Universidad Nacional Autónoma de México

FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA
DEPARTAMENTO DE INGENIERÍA DE CONTROL Y ROBÓTICA

SISTEMA DE VISIÓN PARA EL CONTROL DE ROBOTS MANIPULADORES

T E S I S

QUE PARA OBTENER EL GRADO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

LAURA GARCÍA LUCIANO

Y EL DE

INGENIERO ELÉCTRICO ELECTRÓNICO

CARLOS ALEJANDRO AGUILAR GUEVARA

TUTOR:

Dr. MARCO ANTONIO ARTEAGA PÉREZ

CIUDAD UNIVERSITARIA, MÉXICO D.F. OCTUBRE 2013

Jurado asignado

Presidente: Dr. Heriberto de Jesús Aguilar Juárez

Secretario: Ing. Román Victoriano Osorio Comparán

Vocal: Dr. Marco Antonio Arteaga Pérez

1$^{er.}$ Suplente: Ing. Vicente Flores Olvera

2$^{do.}$ Suplente: Dr. Paul Rolando Maya Ortíz

La tesis se realizó en el Laboratorio de Robótica de la División de Estudios de Posgrado de la Facultad de Ingeniería de la UNAM.
México, D.F., México

TUTOR DE TESIS:
Dr. Marco Antonio Arteaga Pérez

FIRMA

Agradecimientos

Agradecemos a CONACYT por el apoyo para este proyecto que está basado en una investigación referenciada con el No.58112.

Le agradecemos ampliamente a nuestro tutor el Dr. Marco Antonio Arteaga Pérez por su apoyo, paciencia y respaldo total en este proyecto de tesis.

Agradezco también a los sinodales por orientarnos en las correcciones que finalmente hicieron una mejora a nuestro trabajo.

...A mis padres Teresa y Benito por su incesante dedicación y lucha por que yo sea mejor persona en todos los aspectos.

...A mi hermana Mónica por estar siempre a mi lado y permitirme ser su amiga.

...A Gabriel por ser mi amigo y angel incondicional.

...A Juvenal Villanueva por su apoyo y amor en todo momento.

...A Maximiliano Bueno por guiarnos y por el apoyo sin precedentes durante y despues de estar en el laboratorio.

Laura

Y por último, y no menos importante, a la Facultad de Ingeniería de la Universidad Nacional Autónoma de México por brindarnos la preparación necesaria para seguir con nuestro desempeño profesional.

“Agradezco a,

...la Universidad Nacional Autónoma de México con la esperanza de que siga siendo un gran motor del crecimiento social en México.

...a su Facultad de Ingeniería y a toda la gente que en ella da su máximo esfuerzo para que siga siendo la mejor cuna de ingenieros en nuestro país; especialmente al Doctor Marco Antonio Arteaga Pérez por su apoyo y sincero interés en el éxito de este trabajo y a Maximiliano Bueno López por su desinteresada e incondicional ayuda.

También a mi familia que siempre ha sido sostén y ejemplo y sin la cual no sería nada.

Por último, a mis estimados amigos que hicieron que el paso por esta facultad fuera una experiencia única y digna de recordarse toda la vida”

Carlos

Índice general

Índice de figuras

Índice de tablas

Resumen

El presente trabajo es una de las varias columnas que sostienen un gran proyecto que hará uso de un sistema de visión para controlar cualquier robot manipulador con fines de investigación.

Para lograr obtener un sistema de visión es necesario un equipo de cómputo, un par de cámaras, un brazo robótico industrial, tarjetas de adquisición del tipo FPGA[1], una o varias leyes de control a experimentar y los programas correspondientes para desarrollar una interfaz que nos permita la comunicación entre la PC y el brazo. De lo anterior, nosotros nos encontramos con lo necesario para trabajar en ello.

Nuestra columna, refiriéndonos en parte a la interfaz de software que en un principio sólo contaba con la parte de visión para 2D por un lado y por el otro la parte de control, pretende ubicar a través de dos cámaras(3D) el efector final en el espacio de trabajo con una marca de leds de luz blanca que facilita dicha ubicación(3.2.1). Por ahora nuestros experimentos se tienen que realizar en un ambiente muy cuidado de luminosidad, donde la única fuente de luz será la marca luminosa del efector final, para esto se colocan cortinas que impiden la detección de otro punto de luz.

La interfaz de software, la parte de visión explícitamente, despliega en cuadros de texto las coordenadas en (x, y) y (y, z) de las cuales se deduce la triada (x, y, z) del efector final. Ya teniendo la ubicación antes mencionada es posible darle una trayectoria al brazo por medio de una ley de control que no necesita el modelo del robot para ser implementada,y al mismo tiempo con un observador son medidas las velocidades. Éste controlador es un tema ya estudiado en [13] y modificado en [12], el cual al implementarlo en nuestra aplicación, se experimentó con la interfaz que ya cuenta con dos ventanas correspondientes a las dos cámaras, así como la configuración necesaria para comenzar con el análisis de la ley de control en el mismo programa de manera inmediata.

Finalmente, existía una tarjeta de circuito impreso que comunicaba al robot manipulador con las tarjetas de adquisición CompactRio que se rediseñó pues contenía falsos contactos que daban lecturas poco confiables o inexistentes. En el Laboratorio de Robótica de Posgrado se cuenta también con un robot manipulador A255

[1]Es utilizada una unidad FPGA por las propiedades que benefician al procesamiento de algoritmos de visión listadas en la Tabla3.2

comunicado con la misma tarjeta, así que se le aumentó a la misma una funcionalidad que permite escoger de una lista de acciones la deseada, por ejemplo, la opción que enciende los dos robots o la opción que enciende sólo uno de ellos, etc. explicado en 3.3.5.1.

Capítulo 1

Introducción

1.1. Objetivos

Los objetivos de esta tesis son aprovechar los recursos existentes en el Laboratorio de Robótica del Departamento de Control y Robótica de la División de Ingeniería Eléctrica de la Facultad de Ingeniería de la UNAM para lograr implementar controles servovisuales. Así mismo, se desea mejorar la aplicabilidad de los robots con que cuenta actualmente este laboratorio. Los controles servovisuales utilizados se basan en imagen (ver clasificación en la Sección 2.1.2).

1.2. Planteamiento del Problema

Los algoritmos de control servovisual que se quieren utilizar son tales que no requieren del modelo dinámico del robot pero requieren del Jacobiano geométrico y por lo tanto de las señales de posición articular de los encoders. El robot que se utilza es un A465 de CRS Robotics (Figura 1.1) que lleva varios años en operación y que envía las señales de enconders a módulos de propósito general de un CompactRIO que se comunica por un puerto Ethernet a una PC. Por otro lado, a través de la misma computadora y del mismo CompactRIO se maneja otro robot (CRS A255) por lo que sería deseable poder selecciónar, activar y desactivar fácilmente cualquiera de los dos robots que se desee.

Para resolver estos dos problemas, existe ya una tarjeta diseñada en el laboratorio que no funciona adecuadamente. Lo anterior causa una operatividad disminuida de los robots y entorpece el trabajo de los usuarios del laboratorio al punto de hacer

imposibles cierto tipo de pruebas.

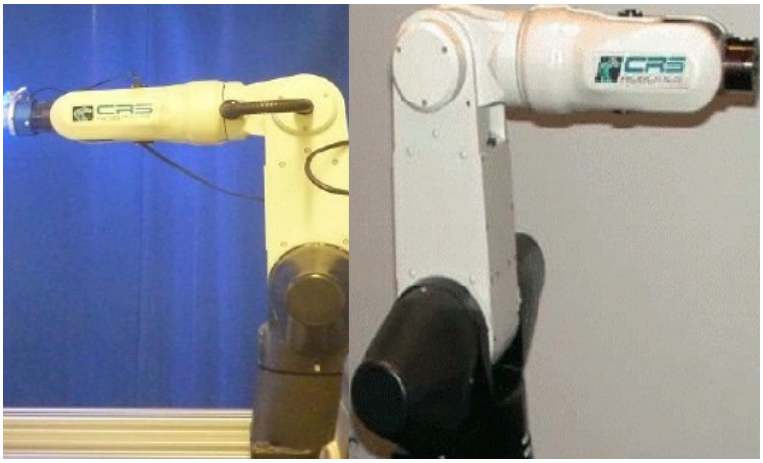


Figura 1.1: Brazo robótico de 6 grados de libertad con que se cuenta en el laboratorio de robótica de la Facultad de Ingeniería UNAM.

Finalmente, también es necesaria una plataforma en la cual, se puedan juntar todos los elementos y las herramientas que se necesitan para la implementación de los controles de interés, se concentre la información relevante de los sensores necesarios y se puedan programar los diferentes controles a experimentar así como los parámetros de estos. Existe una plataforma previa a este proyecto que permite básicamente lo descrito en este párrafo pero requiere ser completada, además que el programa carece de estructura que imposibilita actualizarlo.

1.3. Contribución

La contribución de esta tesis es principalmente un mejor aprovechamiento del laboratorio y sus recursos que facilitarán el trabajo de los usuarios de los robots por varios años. En concreto, las principales aportaciones de este proyecto son:

- Interfaz de hardware diseñada y probada para controles servovisuales y que será de gran utilidad y provecho para cualquier experimento que se desee realizar con los robots y que también facilitará la selección del robot con el que se quiera trabajar entre otras cosas.

- Una plataforma útil como reflejo de un programa analizado y estructurado desde el punto de vista del software, que evidentemente beneficia los experimentos de servocontrol y facilita la adaptación para otros proyectos de este y otro tipo.
- Se considera una contribución la implementación de prueba de controles servovisuales utilizando este sistema ya que servirá como una referencia de su uso.

Capítulo 2

Estado del Arte

Mientras que el concepto de robot nació en un ámbito de ciencia-ficción, hoy definitivamente es al menos un 90 % ciencia y solo 10 % ficción. Desde el nacimiento del termino en 1920, se ha llamado así a prácticamente cualquier cosa operada con cierto grado de autonomía. Para lo que atañe a esta tesis, un robot es un manipulador industrial controlado por computadora.

Este tipo de robots nació de una mezcla de los teleoperadores creados durante la segunda guerra mundial para manejo de desechos radioactivos y máquinas CNC[1] surgidas de la necesidad de realizar tareas con suma precisión.

El campo de aplicación de estas máquinas es muy extenso y el potencial de expansión de éste aun mayor. Una de las primeras aplicaciones populares de los robots fue transportar cosas pesadas o estampar productos. Más tarde con su creciente capacidad sensorial se les empezó a utlizar en actividades más complejas como soldar, lijar, pulir o ensamblar.

Actualmente los robots son muy útiles para realizar tareas en lugares remotos o con ambientes adversos para la vida humana como el espacio exterior o las plantas nucleares. Así mismo son utilizados en medicina para la realización de telecirugías, atención de pacientes o como prótesis, algunas de éstas controladas directamente por señales cerebrales del usuario. También llevan a cabo misiones militares peligrosas y ayudan en labores domésticas. Esta generalización en su uso es posible en buena medida, a su capacidad para sensar su entorno y tomar decisiones "inteligentes"". De estos sensores el que mayor información es capaz de proporcionar acerca del entorno y del robot mismo es definitivamente el de visión.

[1] Control Numérico por Computadora.

A continuación se presenta una breve explicación de la idea tras el control visual de robots, más adelante un pequeño histórico que pretende resumir la evolución de esta rama de la robótica y finalmente, se presentan algunos ejemplos de los frutos actuales de este enfoque para el control de robots.

2.1. Control Visual de Robots

El control visual de robots, como su nombre lo indica, emplea métodos de visión por computadora por medio de una o varias cámaras como sensor. A diferencia de otros sensores, las variables a controlar no siempre pueden ser medidas directamente por la o las cámaras. Por ejemplo, si lo que se desea es llevar al robot a un punto dado del espacio de trabajo, la variable a controlar es la posición del robot mientras que la información arrojada por el sensor (cámara) es un arreglo de valores de intensidades luminosas. Para poder inferir las variables de interés a partir de estos arreglos, es necesario aplicar algoritmos de visión por computadora. A lo largo de los años, se han desarrollado varios enfoques para solucionar el problema del control visual de manipuladores. Éstos varían dependiendo de cómo se utilicen los datos de la cámara, la posición de las cámaras relativa al manipulador, la selección de sistemas coordenados, etc.

2.1.1. Control Visual de Robots a través de los Años

El control servovisual data de incios de los años 70s cuando más de un grupo de investigadores pensaban que podría ser de mucho provecho utilizar cámaras como sensores. De hecho la idea no solo se aplicó al control de robots. Debido a las limitaciones tecnológicas de esos años, los experimentos que implementaron este enfoque no pueden clasificarse estrictamente como control servovisual. Sin embargo la tecnología avanzó de manera bastante rápida y a fines de esa misma década los sistemas implementados con control servovisual eran de hasta 10Hz y capaces de posicionar puntos de interés en tercera dimensión para seguimiento, soldadura y manipulación de objetos en movimiento.

En la tabla (vease Tablas 2.1 y 2.2)se resumen cronológicamente los avances más importantes en cuestión de control servovisual. Desde las primeras referencias en que Shiriai e Inoue [19] describieron cómo un lazo de realimentación visual podía ser usado para corregir la posición de un robot para incrementar la precisión de una tarea hasta la aplicación en controles sofisticados en el año 2002.

Año	Autor(es)	Aportación
1973	Shirai e Ioune	Sistema que permite la colocación de un objeto en una caja utilizando un lazo de realimentación visual y reconocimiento de contornos.
1976-1978	SRI International	Uso de realimentación visual para inserción de tornillos y recolección de objetos de una banda transportadora en movimiento.
1979	Hill y Park	Descripción de control servovisual de un robot Unimate usando imagenes binarizadas. Experimentaron con posicionamiento, estimación de profundidad y orientación así como movimiento planar y 3D guiado por visión.
1979	R. E. Prajoux	Control servovisual en un mecanismo de dos grados de libertad para seguir un gancho oscilante. Utilizó un predictor que estimaba la posición futura del gancho. Logró tiempos de establecimiento del orden de 1s.
1980	A. L. Gilbert	Descripción de cámara autorastreadora utilizada en un cohete. Mantiene el objetivo centrado en el plano de imagen de la cámara mediante movimientos laterales y de inclinación.
1984	L. E. Weiss	Uso del control adaptable para relaciones no lineales variables en el tiempo entre la pose del robot y las características de la imagen en servocontroles basados en imagen.
1985	A. G. Makhlin	Discusión sobre estabilidad, precisión y velocidad de rastreo para un sistema servovisual.
1989	Harrel y Slaughter	Control servovisual de dos grados de libertad para un robot recolector de fruta.
1989	Hashimoto,Kubota y Harashima	Control servovisual basado en redes neuronales. Los sistemas requieren entrenamiento pero la necesidad de relaciones analíticas complejas entre características de imagen y ángulos de articulaciones se eliminan.

Tabla 2.1: Aportaciones al control servovisual. Parte I

Año	Autor(es)	Aportación
1989-1991	J. T. Feddema	Desarrollo de la generación de trayectorias en el espacio de las características de imagen con control articular de lazo cerrado para superar problemas de baja tasa de muestreo visual.
1990	Zhang et al.	Control servovisual estocástico para un robot capaz de colectar objetos de una banda transportadora avanzando a 300 mm/s.
1991	Negahdaripour y Fox	Control de robots subaquaticos utilizando puntos visuales de referencia.
1995	Weis, Papanikolopoulos	Aplicación de controles PI, PID, LQG y asignación de polos.
1997	De Schutter,Morel y Malis	Integran los sensores de visión y fuerza.
1998	K. Hashimoto	Implementación del control óptimo.
1998	Khadraoui	Empleo del control robusto y H1.
2002	Mezouar	Estudio de la generación de trayectorias para control visual.
2002	Malis	Control servovisual invariante.
2002	Corke, Hutchinson	Control servovisual particionado.
2002	Kragic	Control servovisual robusto.
2011	Gerndt; Ostfalia Univ., Germany; Michalik y Krupop	Sistema de visión para robots móviles.
2011	Biao Zhang, Jianjun Wang, Gregory Rossano y Carlos Martinez	Método de ensamblaje robótico guiado por visión sin calibrar.
2012	Mohamad Bdiwi y Jozef Suchý	El sistema de visión detecta la posición y orientación de los objetos clasificándolos, identificar los códigos asignados a los objetos y proporciona las decisiones automáticamente que definen el arte del control.

Tabla 2.2: Aportaciones al control servovisual. Parte II

2.1.2. Modelos de Control Visual

Quizá lo primero que habría que decidir al momento de armar un sistema de control visual sería el lugar en el cuál colocar las cámaras. En general, existen sólo dos posiblidades para colocar las cámaras y dependiendo del lugar elegido el sistema de control visual se puede clasificar en: *cámara fija o cámara ojo en mano* [[14], Capítulo 12].

En un sistema de control visual cámara fija, la cámara se coloca en un lugar fijo y con una orientación fija tal que pueda observar al manipulador y a cualquier objeto que este vaya a manipular. Este enfoque presenta varias ventajas. Dado que la cámara se encuentra en una posición fija, el campo de visión no cambia conforme el manipulador se mueve. La relación geométrica entre la cámara y el espacio de trabajo es constante y puede ser calibrada fuera de línea. La desventaja de esta configuración es que conforme el maipulador se mueve a través del espacio de trabajo, puede obstaculizar el campo de visión de la cámara lo cual puede ser determinate en tareas que requieren alta precisión.

En un sistema cámara ojo en mano, la cámara frecuentemente se fija al manipulador sobre la muñeca de tal forma que el movimiento de muñeca no afecte al de la cámara. En esta forma, la cámara puede observar el movimiento de el efector final con una resolución constante y sin riesgo de oclusión por el manipulador. Un reto que presenta este tipo de sistemas es el de conocer la relación geométrica entre la cámara y el espacio de trabajo en todo momento ya que dicha relación es continuamente cambiante. El campo de visión puede cambiar drásticamente incluso con pequeños movimientos del robot, especialmente si el eslabón al que se encuentra fija cambia su orientación.

En cuanto a la forma en que se usan los datos dados por la cámara, se pueden adoptar dos enfoques distintos. El primero es aplicar algoritmos para extraer información del mundo real a partir de la imagen, por ejemplo infi- riendo a partir de la imagen la distancia en metros que recorre el efector final del manipulador y controlando su posición en el espacio de trabajo. El segundo es utilizar como variables de control magnitudes directamente medibles en la imagen, por ejemplo midiendo el desplazamiento del efector final del maipulador directamente en pixeles y controlando su posición en la imagen generada por la cámara. De cualquier forma, también es posible combinarlos para generar varios esquemas llamados esquemas particionados.

La primera forma de hacer un control visual es conocida como servo control basado en posición. Este enfoque consiste en usar los datos de visión para construir una representación 3D parcial del espacio de observación. Por ejemplo, en el caso en que se quiere llevar el efector de un robot hasta un objeto que se encuentra dentro del espacio de trabajo, se busca determinar las coordenadas 3D del objeto y del efector relativas al sistema coordenado de la cámara. Si es posible obtener estas coordenadas en tiempo real pueden utilizarse en un control que lleve al robot a la configuración necesaria. El problema de este tipo de control es obtener estas coordenadas en tiempo real sin causar inestabilidad en el sistema.

Un segundo método conocido como control servo visual basado en imagen, consiste en usar los datos recolectados por la cámara directamente (en 2D) para controlar el movimiento del robot. Una función de error es definida en términos de cantidades directamente medibles en la imagen (por ejemplo, coordenadas imagen de puntos y orientación de líneas en una imagen) y se construye una ley de control que mapea dicho error directamente en movimiento del robot. A la fecha, la forma más común ha sido usar puntos fáciles de detectar en un objeto como puntos de referencia. Luego, la función de error es el vector diferencia entre la ubicación deseada de estos puntos y la medida en la imagen.

Por otro lado, los controles visuales también suelen clasificarse en indirectos o directos. Indirectos es cuando la ley de control se plantea como un control articular que deduce las posiciones articulares del robot a partir de la información de la cámara y busca llevarlas a la posición necesaria para resultado deseado. En los controles directos, se elimina el control articular definiendo la ley de control en el espacio cartesiano. Las diferentes combinaciones de estos enfoques da origen a la clasificación en cuatro diferentes tipos: *control servovisual indirecto basado en posición, control servovisual directo basado posición, control servovisual indirecto basado en imagen y control servovisual directo basado en imagen, [11] y [12].*

2.2. Control Visual en la Industria

En la industria, es cada vez más común encontrar plantas y líneas de producción completamente automatizadas. En esto los robots ocupan un papel muy importante. En septiembre de 2011 la IFR (Federación Internacional de Robótica) reportó un máximo histórico de robots industriales alimentado por un alza de 18 % en ventas durante ese año. Se espera que la cantidad robots industriales a finales de 2014 sea de 1.3 millones [16].

El objetivo de esta sección es dar a conocer algunos ejemplos que encontramos relevantes de aplicaciones industriales actuales del control visual de robots para mostrar que es una técnica que jugará un papel muy importante en esta expansión por la gran flexibilidad de que dota a las líneas de producción. Actualmente las aplicaciones más comunes de los robots son:

1. Soldadura (por arco y por puntos)
2. Manejo de material
3. Carga y descarga de máquinas
4. Pintura
5. Empaque, embalaje y paletizado
6. Ensamblaje
7. Cortado, pulido y lijado
8. Pegado y sellado

Es fácil ver que muchas de estas se pueden beneficiar del control visual y prueba de ello son las siguientes aplicaciones.

FlexPicker - ABB

La empresa especializada en robots ABB ofrece un producto que promete ser una base para implementar aplicaciones de recolección a alta velocidad fácilmente y con un mínimo de programación sumamente simple. La unidad básica del sistema consta de una o dos bandas trasportadoras, una cámara y un robot IRB360

(paralelo) todos orquestados por el software *PickMaster*.

El software ofrece la facilidad de agregar elementos a la línea de producción de manera gráfica e integrarlos sin necesidad de programación. El verdadero meollo de la aplicación y lo que la hace sumamente flexible, es que cuenta con una cámara perfectamente integrada al sistema que encuentra los objetos de interés, pudiendo discernirlos de otros objetos en la misma banda y le permite al robot conocer el punto exacto en el espacio tridimensional al cual se tiene que dirigir para recolectarlo. La calibración de la cámara se hace de un modo muy sencillo utilizando una cuadrícula especial que permite conocer el origen de los ejes xy, el tamaño real de los objetos y su posición en la banda trasportadora. Gracias a estas características echar a andar aplicaciones de colección y colocación a alta velocidad es expedito. Este tipo de aplicaciones es muy común en líneas de producción que se dedican a empaque de productos.

Sellado - Mercedes-Benz

En la planta de ensamblaje de coches de Mercedes-Benz en Alemania se utilizan robots KUKA articulados guiados por visión utilizando un modelo cámara ojo en mano para aplicar sellador al chasis de los automóviles.

Recubrimiento de Aspas - Enercon

La empresa alemana Enercon se dedica a la fabricación de turbinas de viento para generadores eólicos que van desde los 800 hasta los 7,580KVA. Para aplicar el recubrimiento que protege a las aspas del deterioro por factores ambientales, utilizan una cámara de recubrimiento y secado automatizada con dos robots articulados montados sobre rieles que les permiten desplazarse a lo largo de la cámara (Figura 2.1).

Dado que la empresa produce una variedad de rotores y por lo tanto de aspas, era necesario lograr flexibilidad en la automatización de esta cámara de pintura. Esta flexibilidad se logró implementando un sistema de servocontrol (Figura 2.2) en el que se utilizan varias cámaras, debido a la longitud del espacio de trabajo, para detectar la posición del aspa y dirigir los robots pintores.

Mediante este sistema es posible lograr un excelente control de calidad y gracias a su precisión, también se logra optimizar, ya que el recubrimiento es tan grueso como se necesita pero al mismo tiempo tan delgado como sea posible.

Figura 2.1: Cámara de aplicación de recubrimiento y secado de aspas de la empresa Enercon, automatizada por dos robots ABB articulados servocontrolados.

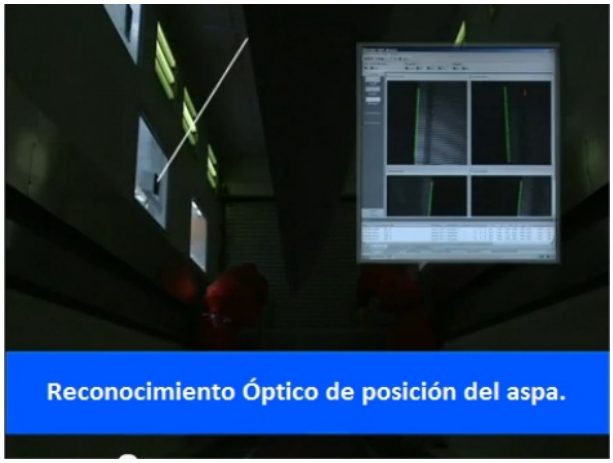


Figura 2.2: Por medio de cámaras es posible detectar la posición y el tamaño del aspa en cuestión.

Capítulo 3

Descripción de la Interfaz

3.1. Antecedentes

3.1.1. Operaciones

La transformación de imágenes responde al proceso de modificar el contenido de una imagen original para obtener una nueva, es decir, se prepara la imagen y posteriormente analizarla llevándonos a la percepción de nivel superior como en cualquier aplicación de visión artificial. Básicamente existen operaciones que transforman el contenido de la imagen con respecto a la intensidad y otras netamente geométricas.

Por mencionar algunas, entre las operaciones de vecindad está el filtrado de imagenes por convolución, así como el suavizado que elimina el ruido subyacente y por supuesto las operaciones de extracción de bordes.

Cuando se procesan datos en un sistema de visión, se distinguen:

- a.Operaciones individuales: Se altera pixel a pixel en escala global.
- b.Operaciones de vecindad: Operaciones basadas en muchos puntos.

3.1.1.1. Operaciones Individuales

Las operaciones individuales ejecutan una regla general a cada pixel. Primero se obtiene el valor del pixel localizado en la imagen, modificándolo por una operación lineal o no lineal que reemplazará dicho valor. Finalmente este procedimiento se aplica a todos los pixeles de la imagen. Por eso se dice que un operador lineal es una transformación uno a uno (vease Figura 3.1).

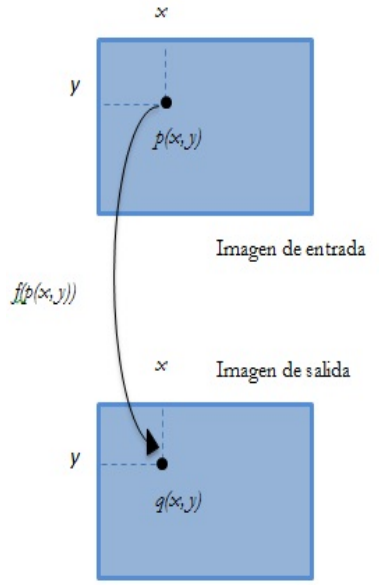


Figura 3.1: Operación Individual.

$$q(x,y) = f(p(x,y)) \tag{3.1}$$

La imagen nueva resulta ser de la misma dimensión que la original [1].

a) Operador Identidad. Crea una imagen de salida que es idéntica a la imagen de entrada.

$$q = p \tag{3.2}$$

b) Operador Negativo. Crea una imagen de salida que es la inversa de la imagen de entrada.

c) Operador umbral. Crea una imagen de salida binaria a partir de una imagen de grises, tomando un parámetro como entrada $p1$.

$$q = \begin{cases} 0 & si \quad p <= p1 \\ 1 & si \quad p > p1 \end{cases} \tag{3.3}$$

[1]Los procesos matemáticos con respecto a las relaciones 3.1, 3.2, 3.3 y 3.4 son simples y podemos encontrarlas en [6]

En el operador umbral existen variaciones del mismo, en los cuales se tiene uno o más parámetros, cuyos valores de gris se encuentran entre p1 y p2, por ejemplo, y los valores que se encuentran fuera tomarán 0 ó 255.

d) Transformación de dos imagenes punto a punto. Esta transformación utiliza la información contenida en la misma localización de dos imágenes de entrada R y S para crear una nueva imagen T.

$$T(x, y) = f_D(r(x, y), s(x, y)) \tag{3.4}$$

3.1.1.2. Operaciones de Vecindad

El procedimiento a seguir para las las operaciones de vecindad es el mismo que para las operaciónes individuales, tomando en cuenta siempre los valores de los pixeles vecinos.

