apartado anterior.

4.1. Especificación de los tamaños de las Herramientas

Aquí especificaremos cómo se ha nombrado a cada parámetro necesario para la definición de cada herramienta.

Primeramente cabe aclarar que para todo lo relacionado con Procesado de imágenes y para las herramientas se ha optado por elegir un sistema de coordenadas típico de gráficos por computador. Esto quiere decir, que el origen del sistema de coordenadas para todas las imágenes es la esquina superior izquierda, siendo el sentido positivo de las X hacia la derecha. Y el sentido positivo de la Y hacia abajo.

Para las herramientas los sistemas de coordenadas se fijan de igual manera, siendo este situado en las esquinas superiores izquierdas de Herramientas y módulos utilizados aquí.

4.1.1. Herramienta 2x2 y Herramienta 2x1

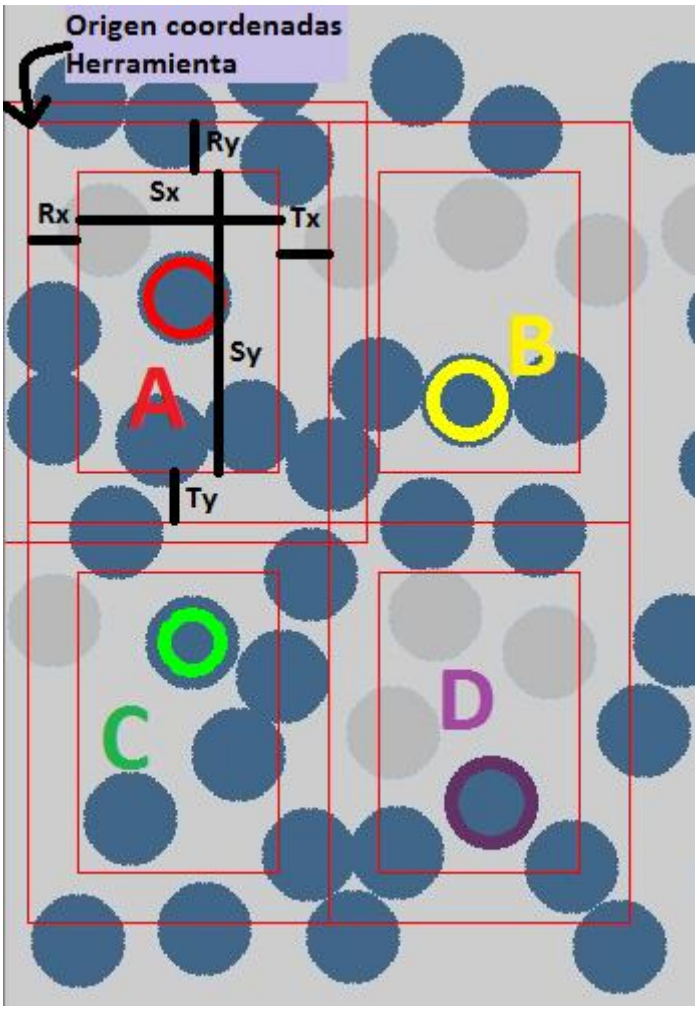

Figura 20: Tamaños herramienta 2x2 y 2x1

En la imagen se puede apreciar cómo se ha denominado a las variables que representan los tamaños de la herramienta. Estos variables pueden tomar cualquier valor razonable. Esto se ha hecho así pensando en que la herramienta podía tomar cualquier tamaño y forma (dentro de los rectángulos para este caso).

Se incluye en esta imagen las herramientas 2x2 y 2x1 rectangulares puesto que los parámetros de las medidas para cada módulo son idénticos.

En la imagen se puede apreciar los 4 módulos correspondientes a la herramienta 2x2. Para la herramienta 2x1 sería igual solo que únicamente con los dos módulos superiores A y B.

Se puede apreciar como las medidas Sx y Sy corresponden al Área de Trabajo del módulo. Cada módulo es denominado con una letra (A, B, C y D). Para acceder a cada parámetro de cada módulo simplemente se hace uso del operador punto (.) A.sx para acceder al parámetro Sx del módulo A. Y sucesivamente para cada módulo.

Los parámetros Rx y Tx corresponden a los márgenes externos izquierdo y derecho correspondientemente de un módulo cualquiera en la Coordenada X. De igual manera es para la coordenada Y, siendo estos Ry y Ty.

Por lo tanto para cada módulo tenemos 6 parámetros que definen su tamaño. Estos son: Rx, Sx, Tx y Ry, Sy, Ty.

Además de estos 6 parámetros tenemos otros dos que definen el offset de cada módulo. Offset[0] para la coordenada X y offset[1] para la coordenada Y Esto es necesario para indicar el punto exacto dónde comienza cada módulo. Cada módulo tiene su origen de coordenadas propio también situado en su esquina superior izquierda.

De este modo, por ejemplo para el módulo de tipo 2, es decir, el módulo B tendremos el siguiente offset.

B.offset[0] = A.rx + A.sx + A.tx; B.offset[1] = 0;

Para la coordenada X tenemos un offset igual al tamaño horizontal del módulo A y para la coordenada Y tendremos un offset de 0 puesto que se encuentra a la misma altura.

4.1.2. Herramienta Circular 2x1Original y Modificada.

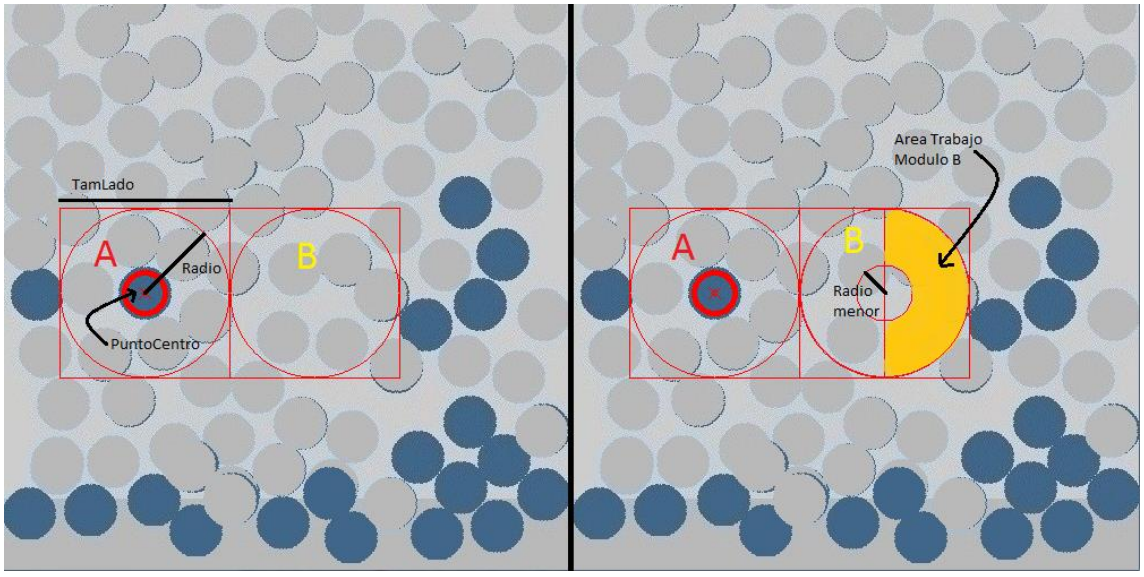

Figura 21: Herramienta 2x1 Circular Original y Modificada

En la imagen izquierda podemos observar la Herramienta circular original y a la derecha la Herramienta circular Modificada para usarse en el caso real de prueba.

La herramienta circular presenta 4 parámetros visibles en la imagen. Estos son: TamLado, Radio, Radio Menor y PuntoCentro.

La herramienta Circular Original tiene de RadioMenor el valor cero. Este parámetro solo es usado por la herramienta modificada. En la original las áreas de trabajo de ambos módulos son circulares.

En la herramienta Circular Modificada, se han añadido una serie de variaciones en el modelo adecuándolo a las limitaciones que la herramienta diseñada en la realidad supone.

En concreto, el módulo izquierdo es fijo, por lo que solo puede agarrarse un objeto con el centro del módulo A.

Para el módulo derecho, las cosas aun cambian más. Tenemos una guía lineal que puede rotar. Pero esta solo rota 180º, los correspondientes a los 180º grados exteriores de la circunferencia. Además cabía la posibilidad de que el TCP del módulo no alcanzase el centro de rotación por lo que tendríamos una pequeña circunferencia en el centro del módulo que nos impediría agarrar objetos en esa zona. De ahí surge el Radio Menor, que define esta pequeña circunferencia central. En definitiva, el área de trabajo del módulo B queda marcada en color naranja en la imagen anterior.

Posteriormente se supo que la ventosa del módulo derecho sí que alcanzaría el centro de rotación de la guía por lo que el RadioMenor en el ejemplo real será finalmente cero, al igual que para la herramienta Original. Pero seguimos manteniendo la rotación de 180º.

