ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO Y MANIPULACIÓN DE FRUTAS UTILIZANDO VISIÓN ARTIFICIAL Y BRAZO ROBÓTICO INDUSTRIAL"

INFORME DE MATERIA INTEGRADORA

Previo a la obtención del Título de:

INGENIERO EN ELECTRICIDAD, ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

ÁNGEL JOSUÉ VALENCIA ARMIJOS

ROGER MICHELL IDROVO URGILÉS

GUAYAQUIL – ECUADOR

AÑO: 2016

DEDICATORIA

A Dios por guiarme por el camino correcto y darme fuerzas para seguir adelante en momentos difíciles y encarar las adversidades de la mejor manera.

A mis padres Ángel Idrovo y Bertha Urgilés, que gracias a su apoyo y amor incondicional me ayudaron a cumplir con esta meta. A mis hermanos Jonathan, Ariel y Anggie por ser esa bella responsabilidad que tengo como hermano mayor.

A mis amigos por brindarme su fiel y hermosa amistad durante esta etapa de mi vida.

Roger Michell Idrovo Urgilés

A mi madre Valeria Armijos, por educarme con ese amor incondicional y siempre apoyarme en cada una de las etapas de mi vida. A mis hermanos Nicolle y Gustavo por ser ese complemento de amor y responsabilidad que me hacía falta.

A mis abuelos, Manuel Armijos y Bertha León, por realizar su labor de padres más de lo que deberían, siendo un apoyo fundamental en mi vida haciendo que nunca me falte nada y preocupándose por mi bienestar.

A mis amigos que he ido encontrando durante el trayecto de mi vida, por saber tratar conmigo y mantener esa amistad a pesar de mis defectos.

Ángel Josué Valencia Armijos

TRIBUNAL DE EVALUACIÓN

PhD. Wilton Edixon Agila Gálvez

PROFESOR EVALUADOR

PhD. Douglas Antonio Plaza Guingla

PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

.....

Ángel Josué Valencia Armijos

.....

Roger Michell Idrovo Urgilés

RESUMEN

La incorporación de los robots en los procesos industriales ha generado nuevas posibilidades para la automatización, transformando los métodos de trabajo que se han venido desarrollando. Otro campo del cual se complementa los sistemas robotizados es la visión artificial, conceder al robot el sentido de la vista proporciona mayor flexibilidad, escalabilidad e inteligencia al sistema.

La Facultad de Ingeniería en Electricidad y Computación (FIEC), recientemente inauguró el laboratorio de control de procesos, que cuenta en sus instalaciones con un brazo robótico industrial. Este equipo actualmente forma parte de un sistema continuo de envasado, pero se ha considerado realizar como proyecto una aplicación adicional enfocada a la industria de alimentos, un sector muy importante para el país.

El proyecto consiste en diseñar e implementar un sistema automático de reconocimiento y manipulación de frutas, utilizando el robot Kawasaki RS03N en conjunto con una aplicación de visión artificial basado en la librería de uso libre OpenCV.

En el capítulo 1, se muestran los objetivos generales y específicos, y también se presenta en más detalle el problema y la solución propuesta.

En el capítulo 2, se describen las herramientas y los métodos que se utilizan para el desarrollo del proyecto.

En el capítulo 3, se realiza un análisis de los resultados obtenidos durante la ejecución del proyecto.

Finalmente, en los anexos, se presenta el código utilizado en el robot y en el sistema de visión artificial.

ÍNDICE GENERAL

DEDICATORIA ii
TRIBUNAL DE EVALUACIÓN iii
DECLARACIÓN EXPRESA iv
RESUMEN v
ÍNDICE GENERAL vi
ÍNDICE DE TABLAS ix
ÍNDICE DE FIGURAS x
CAPÍTULO 1 1
1. INTRODUCCIÓN 1
1.1 Antecedentes 1
1.2 Descripción del problema 2
1.3 Justificación del Proyecto 2
1.4 Propuesta de Solución 3
1.5 Objetivos 3
1.5.1 Objetivo General 3
1.5.2 Objetivos Específicos 3
CAPÍTULO 2 4
2. CONCEPTOS TEÓRICOS Y DESCRIPCIÓN DEL SISTEMA DE VISION ARTIFICIAL Y ROBOT INDUSTRIAL. 4
2.1 Definición del sistema 4
2.2 Condiciones del sistema 5
2.3 Sistema de visión artificial 6
2.3.1 Descripción 6
2.3.2 Etapas del sistema de visión artificial 7
2.4 Captura 7
2.4.1 Cámara Basler acA1300-75gc 7
2.4.2 Óptica Fujinon DF6HA-1B 8