Dada una imagen $f(i, j)$ se obtiene a partir de ella una nueva imagen $g(i, j)$ promediando para cada pixel (i, j) los valores de intensidad de los pixeles incluidos en un entorno de vecindad de (i, j), previamente definido.

a) Filtro de la mediana. Es un filtro de orden. Opera en la vecindad de un determinado pixel, ordenándolos por valores crecientes de nivel de gris y remplazándolo por el valor central que es la mediana.

b) Filtro de la media aritmética. Crea una imagen de salida que es la inversa de la imagen de entrada.

$$g(x, y) = \frac{1}{nm} \sum f(x, y) \tag{3.5}$$

donde nm es el número de pixeles de la ventana. Este filtro suaviza las variaciones locales [6].

3.1.2. Transformaciones Lógicas

Con las transformaciones lógicas, se entiende que los dos niveles de gris 0 y 255, son considerados como el 0 lógico y el 1 lógico. Ya en este punto, es factible operar lógicamente y relacionalmente.

3.1.3. Transformaciones Geométricas

Se llama Región de Interés al área de la imagen que nos interesa analizar, para ello se utilizan las operaciones geométricas que modifican las coordenadas espaciales de la imagen. Las operaciones geométricas transformarán los valores de una imagen para observarla desde otro punto de vista.

Interpolación

La interpolación es el cálculo del valor de intensidad de un pixel [6], en una posición cualquiera, como una función de los pixeles que le rodean.

$$p(x,y) = \sum_{i=-n}^{n} \sum_{j=-m}^{m} p(i,j)h(x-i, y-j) \tag{3.6}$$

El parámetro $h(x,y)$ se conoce como núcleo de interpolación, del cual existen varios tipos: vecino más próximo, bilineal y bicúbico[2].

Interpolación por vecino más próximo: Se decide el pixel más cercano entre otros cuatro, pudiéndose utilizar la distancia euclídea.

Interporlación bilineal: Asigna al pixel en cuestión un valor medio ponderado de las intensidades de los cuatro píxeles que le rodean. Cabe mencionar que este proceso tiene un coste mayor.

Interpolación bicúbica: Aquí intervienen 16 puntos vecinos del pixel que se está interpolando, y por supuesto que se obtienen mejores resultados.

[2]En la tabla 3.1 resume las características de los valores de pixel imagen resultante de la interpolación a partir del valor posicional de un pixel original [6]

Traslación

El desplazamiento consiste en mover el pixel (i, j) de la imagen original hasta que se encuentre en la posición $(i + i_d, j + j_d)$, sustituyendo también el valor de intensidad. La traslación de la imagen original viene dada por la siguiente transformación (vease [6], página 73):

$$x = i + i_d \tag{3.11}$$
$$y = j + j_d \tag{3.12}$$

Rotación

La transformación referida a la rotación es un giro, por lo tanto, hablamos de un ángulo con respecto al origen de coordenadas de la imagen (vease [6], página 73).

$$x = \cos(\Theta i) - \sin(\Theta j) \tag{3.13}$$
$$y = \sin(\Theta i) + \cos(\Theta j) \tag{3.14}$$

3.2. Adquisición de la imagen

3.2.1. Detección del Punto

La introducción de elementos externos permite al robot interactuar en el sistema de manera flexible, adaptándolo a una gama de tareas que a largo plazo corresponde a bajos costes de producción y de mantenimiento. Un sensor de alcance mide la distancia desde un punto de referencia hasta objetos en el campo de operación del sensor, sin embargo la visión artificial maneja conceptos más complejos. La

visión artificial puede ser definida como los procesos de obtención, caracterización e interpretación de información de imágenes tomadas de un mundo tridimensional. Estos procesos en la visión por computadora son listados de la siguiente manera:

- Segmentación. Divide la imagen en objetos que sean de nuestro interés.
- Descripción. Se obtienen características discriminantes para poder diferenciarlos, como área, forma, etc .
- Reconocimiento. Identifica los objetos.

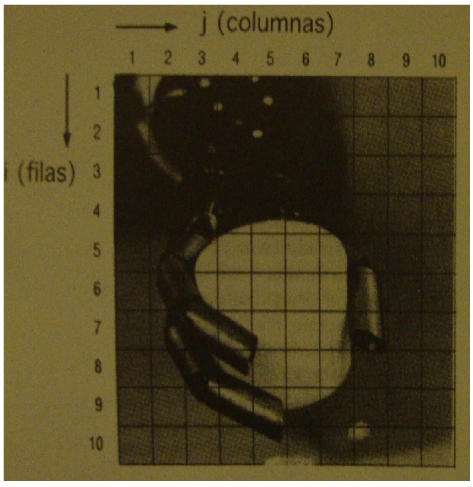
Tabla 3.1: Núcleos de Interpolación usados en el tratamiento geométrico de imágenes.

Tipo	**Valor de la Intensidad de Pixel Interpolado**	**Núcleo de Interpolación** $h(x,y)$
Interpolación por Vecino más Próximo	Toma el valor del pixel más cercano de los cuatro	$h(x) = \begin{cases} 1 & si \quad 0 <\mid x \mid< 0{,}5 \\ 0 & de \quad otro \quad modo \end{cases}$ (3.7)
Interpolación Bilineal	$p(x,y) = a_1 p(i,j) + a_2 p(i,j+1) + a_3 p(i+1,j) + a_4 p(i+1,j+1)$ (3.8)	$h(x) = \begin{cases} 1- \mid x \mid & si \quad 0 <\mid x \mid< 1 \\ 0 & de \quad otro \quad modo \end{cases}$ (3.9)
Interpolación Bicúbica		$h(x) = \begin{cases} 1 - 2 \mid x \mid^2 + \mid x \mid^3 & si \quad 0 <\mid x \mid< 1 \\ 4 - 8 \mid x \mid +5 \mid x \mid^2 - \mid x \mid^3 & si \quad 1 <\mid x \mid< 2 \\ 0 & deotro \quad modo \end{cases}$ (3.10)

- Interpretación. Significado a los objetos reconocidos.

La información visual es interpretada como una imagen digital debido al muestreo y cuantificación en amplitud de las señales eléctricas captadas por los sensores visuales. La función $f(x, y)$ que se obtiene de un dispositivo de imagen, donde x y y indican las coordenadas espaciales y el valor de f en cualquier punto (x, y) es proporcional al brillo de la imagen en ese punto. Una función $f(x, y)$ debe ser digitalizada espacialmente y en amplitud, dando así un muestreo de imagen, mientras que la digitalización de amplitud se llamará cuantización de intensidad o nivel de gris. Suponiendo que se muestrea una imagen continua, se obtiene la matriz de N filas y M columnas donde sus elementos están cuantificados en intensidad (vease Figura 3.2).

Cada elemento de la matriz se llama elemento de imagen o pixel.



(a) Antes de digitalización

j (columnas)

i (filas)	1	2	3	4	5	6	7	8	9	10
1	45	78	35	34	41	40	74	121	126	128
2	42	67	12	27	31	44	66	115	129	124
3	40	63	28	25	20	27	59	109	117	126
4	62	74	49	110	136	145	138	102	112	115
5	124	103	77	226	238	243	240	104	109	191
6	127	117	69	109	221	230	225	109	101	107
7	130	127	66	78	118	210	205	93	85	103
8	134	134	95	73	129	175	190	125	99	100
9	138	135	133	199	80	94	98	104	105	108
10	139	136	133	129	126	122	112	119	139	171

(b) Después de digitalización

Figura 3.2: Imagen original y su digitalización. Imagen ejemplo tomada de [5] página 221

3.2.2. Detección del Efector Final

Para detectar el efector final del robot, se optó por colocarle una marca fácil y certeramente detectable por las cámaras através de un puerto firewire que propiamente maneja datos en serie a gran velocidad [3] (vease Figura 3.3). Para lograr esto, dicha marca debía cumplir dos características. Debia ser semi esférica de modo que pudiera ser vista desde cualquier ángulo y también debia estar blindada contra el ruido ambiental.

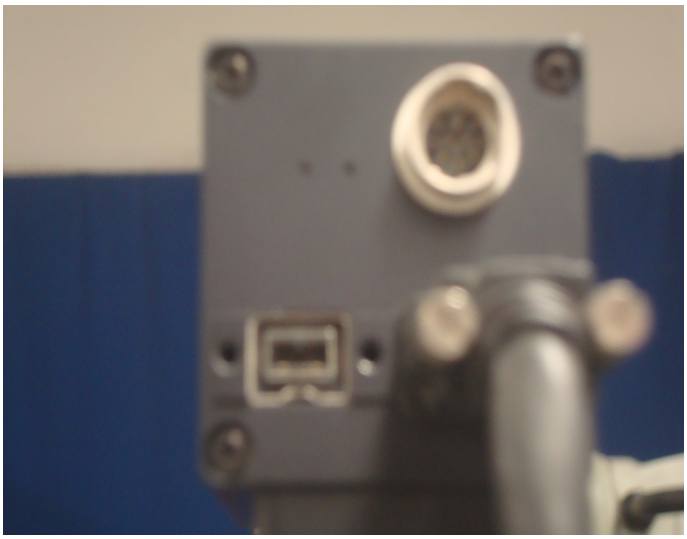


Figura 3.3: El puerto por el cual la cámara se comunica con el ordenador es el FireWire .

Se eligió utilizar LEDs para construir la marca. Esto por su bajo consumo de potencia. Por lo tanto, fue importante considerar que el ángulo de emisión de luz de los diodos a utilizar es de 13^o y por esto, se decidió hacer un arreglo esferico de LEDs equidistantes. Así se logró que la marca sea visible desde cualquier ángulo que forme con la cámara. Para blindarla contra el ruido ambiental, era necesario

buscar que los LEDs emitieran con una longitud de onda poco abundante en la luz solar. Sin embargo, dado que no se quería utilizar demasiada potencia ni correr el riesgo de perder la marca en algun momento se debia buscar también que la

[3] También conocido como IEEE 1394, utilizado para la interconexión de dispositivos digitales

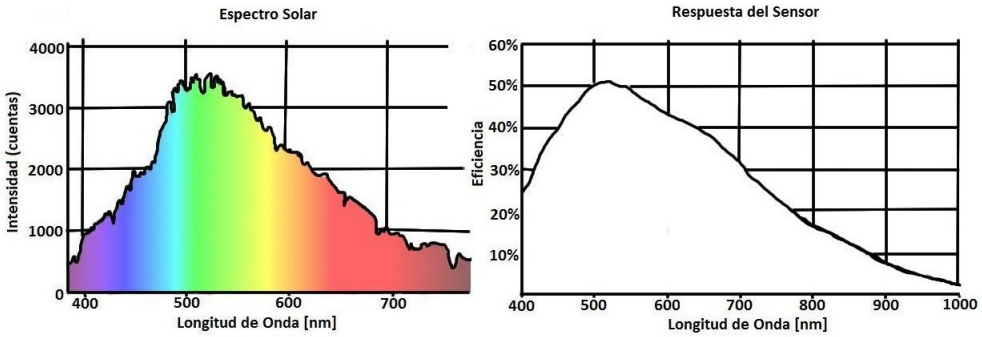


Figura 3.4: Del lado izquierdo el espectro solar y del derecho, la respuesta del sensor de la cámara a las distintas longitudes de onda.

respuesta del sensor de la cámara a esta lóngitud de onda fuera buena. Así, se observaron las gráficas de la Figura 3.4 en busca de la longitud de onda indicada. Es fácil observar que la longitud de onda con mejor equilibrio entre poca presencia en el espectro solar y buena respuesta del sensor (mayor al 40 %) es la de 470 [nm] (azul).

3.2.3. Interfaz Gráfica Diseñada

El lenguaje de programación con el que se desarrolló la interfaz del sistema de visión es el lenguaje orientado a objetos C++, utilizando como IDE o plataforma de desarrollo Visual Studio 6. Siendo más específicos, el proyecto de Visual Studio está basado en aplicaciones de MFC (Microsoft Fundation Class), el cual trabaja con llamadas a mensajes y éstos a su vez realizan acciones determinadas y en conjunto es una aplicación para programar en Microsoft Windows.

Se escogió basarse en MFC ya que es una poderosa herramienta que acorta el tiempo de desarrollo, hace código portable y provee de soporte sin quitar la flexibilidad del mismo. Además, MFC hace fácil la programación de características como ventanas de propiedades, vistas previas, así como barras de herramientas necesarias para una interfaz como este proyecto.

La interfaz se dividirá en dos partes para su precisa explicación, la correspondiente a las cámaras que hará el procesamiento de la imagen adquirida y aquella correspondiente al control del manipulador. En las Figuras 3.5, 3.6 y 3.7 se des-

criben sus componentes; y en la Figura 3.8 se muestra la ubicación de botones y ventanas usadas.

Las cajas de texto X, Y, Z, $y1$, $y2$, $y3$, $Tor1$, $Tor2$, $Tor3$, $z1$, $z2$, $z3$ muestran el valor en cierto momento de variables de importancia. Y la caja de Estado, como su nombre lo indica, despliega el estado de los distintos procesos como son Inicialización, Lectura, y Control.

3.2.3.1. Funcionamiento

El objetivo de la adquisición de la imagen (vease Figura 3.9) por medio de las cámaras es obtener una coordenada (x, y, z) que representa la posición espacial de nuestro efector final. Para ello se le colocó al efector final la marca luminosa hecha a base LEDs descrita en la Sección 3.2.1. Una cámara está ubicada enfrente del manipulador y la otra por encima y alineadas una con la otra en planos perpendiculares. Una descripción más detallada del arreglo de cámaras se da en la Sección 4.1.

Las cámaras captan la imagen (vease Figura 3.10) y el programa hace el análisis del procesamiento de imagen, dejándola binarizada, despues, se ajusta la intensidad de luz que captarán las cámaras. Se selecciona la caja $Grab$ [4] para ver las imágenes vivas, secuencia de imágenes vistas gracias a la programación de un temporizador, posteriormente se selecciona por medio de una caja combo en la cuál aparecen $Tarjeta0$ y $Tarjeta1$ la tarjeta a la cual está conectada cada cámara y por último, dando clic en $Iniciar$ [5] comienza el análisis para encontrar la coordenada correspondiente (vease las Figuras 3.11 3.12) que finalmente se despliega en las cajas de la parte inferior.

[4]Ésta función permite tomar la imagen actual que está viendo la cámara de ese instante y los siguientes, haciendo parecer que la imagen es continua. Los datos de la imagen se guardan en la variable de control mcvImg de la cámara correspondiente, puede verse la función en la **línea 100** del Apéndice B.

[5]Ésta función está perfectamente colocada dentro de la función *Ontimer*, la cual permite iniciar la función principal de visión. Dentro de la tarea de la misma se encuentra el procesamiento de nuestra imagen, el cual la binariza y finalmente encuentra su centroide o para fines prácticos, las coordenadas necesarias (vease la **línea 10** del Apéndice A).

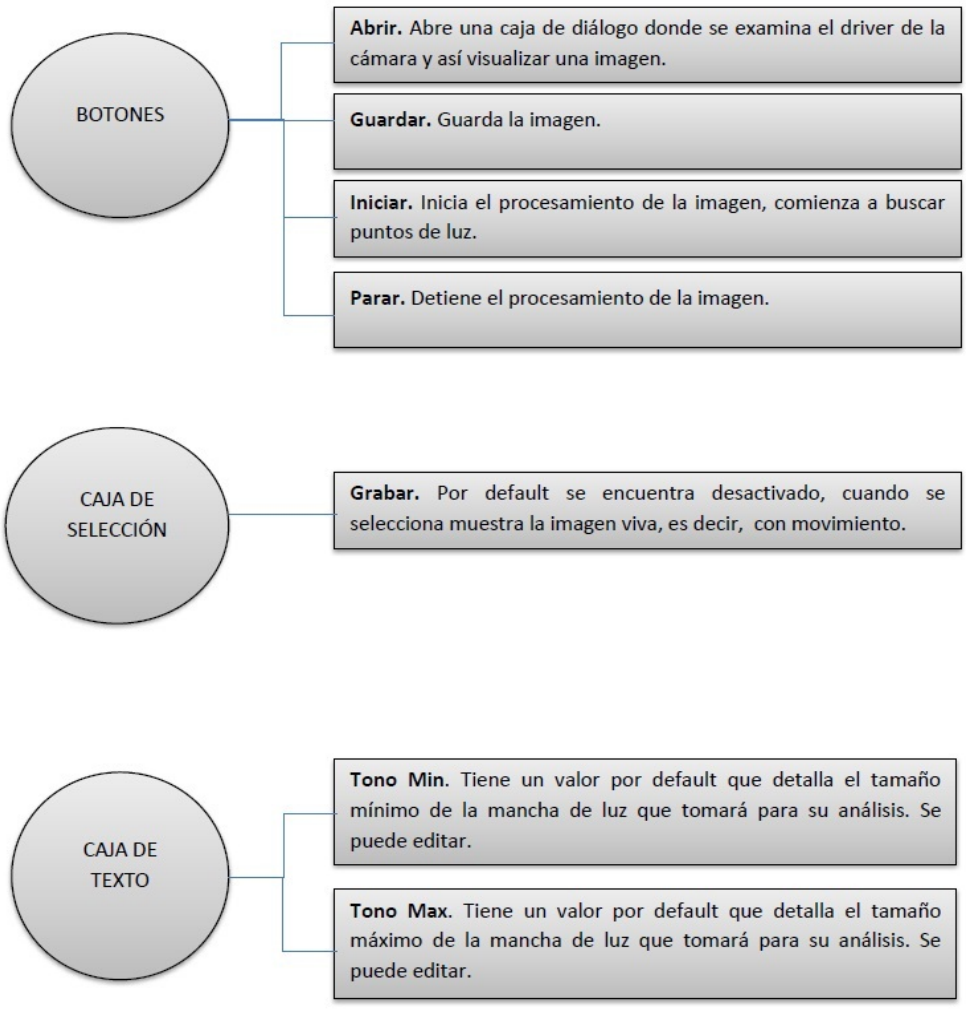


Figura 3.5: Componentes de control de la Interfaz Parte I

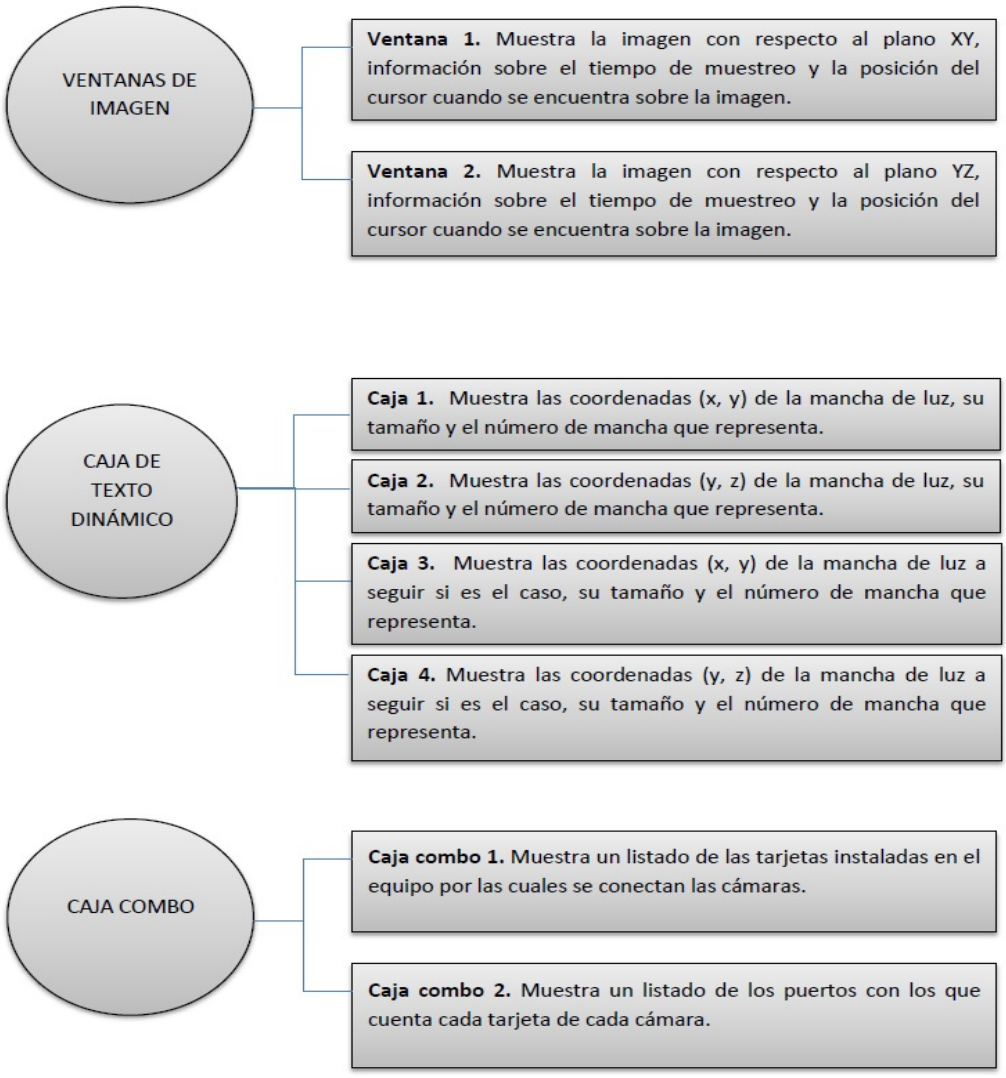


Figura 3.6: Componentes de control de la Interfaz Parte II

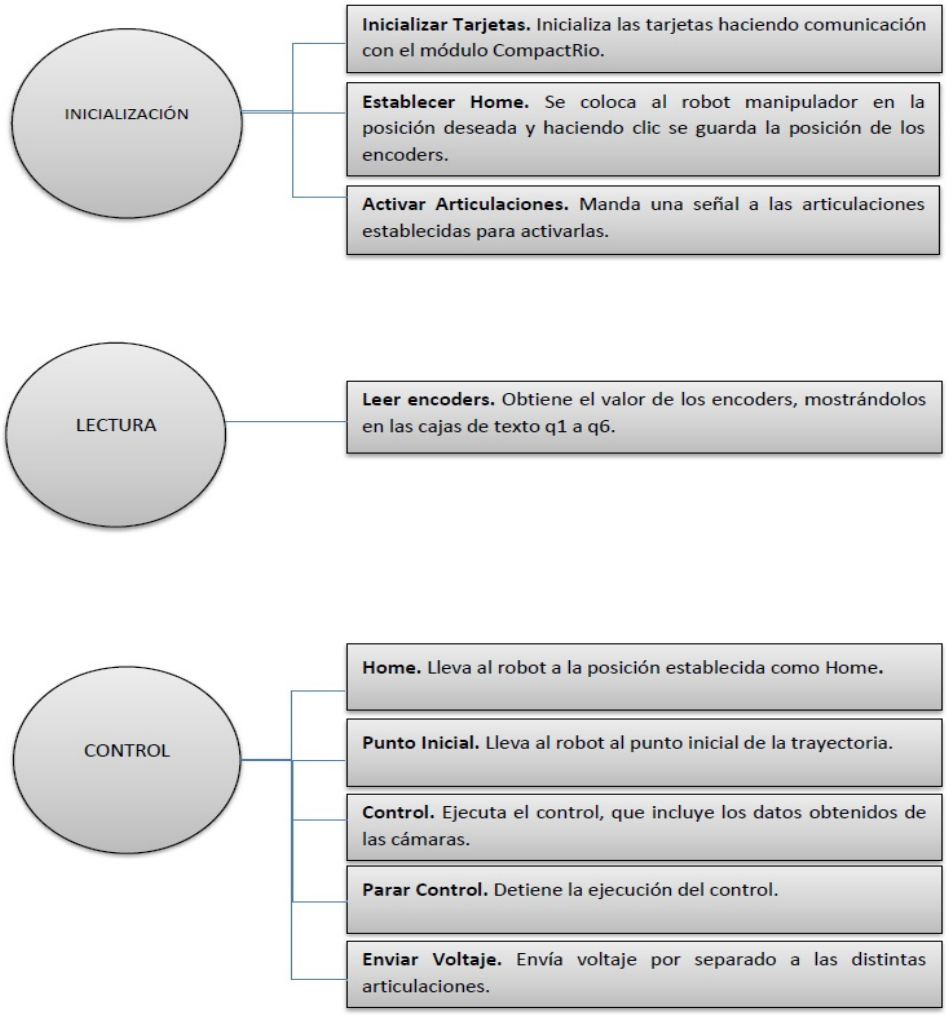


Figura 3.7: Componentes de control de la Interfaz Parte III

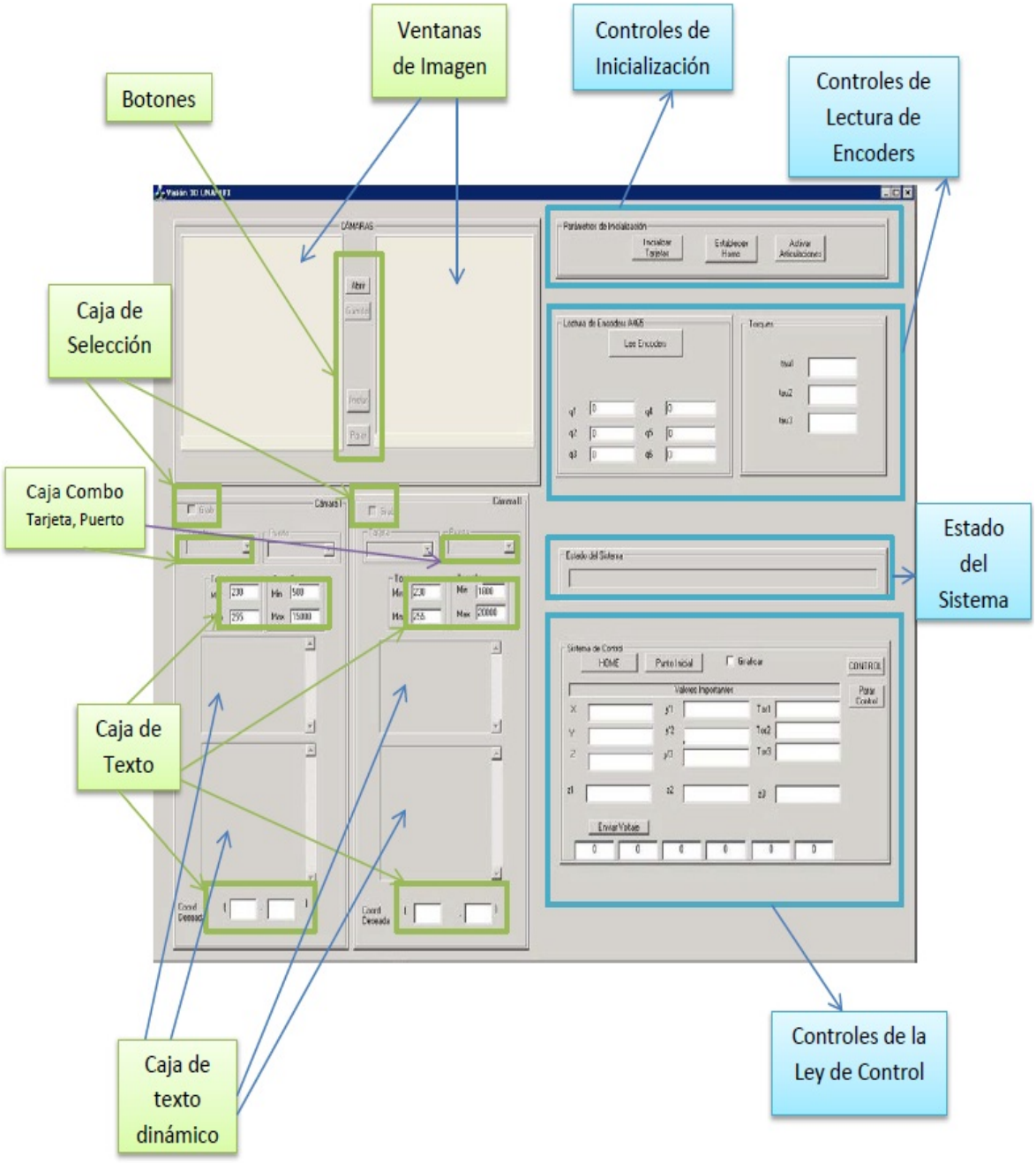


Figura 3.8: Interfaz de interacción con el sistema Visión-Control

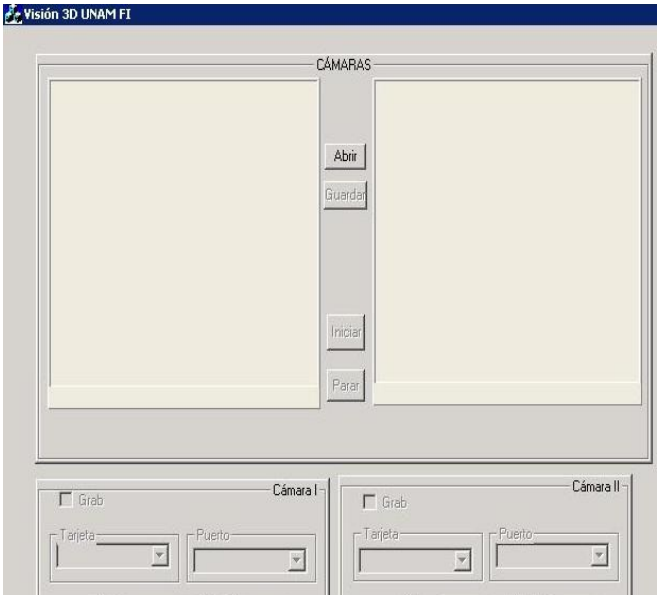


Figura 3.9: Parte de la Interfaz que Despliega la Imagen de las Cámaras

Los procesos a partir de la inicialización de tarjetas muestran su estado en la barra inferior de texto, indicando que llegan cada una a su fin.