4.1.3. Jugando con los parámetros Offset

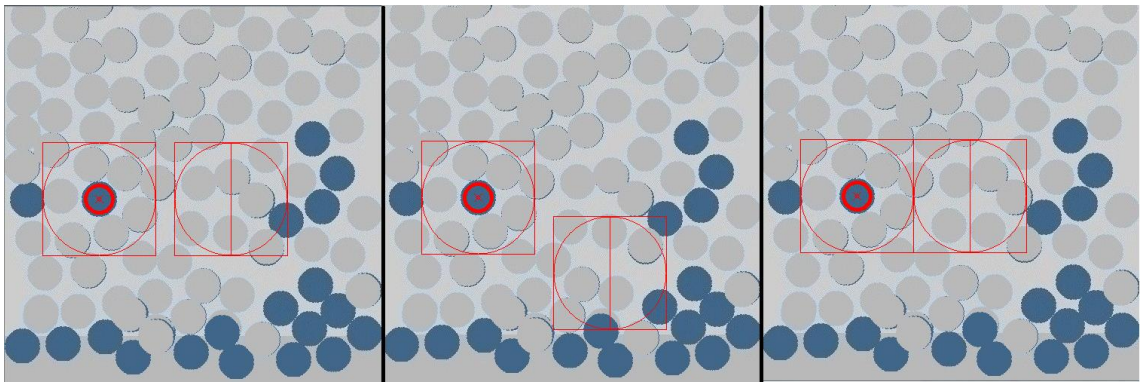

Figura 22: Variación del parámetro Offset

En la imagen podemos ver el resultado de variar los parámetros de Offset para el módulo B. Podemos situar el módulo Derecho básicamente en cualquier sitio. En la imagen Derecha podemos observar cómo sería su situación más normal, un módulo junto a otro.

En la imagen izquierda podemos observar cómo quedarían los módulos si los separamos un poco.

En la imagen central, podemos ver cómo quedarían los módulos si los colocamos en diagonal.

El diseño de la herramienta admite casi cualquier distribución (casi, porque no he probado todas). Para los ejemplos que he probado funciona todo bien, tanto el cálculo de coordenadas desplazadas como de Puntos compatibles.

4.2. Generación de imágenes y objetos [GeneradorPtos.cpp]

En este apartado explicaremos tanto el generador de Imágenes, esto es crear una imagen desde cero. Y también explicaremos la generación de puntos sobre la parte baja de la imagen.

4.2.1. Generación de objetos sobre la parte baja de la imagen

[//ESTRUCTURA][1] Primeramente tenemos una estructura que contiene una serie de parámetros que controlan el comportamiento del generador.

Estos son los que se muestran en el cuadro siguiente.

Por orden de arriba abajo, el primero sirve para controlar el número de pixeles que la imagen avanza tras cada llamada al generador.

```
struct parametrosGenerador{
    int NumPixAvance;
    int NumAvancesPorObjGenerado;
    int RadioObj;
    int Thickness;
    int RadioReal;
};
```

El número de avances que hace la imagen antes de generar un objeto. Es decir, podemos generar un objeto cada dos turnos en los cuales hemos agarrado objetos.

El radio de los objetos a generar en pixeles.

Thickness es la anchura de la línea que forma el círculo. Por ejemplo si tenemos Radio = 10 y Thickness = 25. Tendremos en total un circulo de Radio real = 10 + 25/2 = 22 pixeles aproximadamente. Como 25/2 es mayor que 10. El círculo queda totalmente

[1] Este tipo de notación en forma de clave enlaza la parte explicada aquí junto con otra clave idéntica en el código escrito en C++ en el proyecto. En concreto en el fichero "GeneradorPtos.cpp" se encuentra la clave *//ESTRUCTURA* al principio del fichero, donde se encuentra el trozo de código que hace referencia a lo explicado aquí.
A lo largo de este apartado se hará continuamente referencias de este tipo que enlazan la explicación aquí, con el correspondiente código en C++ en el proyecto.

relleno del color del que pintemos la línea puesto que la línea del círculo es más gruesa que el propio radio del círculo.

4.2.1.1. Movimiento de la cinta

[//AVANCE] Lo primero que hacemos en el proceso de generar un objeto, es simular que avanza la cinta. Esto lo hacemos moviendo hacia arriba en la imagen el valor contenido en cada pixel, uno a uno. En concreto el número de pixeles que movemos cada pixel de la imagen es el valor contenido en NumPixAvance. Los pixeles los movemos hacia la parte alta de la imagen, es decir, su coordenada Y se reduce.

En la parte baja de la imagen nos queda un espacio vacío sin nada. Este es el trozo de cinta transportadora nuevo que ahora se ve. Técnicamente no está vacío, este espacio contiene los píxeles que estaban originalmente en la imagen antes de desplazar. Lo que hacemos es pintarlos de gris.

4.2.1.2. Generación de un objeto

[//COORDENADAS]Para la creación de un nuevo objeto necesitamos generar dos coordenadas, una X y otra Y.

Para la coordenada X generamos un número entre cero y número de pixeles P. El valor de P viene dado por *el ancho de la imagen en pixeles – 2*Radio del objeto*.

A este valor generado le aplicamos un desplazamiento. En concreto sumamos el radio del objeto. Con esto nos aseguramos que el rango de coordenadas X generadas es el siguiente [RadioObjeto , NúmeroPixelesX – RadioObjeto]. Esto nos asegura que ningún objeto va a salirse de la imagen ni va a ser cortado por el borde de esta.

Para la coordenada Y procedemos de forma similar. Solo que el rango de generación son los 50 Pixeles más bajos de la imagen.

Una vez generadas las coordenadas X e Y, debemos comprobar que estas no solapan con otro objeto ya existente. Consideramos que dos objetos pueden estar tocándose, pero nunca uno encima de otro.

Para comprobar esta condición, calculamos la distancia euclidea entre el punto que acabamos de generar y todos aquellos puntos que se encuentren en el rango donde se ha generado el punto. [//NoSOLAPE]

Si solapa con algún punto ya existente, descartamos las dos coordenadas X e Y, y volvemos a intentarlo. Este proceso se repite hasta que consigamos generar un punto válido o se alcance un número máximo de itentos. [//INTENTOS]

4.2.2. Generador de imagen desde cero

En esta parte se explica cómo generamos una imagen nueva con puntos aleatorios. Básicamente hacemos los mismo que en el apartado anterior solo que generamos puntos por todo el espacio de la imagen. Este apartado corresponde con la función "GeneradorVacio" [//GeneradorVacio] en el fichero "GeneradorPtos.cpp".

Se puede comprobar como creamos el rango tanto para X como para Y creando una especie de márgenes alrededor de la imagen de tamaño igual al Radio Real del objeto. De esta forma nos aseguramos que ningún objeto aparezca cortado en la imagen. El procedimiento es el mismo que hemos seguido en el apartado anterior.

A continuación comprobamos que el Objeto generado no solape con ningún otro ya existente comprobando que la distancia entre dicho objeto y todos los ya existentes no sea menor a dos veces el RadioReal del objeto.

Si no solapa con ningún objeto existente perfecto. Dibujamos el objeto, lo introducimos en el vector de objetos e incrementamos el contador de número de objetos.

Si solapa, volvemos a intentarlo hasta conseguir un objeto válido o agotar el número de intentos máximo. Una vez alcanzado el número de intentos máximo, dejamos de generar objetos porque se entiende que la imagen ya está saturada y no hay ningún hueco posible.

4.3. Reconocimiento por visión y extracción de coordenadas

Para poder extraer las coordenadas de los objetos primero es necesario procesar la imagen. Todo el proceso de la imagen se encuentra en el fichero "ImageProcessing.cpp" [//PROCESS]

En concreto primero pasamos la imagen a escala de grises con la función *CvtColor()* de OpenCv. Seguidamente aplicamos un umbral a la imagen para binarizarla. Con esto obtenemos una imagen binaria en blanco y negro. Blanco representa la cinta, y negro representa los objetos.

Tras binarizar la imagen debemos realizar una erosión de los objetos porque en algunos casos estos se encuentran juntos y debemos separarlos. El hecho de que se encuentren juntos ocasiona que ambos objetos sean representados como uno solo.

Por erosión se entiende el proceso por el cual eliminamos un pixel del borde del objeto alrededor de todo su perímetro. Y por dilatación se entiende por el proceso contrario, añadir un pixel adicional a todo el perímetro del objeto, haciéndolo más “gordo”.

Pues bien, en OpenCv esto es al contrario. Por lo tanto hemos aplicado una serie de 10 dilataciones con el objetivo de reducir cada objeto haciéndolo 10 pixeles más “estrecho” por cada lado.