2.5 Procesamiento de imágenes 8
2.5.1 OpenCV 9
2.5.2 Software 9
2.6 Pre procesamiento 9
2.6.1 Filtro gaussiano 10
2.7 Segmentación 10
2.7.1 Espacio de color HSV 10
2.7.2 Histograma 11
2.7.3 Binarización 11
2.7.4 Operaciones morfológicas 12
2.8 Reconocimiento 12
2.8.1 Extracción del contorno 12
2.8.2 Momentos de imágenes 13
2.8.3 Representación visual 14
2.9 Calibración de cámara 14
2.10 Transformación de coordenadas 15
2.10.1 Posición 2D a 3D 15
2.10.2 Posición de la herramienta del robot 17
2.10.3 Orientación de la herramienta del robot 18
2.11 Robot industrial 19
2.11.1 Características generales 20
2.11.2 Características mecánicas 20
2.11.3 Zona de trabajo 21
2.11.4 Componentes del manipulador 22
2.12 Sistema de manipulación de frutas 23
2.12.1 Descripción 23
2.12.2 Algoritmo 24
2.13 Comunicación 26
2.13.1 Cámara - PC 26

2.13.2 Robot - PC 27
CAPÍTULO 3 29
3. RESULTADOS DEL SISTEMA DE RECONOCIMIENTO Y MANIPULACIÓN DE FRUTAS 29
3.1 Funcionamiento del sistema de reconocimiento de frutas 29
3.1.1 Captura 29
3.1.2 Pre-procesamiento 29
3.1.3 Segmentación 30
3.1.4 Reconocimiento 32
3.2 Funcionamiento del sistema de manipulación de frutas 34
3.2.5 Selección de la fruta 34
3.2.6 Agarre y traslado de la fruta 35
3.2.7 Empaquetado de las frutas 36
3.3 Pruebas de agarre de las frutas 37
3.4 Pruebas de velocidad de agarre y traslado de las frutas 39
CONCLUSIONES Y RECOMENDACIONES 41
BIBLIOGRAFÍA 42
ANEXOS 45

ÍNDICE DE TABLAS

Tabla 1. Especificaciones del robot Kawasaki RS03N. [13]..... 20
Tabla 2. Características mecánicas del robot Kawasaki RS03N [13] 21
Tabla 3. Rangos de valores HSV de las frutas 31
Tabla 4. Velocidades y tiempos de manipulación de las frutas..... 40

ÍNDICE DE FIGURAS

Figura 2.1: Componentes del sistema 4
Figura 2.2: Diagrama de bloques del sistema..... 5
Figura 2.3: Entorno del sistema de visión artificial 6
Figura 2.4: Diagrama de flujo de las etapas del sistema de visión artificial 7
Figura 2.5: Parte frontal de la cámara Basler acA1300-75gc [6] 8
Figura 2.6: Parte lateral de la óptica Fujinon DF6HA-1B [7] 8
Figura 2.7: Logos de OpenCV y Visual Studio [9], [10]..... 9
Figura 2.8: Código en OpenCV para la fase de pre procesamiento..... 10
Figura 2.9: Código en OpenCV para la fase de segmentación 10
Figura 2.10: Código en OpenCV para la obtención del histograma..... 11
Figura 2.11: Código en OpenCV para aplicar erosión y dilatación en la imagen 12
Figura 2.12: Código en OpenCV para la extracción del contorno y cálculo de momentos. 13
Figura 2.13: Código en OpenCV para mostrar la información de la fase de reconocimiento..... 14
Figura 2.14: Código en OpenCV para realizar la calibración de la cámara .. 15
Figura 2.15: Código en OpenCV para la transformación de posición 18
Figura 2.16: Código en OpenCV para la transformación de orientación..... 19
Figura 2.17: Robot industrial Kawasaki RS03N 20
Figura 2.18: Representación de las articulaciones del robot Kawasaki RS03N [15]..... 21
Figura 2.19: Área de trabajo del robot Kawasaki RS03N [13]..... 22
Figura 2.20: Robot industrial Kawasaki RS03N del laboratorio de control de procesos 23
Figura 2.21: Diagrama de flujo para el algoritmo del sistema de manipulación [18]..... 24
Figura 2.22: Ubicación de las posiciones finales (P1, P2, P3 y P4) de las frutas 25
Figura 2.23: Diagrama de red ethernet del sistema 26
Figura 2.24: Diagrama de bloques del proceso de comunicación socket entre cliente y servidor [20] 27
Figura 3.1: Captura de imagen 29
Figura 3.2: Imagen luego de la etapa de pre procesamiento. 30
Figura 3.3: Imagen en el espacio de color HSV..... 30