Figura 3.10: Vista que Ejemplifica el Plano XY a una Cámara

Figura 3.11: Detección del punto y sus coordenadas (x, y, z)

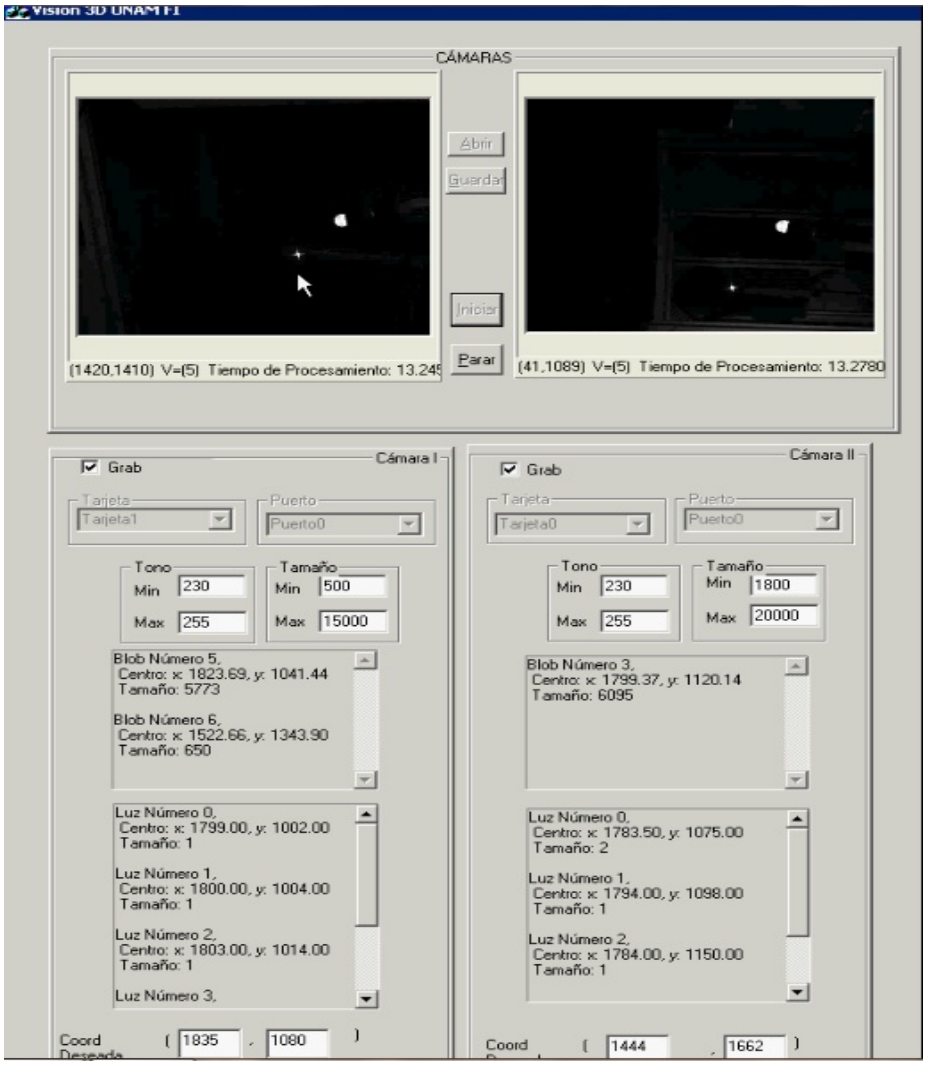


Figura 3.12: Detección del punto y el segundo a seguir con sus coordenadas (x, y, z)

3.2.4. Consideraciones del sistema de hardware

La Tabla 3.2 muestra el rol que juega el hardware cuando se desea implementar algún tipo de algoritmo, que para nuestros fines es importante se conozcan dichas características y obtengamos el mayor provecho. Así pues, la tabla se puede encontrar en [3, pág 819]

Código

El sistema de visión para el control del robot manipulador es usado por un único usuario al cual llamaremos administrador, el cual es guiado por los mensajes y datos que arroja la interfaz. Por lo cual, y para un comprensión global de cualquier profesional se hizo uso del lenguaje unificado de modelado UML, (vease la Figura 3.13). Este lenguaje es asociado muy frecuentemente al modelado de software orientado a objetos. UML es en escencia una sintaxis visual que podemos usar para construir modelos.

El código con el cual trabaja el proyecto de C++ tiene el siguiente diagrama de flujo (vease la Figura 3.15):

Un aspecto importante en la escritura del código fue el uso del timer, pues es quien hace que las diversas funciones usadas se ejecuten las veces necesarias y parar su ejecución para no afectar el tiempo de procesamiento del equipo.

La estructura del temporizador usado se muestra a continuación:

```
1              UINT CTwoCamsDlg::StartTimer(UINT TimerDuration, UINT TimerID↩
                   )
2             {
3               UINT TimerVal;
4               TimerVal = SetTimer(TimerID, TimerDuration, NULL);
5               if (TimerVal == 0)
6               {
7                 MessageBox ("Unable to obtain timer","IDT_TIMER_0",MB_OK|↩
                      MB_SYSTEMMODAL);
8               }
9               timerActive=true;
10              return TimerVal;
11          }
```

Es necesario el temporizador por que windows no es un sistema operativo en tiempo real y existe un desfase significativo cuando se desea ejecutar una acción.

Para la consulta del temporizador y las funciones principales que en él se encuentran, vease el Apéndice A.

Unidad	Función	Propiedades
PC	**Computadora Personal:** cuenta con RAM, HDD y unidades periféricas.Podría necesitar un sistema operativo embebido en una aplicación de tiempo real	- rápido - costo medio - extremadamente flexible - puede ser visto como unidad de software
MP	**Microprocesador :** chip que contiene CPU + RAM caché. El elemento fundamental de una PC	- rápido - bajo costo - extremadamente flexible - puede ser visto como unidad de software
DSP	**Procesador de Señal Digital :** MP de palabra de instrucciones largas, chip diseñado específicamente para procesamiento de señales— alta velocidad de procesamiento sobre una arquitectura restringida.	- muy rápido - bajo costo - altamente flexible - puede ser visto como una unidad de software
FPGA	**Arreglo de Compuertas de Campo Programable :** Arreglo de compuertas de lógica aleatoria con vínculos programables; pueden en ocasiones se dinámicamente reprogramables dentro de la aplicación . Los últimos dispositivos tienen flip-flops y funciones de alto nivel ya contenidos en el chip, listos para unirlos; algunos de estos dispositivos en ocasiones uno o más MP en la tarjeta.	- rápido - bajo a medio costo - extremadamente flexible - puede ser visto como una unidad de hardware, comunmente esclavo de un DSP - puede ser una unidad de software si es controlado sobre un ship MP
LUT	**Búsqueda en un índice RAM o ROM :** Usualmente para búsquedas de funciones cruciales. Normalmente esclavo de un MP o DSP.	- muy rápido - bajo costo - extremadamente flexible, si se construye usando RAM - puede ser una unidad de software esclava
ASIC	**Aplicación Específica de Ciruitos Integrado :** Contiene unidades como Transformadas de Fourier, o una variedad de específico SP o funciones de Visión. Normalmente esclavo de un MP o DSP.	- muy rápido - medio costo - inflexible (flexibilidad sacrificada por rapidéz) - puede ser una unidad de software esclava
Chip de Vision	Los Chips de Vision son ASICs que están específicamente para visió: ellos pueden contener funciones de vision importantes; como detección de bordes, desgloce de algoritmos y relacionado con analizadores de componentes. Normalmente esclavo con un MP o DSP	- muy rápido - costo medio - carece de flexibilidad - debe ser visto como un componente de software
VLSI	**Chip VLSI Personalizado :** comunmente tiene muchos componentes en un solo circuito con una particular aplicación funcional en mente.	- rápido - alto costo - carece de flexibilidad - debe ser visto como un componente de hardware - normalmente esclavo con un MP o DSP

Tabla 3.2: Hardware para la Implementación de Algoritmos de Visión

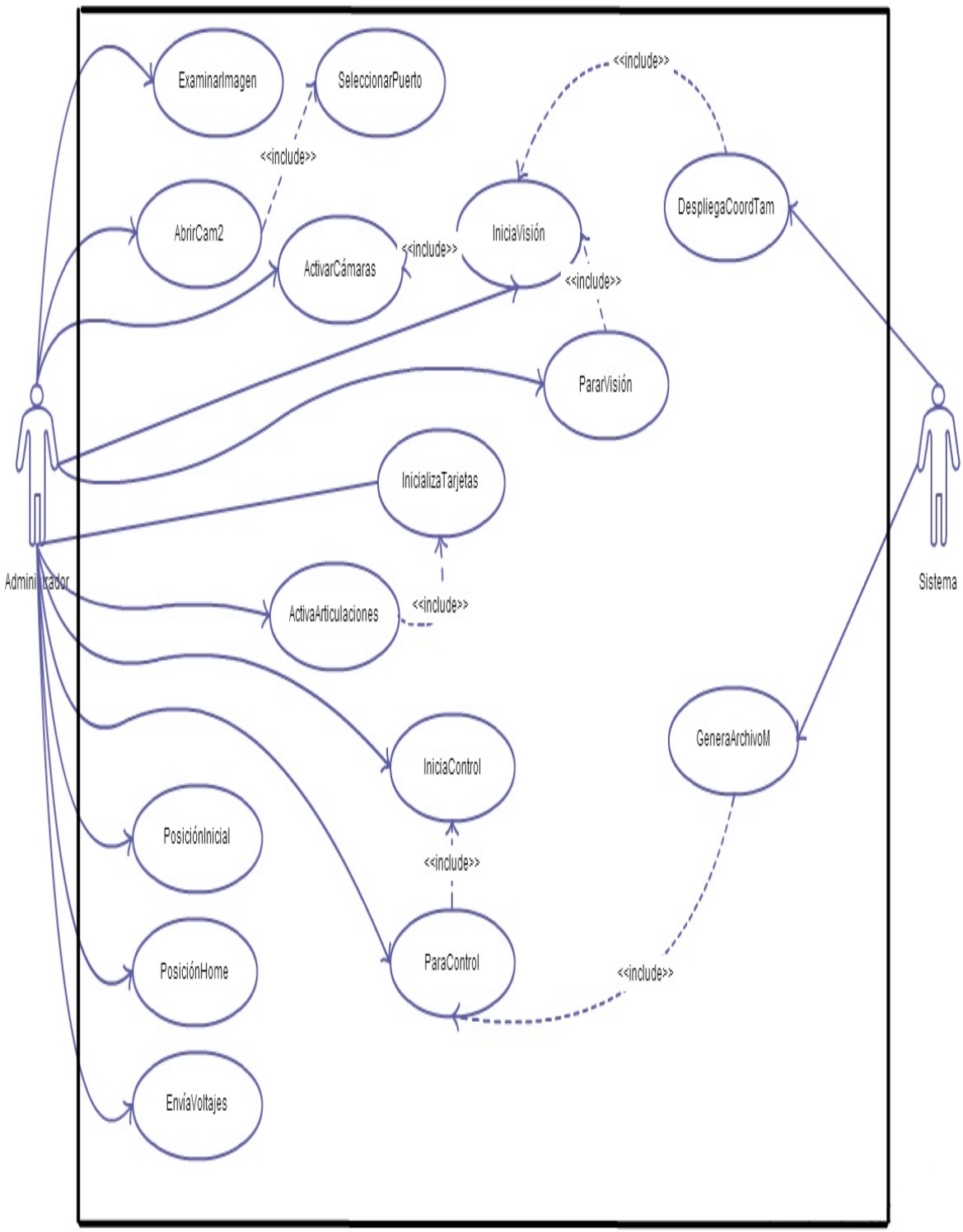


Figura 3.13: Diagrama de Casos del Sistema de Visión para el Control del Robot Manipulador. UML

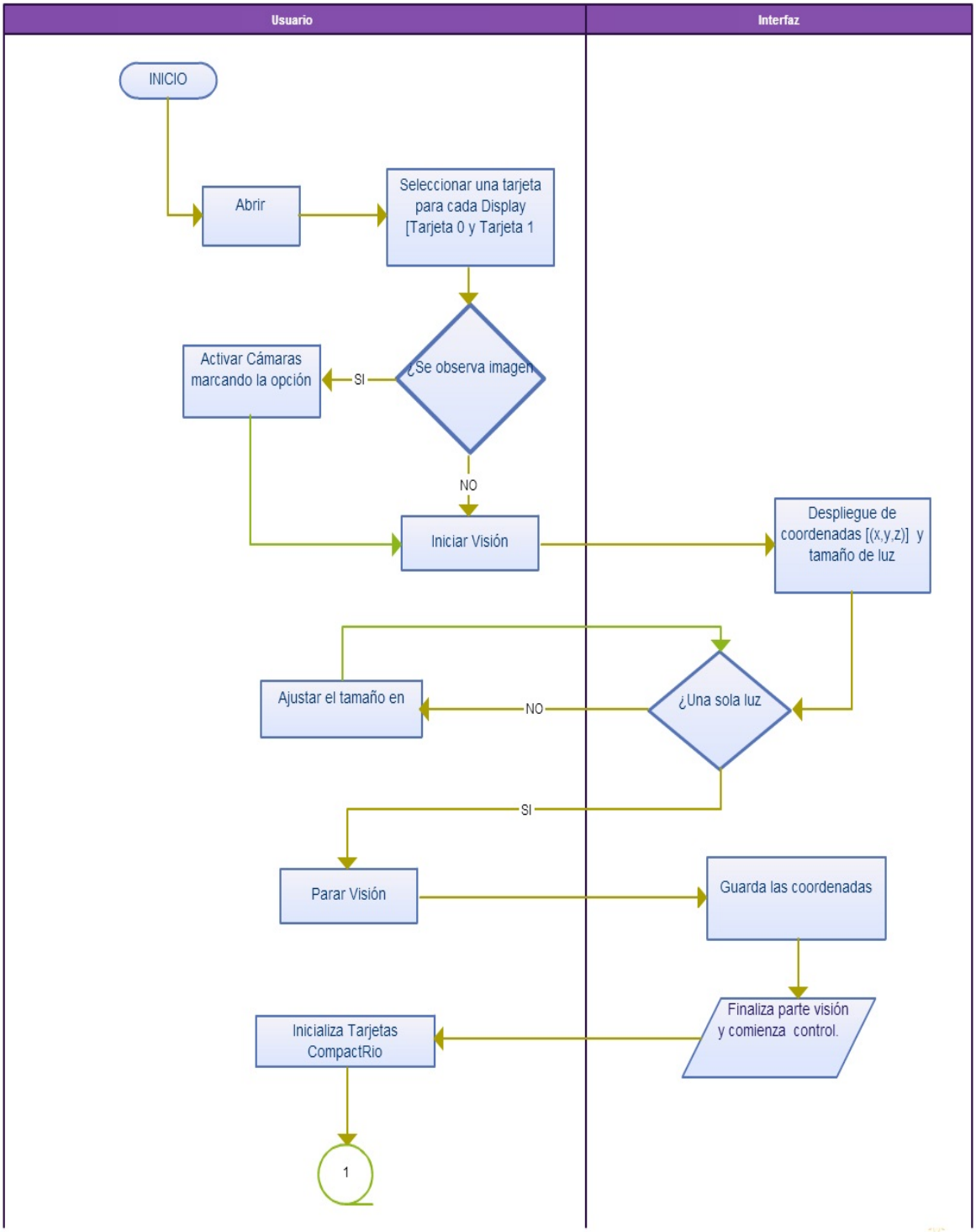


Figura 3.14: Diagrama de Flujo del Sistema de Visión para el Control del Robot Manipulador Parte I.

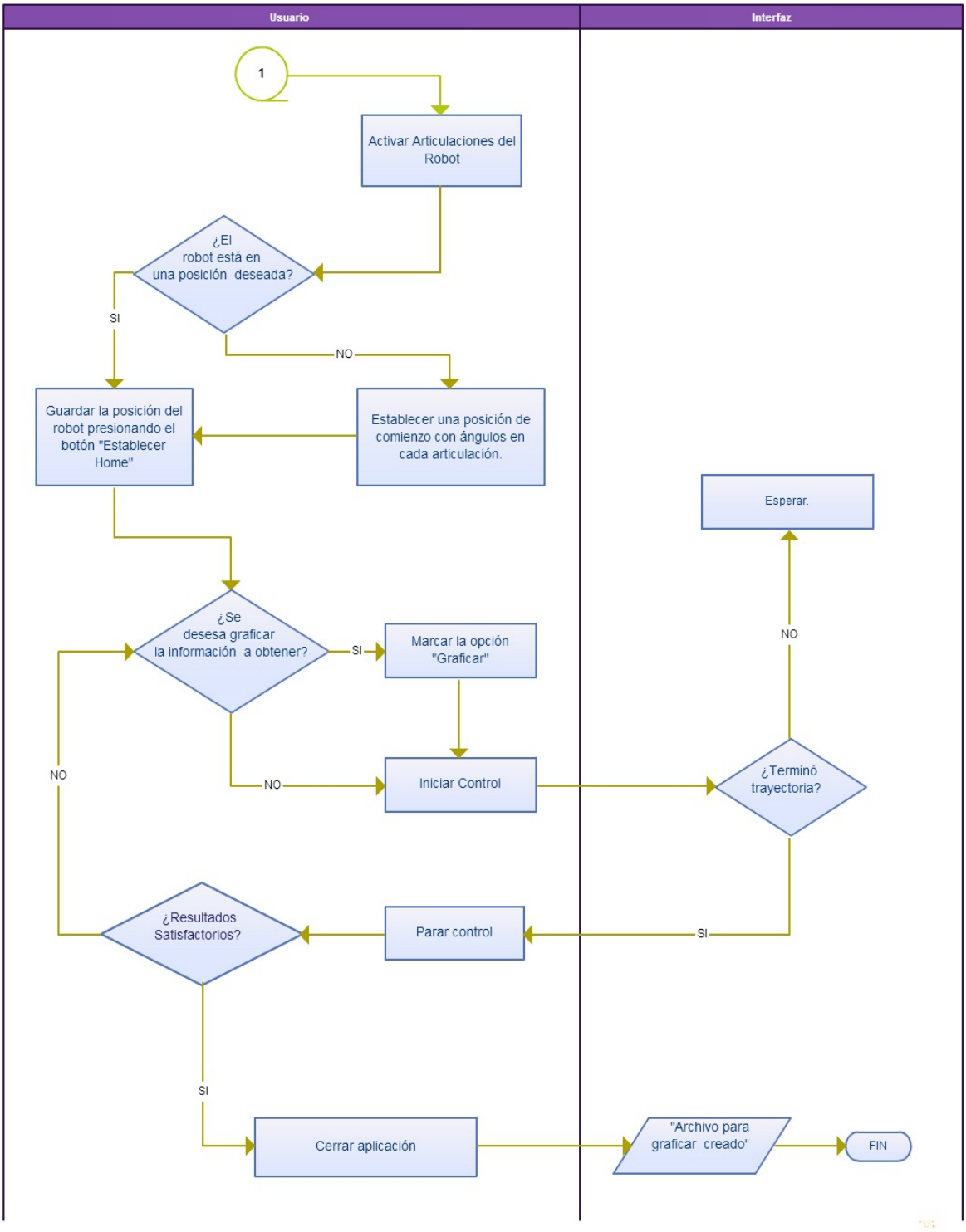


Figura 3.15: Diagrama de Flujo del Sistema de Visión para el Control del Robot Manipulador Parte II.

3.3. Diseño e Implementación de Sistema de Adquisición de Señales para Robots Industriales

Una parte importante de las señales de entrada y salida de nuestra aplicación para la correcta ejecución de un controlador cualquiera, es afirmar sin temor a equivocarnos que éstas tengan como características únicas la precisión y el menor ruido posible. En consecuencia, las lecturas y escrituras de los datos aseguran el mínimo error en los resultados. Para esto se diseñó un circuito que se encarga de acoplar las señales del robot a los módulos del CompactRio, así como también escoger el robot a usar. En las secciones siguientes se explicará a detalle cada una de las etapas que constituye dicho diseño en una placa.

La placa fue hecha en un programa llamado EAGLE v.5.10 (Easily Applicable Graphical Layout Editor), con el cuál se empezó por construir el esquemático para continuar después con el diseñó de dicho PCB. Cabe mencionar el cambio visual que se dió a consecuencia del rediseño de la placa (vease Figura **??**). Y finalmente se muestran los módulos funcionales con que cuenta la placa de trabajo (vease Figura 3.17).

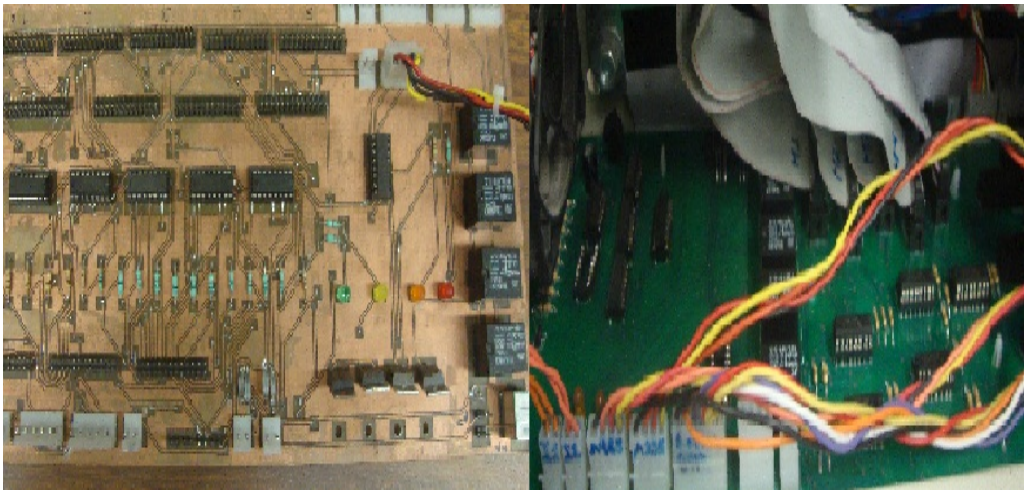


Figura 3.16: Antes y Después de la Placa de Acondicionamiento para el Robot A465 y A255

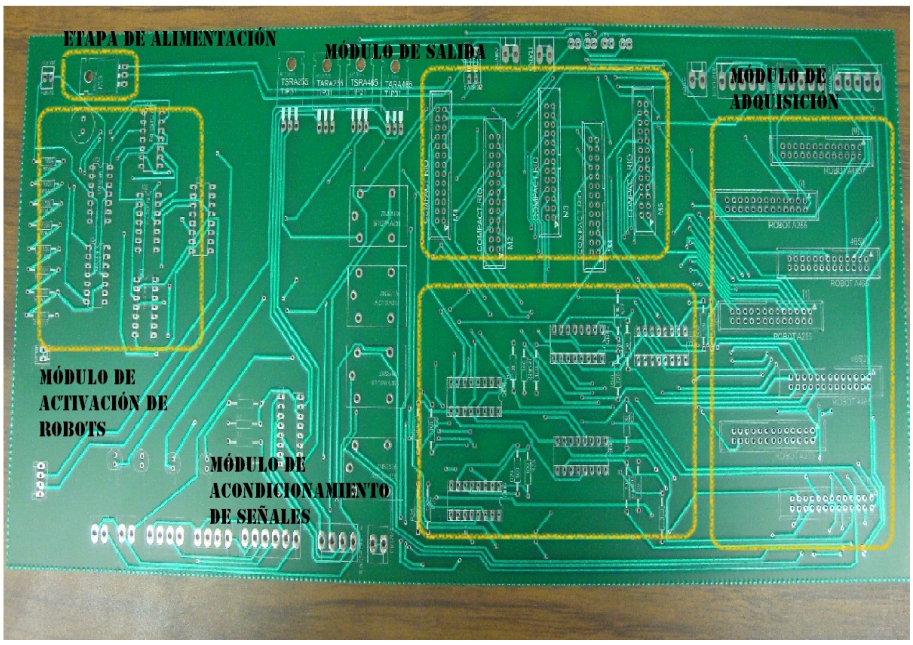


Figura 3.17: Diseño PCB de la Placa de Acondicionamiento para el Robot A465 y A255

3.3.1. Etapa de Alimentación

Como primera etapa se encuentra la alimentación de la placa. Existen dos voltajes, 24 [volts] y 5 [volts], de los cuales la fuente "QUINT POWER"provee el primer voltaje, teniendo las siguientes como principales ventajas:

- Notifica de estados de funcionamiento críticos antes de que se produzcan fallos, monitoreando permanentemente la tensión y la corriente de salida.
- Tensión de salida ajustable en el lado frontal mediante tres fuentes de alimentación con salida de 12, 24 y 48 V CC.
- Carcasa de metal y amplio margen de temperatura de -25 a 70°C.

La fuente QUINT POWER también alimenta al módulo de National Instruments CompactRio que será explicado en el Módulo de Salida.

En la placa se encuentran circuitos operacionales en integrados de la familia lógica TTL (Transistor Transistor Logic), por tanto fue necesario reducir la tensión

Figura 3.18: Fuente de Alimentación de las Tarjetas de Adquisición y Acondicionamiento

a 5 [volts]. Para cambiar el valor de tensión continua para lo que necesitábamos, se hizo uso de un convertidor DC-DC marca Recom con número de parte de fabricante R-785.0-0.5.

El convertidor RECOM es un dispositivo confiable que tiene la misma configuración de conexión de pines que un LM78XX, brindándonos de alta eficiencia, no aislado y que no requiere de componentes externos, ni de disipadores térmicos; cuenta también con protección contra cortocircuitos, bajo rizado y ruido.

3.3.2. Módulo de Adquisición

El brazo robot usado para nuestra aplicación es el CRS A465 que está provisto de seis grados de libertad y un controlador multitarea, que cumple tareas típicas de corte, perforación etc. Originalmente se manejaba con el controlador C500C, sin embargo ahora tomamos las señales directas a la placa que se detalla en este capítulo. Específicamente el módulo de adquisición confiere a las señales del brazo a unos integrados que tratan las señales, trato que es visto en la sección siguiente.

Las señales que se encuentran en contacto directo con el brazo robótico, se conectan a la placa por medio de conectores DB25 hembra, y su distribución se describe en la Tabla 3.3.

Tabla 3.3: Configuración conectores DB25 A465

PIN	DB25[1]	DB25[2]	DB25[I]	DB25[II]
1	A5+	GND	Vcom1	N/C
2	Z4+	GND	Vcom2	N/C
3	B4+	N/C	Vcom3	N/C
4	A4+	GND	Vcom4	N/C
5	Z3+	N/C	HSW1	N/C
6	B3+	N/C	N/C	N/C
7	A3+	N/C	HSW2	N/C
8	Z2+	Z6-	N/C	N/C
9	B2+	N/C	HSW3	N/C
10	A2+	A6-	N/C	N/C
11	Z1+	N/C	HSW4	N/C
12	B1+	B5-	N/C	N/C
13	A1+	N/C	HSW5	N/C
14	Z4-	12 V	Vcom5	N/C
15	B4-	12 V	Vcom6	HSW6
16	A4-	N/C	Vcom7	N/C
17	Z3-	N/C	Vcom8	N/C
18	B3-	N/C	N/C	N/C
19	A3-	N/C	N/C	N/C
20	Z2-	N/C	N/C	N/C
21	B2-	Z6+	N/C	N/C
22	A2-	B6+	N/C	N/C
23	Z1-	A6+	N/C	N/C
24	B1-	Z5+		N/C
25	A1-	B5+	N/C	N/C

3.3.3. Acondicionamiento de Señales

El motivo por el cual se decidió colocar como intermediario un filtrado de señales, fue el ruido abundante que hacía difícil la lectura y asignación de tareas por la variación de voltaje siempre presente. Para ello se incluyeron integrados del tipo Receptor de Línea Diferencial, que nos entrega una señal apta para manejarla.