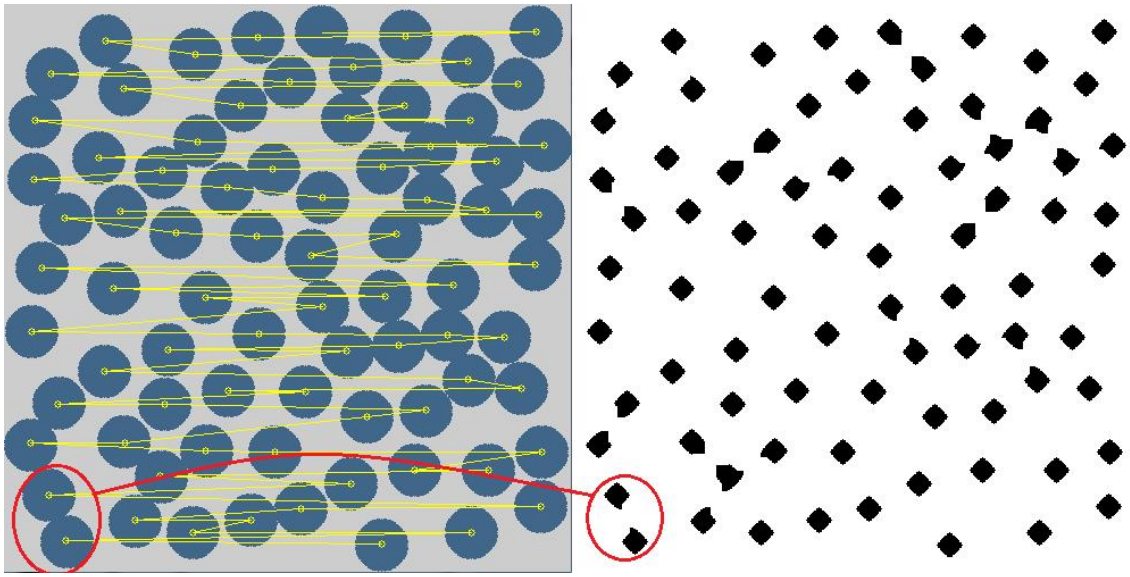

Figura 23: Procesado de imagen.

En la imagen izquierda (quitando los círculos y líneas amarillas) tenemos la imagen de origen recién tomada la foto de la cinta. A partir de esta fotografía aplicamos todo el proceso de tratamiento de imagen descrito anteriormente obteniendo la imagen de la derecha.

Se puede observar cómo hemos enlazado dos objetos de ambas imágenes mediante color rojo. Estos dos objetos se tocan, por lo que si no los separamos en el proceso de tratamiento de imagen hubiesen sido reconocidos como un único objeto. Se puede observar en la imagen derecha como estos dos objetos no son rombos sino que presentan un pequeño pico apuntando al objeto al que estaba unido.

En la imagen derecha se puede apreciar pues todos aquellos objetos que originalmente estaban unidos a otros porque presentan estos picos. Si un objeto tiene forma de rombo perfecto indica que dicho objeto se encontraba separado de los objetos de su alrededor.

Una vez extraidas las coordenadas, es decir, calculados los centroides de todos los objetos, dibujamos sobre la imagen un circulito amarillo por cada centroide que hayamos calculado. Al hacerlo obtenemos la imagen derecha en Figura 23.

Adicionalmente hemos dibujado una línea entre el centroide i y el centroide i+1 para poder ver el orden en el que se encuentran almacenados. Se puede observar que están ordenados por su coordenada Y, es decir, el primer objeto es el que más arriba en la imagen se encuentra.

4.3.1. Extracción de coordenadas

Para calcular el centro de todos los objetos de la imagen recurrimos a dos funciones contenidas en OpenCv: findContours () y moments();

La primera encuentra el contorno de cada objeto. Mientras que moments, devuelve el centro de cada objeto en una estructura especial de OpenCv.

Al fin y al cabo, metemos todos los centroides de los objetos en un vector dinámico de la clase STL <vector> llamado "*centroides*". Este vector dinámico tiene como tipo una estructura de OpenCv llamada Point2f, no es más que una estructura que contiene dos variables tipo float. Una para la coordenada en X y otra para la coordenada en Y.

4.4. Algoritmo para Herramienta 2x2

4.4.1. Función Principal Clasificación(…)

Ahora detallaremos el funcionamiento de la Herramienta con 4 módulos lo primero que hacemos es inicializar todos las estructuras y vectores que vamos a utilizar. Ver en el fichero "ObjectProcessing2x2.cpp" la última función llamada *Clasificación(…)*. Esta función es la principal del algoritmo. A partir de esta se llama a las demás. Aquí podemos ver cómo se inicializa la estructura *objeto2x2 *VObj* cuyo contenido se puede apreciar en el cuadro de arriba. El único miembro de la estructura que no resulta obvio es *errCuadCentro[4]* . Este vector de 4 componentes contiene el error cuadrático en distancia al centro del área de trabajo. Tenemos 4 componentes porque un objeto puede ser de varios tipos de objeto al mismo tiempo como ya se explicó anteriormente.

```
struct objeto2x2{
    float x,y;//coordenadas originales x,y del objeto
    int comA,comB,comC,comD; //compatibilidades
    float xdesp[4],ydesp[4];//coor x,y del objeto desplazado
    int tipoDesplazamiento; //tipo de desplazamiento
    bool fijado; //objeto fijado
    float errCuadCentro[4];
    /*tipo    destino <--- origen
       1        A            A
       2        A            B
       3        A            C
       4        A            D
    */
};
```

Esta función, *Clasificación(),* se encarga de inicializar parámetros y llamar a la función recursiva *fija4puntos()*. La función recursiva nos devolverá *true* en caso de éxito al fijar 4 puntos o *false* en caso contrario.

Si hemos obtenido un *false* como resultado. Repetimos el proceso sobre el mismo objeto líder pero con una diferencia. Ahora en lugar de considerar que lo fijamos sobre el módulo de tipo A, lo fijamos sobre el módulo de tipo B. Es decir, sobre el módulo superior derecho. No tiene sentido seguir intentándolo sobre los módulos C y D porque el objeto líder considerado es el primero en aparecer por la cinta transportadora. Es decir, que es el más avanzado de todos. Todos los demás puntos se encontraran por detrás del líder.

Si tras repetir el proceso considerando el objeto líder como tipo B, fijándolo en dicho módulo, también obtenemos *false* como resultado de la llamada recursiva, nombramos como líder al siguiente objeto en la lista de centroides hasta obtener una respuesta positiva.

4.4.2. Función Recursiva Fija4Puntos(…)

Esta función es una función recursiva cuyo objetivo principal es fijar entre todos los puntos compatibles 4 de ellos para ser agarrados por la herramienta.

Esta función tiene un caso base. Este se alcanza cuando hemos fijado 4 puntos. Para alcanzar el caso base hemos de pasar por 4 niveles.

Nivel cero, fijamos el punto líder inicial a partir del cual fijaremos otros 3. En este nivel fijaremos un segundo punto a partir de este punto líder.

Nivel uno, fijamos a partir de los dos puntos anteriores, otro punto.

Nivel dos, fijamos a partir de los 3 puntos anteriores, el último punto.

Nivel tres, ya tenemos 4 puntos fijados, al alcanzar este nivel hemos alcanzado el caso base. En este momento devolvemos el valor binario *true* para que se propague por la recursión hasta el nivel inicial indicando que el proceso ha concluido correctamente.

En caso de fallar en cualquiera de los casos anteriores, la recursión se corta, devolviendo el valor *false* que se propaga por la recursión hasta el nivel inicial. Esto indica a la función principal *Clasificacíon()* que algo ha fallado.

En cada llamada a esta función recursiva realizamos el proceso de fijado de un punto explicado anteriormente. Calcular Marco Frontera -> Calcular Compatibles -> Calcular de los compatibles el punto óptimo -> Fijar dicho punto.

4.4.2.1. Cálculo del Marco Frontera

Tenemos 3 tipos de Marcos Frontera ya que necesitamos fijar 4 puntos. (Realmente hay implementados 4 tipos de Marco, luego vi que solo se necesitaban 3)

Para calcular el Marco llamamos a la función *CreaMarcoN()* donde N es el tipo de marco a crear. [//CrearMarcos]

Para ello, se calculan las coordenadas máximas y mínimas tanto para X como para Y de aquellos puntos que se consideren a la hora de crear dicho marco. Con esto tenemos 2 puntos extremos a los cuales les aplicamos un desplazamiento igual al tamaño del lado del área de trabajo. Los dos puntos obtenidos son los que conformarán las esquinas superior izquierda y la inferior derecha de dicho Marco Frontera.

Una vez calculado el Marco pasamos a calcular los puntos compatibles a dicho Marco.

4.4.2.2. Cálculo de objetos compatibles

Para el cálculo de objetos compatibles llamamos a la función *Compatibilidades(…)* pasándole un punto únicamente. Esta función se encarga de calcular si dicho punto es compatible con el marco frontera.