Figura 3.4: Gráfica del Histograma para el maracuyá..... 31
Figura 3.5: Imagen en su forma binaria 32
Figura 3.6: Imagen después del proceso de erosión y dilatación 32
Figura 3.7: Detección de los contornos en los objetos..... 33
Figura 3.8: Ubicación del centroide y orientación del objeto en la imagen ... 33
Figura 3.9: Valores de posición y orientación del objeto detectado 34
Figura 3.10: Maracuyás y aguacates en la zona de agarre del robot 35
Figura 3.11: Posición y orientación de la herramienta durante el agarre de las frutas..... 35
Figura 3.12: Robot en proceso de traslado de la fruta..... 36
Figura 3.13: Ordenamiento en forma matricial de las frutas 37
Figura 3.14: Caja llena con el producto final..... 37
Figura 3.15: Frutos distribuidos en la zona de trabajo 38
Figura 3.16: Agarre de la herramienta del robot para las cuatro ubicaciones de las frutas 38
Figura 3.17: Maracuyá en la posición de agarre para la prueba de velocidad 39

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Antecedentes

La necesidad de realizar cambios que aporten mejoras en las actividades económicas y productivas de una industria, hace indispensable la implementación de nuevas tecnologías de automatización. En la actualidad, varios procesos de manufacturación se realizan con máquinas diseñadas exclusivamente para esas tareas, sumado al uso intensivo de mano de obra para estas actividades han llevado a un interés creciente de integrar soluciones basadas en sistemas robotizados.

La incorporación de la robótica ha generado nuevas posibilidades para la automatización, logrando obtener un amplio rango de aplicaciones, dentro de las cuales se puede mencionar: soldadura, manipulación de materiales, paletización, servicio de pintura, inspección, entre otros. Aplicaciones más actuales como la inclusión de visión artificial han logrado una notable evolución en los sistemas robotizados. [1]

En la industria alimentaria los requerimientos de productos con mayor calidad y seguridad a un mejor precio están dominando el mercado. El sector está pasando de ser altamente dependiente de la mano de obra a una industria más automatizada. Debido a las características tan variables de los productos naturales, convierte a este sector idóneo para el empleo de robots en sus procesos.

En la actualidad la principal aplicación de la robótica en la industria alimentaria es el transporte y almacenaje de productos terminados. Con la integración de visión artificial se han aplicado procesos de control de calidad para la identificación de defectos o mal estado de alimentos, al agregar distintos tipos de sensores al sistema han permitido especializar tareas que antes no hubiese

sido posible realizar. Todo esto claramente indica que los usos más determinantes de los sistemas robotizados están aún por desarrollarse. [2]

1.2 Descripción del problema

La industria alimentaria ecuatoriana es de gran importancia para la economía del país, siendo las frutas uno de los principales productos de exportación en este sector. Pese a que cuentan con la tecnología necesaria para cumplir los requisitos de exportación, los métodos aplicados en todos los casos no son los más eficientes. Las tareas repetitivas provocan fatiga, imprecisión y hasta lesiones en los trabajadores, esta situación inherente ha ocasionado problemas como acumulación y deterioro del material, lo que se traduce a pérdidas en la producción.

El proceso de manipulación y empaquetamiento de frutas en las industrias del país se lo ha venido desarrollando con un elevado índice de mano de obra, acarreando todas las complicaciones antes mencionadas. Hasta el momento no se ha considerado una alternativa para este sistema, pero con miras de cumplir las exigencias del mercado y ser más competitivos hace necesaria la implementación de tecnologías basadas en automatización y robótica. [3]

1.3 Justificación del proyecto

El considerable aumento en las ventas de robots industriales cada año está impulsando la demanda de soluciones inteligentes de automatización. Debido a esto, se debe proponer proyectos donde la robótica pueda establecerse en el país, aprovechando los sectores industriales donde exista mayor exportación e inversión.

Además, será necesario contar con personas capaces de programar, instalar, emplear y dar mantenimiento a los robots, generando de esta forma nuevos puestos de trabajo. También se libera de la excesiva carga humana dedicada a las tareas repetitivas, reubicando al personal en otros puestos, como: inspección y control de calidad.