El integrado del cual se habla es el AM 26LS32ACN, en él las transiciones de voltajes y de corrientes en la línea son iguales y opuestas, de esta manera se cancela cualquier ruido. También se genera muy poco ruido de tierra, por lo que no contribuye a introducir ruido en el entorno. Este método de línea diferencial funciona de igual manera para líneas de largas distancias y en entornos eléctricos ruidosos.

3.3.4. Módulo de Salida

La sección posterior al filtrado de señales es el módulo de salida, el cual entrega las mismas a las entradas digitales del Compact Rio de National Instruments.

El CompactRIO es un sistema de control embebido reconfigurable y de adquisición. El sistema CompactRIO incluye un controlador embebido y un chasís reconfigurable. El controlador ofrece la ejecución de gran alcance embebida independiente para aplicaciones determinísticas de Tiempo Real en LabView o aplicaciones flexibles de Windows Embedded Standard 7.

NI CompactRIO provee acceso directo al hardware que comprende la circuitería de entrada/salida de cada módulo I/O, utilizando las funciones elementales de I/O del FPGA de LabVIEW. Cada módulo I/O contiene el acondicionamiento de la señal y la terminal "screw"[6], BNC o conectores D-sub.

El sistema utilizado CompactRIO está integrado por cinco módulos NI 9401. Éstos cuentan con 8 canales digitales bidireccionales de I/O de alta velocidad, configurables por nibble y accesando mediante el estándar DB25.

La comunicación entre el equipo de cómputo y el CompactRIO se lleva a cabo mediante el protocolo TCP/IP. Vease la Figura 3.20.

3.3.5. Módulo de Activación de Robots

En la etapa donde a través de la computadora se deciden acciones reflejadas en combinaciones de 4 bits, se colocó un optoacoplador que aisla los circuitos de manera que las señales sólo se comunican a través de un haz de luz.

El encapsulado optoacoplador usado y encargado de transmitir señales de diferente potencial tiene el número de parte PC-849. Cuando el número binario es recibido por este integrado, se iluminan cuatro leds permitiendo saber dicho número, por ejemplo, sabremos cuando un robot es escogido y activado (ver Tabla 3.4).

[6]terminal de tornillo

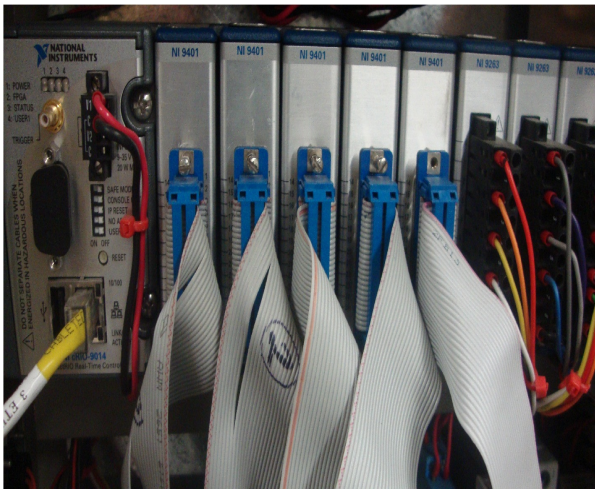


Figura 3.19: Módulo CompactRio NI 9401

3.3.5.1. Convertidor A/D cuatro bits

El convertidor usado en este proyecto es de tipo directo, contando con la característica de alta velocidad de conversión, ideal para resoluciones menores, pues cuando el número de los componentes se empieza a extender el diseño se complica.

La resolución de cuatro bits fue escogida para que en futuras tareas fuera posible agregar más acciones a conveniencia de las necesidades, en este proyecto usamos sólo cuatro del total de combinaciones (vease la combinación usada para cada opción en la Tabla 3.4):

1. Brazo A465 encendido
2. Brazo A255 encendido
3. Brazo A465 y Brazo 255 encendidos
4. Brazo A465 y Brazo 255 apagados

Para una mejor descripción del diseño del convertidor A/D, hacemos uso del esquemático de Eagle que se muestra en la Figura **??**. Éste último describe las interconexiones del convertidor analógico-digital.

El circuito es alimentado con 24 volts los cuales entran a una cascada de resistencias para obtener nuestros voltajes de referencia, y con los cuales se compara el voltaje de entrada que tiene que variar para decidir el nivel de tensión en el que se encuentra. Finalmente, las señales de salida de los operacionales con número de parte LM324 entran a un codificador de entradas y salidas negadas 74LS147 (10:4) al que se le conectan las 7 salidas de los comparadores. Dicho codificador nos devolverá un número en base binaria de cuatro bits, respondiendo a la relación

$$resolucin = 2n - 1$$

donde n=4, resultando así los 7 amplificadores operacionales que necesitamos para que todo funcione.

Tabla 3.4: Relación de entradas y salidas del A/D

				Q_0	Q_1	Q_2	Q_3	Opción
0	0	0	0	1	1	0	1	4
0	0	0	1	0	0	1	1	3
0	0	1	1	1	0	1	1	2
0	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0

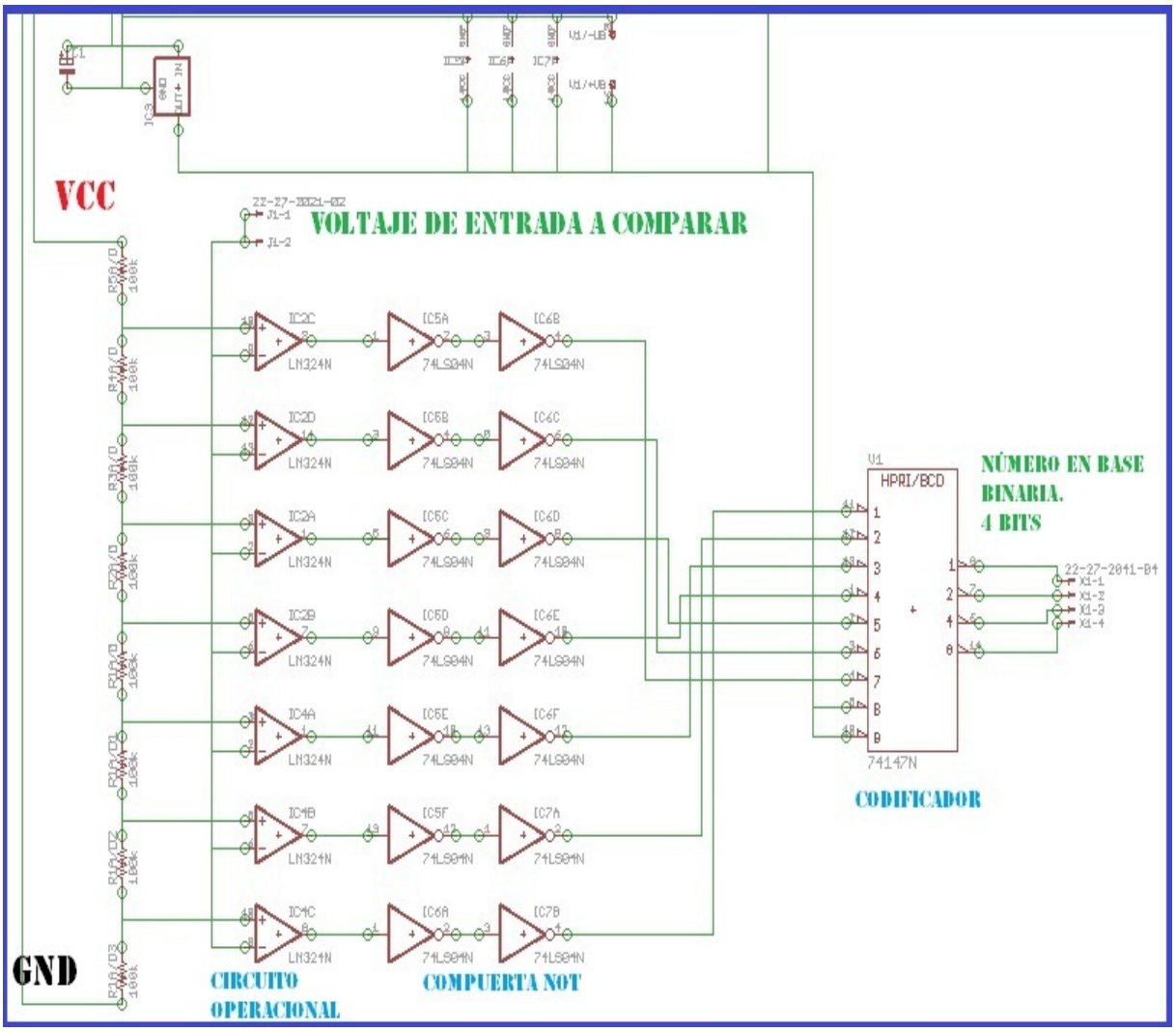


Figura 3.20: Esquemático que muestra los componentes y su interconexión del Convertidor Analógico-Digital de 4 bits.

Capítulo 4

Diseño del Control Servovisual 3D

En este capítulo se explica el diseño del control servovisual en 3D, basado en imagen, cuyo objetivo es seguir una trayectoria especificada en coordenadas de imagen. Así, en la primera parte de este capítulo se describe el arreglo de cámaras utilizado y el modelo que permite relacionar coordenadas de imagen con coordenadas cartesianas. Dicha relación permite adaptar el control originalmente en espacio cartesiano a espacio de coordenadas de imagen. Posteriormente se describen el control y el observador utilizados para seguir una trayectoria en 2D y en 3D.

4.1. Arreglo de Cámaras

Mientras que la imagen producida por una cámara es una representación bidimensional (proyección) de lo que se encuentra en su campo de visión, mediante distintas estrategias es posible recuperar la información tridimensional a partir de este tipo de representaciones. Algunas de estas estrategias se basan en múltiples cámaras colocadas en puntos cuya ubicación es conocida, otras en varias imágenes con distintas iluminaciones o distintos enfoques, etc. [21]. A pesar de que toda esta variedad de técnicas existe, los algoritmos para llevarlas acabo son complicados y requieren de muchos cálculos.

La solución más sencilla, aunque requiere mayor capacidad de procesamiento, es colocar dos cámaras. Dado que en el laboratorio ya se disponía de las cámaras necesarias y de la capacidad de procesamiento, se optó por un arreglo como el que se muestra en la Figura 4.1.

El arreglo consta de una cámara denominada cámara uno cuyo plano de imagen es paralelo al plano xz del robot (puede presentar una rotación respecto de su eje

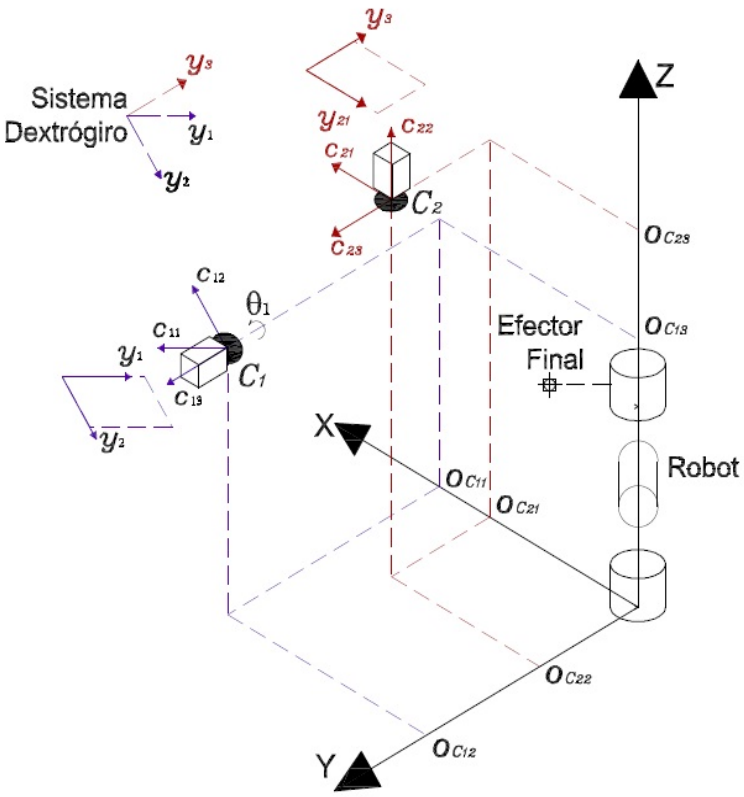


Figura 4.1: Arreglo de las cámaras relativo al robot.

óptico c_{13}) y otra denominada cámara dos cuyo eje óptico c_{22} es perpendicular al plano xy y su eje c_{23} es paralelo al eje y del robot. De esta forma, es sumamente sencillo extraer la información tridimensional del espacio de trabajo del robot ya que las coordenadas de imagen de la cámara uno (y_1, y_2) corresponden a las coordenadas (x, z) y la del eje y_3 de la cámara dos al eje y. Basta ignorar la información del eje y_{21} para tener una representación del espacio de trabajo en las coordenadas (y_1, y_2, y_3).

Finalmente, para encontrar un mapeo que lleve de las coordenadas cartesianas del espacio de trabajo del robot (x, y, z) a las coordenadas de imagen en pixeles (y_1, y_2, y_3) que emplearemos se utiliza el modelo de proyección perspectiva. A este mapeo se le llama modelo del sistema de visión y el desarrollo que se presenta a continuación para obtenerlo está basado en [9].

4.1.1. Modelo del Sistema de Visión

Cámara Uno

Para poder aplicar el modelo de proyección perspectiva, es necesario primero realizar una transformación que lleve del sistema coordenado base $Oxyz$, al sistema coordenado con origen en el centro de proyección de la cámara uno $C_1c_{11}c_{12}c_{13}$. (Ver Figura 4.2.)

Así, un punto X_R con coordenadas $\mathbf{x}_R^O = [x, y, z]^T$ en el sistema base, tendría coordenadas del sistema del centro de proyeccion de la cámara uno C_1 dadas por:

$$X_R^{C_1} = \begin{bmatrix} x_{c_{11}} \\ y_{c_{12}} \\ z_{c_{13}} \end{bmatrix} = R_{C_1}^O \begin{bmatrix} x - o_{c_{11}} \\ y - o_{c_{12}} \\ z - o_{c_{13}} \end{bmatrix}$$

dónde:

$$R_{C_1}^O = \begin{bmatrix} cos\,\theta_1 & 0 & -sin\,\theta_1 \\ sin\,\theta_1 & 0 & cos\,\theta_1 \\ 0 & 1 & 0 \end{bmatrix}$$

Una vez que se cuenta con el vector de posición de cualquier punto expresado en coordenadas del sistema de proyección y dado que el plano de proyección es paralelo al plano xy del sistema de referencia base, solo es necesario aplicar los factores derivados de la proyección perspectiva para conocer los valores de $[y_{11}, y_{12}]^T$ que corresponden a $[x, z]^T$.

Primeramente es necesario dividir las componentes $x_{c_{11}}$ y $z_{c_{12}}$ entre la distancia entre el punto de proyección y el punto proyectado en dirección de c_{12} que es

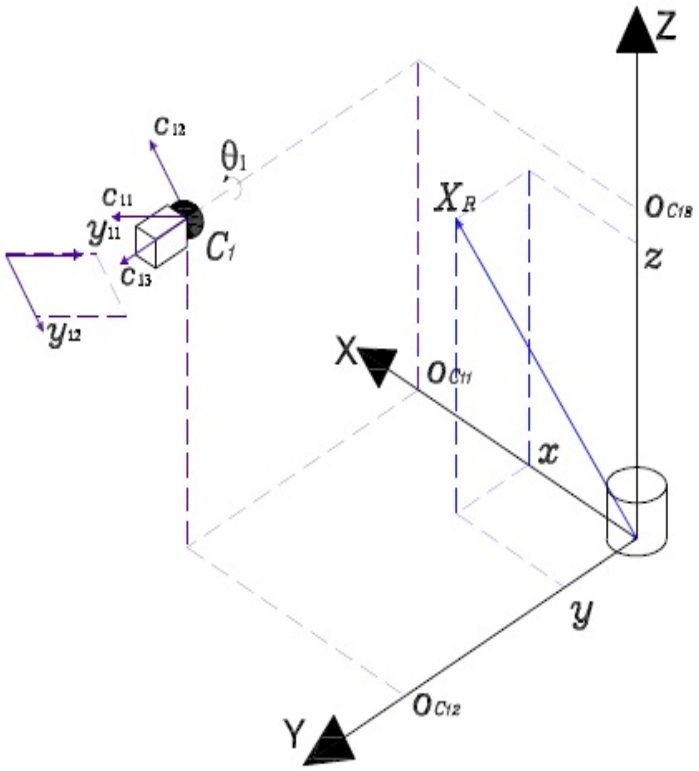


Figura 4.2: Relación entre el sistema coordenado base del robot y el sistema cooredenado con origen en el centro de proyección.

$\lambda_1 + x_{c_{13}}$. También multiplicarlas por el factor de conversión de metros a pixeles α_1[1]. En seguida multiplicar por la distancia al CCD[2] de la cámara, que asumiendo que el punto proyectado está bien enfocado y que la distancia focal de la lente es λ_1, es precisamente λ_1. Suponiendo las lentes de las cámaras como lentes ideales, se ignoran los coeficientes de corrección por distorsión radial y entonces por último, hay que sumar un desplazamiento $[u_{01}, v_{01}]^T$ que considera que el origen del sistema asociado al CCD no esta centrado sino normalmente en una esquina. Un desarrollo detallado del modelo de proyección perspectiva, explicación de sus parámetros y cómo obtenerlos puede encontrarse en [4]. En términos de las componentes del sistema base esto es:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_{11} \\ y_{12} \end{bmatrix} = \alpha_{\lambda_1} \begin{bmatrix} cos\,\theta_1 & -sin\,\theta_1 \\ sin\,\theta_1 & cos\,\theta_1 \end{bmatrix} \begin{bmatrix} x - o_{c_{11}} \\ z - o_{c_{13}} \end{bmatrix} + \begin{bmatrix} u_{01} \\ v_{01} \end{bmatrix} \tag{4.1}$$

dónde

$$\alpha_{\lambda_1} = \frac{\alpha_1 \lambda_1}{\lambda_1 + y - o_{c_{12}}} \tag{4.2}$$

Este modelo es válido si el punto X_R se encuentra frente a la cámara, más allá del foco ($x_{c_{13}} < 0$ y $|x_{c_{13}}| > \lambda_1$) y está bien enfocado. Así mismo, en este modelo, la distancia focal efectiva $\lambda_1/(\lambda_1 + x_{c_{13}})$ es negativa.

Cámara Dos

En una manera completamente análoga, se puede encontrar la relación entre las coordenadas de imagen de cualquier punto en el campo de visión de la cámara dos, colocada por encima del robot (Figura 4.1) y el sistema coordenado base del robot. Como ya se mencionó, tiene ejes paralelos a los del sistema base.

Observando la Figura 4.3 es fácil ver que la transformación de coordenadas de la base a coordenadas del punto de proyección son:

$$X_R^{C_2} = \begin{bmatrix} x_{c_{21}} \\ y_{c_{22}} \\ z_{c_{23}} \end{bmatrix} = R_{C_2}^O \begin{bmatrix} x - o_{c_{21}} \\ y - o_{c_{22}} \\ z - o_{c_{23}} \end{bmatrix}$$

[1]En general, es distinto para la dirección vertical y la horizontal, pero en este caso consideramos pixeles cuadrados para simplificar y tener un solo factor.

[2]En el caso de las cámaras digitales como las que se utilizaron, este sensor llamado charge-coupled device (CCD) es el lugar donde se proyecta la imagen.

dónde:

$$R_{C_1}^{O} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

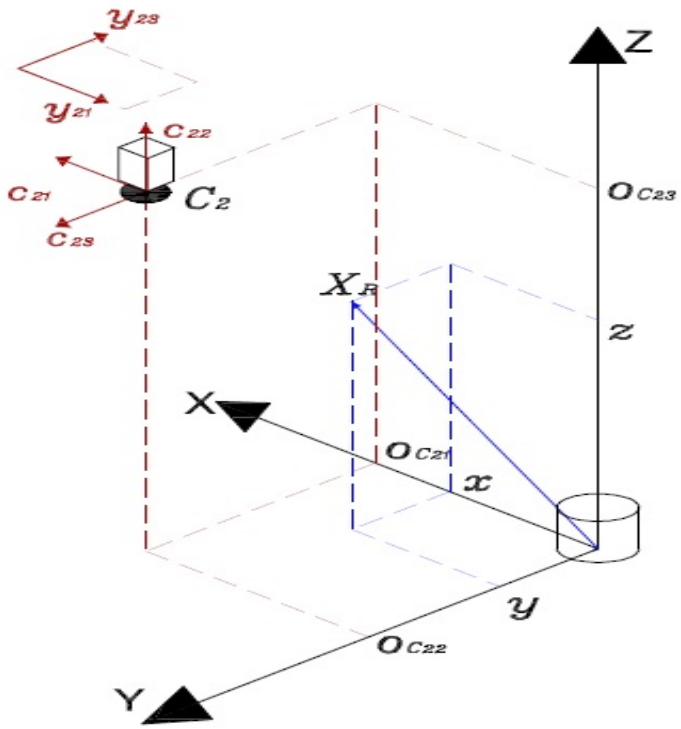


Figura 4.3: Relación entre el sistema coordenado base del robot y el sistema coordenado con origen en el centro de proyección de la cámara dos.

Del mismo modo que con la cámara uno, hecha esta transformación, sólo falta aplicar los factores procedentes de la proyección perspectiva para obtener coordenadas del sistema fijo al plano de proyección, el CCD en las cámaras digitales. Esto es, multiplicar la cantidad de pixeles por metro del sensor de la cámara dos α_2, por la distancia del punto de proyección al plano de proyección, la longitud focal λ_2 de la cámara dos. Así mismo, dividir entre la distancia en dirección perpendicular al plano de proyección entre el punto de proyección y el punto del espacio de trabajo,

$\lambda_2 + z - o_{c_{23}}$. Y por último, agregar el desfasamiento entre el origen del sistema del punto de proyección y el del sistema anclado al plano de proyección. De esta forma tenemos:

$$\begin{bmatrix} y_{21} \\ y_3 \end{bmatrix} = \begin{bmatrix} y_{21} \\ y_{23} \end{bmatrix} = \alpha_{\lambda_2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x - o_{c_{21}} \\ y - o_{c_{22}} \end{bmatrix} + \begin{bmatrix} v_{02} \\ u_{02} \end{bmatrix} \tag{4.3}$$

con

$$\alpha_{\lambda_2} = \frac{\alpha_2 \lambda_2}{\lambda_2 + z - o_{c_{23}}} \tag{4.4}$$

Combinación de los modelos

Como se había planteado originalmente, el objetivo es conformar un sistema dextrógiro, en tres dimensiones, en coordenadas de imagen y tener una transformación que lo relacione con las coordenadas del sistema base del robot. Para esto, solo hace falta combinar las ecuaciones 4.1 y 4.3. En la Figura 4.4 se puede ver que los ejes y_{11}, y_{12} y y_{23} cumplen las condiciones de orientación para formar un sistema coordenado dextrógiro y a pesar de no contar con un origen común o la misma escala, son útiles para el diseño de la ley de control con solo hacer algunas consideraciones para compensar esas diferencias.

De esta forma, el modelo combinado que relaciona al sistema coordenado base del robot, $Oxyz$ y al sistema de coordenadas de imagen en pixeles $Yy_1y_2y_3$ es:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \alpha_{\lambda_1} c_{\theta_1} & 0 & -\alpha_{\lambda_1} s_{\theta_1} \\ \alpha_{\lambda_1} s_{\theta_1} & 0 & \alpha_{\lambda_1} c_{\theta_1} \\ 0 & \alpha_{\lambda_2} & 0 \end{bmatrix} \begin{bmatrix} x - o_{c_{11}} \\ y - o_{c_{22}} \\ z - o_{c_{13}} \end{bmatrix} + \begin{bmatrix} u_{01} \\ v_{01} \\ u_{02} \end{bmatrix} \tag{4.5}$$

Por facilidad de notación, en las matrices $cos\ \theta$ se abrevia como c_θ así como $sin\ \theta$ se abrevia como s_θ. Si hacemos las siguientes definiciones,

$$R_\theta = \begin{bmatrix} c_{\theta_1} & 0 & -s_{\theta_1} \\ s_{\theta_1} & 0 & c_{\theta_1} \\ 0 & 1 & 0 \end{bmatrix} \tag{4.6}$$

$$A = \begin{bmatrix} \alpha_{\lambda_1} & 0 & 0 \\ 0 & \alpha_{\lambda_2} & 0 \\ 0 & 0 & \alpha_{\lambda_1} \end{bmatrix} \tag{4.7}$$

$$\mathbf{o_c} = \begin{bmatrix} o_{c_{11}} \\ o_{c_{22}} \\ o_{c_{13}} \end{bmatrix} \tag{4.8}$$

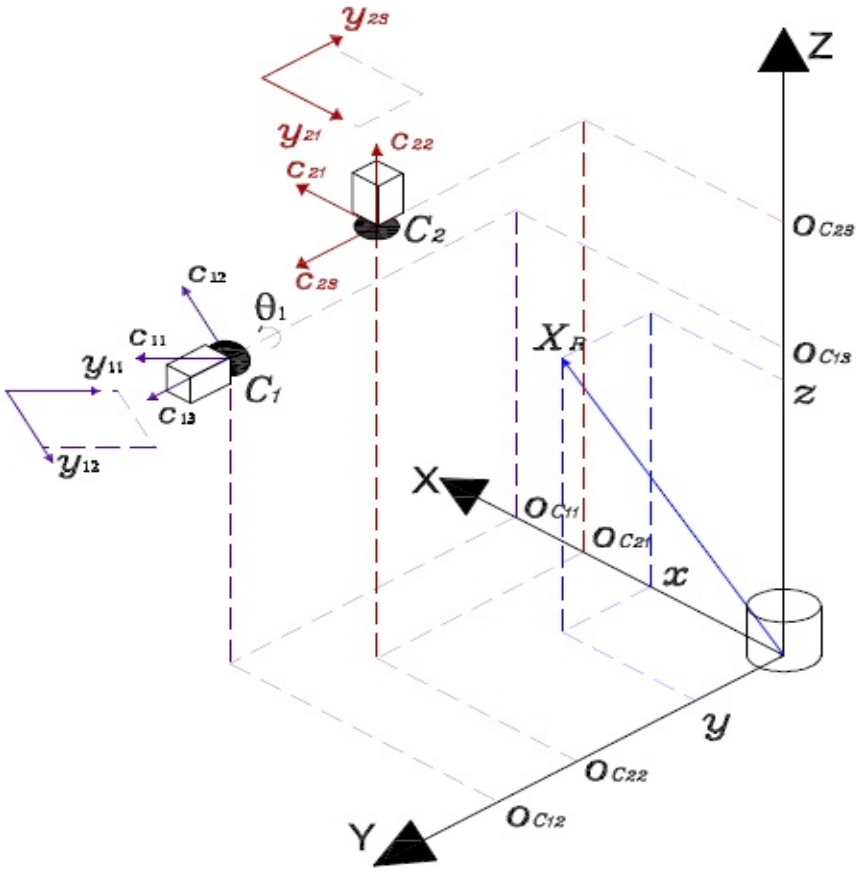


Figura 4.4: Relación entre el sistema coordenado base del robot y el sistema de coordenadas de imagen.

$$\mathbf{u_0} = \begin{bmatrix} u_{01} \\ v_{01} \\ u_{02} \end{bmatrix} \tag{4.9}$$

Sustituyendo estas definiciones, podemos reescribir la ecuación (4.5) en forma matricial como,

$$\mathbf{y} = R_\theta A(\mathbf{x_R} - \mathbf{o_c}) + \mathbf{u_0} \tag{4.10}$$

Evidentemente, la derivada temporal de $\mathbf{y}$ es:

$$\dot{\mathbf{y}} = R_\theta A\dot{\mathbf{x}}_\mathbf{R} + R_\theta \dot{A}(\mathbf{x_R} - \mathbf{o_c}) \tag{4.11}$$

donde $\dot{A}$, al ser función de $\mathbf{x_R}$, ecuaciones (4.2), (4.4) y (4.7), se calcula como:

$$\dot{A} = \begin{bmatrix} \frac{d\alpha_{\lambda_1}}{dy}\dot{y} & 0 & 0 \\ 0 & \frac{d\alpha_{\lambda_2}}{dz}\dot{z} & 0 \\ 0 & 0 & \frac{d\alpha_{\lambda_1}}{dy}\dot{y} \end{bmatrix} \tag{4.12}$$

con:

$$\frac{d\alpha_{\lambda_1}}{dy}\dot{y} = \frac{-\alpha_1\lambda_1}{(\lambda_1 + y - o_{c_{12}})^2}\dot{y} = -\frac{\alpha_{\lambda_1}^2}{\alpha_1\lambda_1}\dot{y}$$

$$\frac{d\alpha_{\lambda_2}}{dz}\dot{z} = \frac{-\alpha_2\lambda_2}{(\lambda_2 + z - o_{c_{23}})^2}\dot{z} = -\frac{\alpha_{\lambda_2}^2}{\alpha_2\lambda_2}\dot{z}$$