Consideramos que el punto en cuestión puede ser de cualquiera de los 4 tipos de puntos. Por lo tanto hacemos el cálculo de compatibilidad 4 veces, una por cada tipo de punto. Como ya hemos dicho anteriormente un punto puede ser de varios tipos a la vez. Por eso mantenemos en la estructura de puntos vectores de 4 dimensiones para coordenadas de desplazamiento y Errores cuadráticos.

Si dicho punto es considerado compatible como algún tipo de punto, el índice de dicho punto es almacenado en un vector de compatibilidades. Tenemos 4 vectores, uno para cada tipo de punto. Estos vectores se llaman “compatibleN” siendo N el tipo de punto (A,B,C,D). El objetivo de estos vectores es mantener 4 listas con los objetos que son compatibles en todo momento. Un punto puede pertenecer a varios vectores de compatibilidad a la vez.

El cálculo de compatibilidad simplemente se trata de comprobar que las coordenadas desplazadas de dicho punto caen dentro del Marco Frontera calculado en el paso anterior.

En este punto tenemos calculados todos los puntos compatibles en 4 vectores cada uno con un tipo de objeto. Ahora mismo podríamos elegir cualquier punto compatible de cualquier tipo y pasar al siguiente nivel de recursión. Al final obtendríamos 4 puntos fijados. Pero, ¿podríamos mejorar alguna característica de estos puntos fijados?

Se pensó en minimizar el desplazamiento que debían moverse las guías lineales de los módulos. Por eso se pensó en calcular para cada punto compatible un factor para definir la "bondad" de cada punto compatible.

4.4.2.3. Cálculo de Error Cuadrático

Este factor de "bondad" es el Error cuadrático en distancia entre objeto líder y las coordenadas desplazadas del punto sobre el que se quiere calcular dicho factor.

Las coordenadas desplazadas de un punto son el desplazamiento aplicado a las coordenadas reales del objeto para colapsarlo sobre el módulo tipo A que es donde se encuentra el Marco Frontera. Esto como ya se explicó, se hace para solamente utilizar un Marco Frontera en lugar de uno para cada tipo.

Este factor de bondad premia a aquellos puntos que se encuentren (en coordenadas desplazadas) más cerca del punto líder. Es decir, favorecemos aquellos puntos que más centrados se encuentren.

El cálculo de este factor se hace en la función *ObjetoCentrado(…)*. En dicha función, aparte de calcular el Error Cuadrático, también calculamos el mínimo error cuadrático de cada tipo, y posteriormente el menor Error Cuadrático general. Esta función deposita en las variables *indiceVectorObj* y *tipoObjMasCentrado* el índice en el vector de objetos y el tipo respectivamente, del objeto que tiene el menor Error Cuadrático de todos los puntos compatibles.

4.4.2.4. Fijar punto y siguiente nivel

Una vez calculado todo esto fijamos dicho punto. Para fijar un punto tenemos diversos parámetros que nos ayudan a fijarlo de forma correcta.

Tiposfijados[4]: Es un vector de 4 elementos tipo *bool* que contiene en cada posición si un tipo ha sido fijado o no. Tiposfijados = {false, true, false, false}; Indica que solamente ha sido fijado un objeto de tipo B.

Selec[4]: Es un vector de 4 elementos tipo *int* que contiene en cada posición el índice del objeto que ha sido fijado. La posición 0 contiene el índice del vector cuyo objeto ha sido fijado como tipo A. Y así sucesivamente hasta la posición 3 que contiene el objeto fijado de tipo D.

Orden[4]: Es un vector de 4 elementos tipo *int* que contiene en cada posición qué tipo de objeto ha sido fijado. Este vector guarda el orden en el que han sido fijados los objetos. Orden[0] contiene el tipo de elemento que ha sido fijado en primer lugar. Mientras que Orden[3] contiene el tipo que ha sido fijado en último lugar.

Estos vectores son necesarios porque los tipos de objetos pueden ser fijados en cualquier orden. Es decir, este orden no viene dado por el tipo de objeto, sino por el Error cuadrático. Será fijado primero un objeto de aquel tipo que presente un menor Error Cuadrático.

Una vez fijado dicho punto, pasamos al siguiente nivel de profundidad. Por lo tanto llamamos a la función Fijar4Puntos(...) con un nivel más de profundidad. [//NuevoNivel]

Una vez alcanzado el caso base, devolvemos *true*. Este valor se propaga por la recursión hasta el nivel inicial indicando que se han fijado 4 puntos de manera correcta.

4.5. Algoritmo para Herramienta 2x1

Como anteriormente se ha comentado. Esta herramienta es una simplificación de la herramienta 2x2. Por lo tanto su algoritmo de control también es similar y más sencillo.

Se podría haber empezado por explicar esta herramienta en primer lugar al ser más sencilla que la 2x2. Pero se decidió explicar las herramientas en el orden en el que se crearon, al igual que sus algoritmos.

Este algoritmo funciona de manera muy similar al anterior, salvo por algunos detalles que cometamos a continuación.

En primer lugar presentamos la estructura sobre la que se guarda la información relevante a un objeto. Se puede observar en el cuadro de la derecha.

```
struct objeto2x1{
    float x,y;//coordenadas originales x,y del objeto
    float xdesp[2], ydesp[2]; //coordenadas desplazadas
    bool fijado; //Ha sido fijado?
    bool compatible[2]; //Compatibilidades con lider
    float errCua[2]; //Error cuadratico en distancia
};
```

Esta estructura es prácticamente idéntica a la de la herramienta 2x2 salvo por algunos detalles.

El algoritmo comienza con la llamada a la función *Selecciona2ptos(...)* del fichero "ObjectoProcessing2x1.cpp". Esta función se encuentra al final del fichero [//selecciona2x1].

El algoritmo comienza con un bucle For que recorre todos los objetos contenidos en el vector. Típicamente solo será necesaria una iteración del bucle. Esto es así porque este bucle solo saltará a la siguiente iteración (siguiente objeto) cuando el resultado de fijar 2 puntos sobre el objeto anterior como líder haya fallado.

4.5.1. Inicialización

En dicho bucle For primeramente inicializamos el vector de estructuras tipo *objeto2x1*. Esta inicialización rellena todas las coordenadas de cada objeto. Calcula todas las coordenadas desplazadas. Inicializa a false los datos de *fijado y compatibilidades*. Inserta un valor Infinito en el error cuadrático de cada objeto.

4.5.2. Cálculo del Marco Frontera

A continuación se calcula el marco Frontera de tipo 1 sobre el objeto líder. [//MF2x1] Contiene dos partes, dependiendo de si tenemos uno de los módulos fijos.

Si tenemos un módulo fijo, el izquierdo normalmente. El marco frontera tipo 1 es exactamente igual al área de trabajo del módulo derecho.

Si ambos módulos son libres, el marco frontera de tipo 1 lo calculamos como se explicó en el apartado anterior. En este caso, se calculan dos puntos extremos (Superior izquierda e Inferior derecha) que conforman los puntos extremos del Marco. Estos puntos los calculamos aplicando un desplazamiento a las coordenadas del objeto líder iguales al lado del área de trabajo.

Adicionalmente calculamos un marco frontera tipo 2 una vez hallamos fijado 2 puntos porque nos será útil a la hora de calcular el punto donde situar la herramienta2x1.

4.5.3. Cálculo de objetos compatibles

El cálculo de los objetos compatibles es idéntico a la forma en la que se calculan para la herramienta 2x2. Es decir, comprobamos que las coordenadas desplazadas del objeto en cuestión caigan dentro del Marco Frontera. Esto lo hacemos en la función *compatibilidades2x1(…)* Los objetos compatibles para esta herramienta, solo pueden ser de dos tipos. Tipo 1, correspondiente al módulo izquierdo. Tipo 2, correspondiente al módulo derecho.

4.5.4. Cálculo de Error Cuadrático

En el caso de que el objeto resulte compatible por algún tipo, se calcula el error cuadrático, que como se ha explicado es la distancia al centro del Marco Frontera. Este error cuadrático se calcula en la función *errCuadratico(…)* cuyas llamadas se encuentran dentro de la función *compatibilidades2x1(…)*.

4.5.5. Fijar punto

Una vez hemos calculado el error Cuadrático, ya tenemos todo listo para fijar el punto que nos falta. Esto lo hacemos en la función *fijaPunto(…)*. En esta función simplemente calculamos de entre los objetos compatibles del tipo que deseemos fijar, aquel que

presente el menor error cuadrático. Una vez fijado este punto, junto con el punto líder, ya tenemos los dos puntos necesarios a fijar. Por eso no ha sido necesario crear una función recursiva en este caso.

Una vez fijados los dos puntos calculamos el marco Frontera tipo2 que nos ayuda a calcular la posición donde situaremos la herramienta.

Para calcular el punto donde situaremos la herramienta calculamos el punto central del Marco Frontera de tipo 2. Ha de ser el de tipo 2 porque este es el que contiene a los dos puntos fijados. El de tipo 1 solamente contiene al líder.