Si definimos

$$-\frac{\alpha_{\lambda_1}^2}{\alpha_1\lambda_1} = \alpha_1'$$

$$-\frac{\alpha_{\lambda_2}^2}{\alpha_2\lambda_2} = \alpha_2'$$

entonces,

$$\dot{A} = \begin{bmatrix} \alpha_1' \dot{y} & 0 & 0 \\ 0 & \alpha_2' \dot{z} & 0 \\ 0 & 0 & \alpha_1' \dot{y} \end{bmatrix} \tag{4.13}$$

Dado que,

$$\dot{A}(\mathbf{x_R} - \mathbf{o_c}) = \begin{bmatrix} \alpha_1'(x - o_{c_{11}})\dot{y} \\ \alpha_2'(y - o_{c_{22}})\dot{z} \\ \alpha_1'(z - o_{c_{13}})\dot{y} \end{bmatrix} = \begin{bmatrix} 0 & \alpha_1'(x - o_{c_{11}}) & 0 \\ 0 & 0 & \alpha_2'(y - o_{c_{22}}) \\ 0 & \alpha_1'(z - o_{c_{13}}) & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

y definiendo

$$\Psi = \begin{bmatrix} 0 & \alpha_1'(x - o_{c_{11}}) & 0 \\ 0 & 0 & \alpha_2'(y - o_{c_{22}}) \\ 0 & \alpha_1'(z - o_{c_{13}}) & 0 \end{bmatrix}$$

la ecuación (4.11) se puede escribir como

$$\dot{\mathbf{y}} = R_\theta A \dot{\mathbf{x}}_\mathbf{R} + R_\theta \Psi \dot{\mathbf{x}}_\mathbf{R} = R_\theta (A + \Psi) \dot{\mathbf{x}_\mathbf{R}} \tag{4.14}$$

si por último definimos

$$B = A + \Psi$$

tendremos finalmente que la ecuación que relaciona la derivada temporal de la representación en coordenadas de imagen del efector final y la de éste en el sistema base del robot es:

$$\dot{\mathbf{y}} = R_\theta B \dot{\mathbf{x}}_\mathbf{R} \tag{4.15}$$

Sabemos también que $\dot{\mathbf{x}}_\mathbf{R}$ está vinculado a la derivada de las coordenadas articulares del robot por medio del jacobiano geométrico del mismo [14]. Entonces,

$$\dot{\mathbf{x}}_\mathbf{R} = J_v \dot{\mathbf{q}}$$

y sustituyendo en la ecuación (4.15) tenemos

$$\dot{\mathbf{y}} = R_\theta B J_v \dot{\mathbf{q}} \tag{4.16}$$

Esta última ecuación (4.16) y la (4.15) dan una relación entre la velocidad en coordenadas de imagen y las misma en espacio articular y en el sistema base del robot respectivamente. Estas relaciones son útiles solo si podemos ir en sentido inverso y saber las velocidades en el espacio articular y cartesiano a partir de la velocidad en imagen. Para asegurar que esto sea posible, las matrices R_θ, B y J_v deben ser no singulares. Para esto, en el caso de J_v, es necesario garantizar que el robot no pasa por ninguna singularidad, por lo que debe mantenerse en su espacio de trabajo.
En el caso de B

$$B^{-1} = \frac{1}{det\, B} adj\, B \tag{4.17}$$

donde el determinante de B resulta ser

$$det\, B = \alpha_{\lambda_1}^2 \alpha_{\lambda_2} \left(1 - \frac{(z - o_{c_{13}})(y - o_{c_{22}})}{(\lambda_1 + y - o_{c_{12}})(\lambda_2 + z - o_{c_{23}})}\right)$$

Es fácil observar que $|\alpha_{\lambda_1}^2 \alpha_{\lambda_2}| > 0$. Por esto, mientras $(z - o_{c_{13}})(y - o_{c_{22}}) \neq (\lambda_1 + y - o_{c_{12}})(\lambda_2 + z - o_{c_{23}})$, B^{-1} existe y es distinta de cero. Un caso sería aquel en que $(z - o_{c_{13}})(y - o_{c_{22}}) = (\lambda_1 + y - o_{c_{12}})(\lambda_2 + z - o_{c_{23}})$, en dado caso bastaría modificar ligeramente la posición de alguna de las cámaras para eliminar la singularidad.
Por último, en el caso de R_θ^{-1}, ésta existe y es,

$$R_\theta^{-1} = R_\theta^T = \begin{bmatrix} c_{\theta_1} & s_{\theta_1} & 0 \\ 0 & 0 & 1 \\ -s_{\theta_1} & c_{\theta_1} & 0 \end{bmatrix} \tag{4.18}$$

4.2. Modelo Dinámico del Robot

Antes de describir el control utilizado, en está sección se presenta el modelo dinámico del robot y se mencionan algunas propiedades de éste, que son útiles para el control mencionado.
La dinámica de un brazo robótico de elementos rígidos es descrita de forma conveniente por las ecuaciones de Euler-Lagrange como se puede ver en [[18], [14]]. El modelo es:

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) = \tau \tag{4.19}$$

En donde $q \in \Re^n$ es el vector de coordenadas generalizadas articulares, $H(q) \in \Re^{n\times n}$ es la matriz simétrica positiva definida de inercias, $C(q,\dot{q})\dot{q} \in \Re$ es el vector cuyos términos cuadráticos representan las fuerzas centrifugas y cuyos términos cruzados representan las fuerzas de Coriolis, $D(q) \in \Re^{n\times n}$ es la matriz diagonal de coeficientes de fricción viscosa, $g(q) \in \Re^n$ es el vector de fuerzas gravitacionales y τ es el vector de pares actuando las articulaciones.

4.2.1. Propiedades delModelo Dinámico del Robot

El modelo dinámico presentado tiene algunas propiedades que facilitan el desarrollo de algoritmos de control. A continuación se presentan las relevantes para este trabajo.

Simetría

La forma de construir la matriz $C(q,\dot{q})$ no es única, pero si se define como

$$c_{kj} = \sum_{i=1}^{n} c_{ijk}(q)\dot{q}_i$$

$$= \sum_{i=1}^{n} \frac{1}{2}\left\{ \frac{\partial h_{kj}}{\partial q_j} + \frac{\partial h_{ki}}{\partial q_j} - \frac{\partial h_{ij}}{\partial q_k} \right\} \dot{q}_i$$

entonces, la matriz $\dot{D}(q) - 2C(q, \dot{q})$ es antisimétrica.

Límites de la Matriz de Inercia

Como ya se mencionó antes, la matriz de inercia de un robot de n eslabones rígidos es simétrica y positiva definida. Para un valor dado de las coordenadas generalizadas q, sean $0 < \lambda_1(q) \leq \cdots \leq \lambda_n(q)$ los n valores propios de $H(q)$. Al ser $H(q)$ positiva definida, dichos valores propios son positivos. Es fácil ver entonces que

$$\lambda_1(q) I_{n\times n} \leq H(q) \leq \lambda_n(q) I_{n\times n} \tag{4.20}$$

Donde I_n es la matriz identidad de $n \times n$.

Además, si todas las articulaciones son de revolución, la matriz de inercia contiene sólo términos con senos y cosenos y por lo tanto es una función acotada en términos de las coordenadas generalizadas. Por esto, es posible encontrar constantes λ_m y λ_M tales que

$$\lambda_m I_{n\times n} \leq H(q) \leq \lambda_M I_{n\times n} \leq \infty \tag{4.21}$$

Linealidad Respecto a los Parámetros Dinámicos

Las ecuaciones de movimiento de un robot son función de parámetros como las masas de los eslabones, los momentos de inercia, etc., que se deben determinar para cada robot en particular. Sin embargo, existe una función $Y(q, \dot{q}, \ddot{q}) \in \Re^{n\times p}$ y un vector $\varphi \in \Re^p$ tal que

$$H(q)\ddot{q} + C(q, \dot{q})\dot{q} + D\dot{q} + g(q) = \tau = Y(q, \dot{q}, \ddot{q})\varphi \tag{4.22}$$

A la función $Y(q, \dot{q}, \ddot{q})$ se le llama regresor y a φ, vector de parámetros. El número p de parámetros necesarios para describir la dinámica en esta manera no es único, depende de cómo se seleccionen. Por otro lado, n es el número de grados de libertad del manipulador.

4.3. Control Servovisual en 3D con Observador

El control utilizado es una adaptación para control servovisual del control que se presenta en [13]. Como no se dispone de mediciones de velocidad, una estimación de y está dada por $\hat{y}$ y el error de observación es

$$\mathbf{z} = \mathbf{y} - \hat{\mathbf{y}} \tag{4.23}$$

y se define una variable auxiliar

$$\dot{\mathbf{y}}_\mathbf{o} = \dot{\hat{\mathbf{y}}} - \Lambda_z \mathbf{z} \tag{4.24}$$

donde $\Lambda_z \in \Re^{3\times 3}$ es una matriz diagonal positiva definida.

A continuación se propone el observador lineal

$$\dot{\hat{\mathbf{y}}} = \dot{\hat{\mathbf{y}}}_\mathbf{o} + \Lambda_z \mathbf{z} + k_d \mathbf{z} \quad \dot{\hat{\mathbf{y}}}(0) = 0 \tag{4.25}$$

$$\dot{\hat{\mathbf{y}}}_\mathbf{o} = \dot{\mathbf{y}}_d - \Lambda_y(\hat{\mathbf{y}} - \mathbf{y_d}) + \mathbf{s_d} + k_d \Lambda_z \int_0^t \mathbf{z}(\vartheta) d\vartheta \tag{4.26}$$

donde $\mathbf{y}_d$ es la trayectoria que se desea seguir con primera y segunda derivadas acotadas y elegida de forma que no pase por ninguna singularidad, $\Lambda_y \in \Re^{3\times 3}$ es una matriz positiva definida, k_d es una constante positiva y $\mathbf{s}_d \in \Re^3$ se define más adelante. Una explicación más detallada del significado de $\mathbf{s}_d$ se encuentra en [13].

El error de seguimiento en coordenadas de imagen está dado por

$$\Delta\mathbf{y} = \mathbf{y} - \mathbf{y}_d \tag{4.27}$$

Para diseñar el controlador de seguimiento de trayectoria, se define

$$\dot{\mathbf{y}}_r = \dot{\mathbf{y}}_d - \Lambda_y(\hat{\mathbf{y}} - \mathbf{y}_d) + \mathbf{s}_d - K_\gamma \sigma, \tag{4.28}$$

donde $K_\gamma \in \Re^{3\times 3}$ es una matriz diagonal positiva definida y $\sigma \in \Re^3$, con

$$\mathbf{s} = \dot{\hat{\mathbf{y}}} - \dot{\mathbf{y}}_d + \Lambda_y(\hat{\mathbf{y}} - \mathbf{y}_d) = \dot{\hat{\mathbf{y}}} + \Lambda_y \bar{\mathbf{y}}, \tag{4.29}$$

$$\mathbf{s}_1 = \mathbf{s} - \mathbf{s}_d, \tag{4.30}$$

$$\mathbf{s}_d = \mathbf{s}(0)e^{-kt}, \tag{4.31}$$

$$\sigma = \int_0^t \{K_\beta \mathbf{s}_1(\vartheta) + sign(\mathbf{s}_1(\vartheta))\}d\vartheta \qquad \sigma(0) = \mathbf{0}, \tag{4.32}$$

donde k es una constante positiva, $K_\beta \in \Re^{3\times 3}$ es una matriz diagonal positiva definida y

$$sign(\mathbf{s}_1) = [sign(s_{11}) \cdots sign(s_{1n})]^T, \tag{4.33}$$

con los elementos s_{1i} de $\mathbf{s}_1$ e $i = 1, 2, \cdots, n$. Alternativamente a (4.32), se puede utilizar

$$\dot{\sigma} = K_\beta \mathbf{s}_1 + sign(\mathbf{s}_1). \tag{4.34}$$

Finalmente, se define

$$\mathbf{s}_o = \dot{\mathbf{y}}_o - \dot{\mathbf{y}}_r. \tag{4.35}$$

Puede verse que la única información necesaria para el observador es el vector $\mathbf{q}$ [3] y que el resto de las variables, se pueden calcular en el orden mostrado. La ley de control está dada por

$$\tau = -K_p J_v^T(\mathbf{q}) \tilde{R}_\theta^T \mathbf{s}_o, \tag{4.36}$$

donde $K_p \in \Re^{3\times 3}$ es una matriz diagonal positiva definida y $\tilde{R}_\theta^T$ es una matriz de rotación que se define igual que en 4.18 pero con un ángulo θ' que es el valor nominal del ángulo θ.

[3] A diferencia de [13], las posiciones articulares sólo se necesitan para el cálculo del Jacobiano ya que no es necesario utilizar la cinemática directa para encontrar la posición del efector final, ésta se encuentra por medio de las cámaras.

4.3.1. Prueba de Estabilidad y Convergencia del Controlador y el Observador

A continuación se presenta el análisis de estabilidad y convergencia a cero de los errores del control y el observador propuestos previamente. Un análisis equivalente para el control de seguimiento en el que se basa este trabajo puede encontrarse en [13]. Así mismo, un análisis completo para la adaptación aquí presentada se puede encontrar en [11] y [12].

Se busca demostrar que considerando una trayectoria acotada y continua y_d, con primera y segunda derivada acotadas y que no pasa por ninguna singularidad del robot, para la ley de control de 4.36, siempre existe una combinación apropiada de ganancias k, k_d, Λ_x, Λ_z, K_β, K_γ y K_p tales que el error de seguimiento y los errores de observación $(\Delta\dot{x}, \Delta x, \dot{z}, z)$ son acotados y tienden a cero.

Para dicha demostración es necesario primeramente definir las siguientes variables

$$\mathbf{r} = \dot{\mathbf{y}} - \dot{\mathbf{y}}_o \tag{4.37}$$

$$\dot{\mathbf{q}}_r = J^{-1}(\mathbf{q})B^{-1}R_\theta^{-1}\dot{\mathbf{y}}_r \tag{4.38}$$

$$\mathbf{s}_r = \dot{\mathbf{q}} - \dot{\mathbf{q}}_r \tag{4.39}$$

$$\mathbf{s}_y = \dot{\mathbf{y}} - \dot{\mathbf{y}}_r. \tag{4.40}$$

A partir de las ecuaciones (4.38) a (4.40), $\mathbf{s}_r$ y $\mathbf{s}_y$ se relacionan por

$$\mathbf{s}_r = J^{-1}(\mathbf{q})B^{-1}R_\theta^{-1}\mathbf{s}_y, \tag{4.41}$$

ó

$$\mathbf{s}_y = R_\theta B J(\mathbf{q})\mathbf{s}_r. \tag{4.42}$$

Ahora es posible reescribir la ecuación (4.19) como

$$H(\mathbf{q})(\dot{\mathbf{s}}_r + \ddot{\mathbf{q}}_r) + C(\mathbf{q}, \dot{\mathbf{q}})(\mathbf{s}_r + \dot{\mathbf{q}}_r) + D(\mathbf{s}_r + \dot{\mathbf{q}}_r) + g(\mathbf{q}) = \tau - \tau_p, \tag{4.43}$$

donde τ_p es una perturbación en la que se consideran efectos dinámicos no modelados. Agrupando términos tenemos

$$H(\mathbf{q})\dot{\mathbf{s}}_r + C(\mathbf{q},\dot{\mathbf{q}})\mathbf{s}_r + D\mathbf{s}_r = \tau - \mathbf{y}_a \tag{4.44}$$

donde utilizando la propiedad de linealidad respecto a los parámetros dinámicos presentada en 4.22

$$\mathbf{y}_a = H(\mathbf{q})\ddot{\mathbf{q}}_r + C(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_r + D\dot{\mathbf{q}}_r + g(\mathbf{q}) + \tau_p. \tag{4.45}$$

Por otro lado, hay que aplicar la siguiente manipulación a la ley de control de la ecuación (4.36),

$$\tau = -K_p J_v^T(\mathbf{q})\tilde{R}_\theta^T \underbrace{R_\theta B J(\mathbf{q}) J^{-1}(\mathbf{q}) B^{-1} R_\theta^{-1}}\mathbf{s}_o \tag{4.46}$$

ó

$$\tau = -\bar{K}_p J^{-1}(\mathbf{q}) B^{-1} R_\theta^{-1}\mathbf{s}_o \tag{4.47}$$

con

$$\bar{K}_p = K_p J^T(\mathbf{q}) E J(\mathbf{q}) \tag{4.48}$$

$$E = R_{\tilde{\theta}} B \tag{4.49}$$

$$R_{\tilde{\theta}} = \tilde{R}_\theta^T R_\theta = \begin{bmatrix} c_{\tilde{\theta}} & 0 & -s_{\tilde{\theta}} \\ 0 & 1 & 0 \\ s_{\tilde{\theta}} & 0 & c_{\tilde{\theta}} \end{bmatrix} \tag{4.50}$$

Utilizando las ecuaciones (4.35), (4.40), (4.37) y (4.42), $\mathbf{s}_o$ se escribe como

$$\mathbf{s}_o = \dot{y}_o - \dot{y}_r \tag{4.51}$$

$$= \dot{y}_o - (\dot{y} - s_y) \tag{4.52}$$

$$= s_y - r \tag{4.53}$$

$$\mathbf{s}_o = R_\theta B J(\mathbf{q})\mathbf{s}_r - r \tag{4.54}$$

Sustituyendo (4.54) en (4.47) y tomando en cuenta (4.44) tenemos

$$H(\mathbf{q})\dot{\mathbf{s}}_r + C(\mathbf{q},\dot{\mathbf{q}})\mathbf{s}_r + D\mathbf{s}_r = -\bar{K}_p\mathbf{s}_r + K_p J^T(\mathbf{q})\tilde{R}_\theta^T\mathbf{r} - \mathbf{y}_a.$$

Finalmente, definiendo $K_{DP} = D + \bar{K}_p$ se obtiene la ecuación en lazo cerrado

$$H(\mathbf{q})\dot{\mathbf{s}}_r + C(\mathbf{q}, \dot{\mathbf{q}})\mathbf{s}_r + K_{DP}\mathbf{s}_r = K_p J^T(\mathbf{q})\tilde{R}_\theta^T \mathbf{r} - \mathbf{y}_a. \tag{4.55}$$

Ahora, para calcular la dinámica del error de observación, se sustituye $\dot{\hat{\mathbf{y}}}_o$ ecuación(4.26) en la ecuación (4.25)

$$\dot{\hat{\mathbf{y}}} = \dot{\mathbf{y}}_d - \Lambda_y(\hat{\mathbf{y}} - \mathbf{y}_d) + \mathbf{s}_d + k_d\Lambda_z \int_0^t \mathbf{z}(\vartheta)d\vartheta + \vartheta_z\mathbf{z} + k_d\mathbf{z}, \tag{4.56}$$

Sumando $\dot{\mathbf{y}}$ en ambos lados de la ecuación y reordenando términos obtenemos

$$\dot{\mathbf{y}} - \dot{\hat{\mathbf{y}}} + \Lambda_z\mathbf{z} + k_d\left(\mathbf{z} + \Lambda_z\int_0^t \mathbf{z}(\vartheta)d\vartheta\right) = \dot{\mathbf{y}} - \dot{\mathbf{y}}_d + \Lambda_y(\hat{\mathbf{y}} - \mathbf{y}_d) - \mathbf{s}_d$$

para, utilizando (4.27), (4.29) y (4.37), obtener

$$\mathbf{r} + k_d\int_0^t \mathbf{r}d\vartheta = \Delta\dot{\mathbf{y}} + \Lambda_y\bar{\mathbf{y}} - \mathbf{s}_d$$

Tomando en cuenta (4.31) y derivando, se puede

$$\dot{\mathbf{r}} + k_d\mathbf{r} = \Delta\ddot{\mathbf{y}} + \Lambda_y\dot{\hat{\mathbf{y}}} + k\mathbf{s}_d \tag{4.57}$$

Considerando las ecuaciones de error de seguimiento (4.55) y de error de observación 4.57 se define el vector de estado

$$\omega = \begin{bmatrix} \mathbf{s}_r \\ \mathbf{r} \end{bmatrix} \tag{4.58}$$

Finalmente, se puede demostrar que el estado ω está acotado, es decir que $\|\omega\| \leq \omega_{max}$. Así mismo, cualquier otra señal involucrada en el control servovisual permanece acotada. También es posible demostrar que los errores de seguimiento $\Delta\mathbf{y}$, $\Delta\dot{\mathbf{y}}$ y de observación $\mathbf{z}$, $\dot{\mathbf{z}}$ tienden a cero cuando el tiempo tiende a infinito. Para esto, es necesario que se cumplan las condiciones mencionadas al final de la sección (4.1.1) es decir, que el robot no pasa por ninguna singularidad y que la inversa de la matriz B existe y que la ganancia $\bar{K}_p$ sea positiva definida. Para un desarrollo detallado de estas tres demostraciones y una discusión de las condiciones se pueden consultar [11] y [12].

Capítulo 5

Resultados Experimentales

En este capítulo se presentan los resultados de una serie de pruebas realizadas al software y hardware diseñado en este proyecto y también resultados de controles servovisuales obtenidos con el sistema presentado. Primero, se presentan resultados acerca de la lectura de encoders[1] con la nueva tarjeta descrita en el Capítulo 3 así como la verificación de las operaciones que debe realizar. En seguida se presentan algunas pruebas de detección de la marca diseñada para identificación del efector final. Finalmente, se presentan resultados del control visual descrito en este trabajo y de otro control servovisual implementado utilizando la misma interfaz.

5.1. Tarjeta para Adquisición de Señales

Para probar la correcta lectura de los encoders utilizando la tarjeta fabricada durante este proyecto, se realizó una serie de movimientos de extremo a extremo de cada articulación y la cantidad de grados leídos se comparó con la cantidad nominal de grados que según el manual del robot cada articulación puede girar. En la Tabla 5.1, se resumen los datos obtenidos de esta prueba.

Con los datos obtenidos, es posible ver que la lectura de los encoders usando nuestra tarjeta es bastante aceptable. Así mismo, se verificó que es posible seleccionar el robot a utilizar, a través de la computadora.

[1]Sensores que generan señales digitales en respuesta al movimiento del brazo

Articulación	Desplazamiento Medido Promedio[°]	Desplazamiento Nominal [°]	Error Promedio [°]	Porcentaje de Error
1	348.1	350	1.9	0.54
2	182	180	2	1.13
3	223.1	220	3.1	1.41
4	374	360	14	3.89
5	213.9	210	6.1	1.85
6	368.2	360	8.2	2.27

Tabla 5.1: Lectura de encoderes para el robot A465.

5.2. Detección del Efector Final

Esta sección se compone de dos partes. Primero se muestra una superposición del espacio de trabajo del robot, con el campo de visión de las cámaras. En la segunda parte se muestra el rango de detección de la marca.

En la Figura 5.1 se muestra el espacio de trabajo del robot y superpuestas están las imágenes tomadas con las cámaras en la posición descrita en el Capítulo 4. Las imágenes están puestas a la misma escala y centradas utilizando varios puntos de referencia, con el objetivo de mostrar la porción del espacio de trabajo que puede utilizarse sin que el efector final salga del área de visión de las cámaras y por lo tanto sin que se pierda el control.

Por otro lado, se llevó el robot a las posiciones extremas para mostrar la visibilidad de la marca y el rango de detección con el arreglo de cámaras utilizado. En la Figura 5.2 se puede ver el robot en la posición más lejana hacia el fondo a la que puede ir sin salir del área de visión de la cámara superior. Se puede observar claramente que aún en esa posición, la marca es suficientemente visible para la cámara uno (frontal) como para ser detectada. Si en ese extremo, se moviera el robot hacia arriba, la marca seguiría siendo visible para ambas cámaras hasta salir del rango de visión de la cámara superior.

En la Figura 5.3 se movió el robot hasta el extremo posterior e inferior del campo de visión de las cámaras. Se puede observar que la marca sólo se perderá de vista si se rota el robot en esa posición hacia la izquierda, cuando el mismo robot obstruya la visión de la cámara. Para este problema, no hay ningún cambio que se pueda hacer a la marca para prevenir ese punto ciego de la cámara uno. Por otro

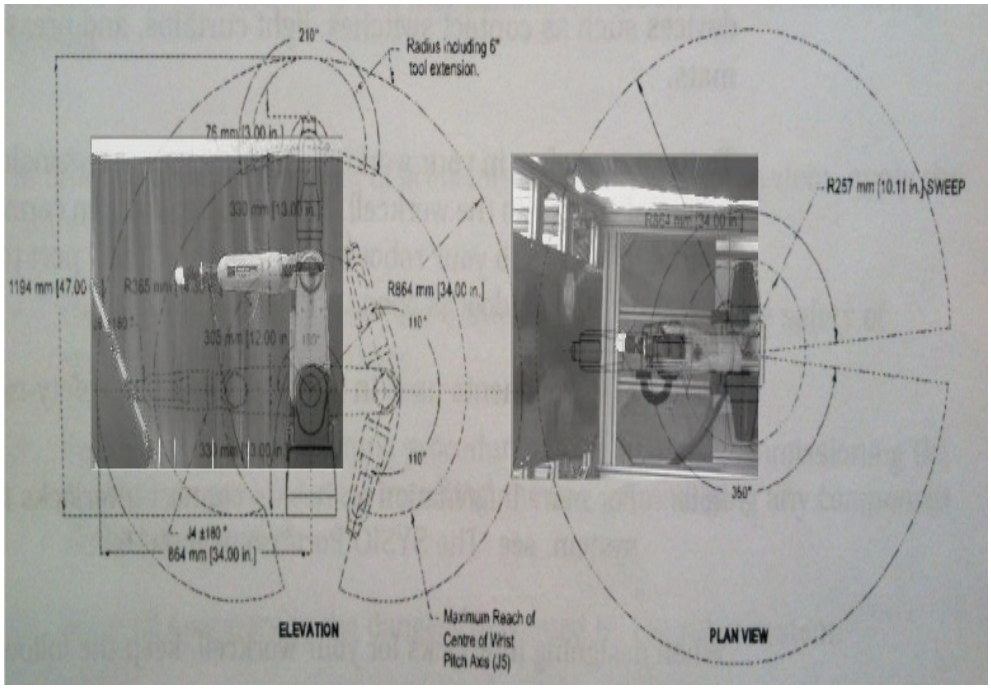


Figura 5.1: Limitación del espacio de trabajo del robot A465 por el área de visión de las cámaras.

Figura 5.2: Visión de las cámaras al llevar el efector final hacia el fondo del área de visión de las mismas.

lado, se puede observar que la cámara dos (superior) no la perderá de vista.

Figura 5.3: Visión de las cámaras al llevar el efector final hacia la parte inferior del espacio de trabajo del robot.

En la Figura 5.4 se puede observar la perspectiva de las cámaras cuando el efector final se aproxima al borde anterior del área de visión. Es claro que en esta dirección, no existe peligro de que se pierda visibilidad de la marca antes de salir de la zona abarcada por las cámaras.

Por último, en la Figura 5.5 se ve la perspectiva de las cámaras al mover el robot hacia arriba. También es muy claro que la marca solo se perderá de vista al llegar al borde de visión de las cámaras.

Con esta serie de fotos es posible notar que el tamaño y forma de la marca es adecuado para limitar lo menos posible la zona de operación del robot. De hecho, la limitación del espacio de trabajo viene esencialmente del campo de visión de las cámaras.

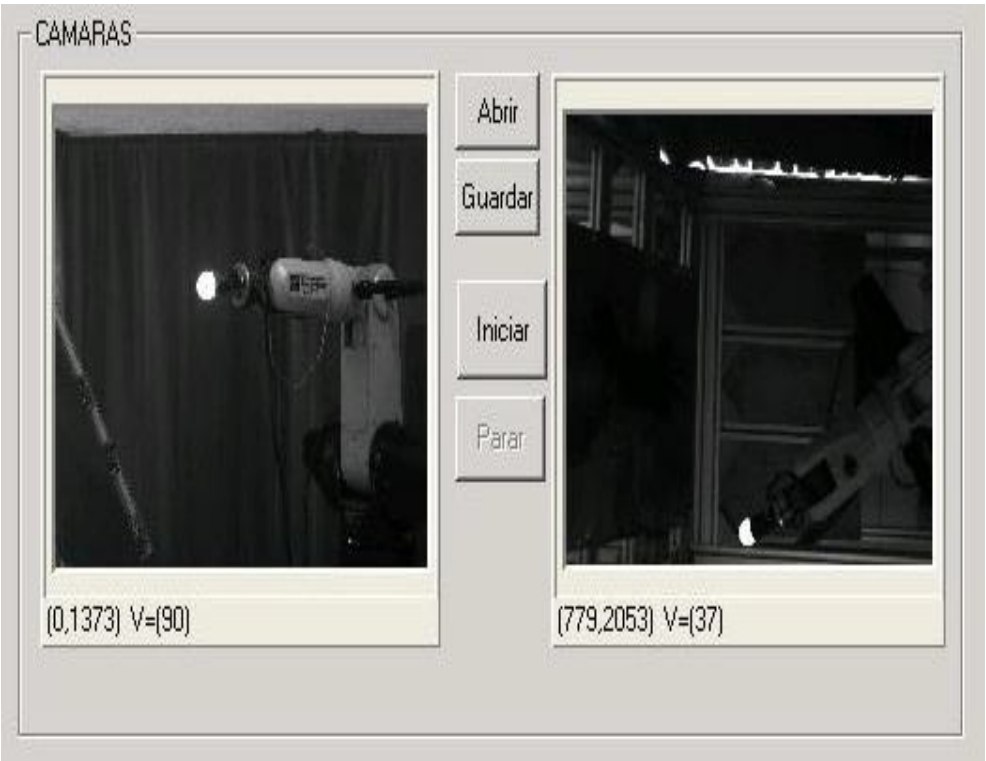


Figura 5.4: Visión de las cámaras al llevar el efector final hacia el frente del área de visión de las mismas.

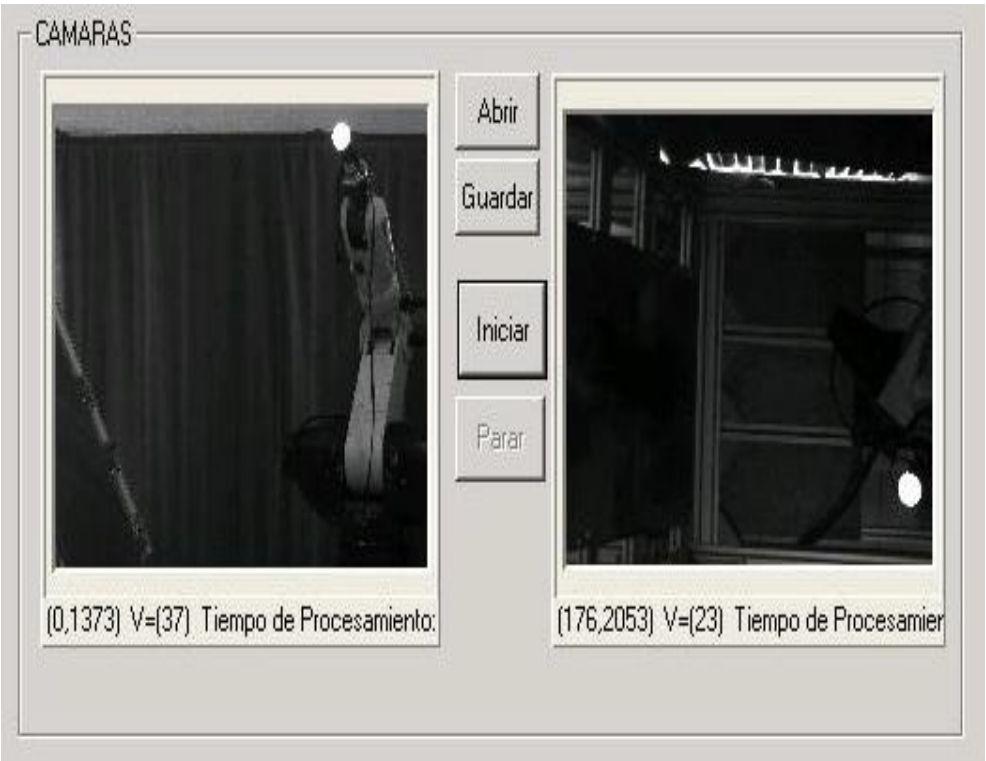


Figura 5.5: Visión de las cámaras al llevar el efector final hacia la parte superior del espacio de trabajo del robot.

5.3. Resultados del Control PID Utilizando la Interfaz Diseñada

En esta sección se presentan los resultados del control descrito en el Capítulo 4 y que se implementó utilizando la interfaz diseñada en este proyecto. El experimento consistió en llevar el efector final de su posición inicial $\mathbf{y}(0) = [y_1(0), y_2(0), y_3(0)]^T$ hasta una posición final arbitraria $\mathbf{y}_f$ siguiendo una trayectoria generada por

$$\mathbf{y}_d = \int_0^t \dot{\mathbf{y}}_d d\vartheta + \mathbf{y}(0) \tag{5.1}$$

$$\dot{\mathbf{y}}_d = -\frac{k_0}{\|\tilde{\mathbf{y}}\| + \epsilon}\tilde{\mathbf{y}} - k_1(\mathbf{y}_d - \mathbf{y}_f), \tag{5.2}$$

donde $\tilde{y} \stackrel{\triangle}{=} \mathbf{y} - \mathbf{y}_f$, $\epsilon = 0{,}1$, $k_0 = 0{,}1$ y $k_1 = 2$.Los parámetros empleados en el experimento son $k_d = 400$, $k = 0{,}1$, $\mathbf{Kp} = \mathbf{0{,}15I}$, $\mathbf{\Lambda}_z = 50\mathbf{I}$, $\mathbf{\Lambda}_y = 30\mathbf{I}$, $\mathbf{K}_\gamma = 0{,}1\mathbf{I}$ y $\mathbf{K}_\beta = 5{,}2\mathbf{I}$.

A continuación se presentan las gráficas resultantes del experimento. En las Figuras 5.6, 5.7 y 5.8 se muestran la trayectoria generada y el seguimiento de la misma para las coordenadas de imagen y_1, y_2 y y_3, respectivamente. En las figuras se puede ver que el seguimiento realizado por el robot con esté control es muy bueno. Especialmente en las coordenadas y_2 y y_3. En el caso de la primera coordenada la convergencia a cero toma más tiempo sin embargo, este efecto no es imputable a la interfaz sino al uso moderado de la potencia de los motores del robot. (Se pueden ver más detalles y gráficas sobre los voltajes en [12].) De hecho, en las Figuras 5.9, 5.10 y 5.11 se puede ver el error de seguimiento para cada una de las coordenadas y cómo éste converge a cero en los tres casos. Si el error fuera producto de una mala detección del efector final o de una mala lectura de los encoders, se esperarían picos aleatorios en el error. La congruencia en el comportamiento del error hace ver que el ruido en las señales de los encoders es moderado y que la detección el efector es continua. También se presentan las gráficas del error de observación en las que se pude ver que el desempeño del observador es bastante bueno.

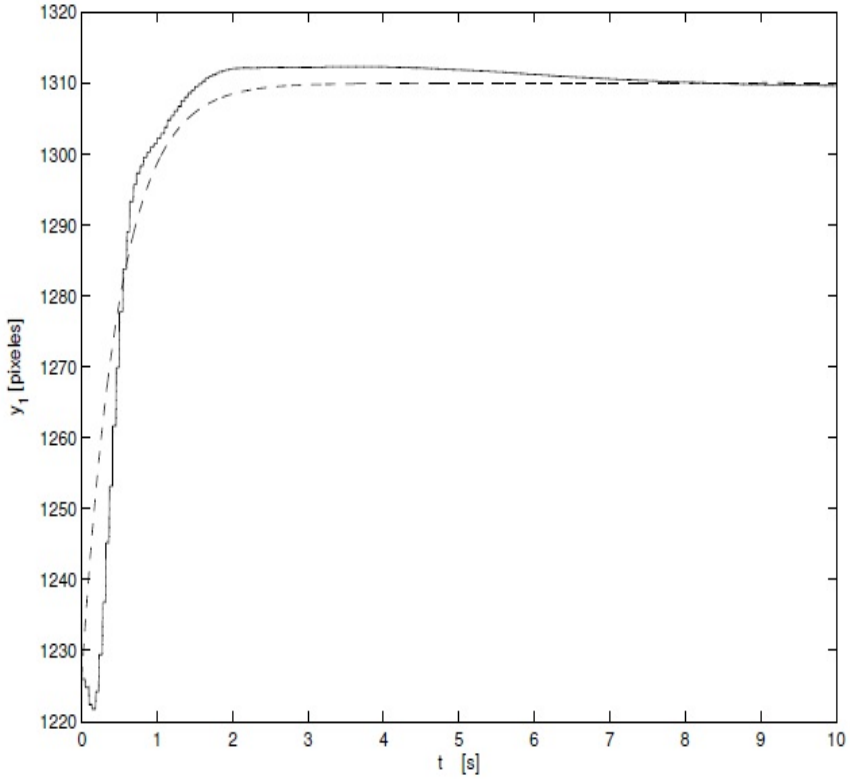


Figura 5.6: Seguimiento de trayectoria de la primera coordenada. y_1 (–), $\dot{y_{d1}}$ (- - -).

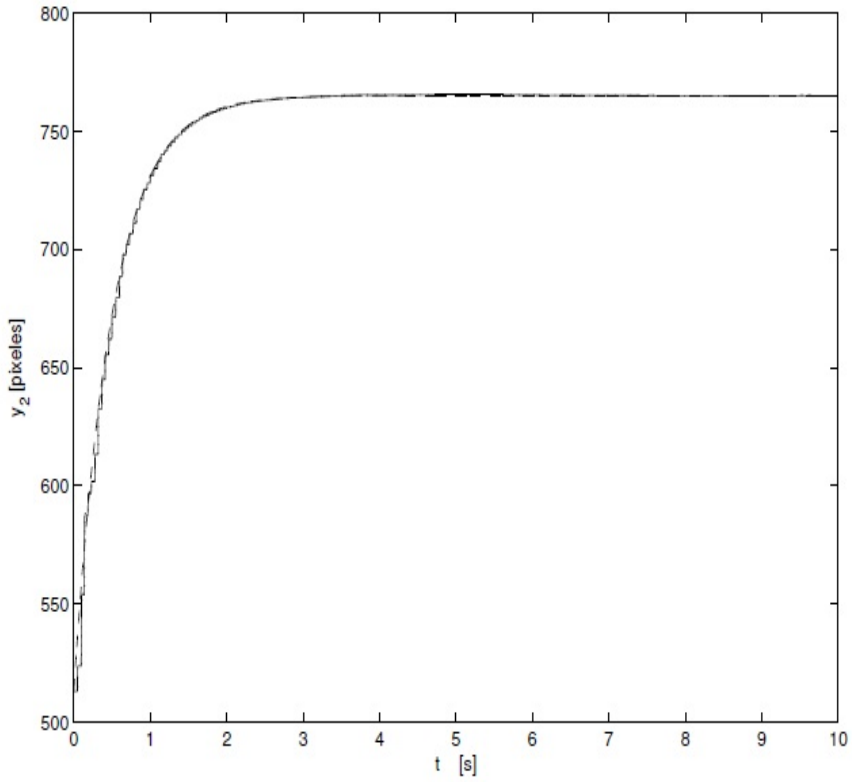


Figura 5.7: Seguimiento de trayectoria de la segunda coordenada. y_2 (–), $\dot{y_{d2}}$ (- - -).

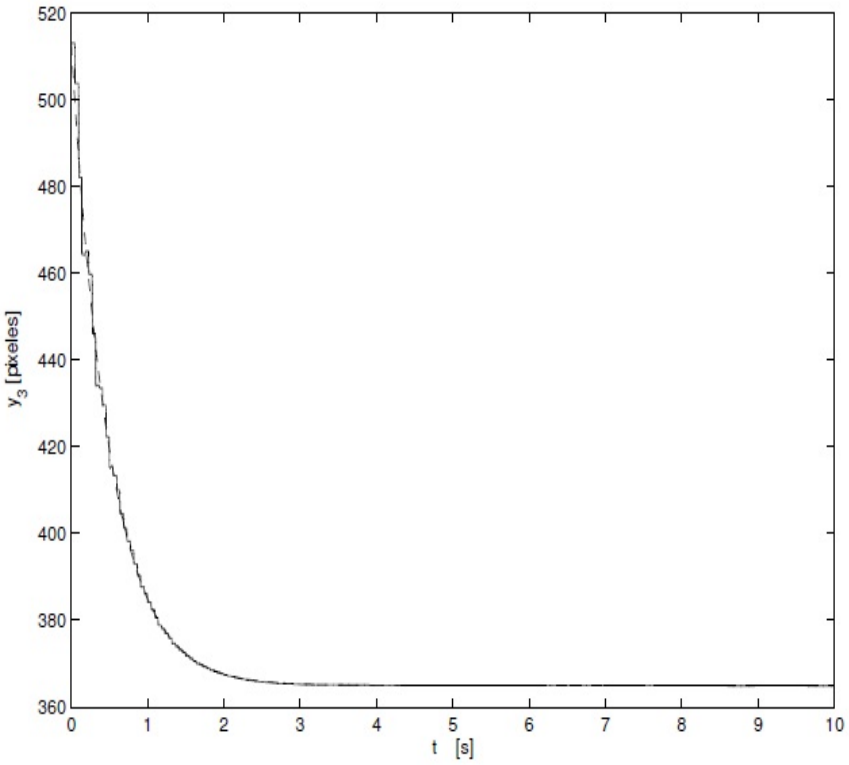


Figura 5.8: Seguimiento de trayectoria de la tercera coordenada. y_3 (–), y_{d3} (- - -).

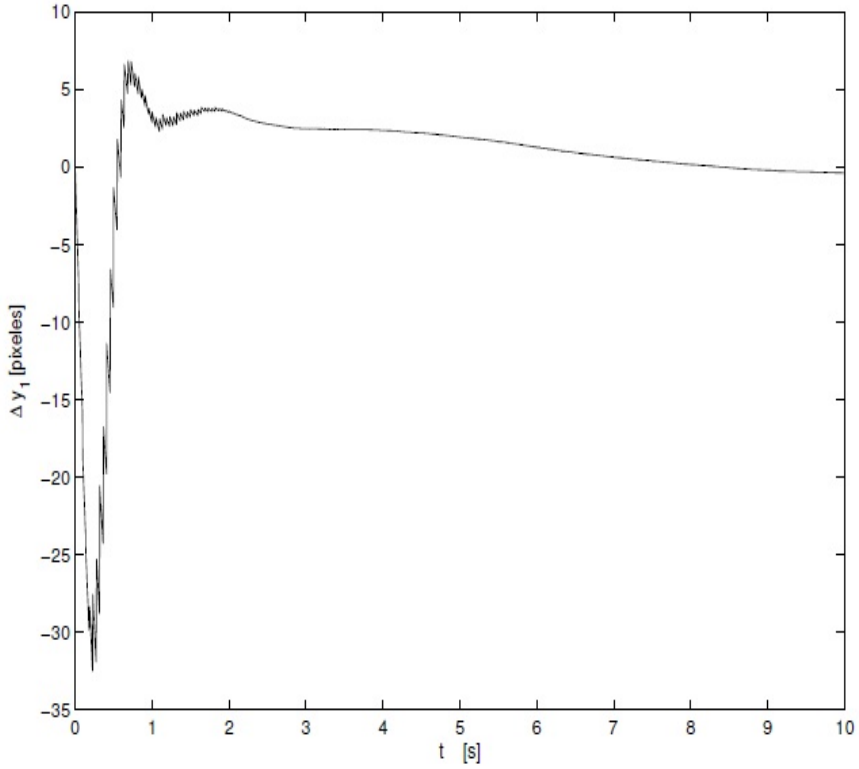


Figura 5.9: Error de seguimiento de trayectoria y_1.

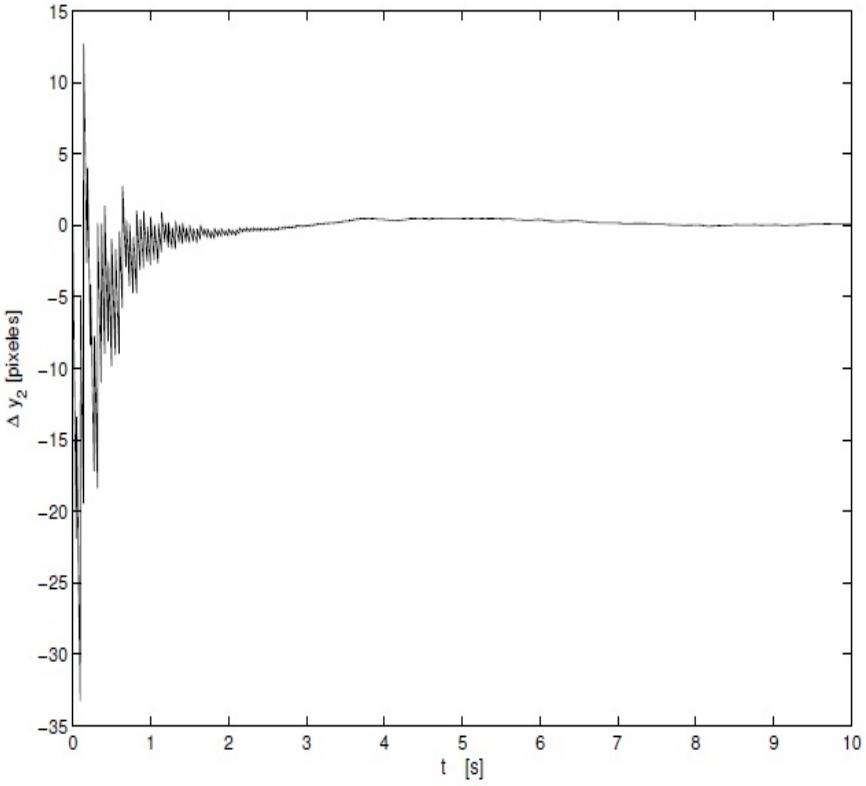


Figura 5.10: Error de seguimiento de trayectoria y_2.

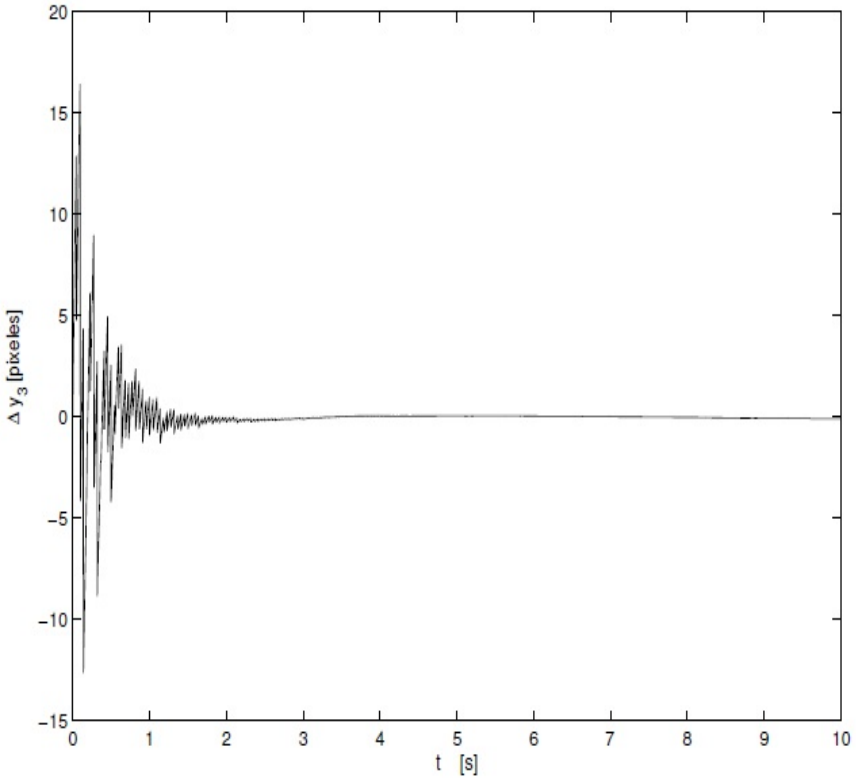


Figura 5.11: Error de seguimiento de trayectoria y_3.

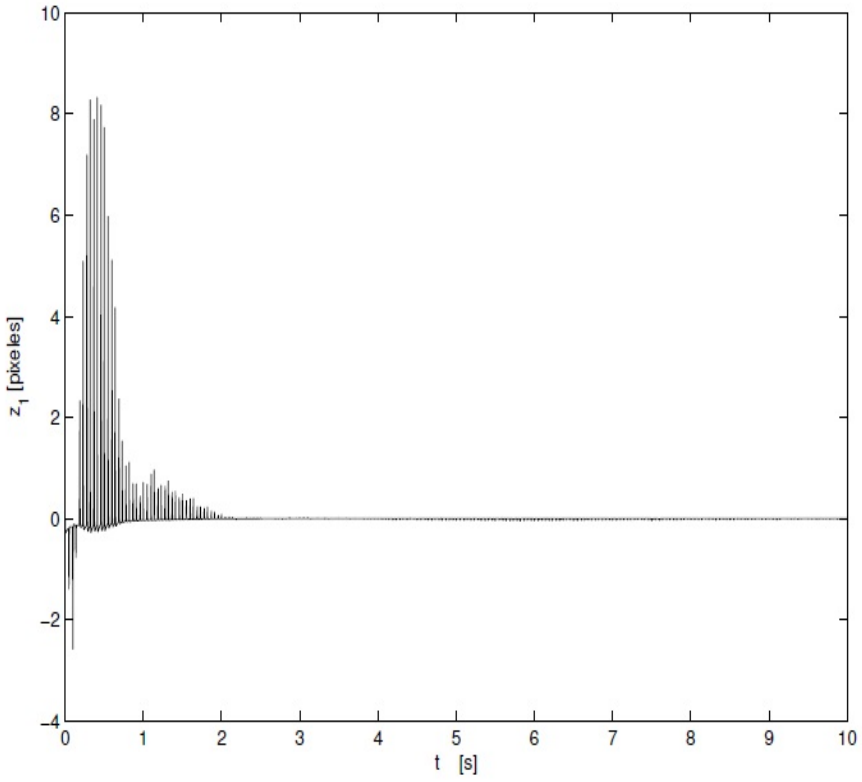


Figura 5.12: Error de observación $\dot{z_1}$.

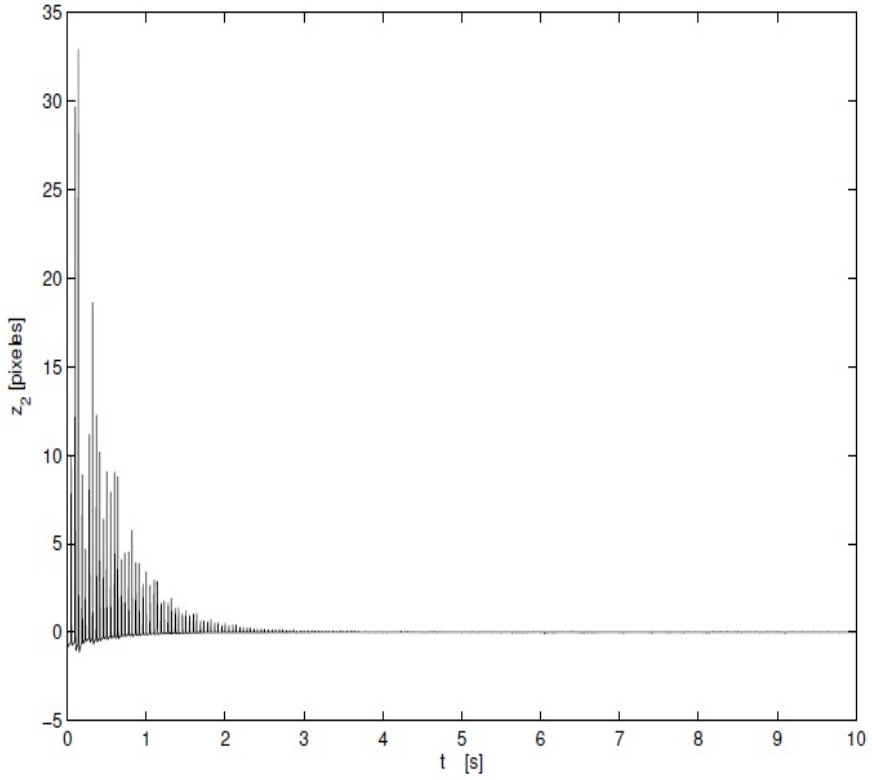


Figura 5.13: Error de observación z_2.

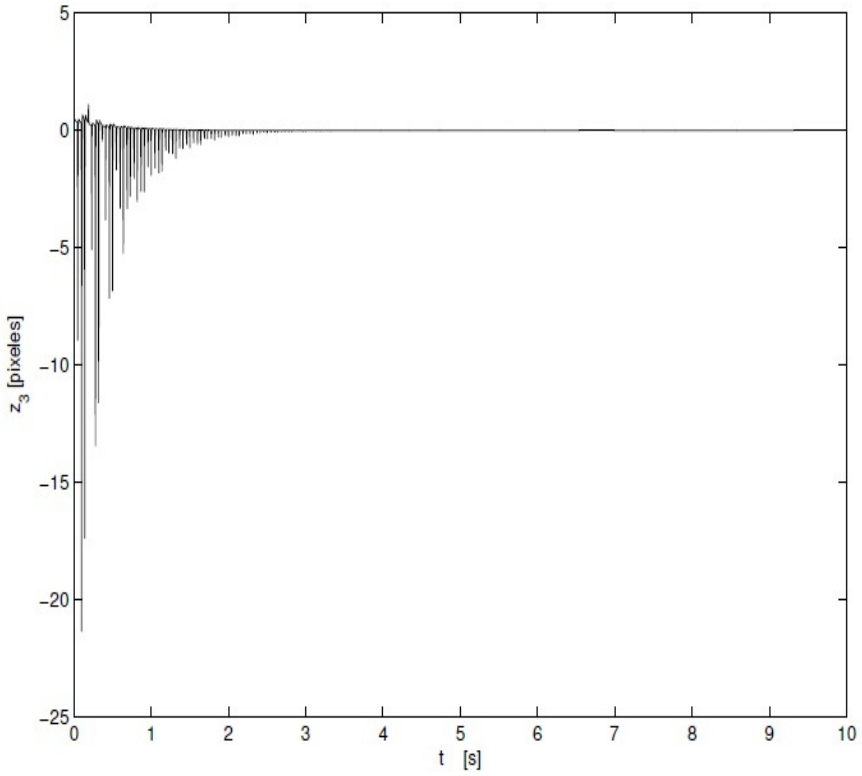


Figura 5.14: Error de observación z_3.

Capítulo 6

Conclusiones

En esta tesis se usaron los recursos disponibles y se aumentaron otros para realizar el sistema de visión. Se diseñó e implementó una interfaz para la realización de control de robots por visión, abarcando tanto el aspecto de hardware como el de software. En cuanto a hardware, se construyó una tarjeta que mejorara la adquisición de señales del robot y facilitara el uso de los dos robots con que se cuenta en el laboratorio. También se diseñó una marca, descrita en la Sección 3.2.2, de luz blanca para la correcta identificación del efector final del robot mediante cámaras blanco y negro. Por último, se buscó la colocación más adecuada para estas cámaras de forma que se tuvieran las mejores condiciones para la experimentación con controles por visión. En cuanto a software, se rediseñó completamente el programa mediante el cual se implementaron los algoritmos servovisuales. Este programa cuenta con un diseño fácil de usar y que aporta mucha información interesante acerca del control implementado. Por último, se presentó un algoritmo de control servovisual 3D en la Sección 4.3 como muestra del buen funcionamiento del sistema y como muestra de lo que es posible implementar en él.

El sistema implementado permite experimentar bastante bien controles visuales, lo cual será de gran utilidad en el laboratorio dadas las bondades de este tipo de controles. No obstante, el alcance de este proyecto es limitado dado el límite de tiempo y los recursos disponibles al momento de realizarlo y por lo tanto se identifican algunos aspectos que sería posible mejorar en el futuro. A continuación se enumeran los más importantes.