Una vez calculado el punto central del Marco Frontera, aplicamos al punto un desplazamiento. En concreto restamos Rx y Sx/2 a la coordenada X y su correspondiente para la Y al punto central calculado anteriormente. Esto es así porque el origen de coordenadas de la herramienta se encuentra en la esquina superior izquierda de esta. (Ver apartado 4.1 Especificación de tamaños).

4.6. Algoritmo para 2x1Circular Original y Modificada

En general el proceso para fijar puntos para la herramienta 2x1 Circular es el mismo. Únicamente cambia un poco algunos detalles de implementación.

A continuación explicaremos el funcionamiento de la Herramienta 2x1Circular. Al final explicaremos las diferencias que tiene la Herramienta Modificada con la original.

La estructura que forma el vector de los objetos es idéntica a la usada para la herramienta 2x1 rectangular.

Se muestra en el cuadro siguiente.

```
struct objetoCirc2x1{
    float x,y;//coordenadas originales x,y
    float xdesp[2], ydesp[2];
    bool fijado;
    bool compatible[2];
    float errCua[2];
};
```

4.6.1. Inicialización

En dicho bucle For primeramente inicializamos el vector de estructuras tipo *objetoCirc2x1*. Esta inicialización rellena todas las coordenadas de cada objeto. Calcula todas las coordenadas desplazadas. Inicializa a false los datos de *fijado y compatibilidades*. Inserta un valor Infinito en el error cuadrático de cada objeto.

4.6.2. Cálculo Marco Frontera circular

Para el caso de herramientas circulares como este, el marco frontera de tipo 1 es una circunferencia. Para tipos mayores a 1 la forma del marco frontera no sería una circunferencia porque consideramos más de un punto en su interior. Para este caso solo es necesario considerar un marco frontera, y este es el de tipo 1.

Tenemos 2 posibilidades. Que la herramienta circular tenga el módulo izquierdo fijo o libre.

En el caso de que sea fijo (como en el caso real tendremos) el Marco Frontera Tipo 1 es una circunferencia de igual radio que la circunferencia que describe el área de trabajo del módulo derecho. Esta circunferencia se haya centrada sobre el objeto líder en el módulo izquierdo.

En el caso contrario de que tengamos el módulo izquierdo libre. El Marco Frontera Tipo 1 es una circunferencia de radio el doble al área de trabajo de los módulos de la Herramienta circular.

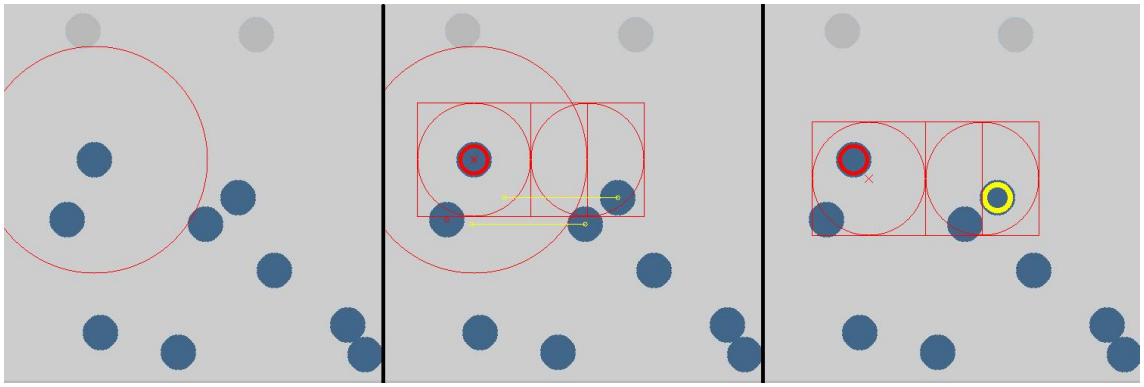

Figura 24: Proceso Fijado Herramienta 2x1 Circular Ejemplo 1

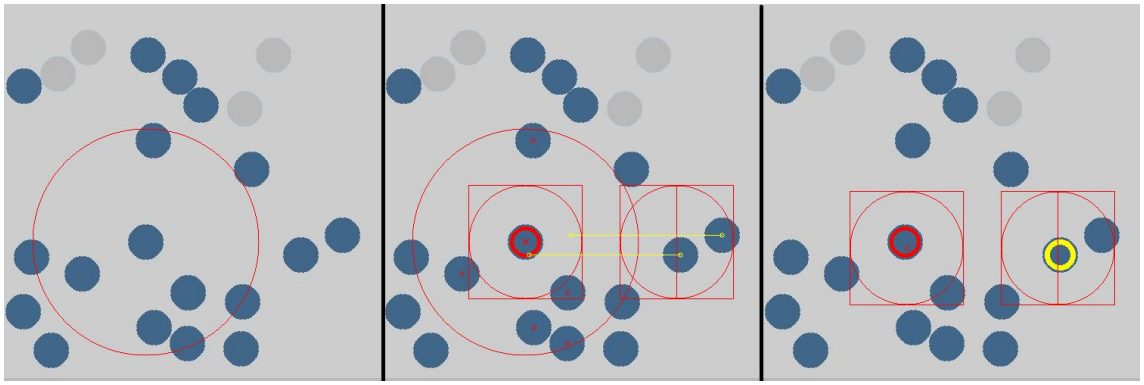

Figura 25: Proceso Fijado Herramienta 2x1 Circular Ejemplo 2

En las dos imágenes anteriores podemos ver dos ejemplos del proceso de fijado de dos puntos para una Herramienta 2x1 circular con ambos módulos libres. Se puede apreciar

en las imágenes izquierdas como el tamaño del Marco Frontera es el doble que el del área de Trabajo. Al ser módulos libres, podemos apreciar en las imágenes derechas como la herramienta se posiciona en un punto intermedio para los dos objetos. En el caso de módulo izquierdo fijo, era obligatorio agarrar el objeto izquierdo en el centro del módulo. Aquí se observa que ambos se encuentran desplazados del centro de sus correspondientes módulos.

4.6.3. Cálculo de Objetos compatibles

El cálculo de objetos compatibles lo hacemos uno a uno. Para ello calculamos la distancia entre el objeto líder que se encuentra en el centro del Marco Frontera calculado y entre el punto que consideremos.

Para la Herramienta Original consideraremos que es compatibles si dicha distancia es menor que el Radio del Marco Frontera.

Para la Herramienta Modificada añadimos un par de condiciones para conseguir el efecto de los 180º grados exteriores, y el círculo interior explicado en el apartado 4.1 en la parte de la Herramienta circular 2x1.

Para conseguir el efecto de considerar la semicircunferencia que abarca los 180º exteriores, añadimos la condición de que la coordenada en X del objeto líder sea menor que la coordenada desplazada en X del objeto que estemos considerando. Si es mayor significa que se encuentra en la mitad derecha. Consiguiendo así los 180º exteriores del área de trabajo del módulo derecho.

Para conseguir la circunferencia interior, usamos el parámetro RadioMenor. De manera que si la distancia entre el líder y el punto a considerar es mayor que el valor del RadioMenor, el objeto considerado es compatible.

4.6.4. Cálculo de Error Cuadrático

El cálculo del error cuadrático se hace igual que para la herramienta 2x1 rectangular, dentro de la función *CompatiblidadesCirc2x1(…)* que es la que calcula los objetos compatibles. Se calcula de igual manera que para los casos anteriores, se calcula la distancia euclidea entre el objeto líder, y el punto que se esté considerando.

4.6.5. Fijar punto

Para calcular el segundo punto a fijar hacemos la llamada a la función *fijaPuntoCirc(…)* la cual recorre todo el vector de objetos y calcula aquel objeto que tenga menor error cuadrático de entre todos los compatibles. Una vez calculado insertamos el índice que ocupa en el vector de objetos en la variable *tiposfijados*.

El cálculo del punto central donde colocar la herramienta para el caso de que tengamos el módulo izquierdo libre, lo hacemos de manera diferente al caso anterior. Ahora, calculamos el punto intermedio entre los dos objetos fijados. A este punto le aplicamos un desplazamiento para calcular el punto donde se situaría el origen de coordenadas de la Herramienta (superior izquierda) que es desde donde empezamos a dibujar. Este desplazamiento se calcula simplemente restando a las coordenadas del objeto líder el valor del Radio del área de trabajo.

Para el caso de tener el módulo izquierdo fijo, el punto donde comenzar a dibujar la herramienta simplemente es aquel cuyo resultado se obtiene de restar a las coordenadas del objeto líder el valor del Radio del área de trabajo.

4.7. Main(). Función principal

La parte del simulador trabaja sobre consola de texto, por la cual se va imprimiendo información útil sobre lo que va ocurriendo, y trazas de ejecución.

Paralelamente a la consola de texto, aparecen imágenes en ventanas separadas mostrando el progreso del proceso de reconocimiento de imágenes y fijado de puntos. Ante una imagen el proceso se detiene, para continuar ejecutando el simulador es necesario sobre la imagen pulsar la tecla 'Escape'. Al avanzar, aparecerán sobre la consola nueva información y de nuevo otra imagen.

A continuación explicamos detalladamente cómo funciona esta parte del código:

Primeramente tenemos un menú sobre la consola. En el cual nos pregunta cuál de los 3 tipos de Herramienta queremos utilizar. A continuación nos da la opción de fijar el primer punto en aparecer. O por el contrario fijar un punto que nosotros queramos indicándole las coordenadas de este mirando la información que aparece en la consola. Por último nos pregunta si queremos leer una imagen generada previamente o por el contrario generar una nueva en ese mismo instante. Para ello necesitaremos indicarle el número de pixeles en X y en Y que deseemos en la imagen, y el número de puntos que queremos que se generen aleatoriamente. En ocasiones si especificamos un número muy elevado de puntos puede que no lleguen a generarse todos por falta de espacio en la imagen.

```
int tipoHerramienta;
cout << "\n----Herramientas----\n"
    << "1:  Herramienta cuadrada 2x1\n"
    << "2:  Herramienta cuadrada 2x2\n"
    << "3:  Herramienta circular 2x1\nInserta Codigo:";
cin >> tipoHerramienta;

int forzarPrimerPunto;
cout << "\n----Elección primer punto:----\n"
    << "1: Automático, primer punto en aparecer.\n"
    << "2: Forzar coordenadas punto izquierda.\n";
cin >> forzarPrimerPunto;
float coorX, coorY;

int generarImagen;
cout << "\n---Imagen a usar:---\n"
    << "1: Leer imagen de fichero. \n"
    << "2: Generar imagen aleatoria. \n";
cin >> generarImagen;
int tamX, tamY, numobjaGen;
```

Figura26: Menú

A continuación del menú viene la parte del código en la que definimos todos los parámetros de cada Herramienta. Por lo tanto, si se quiere modificar alguno es en esta parte del código donde se tiene que hacer.

Para cada herramienta, la zona de código donde empieza la definición viene marcada de esta forma:

```
///////////////
//Herr2x2
///////////////
```

Para los tres tipos de herramientas definimos sus parámetros de igual forma. Primero los tamaños, a continuación el offset de desplazamiento de módulo.

El final de la zona de definición de parámetros de las herramientas viene marcada con otro comentario especial, del mismo tipo que el anterior en el que pone *fin de definición de herramientas*.

Después de este comentario realizamos las tareas de reconocimiento de imágenes. Para ello llamamos a la función *imágenes(…)*. La cual hace la llamada a la función *process(…)* cuyo funcionamiento está explicado en el punto 4.3 Reconocimiento por visión. Esta función *imágenes(…)* también muestra el resultado del reconocimiento de los objetos, dibujando sobre una imagen circulitos sobre los centros de cada objeto reconocido para así verificar que se ha reconocido correctamente todos los objetos.

De vuelta a la función Main(), a continuación de la función *imágenes()* llamamos a otra función llamada *invierteCentroides(…)* cuyo nombre indica el trabajo que realiza. Esta

es invertir todas las posiciones del vector de centroides. Esto lo realizamos porque OpenCv nos ha reconocido los centroides de abajo a arriba. Es decir, el primer objeto en el vector de centroides es el que se encuentra más abajo. Como nos interesa que sea al contrario llamamos a esta función.

A continuación llamamos a una función llamada EscribeFichero(…) la cual mete en un fichero de texto las coordenadas del vector de centroides. Esto es útil para comprobar que la librería dll tiene el mismo comportamiento que el simulador. Es decir, ante dos vectores de coordenadas igual, comprobar que obtenemos los mismos resultados.

Después de todo esto nos encontramos un *switch(…)*. En esta parte tenemos un trozo de código para cada herramienta. Dependiendo de la herramienta que hayamos elegido en el menú del principio, ejecutaremos una parte del código u otra.

En este switch realizamos la llamada a la función que hace todo el cálculo de cada herramienta. Contienen los algoritmos explicados anteriormente para todas las herramientas creadas. Para la Herramienta 2x2 tenemos la función Clasificacion(…). Para la Herramienta 2x1 tenemos la función Selecciona2ptos(…) Y para la Herramienta circular tenemos selecciona2ptosCirc(…).

Después de hacer la llamada a la función correspondiente, dibujamos el resultado en una imagen con los objetos fijados y la herramienta correctamente posicionada.

A continuación de dibujar, hacemos la llamada al Generador de Puntos aleatorio para que genere un objeto sobre la parte baja de la imagen.

Por último, después del switch preguntamos por consola si queremos continuar y hacer otra iteración del bucle, o por el contrario queremos finalizar el programa.

5. Ejemplo Real

Para hacer una prueba real se ha utilizando el prototipo diseñado por mi compañero de máster como trabajo de tesis. Mi aplicación como hemos visto se ha adaptado al diseño de este prototipo. Como resultado se ha obtenido la Herramienta 2x1 Circular Modificada. (Ver en Apartado 4.1, imagen 22). Esta es la que se ha usado para la prueba.

Para la prueba se ha usado el Robot ABB flexpicker, al cual se le ha acoplado la herramienta construida por mi compañero.

En concreto mi aplicación debe ejecutarse sobre el software de ABB llamado PickMaster. El cual ya realiza toda la parte de visión de los objetos. Por lo que el desarrollo que se ha hecho en este proyecto sobre visión queda solo para la parte del simulador.

PickMaster ejecuta un programa escrito en C#. Por este motivo se ha pensado que lo más conveniente ha sido crear una librería en C++ de enlaces dinámicos DLL. Por lo tanto para la prueba mi proyecto se ha encapsulado en una librería .dll en la cual se ha eliminado toda la parte de visión e imágenes. Es decir, la librería creada, trabaja con un vector de coordenadas. El cual contiene evidentemente todas las coordenadas de los objetos que se hallen sobre la cinta. El resultado de la llamada a la librería devuelve las coordenadas de aquellos dos objetos que se deben agarrar con la herramienta.

Esta DLL creada sobre C++ debe comunicarse correctamente con un programa escrito en C#, el que utiliza PickMaster. Es decir, que el proyecto de programación sobre C# que utiliza PickMaster debe importar mi librería DLL escrita en C++.

Una vez conseguido esto, este proyecto escrito en C# al compilarse da como resultado otra DLL (la cual incluye la mía). Esta DLL sobre C# creada es la que importa el programa PickMaster para hacer la tarea de empaquetado de objetos.

Uno de los problemas que se ha encontrado ha sido la necesidad de comunicar el proyecto en C# con mi .dll en C++ mediante variables simples. Puesto que las variables no son tratadas de igual forma por ambos lenguajes.

Por ejemplo, no podríamos crear una clase en C# y otra idéntica en C++. Declarar una instancia de la clase en C#, pasar la referencia a través de la API de la dll y tratar de que en C++ podamos usar los métodos de la clase que nos acaban de pasar desde C#.

Por este motivo las variables que utilizamos para pasar datos a través de la API de la dll son variables simples que son reconocidos por igual en ambos lenguajes como varibles tipo *int, float, double, y vectores sobre estas variables.* Pero nada de estructuras ni clases.

5.1. Creación de librería de enlaces dinámicos DLL externa, para ser utilizada en C# en PickMaster.

Para la creación de la librería .dll ha sido necesario la creación de otro proyecto C++, el cual creara la librería. Nos referimos al proyecto contenido en la carpeta "MultipickParaC#" allí se halla todo el proyecto listo para ser compilado mediante Visual Studio 2008.

Dentro de dicho proyecto se halla el fichero "API.cpp" en el cual se encuentran las funciones que serán visibles por el proyecto C# que importará esta librería.

En concreto hay 3 funciones, una para cada herramienta. Herramienta2x2, Herramienta2x1 y Herramienta2x1Circular.

Aunque en la prueba solamente utilizaremos la llamada a la función Herramienta2x1Circular.

Cada una de estas funciones se declara como *extern "C"* para que sean reconocidas como código escrito en C por el compilador. También se añade el modificador *__declspec(dllexport)* el cual habilita a la función a la que acompaña para que sea exportada por la DLL para que pueda ser utilizada por otras aplicaciones. El proyecto escrito en C# en este caso.