Sobre la placa: La placa actual que fue rediseñada tanto en espacio como en aumento de componentes, filtra las señales de encoders del robot A465. Los encoders del robot A255 están siendo sustituidos por encoders nuevos y por lo

tanto requieren en menor medida del filtrado[1]. Además, es capaz de realizar 16 acciones gracias al convertidor analógico-digital nada complejo pero con resultados exactos visualizado para resoluciones pequeñas como la usada. La mejora a la placa aumentó cuando se le colocó un regulador(dc-dc), teniendo menos necesidades de corriente, y resultando 5 volts exactos para aquellos componentes que así lo necesitan. Y no menos importante, el acomodo apropiado de las bases y demás componentes electrónicos que benefician la ubicación rápida a la vista del módulo que se desea usar.

Sobre la marca: Podría mejorar sustancialmente la detección aún requiriendo de mucha menor restricción en las condiciones ambientales como fuentes de luz externas, existencia de superficies reflejantes en el rango de visión de las cámaras, etc. Se identifican tres formas principales de hacerlo. La primera sería adquirir filtros ópticos pasabanda azul para las cámaras y sustituyendo los LEDs de la marca por LEDs azules. La segunda sería hacer detección por colores en cuyo caso tendrían que sustituirse las cámaras actuales por cámaras a color y la marca sustituirse por una esfera del color a detectar o simplemente detectar el color azul de la marca. Las cámaras podrían ser cámaras baratas ya que no es necesaria demasiada resolución ni velocidad, se estima que una webcam común sería suficiente aunque tendrían que hacerse los ajustes correspondientes sobre el programa tales como el cambio de la ecuación de conversión de metros a pixeles y los criterios de reconocimiento de la marca. La última forma y también la más robusta, sofisticada y por supuesto compleja, sería detectar el efector final por reconocimiento de bordes y formas utilizando algoritmos de reconocimiento de imágenes y sin necesidad de una marca.

Sobre el campo de visión: El arreglo de cámaras funciona muy bien sin embargo, sería una mejora sustancial poder abarcar todo el espacio de trabajo del robot o al menos una mayor cantidad de éste. La ubicación de las cámaras que se utilizó en este proyecto, fue la mejor que permitieron los espacios y condiciones del laboratorio en aquel momento.

Se identifican dos formas en que puede mejorarse el campo de visión. La primera sería correr las cámaras de tal forma que el plano en que se encuentran interseque a la primera articulación del robot o visto de otra forma, que incluya el centro del espacio de trabajo del robot. La principal dificultad para esto, es colocar un soporte que permita esto para la cámara superior. La segunda sería sustituir la lente de la cámara superior de tal forma que aleje la imagen y logre ampliar el campo de visión de ésta. Otra opción para lograr ampliar el campo de visión, sería rotar el

[1]Una futura versión de esta placa podría filtrar las señales de los nuevos encoders aunque no de la misma manera pues los encoders nuevos no emiten señales negadas

plano en que se encuentran las cámaras buscando un ángulo en que los espacios del laboratorio permitan mayor amplitud. En este último caso, tendrían que adaptarse las ecuaciones del controlador para reflejar este giro en el sistema de referencia de las cámaras.

Sobre la interfaz: La interfaz cumple con los requisitos de precisión en la ubicación (x, y, z) de los puntos de luz en un umbral determinado, así como los voltajes enviados a los encoders del robot para la ejecución del control. Sin embargo, siempre es factible mejorar el despliegue de información, aumentando gráficos como un histograma para mostrar las características a nivel pixel de la imagen binarizada. También se puede tener una perspectiva amplia y llevar el análisis de la imagen en un equipo independiente, enviando solamente las coordenadas por red, es decir, se podrían utilizar sockets[2]. En general, el programa está estructurado para recibir mantenimiento, para que al final la interacción con el programa sea más amigable todavía.

[2]Un socket se define como un descriptor de un canal de comunicación por medio del cual un proceso puede enviar o recibir información

Apéndice A

Código Funciones Principales Visión-Control.

```
1
2  void CTwoCamsDlg::OnTimer(UINT nnhIDEvent)
3  {
4    CTwoCamsDlg *pMainWnd = (CTwoCamsDlg *) AfxGetMainWnd();
5
6
7    if(goOn){
8
9    // ***********************************Timer referido al bloque de VISION
10     if( nIDEvent == IDT_VISION )
11     {
12         str.Empty();
13     str.Format("0.00 0.00 ");
14     str2.Empty();
15     str2.Format("0.00 0.00");
16
17     UpdateData(TRUE);
18     m_cvDisp.RemoveAllUserObjects();
19     m_cvDisp2.RemoveAllUserObjects();
20
21     if ((BlobExec((DWORD *)m_BlobInstance )) < 0){
22       MessageBox("Error de procesamiento en imagen 1 (No hay blobs)", "↩
             Advertencia", MB_OK);
23       return;
24     }
25
26     if ((BlobExec((DWORD *)m_BlobInstance2 )) < 0){
27       MessageBox("Error de procesamiento en imagen 2 (No hay blobs)", "↩
             Advertencia", MB_OK);
28       return;
29     }
30
31     double dTime;
```

```
32      BlobGetExecTime  ((DWORD *)m_BlobInstance , dTime);
33      csMsg.Format("  Tiempo de Procesamiento: %.4f ms", dTime*1000);
34      m_cvDisp.SetStatusUserText((LPCTSTR)csMsg);
35      BlobGetExecTime  ((DWORD *)m_BlobInstance2 , dTime);
36      csMsg.Format("  Tiempo de Procesamiento: %.4f ms", dTime*1000);
37      m_cvDisp2.SetStatusUserText((LPCTSTR)csMsg);
38
39      BlobGetNumBlobs((void *)m_BlobInstance, lNumObjects);
40      Final.Format("");
41      Finalb.Format("");
42      Finalc.Format("");
43
44      for (i=0; i < lNumObjects; i++){
45
46        Name.Format ("Obj %d", i);
47
48        if ((k = BlobGetCenter((void *)m_BlobInstance, i, x,  y)) < 0){
49          MessageBox("Error al llamar a la función Blob de CVB", "↩
                Advertencia", MB_OK);
50          break;
51          }
52
53        Vert[0].x = x;
54        Vert[0].y = y;
55
56        if (BlobGetBlobSize((void *)m_BlobInstance, i, Size) < 0){
57          MessageBox("Datos de imagen no válidos", "Información", MB_OK);
58          m_ResultBox1 = "No Data";
59          return;
60        }
61
62        if (BlobGetBlobSize((void *)m_BlobInstance, i, Size) < 0){
63          MessageBox("Datos de imagen no válidos", "Información", MB_OK);
64          m_ResultBox1b = "No Data";
65          return;
66        }
67        //Especificamos el área
68        if (Size > m_lowSize && Size<m_upSize){
69          TmpStr.Format("Blob Número %d, \r\n Centro: x: %.2f, y: %.2f \r\n ↩
                Tamaño: %ld \r\n\r\n", i, *GetBlobCentroid(i,m_BlobInstance), ↩
                *(GetBlobCentroid(i,m_BlobInstance)+1), Size);
70          Final += TmpStr;
71          str.Empty();
72          str.Format("%.2f %.2f ",*GetBlobCentroid(i,m_BlobInstance), *(↩
                GetBlobCentroid(i,m_BlobInstance)+1));
73        }
74        if (Size < m_lowSize){
75          TmpStr.Format("Luz Número %d, \r\n Centro: x: %.2f, y: %.2f \r\n ↩
                Tamaño: %ld \r\n\r\n", i, *GetBlobCentroid(i,m_BlobInstance), ↩
                *(GetBlobCentroid(i,m_BlobInstance)+1), Size);
76          Finalb += TmpStr;
77          str.Empty();
78          str.Format("%.2f %.2f ",*GetBlobCentroid(i,m_BlobInstance), *(↩
                GetBlobCentroid(i,m_BlobInstance)+1));
79
80          CString cdXB;
81          cdXB.Format("%ld",x);
```

```
82          pMainWnd->m_cCDXB.SetWindowText(cdXB);
83
84          CString cdYB;
85          cdYB.Format("%ld",y);
86          pMainWnd->m_cCDYB.SetWindowText(cdYB);
87
88        }
89
90
91
92      }
93
94      m_ResultBox1 = Final;
95          m_ResultBox1b = Finalb;
96
97
98      BlobGetNumBlobs((void *)m_BlobInstance2, lNumObjects);
99      Final.Format("");
100
101     for (i=0; i < lNumObjects; i++){
102
103       Name.Format ("Obj %d", i);
104
105       if ((k = BlobGetCenter((void *)m_BlobInstance2, i, x,  y)) < 0){
106         MessageBox("Error al llamar a la función Blob de CVB", "↩
              Advertencia", MB_OK);
107         break;
108         }
109
110       Vert[0].x = x;
111       Vert[0].y = y;
112
113       if (BlobGetBlobSize((void *)m_BlobInstance2, i, Size) < 0){
114         MessageBox("Datos de imagen no válidos", "Información", MB_OK);
115         m_ResultBox2 = "No Data";
116         return;
117       }
118
119       if (BlobGetBlobSize((void *)m_BlobInstance2, i, Size) < 0){
120         MessageBox("Datos de imagen no válidos", "Información", MB_OK);
121         m_ResultBox2b = "No Data";
122         return;
123       }
124
125       //Especificamos el área
126       if (Size > m_lowSize2 && Size<m_upSize2){
127         TmpStr.Format("Blob Número %d, \r\n Centro: x: %.2f, y: %.2f \r\n ↩
              Tamaño: %ld \r\n\r\n", i, *GetBlobCentroid2(i,m_BlobInstance2)↩
              , *(GetBlobCentroid2(i,m_BlobInstance2)+1), Size);
128         Final += TmpStr;
129         str2.Empty();
130         str2.Format("%.2f %.2f",*GetBlobCentroid2(i,m_BlobInstance2), *(↩
              GetBlobCentroid2(i,m_BlobInstance2)+1));
131       }
132       if (Size < m_lowSize2){
133         TmpStr.Format("Luz Número %d, \r\n Centro: x: %.2f, y: %.2f \r\n ↩
              Tamaño: %ld \r\n\r\n", i, *GetBlobCentroid2(i,m_BlobInstance2)↩
```

```
            , *(GetBlobCentroid2(i,m_BlobInstance2)+1), Size);
134         Finalc += TmpStr;
135         str2.Empty();
136         str2.Format("%.2f %.2f",*GetBlobCentroid2(i,m_BlobInstance2), *(↩
                GetBlobCentroid2(i,m_BlobInstance2)+1));
137
138
139         CString cdXC;
140         cdXC.Format("%ld",x);
141         pMainWnd->m_cCDXC.SetWindowText(cdXC);
142
143         CString cdYC;
144         cdYC.Format("%ld",y);
145         pMainWnd->m_cCDYC.SetWindowText(cdYC);
146
147       }
148
149     }
150
151     m_ResultBox2 = Final;
152     m_ResultBox2b = Finalc;
153     str = str+str2;
154
155
156     UpdateData(FALSE);
157     m_cvDisp.Refresh();
158     m_cvDisp2.Refresh();
159   }
160
161   }//CIERRE DE IF nIDEvent
162   // ***************************************
163
164
165   // *********************************Timer referido a la posición HOME ↩
          del robot
166   if( nIDEvent == IDT_HOME )
167   {
168
169     const n = 3;
170     double qi[n], qf[n], difq[n], qdes[n]={0.0}, taus[6], qtilde[n]={0.0};
171     static double iqtilde[n], qtilde_1[n];
172     double a0[n], a3[n], a4[n], a5[n];
173     double kp[n], ki[n], kd[n];
174     double dqtilde[n];
175
176     const double tf = 6.0;
177
178     CTwoCamsDlg *pMainWnd = (CTwoCamsDlg *) AfxGetMainWnd();
179
180     getArtRad();
181
182     /* Trayectoria */
183     // Posición inicial de trayectoria
184     if (iCHome){
185       for(int i=0; i<n; i++){
186         iqtilde[i] = 0.0;
187         qtilde_1[i] = 0.0;
```

```
188         }
189         qi[0] = q[0];
190         qi[1] = q[1];
191         qi[2] = q[2];
192         pMainWnd->m_cstatusText.SetWindowText(_T("*** Ejecutando Home ***"))↩
                ;
193         iCHome = false;
194       }
195       // Posición final de trayectoria
196       qf[0] = 0.0*pi/180;
197       qf[1] = 90.0*pi/180;
198       qf[2] = -90.0*pi/180;
199
200
201       for(i=0; i<n; i++){
202         difq[i] = qf[i]-qi[i];
203         a0[i] = 1.0*qi[i];
204         a5[i] = 6.0/(pow(tf,5))*difq[i];
205         a4[i] = -15.0/(pow(tf,4))*difq[i];
206         a3[i] = 10.0/(pow(tf,3))*difq[i];
207       }
208
209       if(t<tf){
210        for(i=0;i<n;i++)
211          qdes[i] = a5[i]*pow(t,5)+a4[i]*pow(t,4)+a3[i]*pow(t,3)+a0[i];
212       }
213       else
214       {
215         qdes[0] = qf[0];
216         qdes[1] = qf[1];
217         qdes[2] = qf[2];
218         if(t<tf+0.006){
219           pMainWnd->m_cstatusText.SetWindowText(_T("*** En Home ***"));
220         }
221       }
222
223       //ganancias
224       kp[0] = 400.0;
225       kp[1] = 400.0;
226       kp[2] = 600.0;
227       kd[0] = 2.0;
228       kd[1] = 2.0;
229       kd[2] = 4.0;
230       ki[0] = 8.0;
231       ki[1] = 8.0;
232       ki[2] = 12.0;
233
234       for(int i=0; i<3; i++){
235         qtilde[i]=q[i] - qdes[i];
236       }
237
238       /* Inicializacion de variables */
239       if (t<h) {
240         for(int i=0;i<n;i++)
241           qtilde_1[i] = qtilde[i];
242       }
243
```

```
244     // ESTIMACION DE VELOCICDAD
245     for(i=0;i<n;i++){
246       dqtilde[i] = (qtilde[i] - qtilde_1[i])/h;
247     }
248
249       taus[0] = -kp[0]*qtilde[0] -ki[0]*iqtilde[0] -kd[0]*dqtilde[0];
250       taus[1] = -kp[1]*qtilde[1] -ki[1]*iqtilde[1] -kd[1]*dqtilde[1];
251       taus[2] = -kp[2]*qtilde[2] -ki[2]*iqtilde[2] -kd[2]*dqtilde[2];
252
253     ////////////////////////////// Manda voltajes ///////////////
254
255     NiFpga_WriteI16(session,voltA465_art1, taus[0]*DACR);
256     NiFpga_WriteI16(session,voltA465_art2, taus[1]*DACR);
257     NiFpga_WriteI16(session,voltA465_art3, taus[2]*DACR);
258
259     ////////////// Actualizacion de variables ////////////////////
260     for(i=0;i<n;i++){
261       qtilde_1[i] = qtilde[i];
262       iqtilde[i] += qtilde[i]*h;
263     }
264     ts = timeGetTime()-t*1000-tc;
265     t = 1.0*(timeGetTime()-tc)/1000;
266
267   }//CIERRE DE IF nIDEvent
268   // ***************************************
269
270
271   // *********************************Timer referido al PUNTO INICIAL ↩
          del robot
272   if( nIDEvent == IDT_PINICIAL )
273   {
274     const int n = 3;// Numero de articulaciones
275     double c1, s1, c2, s2, c3, s3, s12, c12, c23, s123, c123;
276     // Matriz de Inercia y Jacobianos
277     double H[n][n], InvH[n][n];
278     double J[n][n], JT[n][n], Jm1[n][n], Mq[n][n], Mqi[n][n];
279     // Tiempo
280     const double tf = 8.0;
281     // Cinemática
282     double x[n], xp[n], e[n], ep[n], xf[n], difx[n];
283     static double zg2[n], zg1[n], eg2[n], eg2e[n], eg[n], x_1[n], xi[n], ↩
          e_1[n];
284     static double et[n], egp[n];
285     double xd[n], xdp[n], xdpp[n];
286     float l1 = 0.33f, l2 = 0.35f, l3 = 0.07f + ltool; //longitud de ↩
          eslabones
287     // Para la trayectoria
288     double a0[n], a3[n], a4[n], a5[n];
289
290     // Observador de velocidad
291     const float lam0 = 40960000.0f, lam1 = 2048000.0f, lam2 = 38400.0f, ↩
          lam3 = 320.0f; //polinomio de lambdas de velocidad
292
293     double eg2p[n], egfp;
294     double zg1p[n], zg2p[n];
295     // Parametros del controlador
296     const float eta = 0.9f, wn[n] = {0.5f, 0.5f, 0.5f};
```

```
297     // Entradas de control
298     double tauu[n], tauc[n], taus[6];
299     // Parametros del robot
300     const float m1 = 28.5f, m2 = 16.6f, m3 = 1.0f, lc1 = 0.14f, lc2 = 0.14↩
            f, lc3 = 0.07f, I1 = 0.85f, I2 = 0.7f, I3 = 0.18f;
301
302     CTwoCamsDlg *pMainWnd = (CTwoCamsDlg *) AfxGetMainWnd();
303     // lectura de encoders
304     getArtRad();
305
306     // Definiciones trigonométricas útiles
307     c1 = cos(q[1]);
308     s1 = sin(q[1]);
309     c2 = cos(q[2]);
310     s2 = sin(q[2]);
311     s12 = sin(q[1]+q[2]);
312     c12 = cos(q[1]+q[2]);
313     c23 = cos(q[2]+q[4]);
314     s123 = sin(q[1]+q[2]+q[4]);
315     c123 = cos(q[1]+q[2]+q[4]);
316
317     H[0][0] = m1*lc1*lc1+m2*(l1*l1+2*l1*lc2*c2+lc2*lc2)+m3*(l1*l1+2*l1*l2*↩
            c2+2*l1*lc3*c23+l2*l2+2*lc3*l2*c3+lc3*lc3)+I1+I2+I3;
318     H[0][1] = m2*(l1*lc2*c2+lc2*lc2)+m3*(l1*l2*c2+l1*lc3*c23+l2*l2+2*lc3*↩
            l2*c3+lc3*lc3)+I2+I3;
319     H[0][2] = m3*(l1*lc3*c23+lc3*l2*c3+lc3*lc3)+I3;
320     H[1][0] = H[0][1];
321     H[1][1] = m2*lc2*lc2+m3*(l2*l2+2*lc3*l2*c3+lc3*lc3)+I2+I3;
322     H[1][2] = m3*(lc3*l2*c3+lc3*lc3)+I3;
323     H[2][0] = H[0][2];
324     H[2][1] = H[1][2];
325     H[2][2] = m3*lc3*lc3+I3;
326
327     // Calculo de x y xp
328     x[0] = l1*c1 + l2*c12 + l3*c123;
329     x[1] = l1*s1 + l2*s12 + l3*s123;
330     x[2] = q[1]+q[2]+q[4];
331     for(int i=0; i<n; i++){
332       xp[i] = (x[i] - x_1[i])/h;
333     }
334
335     /* Trayectoria */
336     // Posición inicial de trayectoria
337     if(iCPnt){
338       xi[0] = x[0];
339       xi[1] = x[1];
340       xi[2] = x[2];
341     }
342
343     // Posición final de trayectoria
344     xf[0] = beta + di*calpha;
345     xf[1] = di*salpha;
346     xf[2] = alpha - pi/2;
347
348     for(i=0; i<n; i++){
349       difx[i] = 1.0*(xf[i]-xi[i]);
350       a0[i] = 1.0*xi[i];
```

```
351         a5[i] = 6.0/(pow(tf,5))*difx[i];
352         a4[i] = -15.0/(pow(tf,4))*difx[i];
353         a3[i] = 10.0/(pow(tf,3))*difx[i];
354       }
355
356       if(t<tf){
357         for(int i=0;i<n;i++) {
358          xd[i] = a5[i]*pow(t,5)+a4[i]*pow(t,4)+a3[i]*pow(t,3)+a0[i];
359          xdp[i] = 5*a5[i]*pow(t,4) + 4*a4[i]*pow(t,3) + 3*a3[i]*pow(t,2);
360          xdpp[i] = 20*a5[i]*pow(t,3) + 12*a4[i]*pow(t,2) + 6*a3[i]*t;
361         }
362       }
363       else
364       {
365         for(i=0;i<n;i++){
366           xd[i] = xf[i];
367           xdp[i] = 0;
368           xdpp[i] = 0;
369         }
370
371         if(t<tf+0.006){
372           pMainWnd->m_cstatusText.SetWindowText(_T("*** En Punto de Inicio ↩
                  ***"));
373         }
374       }
375
376
377       // Variables auxiliares
378       for(i=0; i<n; i++){
379         e[i] = x[i] - xd[i];
380         ep[i] = xp[i] - xdp[i];
381       }
382       // Inicializacion de variables
383       if(iCPnt){
384         for(i=0; i<n; i++){
385           xp[i] = 0.0;
386           eg[i] = 0.0;
387           egp[i] = 0.0;
388           eg2[i] = 0.0;
389           et[i] = 0.0;
390           zg1[i] = 0.0;
391           zg2[i] = 0.0;
392           x_1[i] = x[0];
393         }
394         pMainWnd->m_cstatusText.SetWindowText(_T("*** Llevando a Punto de ↩
                Inicio ***"));
395         iCPnt = false;
396       }
397       // Calculo del Jacobiano
398       J[0][0] = -l1*s1-l2*s12-l3*s123;
399       J[0][1] = -l2*s12-l3*s123;
400       J[0][2] = -l3*s123;
401       J[1][0] = l1*c1+l2*c12+l3*c123;
402       J[1][1] = l2*c12+l3*c123;
403       J[1][2] = l3*c123;
404       J[2][0] = 1.0;
405       J[2][1] = 1.0;
```

```
406     J[2][2] = 1.0;
407     //Transpose (J, n, n, JT);
408     //Estimacion de la velocidad
409     for(i=0; i<n; i++){
410       eg2e[i] = (e[i]-e_1[i])/h;
411     }
412     InvMatrix (J, n, Jm1);
413     MatrixMul (H, Jm1, n, n, n, Mq);
414     for(i=0; i<n; i++){
415       tauu[i] = Mq[i][0]*(-2.0*eta*wn[i]*eg2[0]-wn[i]*wn[i]*e[0]-zg1[0])+↩
              Mq[i][1]*(-2.0*eta*wn[i]*eg2[1]-wn[i]*wn[i]*e[1]-zg1[1])+Mq[i↩
              ][2]*(-2.0*eta*wn[i]*eg2[2]-wn[i]*wn[i]*e[2]-zg1[2]);
416     }
417
418     for(i=0; i<n; i++){
419       taus[i] = tauu[i];
420     }
421
422     // Observador de velocidad
423     for(i=0; i<n; i++)
424       et[i] = e[i] - eg[i];
425     InvMatrix (Mq, n, Mqi);
426     for(i=0; i<n; i++){
427       egp[i] = eg2[i] + lam3*et[i];
428       eg2p[i] = Mqi[i][0]*tauu[0]+Mqi[i][1]*tauu[1]+Mqi[i][2]*tauu[2] + ↩
              zg1[i] + lam2*et[i];
429       zg1p[i] = zg2[i] + lam1*et[i];
430       zg2p[i] = lam0*et[i];
431     }
432
433       //↩
              ////////////////////////////////////////////////////////////////////////↩

434     ////////////////////////////// Manda voltajes ///////////////

435
436     NiFpga_WriteI16(session,voltA465_art2, taus[0]*vpn[1]*DACR);
437     NiFpga_WriteI16(session,voltA465_art3, taus[1]*vpn[2]*DACR);
438     NiFpga_WriteI16(session,voltA465_art5, taus[2]*vpn[4]*DACR);
439
440     //↩
            ////////////////////////////////////////////////////////////////////////↩

441     ////////////// Actualizacion de variables ///////////////////

442
443     for(i=0; i<n; i++){
444       x_1[i] = x[i];
445       e_1[i] = e[i];
446       eg[i] += h*egp[i];
447       eg2[i] += h*eg2p[i];
448       zg1[i] += h*zg1p[i];
449       zg2[i] += h*zg2p[i];
450     }
451
452
453     ts = timeGetTime()-t*1000-tc;// Periodo de muestreo real (ms).
454     t = 1.0*(timeGetTime()-tc)/1000; //Tiempo transcurrido (seg).
455
```

```
456
457   }//CIERRE DE IF nIDEvent
458   // ***************************************
459
460
461   // *********************************Timer referido al CONTROL del ↩
          robot
462   if( nIDEvent == IDT_CONTROL )
463   {
464
465     const  n=3;
466   double Kp[3][3];
467     double Kpn[3][3];
468     double lambday1=30;
469     double lambday2=30;
470     double lambday3=30;
471   double yrp[3][1];
472     double yop[3][1];
473     double taus[6];
474   double kd=500;
475     double kg1=0.1;
476     double kg2=0.1;
477   double kg3=0.1;
478     double kb1=5.2;
479     double kb2=5.2;
480   double kb3=5.2;
481     double k=0.1;
482     double s10=0;
483     double s20=0;
484     double s30=0;
485     double lambdaz1=30;
486   double lambdaz2=30;
487   double lambdaz3=30;
488     static double yh[3][1],yd[3][1],intez1=0,intez2=0,intez3=0,yohp1,yohp2↩
          ,yohp3,yhp1,yhp2,yhp3,sigma1=0,sigma2=0,sigma3=0;
489     double y_f[3][1],ytilde1,ytilde2,ytilde3,yd1,yd2,yd3;
490     double k_0=0.1;
491     double k_1=2;
492     double epsilon=0.1;
493     double taui1[3][1],taured[3][1],y_tilde[3][1],y_tildet[1][3],so[3][1];
494     double sd1,sd2,sd3,ese1,ese2,ese3,ydp[3][1],s11,s12,s13,sigmap1,↩
          sigmap2,sigmap3;
495     double yop1,yop2,yop3,yrp1,yrp2,yrp3,prue1[1],n_y_tilde[1];
496     float a2 = 0.330f, a3 = 0.305f,te1p=30*(3.1416/180);;
497     double offset=0.0,oc1=0.647,oc2=0.616,oc3=1.52,k0=20;
498     double factor1,factor2,lamda=0.0085,alphau=67567,alphav=67567,ucam↩
          =182,vcam=124;
499     double J[3][3],Rphi[3][3];
500     double JT[3][3];
501     double sat=7.0;
502     double kdz[3][1],Variable1[3][3],Variable2[3][3];
503     double sign_s11,sign_s12,sign_s13;
504     int i,y[3][1],z1,z2,z3,volt1,volt2,volt3;
505     double tau[6]={0.0};
506     const double tf = 10;
507   Kp[0][0]=0.025;
508   Kp[0][1]=0;
```

```
509    Kp[0][2]=0;
510    Kp[1][0]=0;
511    Kp[1][1]=0.025;
512    Kp[1][2]=0;
513    Kp[2][0]=0;
514    Kp[2][1]=0;
515    Kp[2][2]=0.025;
516    Kpn[0][0]=Kp[0][0]*(-1);
517    Kpn[0][1]=Kp[0][1]*(-1);
518    Kpn[0][2]=Kp[0][2]*(-1);
519    Kpn[1][0]=Kp[1][0]*(-1);
520    Kpn[1][1]=Kp[1][1]*(-1);
521    Kpn[1][2]=Kp[1][2]*(-1);
522    Kpn[2][0]=Kp[2][0]*(-1);
523    Kpn[2][1]=Kp[2][1]*(-1);
524    Kpn[2][2]=Kp[2][2]*(-1);
525        //↩
               ////////////////////////////////////////////////////////////////////////↩

526      CTwoCamsDlg *pMainWnd = (CTwoCamsDlg *) AfxGetMainWnd();
527    // lectura de encoders
528    getArtRad();
529    /*if(lambda>100){
530      NiFpga_MergeStatus(&status, NiFpga_WriteI16(session,activaADC, ↩
           paroA465));
531    }*/
532      // Definiciones trigonometricas utiles
533      ts = timeGetTime()-t*1000-tc;// Periodo de muestreo real (ms).
534      t = 1.0*(timeGetTime()-tc)/1000; //Tiempo transcurrido (seg).
535  //------Calcula el factor de convercion entre el espacio imagen y el ↩
       cartesiano
536     factor1=(alphau*lamda)/(oc3-lamda);
537     factor2=(alphav*lamda)/(oc3-lamda);
538    //-------Definicion del campo de velocidad↩
           ------------------------------------
539       y[0][0]= (int)pMainWnd->centroid[0];
540       y[1][0]= (int)pMainWnd->centroid[1];
541       y[2][0]= (int)pMainWnd->centroid2[1];
542
543
544
545       if(iCCtrl){
546      yd[0][0]=232;
547      yd[1][0]=404;
548      yd[2][0]=1010;
549      yh[0][0]=232;
550      yh[1][0]=404;
551      yh[2][0]=1010;
552        pMainWnd->m_cstatusText.SetWindowText(_T("*** Ejecutando Control ***↩
             "));
553        iCCtrl = false;
554      }
555       y_f[0][0]=400;
556       y_f[1][0]=300;
557       y_f[2][0]=900;
558      //////////----------------------INDICADOR X↩
             ----------------------------///////////
```

```
559       CString indic1;
560      indic1.Format("%d",y[0][0]);
561      pMainWnd->m_cv1.SetWindowText(indic1);
562      //////////----------------------INDICADOR Y↩
             ----------------------------//////////
563      CString indic2;
564      //indic1.Format("%lf",yd[0][0]);
565      indic2.Format("%d",y[1][0]);
566      pMainWnd->m_cv2.SetWindowText(indic2);
567      //////////----------------------INDICADOR Z↩
             ----------------------------//////////
568      CString indic9;
569      indic9.Format("%d",y[2][0]);
570      pMainWnd->m_cv9.SetWindowText(indic9);
571
572      //↩
             ///////----------------------------------------------------------------------//////////↩


573
574      Sub2D(y,y_f,2,1,y_tilde);
575      ytilde1=y[0][0]-y_f[0][0];
576      ytilde2=y[1][0]-y_f[1][0];
577      ytilde3=y[2][0]-y_f[2][0];
578      //////////----------------------INDICADOR ytilde1 ↩
             ----------------------------//////////
579      CString indic5;
580      indic5.Format("%lf",ytilde1);
581      pMainWnd->m_cv5.SetWindowText(indic5);
582      //////////----------------------INDICADOR ytilde2 ↩
             ----------------------------//////////
583      CString indic6;
584      indic6.Format("%lf",ytilde2);
585      pMainWnd->m_cv6.SetWindowText(indic6);
586      //////////----------------------INDICADOR ytilde3 ↩
             ----------------------------//////////
587      CString indic10;
588      indic10.Format("%lf",ytilde3);
589      pMainWnd->m_cv10.SetWindowText(indic10);
590      //↩
             ////////////////////////////////////////////////////////////////////////↩


591      Transpose(y_tilde,3,1,y_tildet);
592      MatrixMul(y_tildet,y_tilde,1,3,1,prue1);
593      n_y_tilde[0]=sqrt(prue1[0]);
594      ydp[0][0]=-k_1*(yd[0][0]-y_f[0][0])-(ytilde1)*(k_0/(n_y_tilde[0]+↩
             epsilon));
595      ydp[1][0]=-k_1*(yd[1][0]-y_f[1][0])-(ytilde2)*(k_0/(n_y_tilde[0]+↩
             epsilon));
596    ydp[2][0]=-k_1*(yd[2][0]-y_f[2][0])-(ytilde3)*(k_0/(n_y_tilde[0]+epsilon↩
           ));
597  //--------------------Otras definiciones↩
       ------------------------------------
598      yd1=yd[0][0]+h*ydp[0][0];
599      yd2=yd[1][0]+h*ydp[1][0];
600    yd3=yd[2][0]+h*ydp[2][0];
601      sd1 = s10*exp(-k*t);
602      sd2 = s20*exp(-k*t);
```

```
603   sd3 = s30*exp(-k*t);
604   yd[0][0] =  yd[0][0]+(h*ydp[0][0]);
605   yd[1][0] =  yd[1][0]+(h*ydp[1][0]);
606   yd[2][0] =  yd[2][0]+(h*ydp[2][0]);
607   //--------------------------------Errores↩
        ----------------------------------------
608     z1 =  y[0][0] - yh[0][0];
609     z2 =  y[1][0] - yh[1][0];
610     z3 =  y[2][0] - yh[2][0];
611      //////////--------------------------INDICADOR z1↩
           ----------------------------///////////
612     CString indic7;
613     indic7.Format("%d",z1);
614     pMainWnd->m_cv7.SetWindowText(indic7);
615     //////////----------------------INDICADOR z2↩
           ------------------------------------///////////
616     CString indic8;
617     indic8.Format("%d",z2);
618     pMainWnd->m_cv8.SetWindowText(indic8);
619
620     //////////----------------------INDICADOR z3↩
           ------------------------------------///////////
621     CString indic12;
622     indic12.Format("%d",z3);
623     pMainWnd->m_cv12.SetWindowText(indic12);
624     //↩
           ////////////////////////////////////////////////////////////////////////////////↩

625     intez1 = intez1 + h*z1;
626     intez2 = intez2 + h*z2;
627     intez3 = intez3 + h*z3;
628     yohp1  = ydp[0][0] - lambday1*(yh[0][0]-yd[0][0]) + sd1 + kd*lambdaz1*↩
          intez1;
629     yohp2  = ydp[1][0] - lambday2*(yh[1][0]-yd[1][0]) + sd2 + kd*lambdaz2*↩
          intez2;
630     yohp3  = ydp[2][0] - lambday3*(yh[2][0]-yd[2][0]) + sd3 + kd*lambdaz3*↩
          intez3;
631     yhp1 = yohp1 + lambdaz1*z1 + kd*z1;
632     yhp2 = yohp2 + lambdaz2*z2 + kd*z2;
633   yhp3 = yohp3 + lambdaz3*z3 + kd*z3;
634   yh[0][0] =  yh[0][0]+(h*yhp1);
635   yh[1][0] =  yh[1][0]+(h*yhp2);
636     yh[2][0] =  yh[2][0]+(h*yhp3);
637   yop1 = yhp1 - lambdaz1*z1;
638     yop2 = yhp2 - lambdaz2*z2;
639     yop3 = yhp3 - lambdaz3*z3;
640     ese1 = yhp1 - ydp[0][0] + lambday1*(yh[0][0]-yd[0][0]);
641     ese2 = yhp2 - ydp[1][0] + lambday2*(yh[1][0]-yd[1][0]);
642   ese3 = yhp3 - ydp[2][0] + lambday3*(yh[2][0]-yd[2][0]);
643   /*if (tiempo< hd2){
644     s10=ese1;
645     s20=ese2;
646     s30=ese3;}*/
647     s11 = ese1 - sd1;
648     s12 = ese2 - sd2;
649   s13=  ese3 - sd3;
```

```
650     if(s11 > 0) sign_s11 = 1; else if(s11 < 0) sign_s11 = -1; else if(s11 ↩
            == 0) sign_s11 = 0;
651     if(s12 > 0) sign_s12 = 1; else if(s12 < 0) sign_s12 = -1; else if(s12 ↩
            == 0) sign_s12 = 0;
652     if(s13 > 0) sign_s13 = 1; else if(s13 < 0) sign_s13 = -1; else if(s13 ↩
            == 0) sign_s13 = 0;
653   sigmap1 = kb1*s11 + sign_s11;
654     sigmap2 = kb2*s12 + sign_s12;
655   sigmap3 = kb3*s13 + sign_s13;
656     sigma1 = sigma1 + h*sigmap1;
657     sigma2 = sigma2 + h*sigmap2;
658   sigma3 = sigma3 + h*sigmap3;
659 //-----------------------------------Se forma s0↩
        ----------------------------------------
660     yrp1 = ydp[0][0] - lambday1*( yh[0][0]-yd[0][0]) + sd1 - kg1*sigma1;
661     yrp2 = ydp[1][0] - lambday2*( yh[1][0]-yd[1][0]) + sd2 - kg2*sigma2;
662   yrp3 = ydp[2][0] - lambday3*( yh[2][0]-yd[2][0]) + sd3 - kg3*sigma3;
663     yrp[0][0]=yrp1;
664     yrp[1][0]=yrp2;
665   yrp[2][0]=yrp3;
666     yop[0][0]=yop1;
667     yop[1][0]=yop2;
668     yop[2][0]=yop3;
669     so[0][0]=yop[0][0]-yrp[0][0];
670     so[1][0]=yop[1][0]-yrp[1][0];
671     so[2][0]=yop[2][0]-yrp[2][0];
672     //Sub2D(yop,yrp,2,1,so);
673 //-----------------------Calculo del Jacobiano↩
        ----------------------------------------
674     J[0][0] = -sin(q[0])*(a2*cos(q[1])) + a3*cos(q[1]+q[2]);
675     J[0][1] = -cos(q[0])*(a2*sin(q[1])) + a3*sin(q[1]+q[2]);
676     J[0][2] = -a3*cos(q[0])*sin(q[1]+q[2]);
677     J[1][0] = cos(q[0])*(a2*cos(q[1])) + a3*cos(q[1]+q[2]);
678   J[1][1] = -sin(q[0])*(a2*sin(q[1])) + a3*sin(q[1]+q[2]) ;
679     J[1][2] = -a3*sin(q[0])*sin(q[1]+q[2]);
680     J[2][0] = 0;
681     J[2][1] = a2*cos(q[1]) + a3*cos(q[1]+q[2]);
682   J[2][2] = a3*cos(q[1]+q[2]);
683   Transpose(J,3,3,JT);
684 //-----------------------Calculo de la Matriz de Rotacion↩
        ----------------------
685     Rphi[0][0]=-cos(te1p);
686     Rphi[0][1]=0;
687     Rphi[0][2]=sin(te1p);
688     Rphi[1][0]=-sin(te1p);
689   Rphi[1][1]=0;
690     Rphi[1][2]=-cos(te1p);
691     Rphi[2][0]=0;
692     Rphi[2][1]=-1;
693   Rphi[2][2]=0;
694   //-----------------------Calculo de la Salida↩
          ----------------------------------------
695     MatrixMul(Kpn,JT,3,3,3,Variable1);
696     MatrixMul(Variable1,Rphi,3,3,3,Variable2);
697     taured[0][0]=Variable2[0][0]*so[0][0]+Variable2[0][1]*so[1][0]+↩
            Variable2[0][2]*so[2][0];
```

```
698      taured[1][0]=Variable2[1][0]*so[0][0]+Variable2[1][1]*so[1][0]+↩
             Variable2[1][2]*so[2][0];///-Kp*JT*Rphi*so
699      taured[2][0]=Variable2[2][0]*so[0][0]+Variable2[2][1]*so[1][0]+↩
             Variable2[2][2]*so[2][0];
700      taus[0] = taured[0][0];
701      taus[1] = taured[1][0];
702      taus[2] = taured[2][0];
703      volt1=taui1[0][0]*DACR;
704      volt2=taui1[1][0]*DACR;
705      volt3=taui1[2][0]*DACR;
706
707
708      //↩
             ********************************************************************↩

709      CString v1;
710      v1.Format("%d",taus[0]);
711      pMainWnd->m_volt1.SetWindowText(v1);
712
713      CString v2;
714      v2.Format("%d",taus[1]);
715      pMainWnd->m_volt2.SetWindowText(v2);
716
717      CString v3;
718      v3.Format("%d",taus[2]);
719      pMainWnd->m_volt3.SetWindowText(v3);
720
721
722      //↩
             ********************************************************************↩

723
724
725
726
727      //////////----------------------INDICADOR TORQUE ↩
             1----------------------//////////
728      CString indic3;
729      indic3.Format("%d",volt1);
730      pMainWnd->m_cv3.SetWindowText(indic3);
731      //////////----------------------INDICADOR TORQUE ↩
             2----------------------//////////
732      CString indic4;
733      indic4.Format("%d",volt2);
734      pMainWnd->m_cv4.SetWindowText(indic4);
735      //////////----------------------INDICADOR TORQUE ↩
             2----------------------//////////
736      CString indic11;
737      indic11.Format("%d",volt3);
738      pMainWnd->m_cv11.SetWindowText(indic11);
739    //----------------------Envio de Voltajes↩
         ----------------------------------

740
741
742      NiFpga_WriteI16(session,voltA465_art1, taus[0]*DACR);
743      NiFpga_WriteI16(session,voltA465_art2, taus[1]*DACR);
744      NiFpga_WriteI16(session,voltA465_art3, taus[2]*DACR);
```

```
745
746
747
748   //----------------------Actualizacion de variables↩
          ------------------------------
749   ////////////// Actualizacion de variables ///////////////////
750
751     if(y[0][0]==y_f[0][0] && y[1][0]==y_f[1][0]){
752
753       pMainWnd->m_cstatusText.SetWindowText(_T("*** Control Finalizado ***↩
              "));
754         }
755 /////////// Para Graficar
756
757     if(pMainWnd->m_grafic){
758
759
760       grafi[0][indx] = t;
761     grafi[1][indx] = y[0][0];
762     grafi[2][indx] = y[1][0];
763         grafi[3][indx] = y[2][0];
764     grafi[4][indx] = yd[0][0];
765     grafi[5][indx] = yd[1][0];
766         grafi[6][indx] = yd[2][0];
767     grafi[7][indx] = yh[0][0];
768     grafi[8][indx] = yh[1][0];
769         grafi[9][indx] = yh[2][0];
770     grafi[10][indx] = volt1;
771     grafi[11][indx] = volt2;
772         grafi[12][indx] = volt3;
773     grafi[13][indx] = taus[0];
774     grafi[14][indx] = taus[1];
775     grafi[15][indx] = taus[2];
776     indx ++;
777
778     }
779
780   }//CIERRE DE IF nIDEvent
781   // ************************************
782
783
784
785   CDialog::OnTimer(nIDEvent);
786 }
```