5.1.1. Parámetros de la cabecera de las funciones contenidas en la API de la DLL

Cada una de estas funciones tiene un aspecto similar a este:

```
extern "C" __declspec(dllexport) void Procesa2x1Circ(int *coordX, int *coordY, int numObj, float *tamA4, float *tamB4, bool *fixed2, int *despX2, int *despY2, int anchoimg, int altoimg, int lider, int *seleccionados2)
```

coordX y coordY: son dos vectores de enteros. Mediante estos vectores pasamos las coordenadas de los objetos reconocidos por PickMaster. Es necesario pasar las coordenadas X e Y en vectores separados por el motivo de utilizar variables los más sencillas posibles.

numObj: Esta variable tipo int contiene en número de objetos que han sido reconocidos.

tamA4 y tamB4: Son dos vectores que contienen los tamaños de cada módulo A y B correspondientemente. Estos son por orden: Radio, RadioMenor, CentroX y CentroY.

fixed2: Vector booleano de 2 componentes. Contiene true o false dependiendo de si el módulo A o B son fijos o libres correspondientemente.

Para la herramienta tenemos el módulo izquierdo que lo declaramos como fijo, y el derecho como móvil. Por lo tanto el vector fixed2 queda así: {true, false};

despX2 y despY2: Son dos vectores que contienen las coordenadas en X y en Y del offset donde se sitúa el origen de coordenadas de cada uno de los módulos. Como tenemos 2 módulos, necesitamos vectores de tamaño 2. Primera componente del vector para el módulo A (izq) y segunda componente para el módulo B (der). Normalmente hubiese sido un vector de tamaño 2 con una estructura que incluye dentro las dos coordenadas.

Anchoimg y altoimg: Valores enteros que contiene el número de pixeles del ancho y algo de la imagen.

Líder: Contiene el valor de aquel objeto que va a ser el líder. Típicamente será valor 0, de esta forma el primer objeto a considerar es el más avanzado. Otros valores de líder son interesantes cuando estamos testeando la aplicación y queremos empezar considerando un objeto que se encuentre por el medio de la imagen. En ese caso daríamos a la variable líder un valor aproximadamente a la mitad del total de objetos reconocidos.

Seleccionados2: Es un vector de enteros de 2 posiciones, a través del cual se devuelven los índices de aquellos dos objetos que han sido seleccionados por la aplicación para ser agarrados. Estos índices corresponden con el vector de objetos.

5.1.2. Implementación de la función

En cada llamada a la función (para el caso real llamamos solo a *Procesa2x1Circ(...)*) hacemos lo siguiente:

-Declaramos una instancia de la Herramienta adecuada llamando a su constructor correspondiente: `HerrCirc2x1 H(tamA4,tamB4,fixed2);` Pasándole los parámetros Tamaños de la ambos módulos, y la especificación de si son libres o fijos.

-Llamamos al método incluido dentro de la clase para definir los offsets de cada módulo: `H.definecoordCirc(desplazamiento);`

-Rellenamos el vector "centroides" que contiene las coordenadas de los objetos mediante los 2 vectores de coordenadas.

-Finalmente llamamos a la función que se encarga de realizar todos los cálculos: `Clasificiacion2x1Circ(&H,&centroides,selec,anchoimg,altoimg,lider);`

Al finalizar la llamada a dicha función tendremos en el vector *selec[2]* los dos índices de los objetos seleccionados. Ahora simplemente los pasamos al vector *seleccionados[2]* que es el que se encarga de transmitirlos a quién llamó a esta función de la DLL. Es decir, a través de este vector los índices llegan hasta el proyecto C# que invocó esta DLL.

5.2. Testeo de la librería de enlaces dinámicos

Para comprobar que el funcionamiento es el correcto se ha creado un nuevo proyecto en C# el cual importa la librería DLL en C++ que hemos creado para usarla.

Este proyecto C# se encuentra en la carpeta "DLLprueba". En dicho directorio se encuentra alojado todo el proyecto listo para compilarlo y ejecutarlo sobre Visual Studio 2008.

Para comprobar que funciona exactamente igual y obtenemos los mismos resultados que en el simulador se ha hecho lo siguiente.

En el proyecto C++ correspondiente al simulador (este proyecto se encuentra alojado en la carpeta "PFC") cada vez que se extraen las coordenadas de los objetos de la imagen, abrimos un fichero de texto y escribimos en él las coordenadas de todos los objetos.

En el proyecto de C# abrimos este fichero de texto para leer las coordenadas y reconstruir el vector de coordenadas. Una vez reconstruido, hacemos la llamada a la dll y obtenemos los resultados, evidentemente utilizamos los mismos parámetros para definir los tamaños de la herramienta para que sean iguales en ambos proyectos. Cabe decir que para todos los casos probados, se han obtenido los mismo resultados en ambos proyectos (simulador en c++ y prueba en c#)

5.3. Uso sobre PickMaster

Para poder ejecutar programas sobre PickMaster, este tiene la posibilidad de importar una dll de un proyecto en C#.

Para crear la dll sobre C# disponemos de un proyecto ya preparado previamente llamado “Hook”. En este proyecto C# es en el que importamos nuestra librería dll de C++ creada anteriormente.

5.4. Detalles de implementación sobre C#

Sobre este proyecto en C# creamos todas las variables necesarias añadiéndoles sus correspondientes valores.

5.4.1. Sistemas de coordenadas

El primer problema surge rápido. Los ejes de coordenadas y el origen de este que hemos considerado desde el principio no tienen nada que ver con el que usa PickMaster:

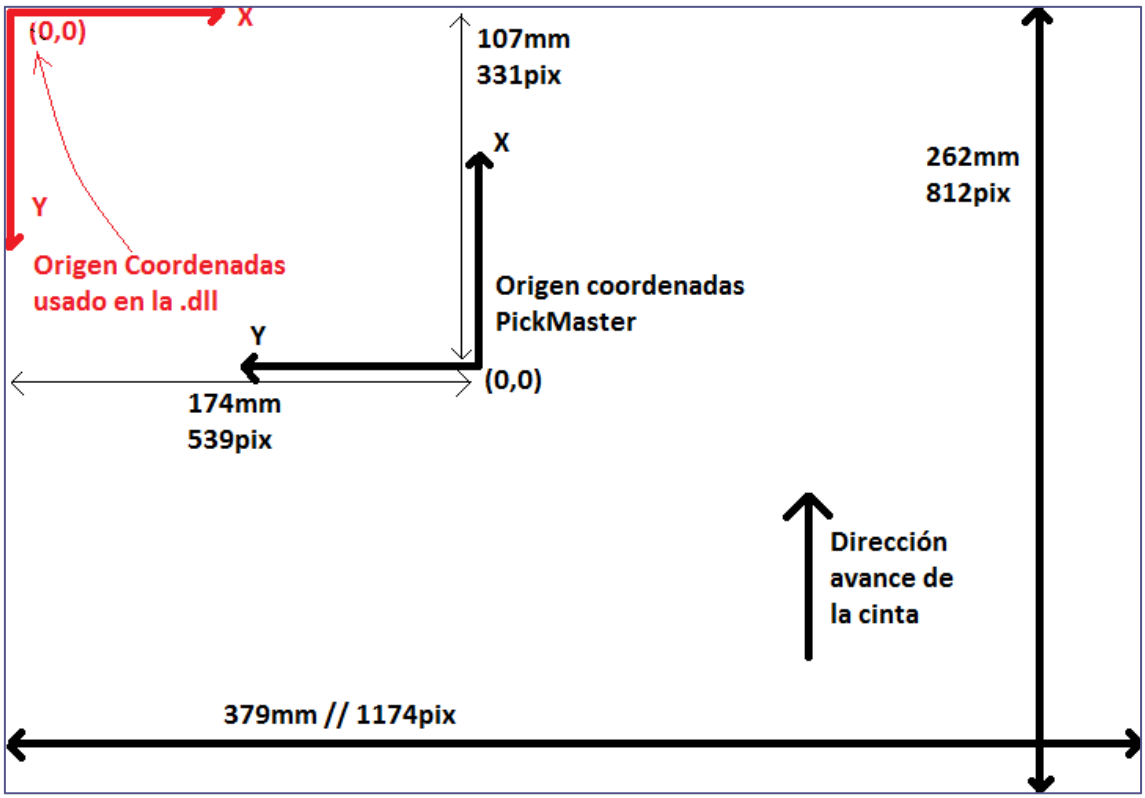

Figura 27: Tamaños imagen en PickMaster

Se puede observar en la imagen que PickMaster trabaja con imágenes de 389x262mm de tamaño. En el ordenador esta imagen ocupa 1174x812pix.

Hacemos el cálculo siguiente: 1174pix / 379mm = 3,097 pix/mm. Obtenemos que por cada milímetro de la imagen tenemos algo más de 3 píxeles.

Podemos observar en negro los ejes de coordenadas y su origen que usa PickMaster. Mientras que en rojo podemos observar el origen de coordenadas usado por la dll, que es el típico usado en gráficos por computador.

El problema es el siguiente: Las coordenadas de los objetos vienen dadas en el sistema de PickMaster, mientras que la dll necesita las coordenadas en base a su propio sistema de coordenadas. Por lo tanto hacemos un cambio de sistema de coordenadas.

Tenemos las X y las Y cambiadas, por lo que habrá que intercambiarlas. Como tanto la X como la Y tienen el sentido también cambiado, habrá que cambiar de signo ambas coordenadas. Por último aplicamos un desplazamiento, que no es más que la distancia entre un sistema y otro, pero también intercambiando la coordenada X con la Y.

Una vez hecho esto hemos pasado las coordenadas de PickMaster, a coordenadas aptas para usar por nuestra librería.

5.4.2. Llamada a la librería

Una vez obtenidos los dos vectores de coordenadas en el sistema de coordenadas de nuestra librería pasamos a realizar la llamada a esta pasándole todos los parámetros que hemos definido anteriormente.

Como respuesta, ésta llamada a la librería nos devuelve en el vector *seleccionados2[2]* las coordenadas de los dos objetos que ha fijado. En caso contrario este vector contiene -1.

5.4.3. Procesado de las coordenadas

Una vez tenemos los dos objetos fijados y sus coordenadas. Necesitamos saber el ángulo y la distancia a la que debemos posicionar la ventosa del módulo móvil.

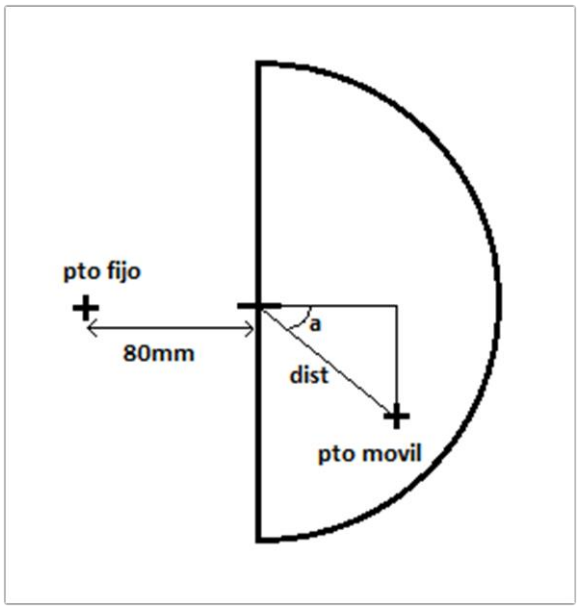

Figura28: Ángulo y distancia al punto móvil.

En la imagen podemos observar dónde consideramos el Ángulo alfa y la distancia al punto móvil. El ángulo 0º corresponde con la horizontal. En el caso del ejemplo de la imagen el ángulo alfa tendría unos 45º positivos aproximadamente. Es decir hacia abajo tenemos ángulos positivos y hacia arriba tenemos ángulos negativos.

El cálculo del ángulo y la distancia lo hacemos de la siguiente manera:

Primero calculamos las coordenadas del punto central del módulo móvil al cual denominaremos punto Central, este se encuentra a 80mm a la derecha del punto fijo. Por lo tanto podemos saber fácilmente sus coordenadas.

Segundo, calculamos las coordenadas del punto móvil respecto al punto central antes calculado. Esto lo hacemos restando las coordenadas del punto móvil menos las coordenadas del punto central. Con esto obtenemos la distancia en componente X e Y entre el punto Central y el punto móvil.

Tercero, el ángulo lo calculamos utilizando el Arcotangente de la componente Y del punto móvil entre la componente X del punto móvil. Este valor lo convertimos a grados, puesto que nos viene calculado en radianes.

Cuarto, para el cálculo de la distancia simplemente aplicamos el teorema de Pitágoras sobre los catetos del triángulo, estos son la componente X e Y del punto móvil. Este valor lo convertimos a milímetros mediante el factor de conversión puesto que nos viene dado en píxeles.

Una vez tenemos las coordenadas de los dos puntos y el ángulo y distancia correspondientes, guardamos los datos en vectores dinámicos, en c# se denominan List<Tvar>.

5.4.4. Sincronización con el Robot

Una vez tenemos los datos guardados, hacemos la sincronización con el Robot. Ello se hace mediante un fichero de texto. El Robot cuando calcula las coordenadas del siguiente objeto que va a agarrar, escribe en un fichero las coordenadas X e Y. Este fichero se encuentra en su propio sistema de ficheros.

Esto supuso un gran problema, puesto que PickMaster se ejecuta sobre un ordenador externo al robot sobre Windows 7, mientras que el robot tiene su propio ordenador con su sistema de ficheros. No se consiguió ni que el robot escribiera el fichero en Windows para poder leerlo. Ni acceder al fichero de manera directa desde Windows al sistema de ficheros del propio robot.

Por suerte se me ocurrió acceder al robot utilizando el protocolo FTP sobre la IP del robot que aparecía en PickMaster. Sorprendentemente el robot tenía el puerto de FTP abierto sin contraseña ni nada. Únicamente sabiendo su IP cualquiera podía acceder al robot mediante FTP.

Mediante un get sobre el fichero conseguíamos descargarnos el fichero en cuestión midante FTP a nuestro Windows 7.

A continuación lo leemos, si las coordenadas han cambiado respecto a la lectura anterior del fichero esto significa que el robot acababa de empezar a moverse hacia un objeto.

Inmediatamente buscamos las coordenadas leídas en los vectores dinámicos que contienen toda la información. Una vez encontrado el índice de los vectores que contienen la información la pasamos por Puerto serie.

5.4.5. Envío de datos por Puerto Serie

Abrimos una conexión por puerto serie acordando los parámetros con mi compañero que se encarga de diseñar la herramienta, para configurar el puerto Serie.

A continuación enviamos los datos, estos deben ser enviados en forma de Bytes. Por suerte como la distancia es un valor positivo entre 0 y 100 mm, y el ángulo es un valor entre 0 y 180º solo necesitamos un byte para cada dato. No es necesaria la partición de ningún dato puesto que estos no superan el valor numérico de 256.

El valor del ángulo realmente es un valor entre -90 y 90º, como no podemos en principio pasar directamente valores negativos como Byte, simplemente sumamos 90 al valor de los grados para tener un rango entre 0 y 180.

Una vez enviados los datos cerramos el puerto serie. Y borramos la entrada de los vectores dinámicos que contenían la información que acabamos de enviar. De esta manera mantenemos en los vectores dinámicos aquellos pares de objetos que son factibles agarrar pero que el robot aún no ha agarrado.

Mi compañero, nada más recibir los datos por Puerto Serie, actúa sobre la herramienta para posicionarla en la posición correcta para agarrar ambos objetos.

6. Conclusiones y trabajos futuros

6.1. Conclusiones

En definitiva, si miramos el trabajo realizado de manera general y no específica para lo que ha sido diseñado, hemos obtenido una aplicación (y una librería dll) que ante coordenadas en un plano X e Y es capaz de emparejar objetos de 2 en 2 o de 4 en 4 especificando áreas rectangulares o circulares bajo las cuales es factible el emparejamiento.

La aplicación más directa al problema resuelto es la recogida de objetos desordenados que circulan por una cinta transportadora. Podría también existir alguna otra aplicación interesante para lo desarrollado aquí.

Este proyecto final de carrera ha tenido desde el principio como objetivo crear un primer prototipo que resolviera el concepto de Multipick. El objetivo global de la idea de Multipick es llegar a implantarlo en la industria.

6.2. Trabajos futuros.

Para el futuro queda diseñar de una interfaz gráfica para el simulador.

Ampliar la aplicación para permitir herramientas modulares. Es decir, la posibilidad de construir una herramienta compuesta de varias herramientas (2x1, 2x2, 2x1Circular) ensamblándolas para poder construir una herramienta del tamaño que necesitemos.

7. Bibliografía

Bjarne Stroustrup. The C++ programming language. Reading Massachusetts etc. Addison-Wesley.

Christoph Wille. C#. Madrid etc. Prentice Hall

Donis Marshall. Programming Microsoft Visual C# 2005 The language. Redmond, Wash. Microsoft Press

Abb PickMaster 3.3 Application Manual

-Crear Biblioteca de vínculos dinámicos .dll para c#

http://msdn.microsoft.com/es-es/library/ms235636.aspx

http://social.msdn.microsoft.com/forums/vstudio/en-US/6215d368-ec60-4712-850d-746c0a078b85/trying-to-call-c-dll-in-c

http://stackoverflow.com/questions/8366590/how-to-create-dll-in-c-for-using-in-c-sharp

http://msdn.microsoft.com/en-us/library/aa288468(v=vs.71).aspx

-Puerto Serie

http://msdn.microsoft.com/es-es/library/system.io.ports.serialport(v=vs.110).aspx

-Lectura y escritura de ficheros no bloqueante

http://balajiramesh.wordpress.com/2008/07/16/using-streamreader-without-locking-the-file-in-c/

-Protocolo FTP

http://msdn.microsoft.com/en-us/library/ms229715(v=vs.110).aspx

http://www.codeproject.com/Tips/443588/Simple-Csharp-FTP-Class

http://www.cintastransportadoras.eu/sites/507535dcaca79e6b1600000e/theme/images/t-banda03-g.jpg