Apéndice B

Código. Funciones que hacen uso de librerías.

```
1
2  #include "stdafx.h"
3  #include "afxinet.h"
4  #include "TwoCams.h"
5  #include "TwoCamsDlg.h"
6  #include "MMSYSTEM.H"
7
8  #include "NiFpga_final.h"
9  //Archivo con los parametros del robot
10 #include "parametros.h"
11 //Librerias de matematicas y análisis de matrices
12 #include  "math.h"
13 #include  "analysis.h"
14
15 //Librerias auxiliares
16 #include "LIB.h"
17 #include "APICRIO.h"
18
19
20
21
22 #ifdef _DEBUG
23 #define new DEBUG_NEW
24 #undef THIS_FILE
25 static char THIS_FILE[] = __FILE__;
26 #endif
27
28
29 //***Defino TIMERS a usar************************************
30
31 #define IDT_VISION WM_USER + 200
32 #define IDT_HOME IDT_VISION + 1
33 #define IDT_PINICIAL IDT_HOME + 1
```

```
34 #define IDT_CONTROL IDT_PINICIAL + 1
35
36 //************************************************************
37
38 // link image manger lib's (LIB file directories must be set properly)
39 #pragma comment (lib, "cvcimg.lib")
40 #pragma comment (lib, "cvcutilities.lib")
41 #pragma comment (lib, "cvcdriver.lib")
42 #pragma comment (lib, "cvcBlob.lib")
43
44
45 // ****************** Funciones que hacen uso de librerías de Common ↩
       Vision Blox
46
47
48 void CTwoCamsDlg::OnBOpen()
49 {
50
51
52   // Cargar imagen usando un cuadro de diálogo
53   //m_cvImg.LoadImageByDialog ();
54   //m_cvImg2.LoadImageByDialog ();
55
56   //Cargar imagen directo de ruta
57   m_cvImg.LoadImage(LPCSTR(_T("C:/Archivos de programa/Common Vision Blox/↩
         Drivers/cvAVT1394.vin")));
58   m_cvImg2.LoadImage(LPCSTR(_T("C:/Archivos de programa/Common Vision Blox↩
         /Drivers/cvAVT1394.vin")));
59
60   // setup UI
61   SetupUI ();
62   SetupBlob();
63
64
65   OnComboBoxBoardCreate();
66   OnComboBoxPortCreate();
67   OnComboBoxBoardCreate2();
68   OnComboBoxPortCreate2();
69
70 }
71
72 void CTwoCamsDlg::OnBSave()
73 {
74   // save image using comdlg
75   m_cvImg.SaveImageByDialog ();
76   m_cvImg2.SaveImageByDialog ();
77
78   // TODO: Add your control notification handler code here
79
80 }
81
82 void CTwoCamsDlg::OnImageSnapedCvImg()
83 {
84
85   // refresh display
86   m_cvDisp.Refresh ();
87
```

```
88    // TODO: Add your image processing functions here or modify the code ↩
          above
89
90  }
91
92  void CTwoCamsDlg::OnImageSnapedCvImg2()
93  {
94    // refresh display
95    m_cvDisp2.Refresh ();
96    // TODO: Add your control notification handler code here
97
98  }
99
100 void CTwoCamsDlg::OnCGrab()
101 {
102   // update checkbox state
103   UpdateData (TRUE);
104   // set grab
105   m_cvImg.SetGrab (m_bGrab);
106
107   // TODO: Add your control notification handler code here
108
109 }
110
111
112 void CTwoCamsDlg::OnCGrab2()
113 {
114   // update checkbox state
115   UpdateData (TRUE);
116   // set grab
117   m_cvImg2.SetGrab (m_bGrab2);
118
119   // TODO: Add your control notification handler code here
120
121 }
122
123 void CTwoCamsDlg::SetupBlob()
124 {
125   if (IsImage((void *)m_cvImg.GetImage()))
126   {
127     // reset textboxes, overlays
128     m_cvDisp.RemoveAllUserObjects();
129
130     m_cvDisp.SetImage(m_cvImg.GetImage());
131   BlobSetObjectFeatureRange((void *)m_BlobInstance, m_lowTres, m_upTres);
132     BlobSetImage((IMG)m_BlobInstance, (void *)m_cvImg.GetImage(), 0);
133
134   }
135   else
136   {
137     //m_ctrlExec.EnableWindow(FALSE);
138   }
139
140   if (IsImage((void *)m_cvImg2.GetImage()))
141   {
142     // reset textboxes, overlays
143     m_cvDisp2.RemoveAllUserObjects();
```

```
144
145     m_cvDisp2.SetImage(m_cvImg2.GetImage());
146   BlobSetObjectFeatureRange((void *)m_BlobInstance2, m_lowTres2, m_upTres2↩
        );
147     BlobSetImage((IMG)m_BlobInstance2, (void *)m_cvImg2.GetImage(), 0);
148
149   }
150   else
151   {
152     //m_ctrlExec.EnableWindow(FALSE);
153   }
154   //m_CtrlStop.EnableWindow(FALSE);
155 }
156
157 int CTwoCamsDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
158 {
159   if (CDialog::OnCreate(lpCreateStruct) == -1)
160     return -1;
161
162   //SetTimer(1,10,NULL); // TimerID, Duration[ms], NULL
163
164   StartTimer(10, IDT_VISION);
165
166   return 0;
167 }
168
169 double* CTwoCamsDlg::GetBlobCentroid(int i,DWORD BlobIns)
170 {
171   long Size;
172   centroid[0]=0;
173   centroid[1]=0;
174
175   BlobGetCenterEx((void *)BlobIns, i, centroid[0], centroid[1]);
176   if (BlobGetBlobSize((void *)BlobIns, i, Size) < 0)
177     {
178     MessageBox("Datos de imagen no válidos", "Información", MB_OK);
179     return &centroid[0];
180     }
181   return &centroid[0];
182 }
183
184 //***************Funciones que se comunican con el Módulo CompactRio
185
186
187 void CTwoCamsDlg::OnInitar()
188 {
189   // TODO: Add your control notification handler code here
190   status = NiFpga_Initialize();
191
192   if (NiFpga_IsNotError(status))
193     {
194     /* opens a session, downloads the bitstream, and runs the FPGA */
195     NiFpga_MergeStatus(&status, NiFpga_Open(NiFpga_final_Bitfile,
196                                 NiFpga_final_Signature, //Firma en ↩
                                      NiFpga_IOCLIENT.h
197                                 "rio://10.52.22.73/RIO0", //ver en Max+↩
                                      devices and interfaces+RIO0
```

```
                                    NiFpga_OpenAttribute_NoRun,
                                    &session));
    NiFpga_MergeStatus(&status, NiFpga_Run(session, 0));
    }
  /* check if anything went wrong */
  if (NiFpga_IsError(status))
    {
    char error[32];
    sprintf(error, "Error %d!", status);
    MessageBox("No se pudo iniciar el FPGA");
    }
  else
  {
    //MessageBox("Tarjetas Inicializadas Correctamente");//
    m_cstatusText.SetWindowText(_T("*** Tarjetas Inicializadas ↩
        Correctamente ***"));
  }
}


void CTwoCamsDlg::OnLeeEnc()
{
  getArtRad();
  CString qdata;
//  qdata.Format("fx: %d",off.fx);
  m_a1 = q[0]*180/pi;
  m_a2 = q[1]*180/pi;
  m_a3 = q[2]*180/pi;
  m_a4 = q[3]*180/pi;
  m_a5 = q[4]*180/pi;
  m_a6 = q[5]*180/pi;
  UpdateData(FALSE);

}

void CTwoCamsDlg::OnACTVart()
{
//Activa las articulaciones
  NiFpga_MergeStatus(&status, NiFpga_WriteBool(session,actA465_art1,1));
  NiFpga_MergeStatus(&status, NiFpga_WriteBool(session,actA465_art2,1));
  NiFpga_MergeStatus(&status, NiFpga_WriteBool(session,actA465_art3,1));
  NiFpga_MergeStatus(&status, NiFpga_WriteBool(session,actA465_art4,0));
  NiFpga_MergeStatus(&status, NiFpga_WriteBool(session,actA465_art5,0));
  NiFpga_MergeStatus(&status, NiFpga_WriteBool(session,actA465_art6,0));
  NiFpga_MergeStatus(&status, NiFpga_Run(session, 0));

  if (NiFpga_IsError(status))
  {
    MessageBox("No se pudieron activar las articulaciones!");
    char error[32];
    sprintf(error, "Error %d!", status);
  }
  else
  {
    m_cstatusText.SetWindowText(_T("*** Articulaciones Activadas ***"));
  }
  //manda limites de voltaje SUPERIOR
```

```
254   NiFpga_WriteI16(session,LVSA465_art1,2*DACR);
255   NiFpga_WriteI16(session,LVSA465_art2,3*DACR);
256   NiFpga_WriteI16(session,LVSA465_art3,3*DACR);
257   NiFpga_WriteI16(session,LVSA465_art4,5*DACR);
258   NiFpga_WriteI16(session,LVSA465_art5,4*DACR);
259   NiFpga_WriteI16(session,LVSA465_art6,5*DACR);
260
261   //manda limites de voltaje INFERIOR
262   NiFpga_WriteI16(session,LVIA465_art1,-2*DACR);
263   NiFpga_WriteI16(session,LVIA465_art2,-3*DACR);
264   NiFpga_WriteI16(session,LVIA465_art3,-3*DACR);
265   NiFpga_WriteI16(session,LVIA465_art4,-5*DACR);
266   NiFpga_WriteI16(session,LVIA465_art5,-4*DACR);
267   NiFpga_WriteI16(session,LVIA465_art6,-5*DACR);
268
269 }
270
271
272
273
274
275 void CTwoCamsDlg::OnMCart()
276 {
277
278   UpdateData(TRUE);
279
280
281   NiFpga_WriteI32(session,rstcuentaA465_art1, ((1.0*m_a1)/gpc[0]) );
282   NiFpga_WriteBool(session,rstA465_art1,1);
283   NiFpga_WriteBool(session,rstA465_art1,0);
284
285   NiFpga_WriteI32(session,rstcuentaA465_art2, ((1.0*m_a2)/gpc[1]) );
286   NiFpga_WriteBool(session,rstA465_art2,1);
287   NiFpga_WriteBool(session,rstA465_art2,0);
288
289   NiFpga_WriteI32(session,rstcuentaA465_art3, ((1.0*m_a3)/gpc[2]) );
290   NiFpga_WriteBool(session,rstA465_art3,1);
291   NiFpga_WriteBool(session,rstA465_art3,0);
292
293   NiFpga_WriteI32(session,rstcuentaA465_art4, ((1.0*m_a4)/gpc[3]) );
294   NiFpga_WriteBool(session,rstA465_art4,1);
295   NiFpga_WriteBool(session,rstA465_art4,0);
296
297   NiFpga_WriteI32(session,rstcuentaA465_art5, ((1.0*m_a5)/gpc[4]) );
298   NiFpga_WriteBool(session,rstA465_art5,1);
299   NiFpga_WriteBool(session,rstA465_art5,0);
300
301   NiFpga_WriteI32(session,rstcuentaA465_art6, ((1.0*m_a6)/gpc[5]) );
302   NiFpga_WriteBool(session,rstA465_art6,1);
303   NiFpga_WriteBool(session,rstA465_art6,0);
304
305 }
306
307
308
309
```

```
310 // *Función que inicializa los timers↩
        **************************************************************************↩
311
312 UINT CTwoCamsDlg::StartTimer(UINT TimerDuration, UINT TimerID)
313 {
314   UINT TimerVal;
315
316   TimerVal = SetTimer(TimerID, TimerDuration, NULL);
317
318   if (TimerVal == 0)
319   {
320     MessageBox ("Unable to obtain timer","IDT_TIMER_0",MB_OK|↩
          MB_SYSTEMMODAL);
321   }
322
323   timerActive=true;
324
325   return TimerVal;
326
327 }
328 // ********************************************************************
329
330 // *Función que para un timer en específico*****************************
331
332 BOOL CTwoCamsDlg::StopTimer(UINT TimerVal)
333 {
334   if (timerActive)
335   {
336     if (!KillTimer (TimerVal))
337     {
338        // CString Message;
339        // Message.Format("Unable to kill timer: %u", TimerVal);
340        // MessageBox (Message, "StopTimer", MB_OK|MB_SYSTEMMODAL);
341
342         return FALSE;
343     }
344   }
345   return TRUE;
346 }
347 // ********************************************************************
```

Bibliografía

[1] ARTHUR BROWNE and LEONARD NORTON-WAYNE. *Vision and Information. Processing for Automation.* PLENUM PRESS, New York, 1986.

[2] Peter I. Corke. *Visual control of robot manipulators - a review. In Visual Servoing.* World Scientific, 1994.

[3] E.R. DAVIES. *Machine Vision. Theory, Algorithms, Practicalities.* ELSEVIER, U.S., 3ª edition, 2005.

[4] David A. Forsyth and Jean Ponce. *Computer Vision: A modern Approach.* TBS, 1ª edition, 2002.

[5] DARÍO MARAVALL GÓMEZ-ALLENDE. *Reconocimiento de Formas y Visión Artificial.* Addison-Wesley Iberoamericana, S.A., Delawere, E.U.A., 1994.

[6] PAJARES MARTINSANZ GONZALO and DE LA CRUZ GARCÍA JESÚS M. *Visión por computador. Imágenes Digitales y Aplicaciones.* Alfaomega, México, 2ª edition, 2008.

[7] Ivor Horton. *Ivor Horton's Beginning Visual C++ 2010.* Wiley Publishing,Inc., Indianapolis, Indiana, 1ª edition, 2010.

[8] R. Kelly. Robust asymptotically stable visual servoing of planar robots. *IEEE Transactions on Robotics and Automation*, 12(5):759–766, 1996.

[9] R. Kelly and F. Reyes. On vision system identification with application to fixed-camera robotic systems. *Int. J. Imaging Syst. Technol.*, 11(3):170–180, 2000.

[10] R. Kelly and V. Santibañez. *Control de Movimiento de Robots Manipulados.* Pearson Educación, MADRID, 2003.

[11] E.C. Dean León. Sistema servo visual no calibrado de fuerza/ posición para robots dinámicos restringidos por superficies de contacto no estructuradas. *Centro de nvestigación y de Estudios Avanzados del IPN*, 2006.

[12] M. Bueno López. Control servovisual de robots manipuladores en 3d. *Universidad Nacional Autónoma de México*, 2012.

[13] Adrián M. Castillo-Sánchez Marco A. Arteaga and Vicente Parra-Vega. Cartesian control of robots without dynamic model and observer design. *Automatica*, pages 473–480, 2006.

[14] Seth Hutchinson Mark W. Spong and M. Vidyasagar. *Robot Modeling and Control.* John Wiley and Sons,Inc., 1ª edition, 2006.

[15] ZHENG NANNING, JIANG XIAOYI, and LAN XUGUANG. *Advances in Machine Vision, Image Processing, and Pattern Analysis.* Springer, 1986.

[16] International Federation of Robotics Statistical Departament. World robotics 2012 - industrial robots. *VDMA Robotics*, 2012.

[17] FU-K. S., GONZALEZ-R.C., and LEE-C. S. G. *ROBOTICA: Control, detección, visión e inteligencia*. McGraw-Hill, España, 1ª edition, 1998.

[18] Lorenzo Sciavicco and Bruno Siciliano. *Modeling and Control of Robot Manipulators*. Springer, 2ª edition, 2000.

[19] Y. Shirai and H. Inoue. Guiding a robot by visual feedback in assembling tasks. *Pattern Recognition*, 5(2):99–108, 1973.

[20] JOHN E. SWANKE. *Visual C++ MFC. Programming by Example*. CMP Books, U.S., 1999.

[21] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, draft edition edition, 2010.