1.4 Propuesta de solución

Se implementará un sistema automático de reconocimiento y manipulación de frutas, la propuesta abarca dos etapas: se inicia con el reconocimiento del objeto, incorporando una solución basada en visión artificial, este sistema comprende una cámara industrial encargada de capturar información del entorno y un computador para el procesamiento de las imágenes basado en herramientas de software libre. Seguidamente el proceso de manipulación, esta tarea la realiza un robot industrial de tipo articulado, encargado de posicionar y orientar correctamente su herramienta para poder agarrar la fruta y colocarla de manera ordenada en una caja de empaque.

1.5 Objetivos

1.5.1 Objetivo general

- Diseñar e implementar un sistema de reconocimiento y manipulación de frutas utilizando visión artificial y brazo robótico industrial.

1.5.2 Objetivos específicos

- Introducir a la programación de robots industriales en sus aplicaciones más comunes.
- Familiarizarse con la arquitectura y entorno de Kawasaki Robotics para el empleo del robot de la serie RS03N.
- Aplicar técnicas basadas en visión artificial para el reconocimiento de objetos y obtención de sus características.
- Desarrollar aplicaciones de visión artificial utilizando la biblioteca libre OpenCV en el entorno Visual C++.
- Configurar e instalar una cámara industrial Basler e integrar su API en nuestra aplicación de visión.
- Realizar un sistema de comunicación basado en TCP/IP que permita integrar las aplicaciones de visión y robótica.

CAPÍTULO 2

2. CONCEPTOS TEÓRICOS Y DESCRIPCIÓN DEL SISTEMA DE VISION ARTIFICIAL Y ROBOT INDUSTRIAL.

En el presente capítulo se describirá los métodos de visión artificial junto con la arquitectura y programación del robot industrial utilizado para el reconocimiento y manipulación de frutas.

2.1 Definición del sistema

El sistema está formado por una aplicación de visión artificial, la cual se encarga de la identificación y reconocimiento de las frutas mediante la extracción de la información de posición y orientación, estos datos se envían desde el computador hacia el robot a través de una red TCP/IP (creada entre los dispositivos del sistema). El robot posteriormente se encarga de manipular el objeto desde una base hasta su caja de empaque. En la figura 2.1 se muestra los componentes que realizan las tareas mencionadas.

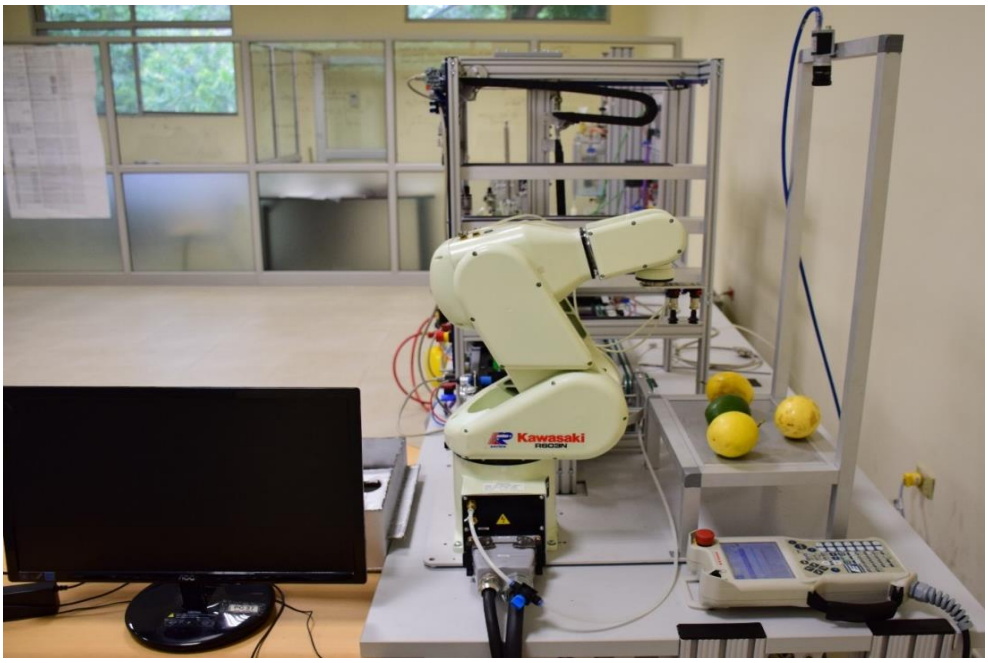

Figura 2.1: Componentes del sistema

El diagrama de bloques del sistema se muestra en la figura 2.2, aquí aparecen los componentes del proyecto y el flujo de operación del mismo. Se observa como la incorporación de visión artificial genera un lazo cerrado en el sistema, también se muestra al computador como el controlador principal del proyecto, este dispositivo maneja toda la información ya sea sensorial como de accionamiento.

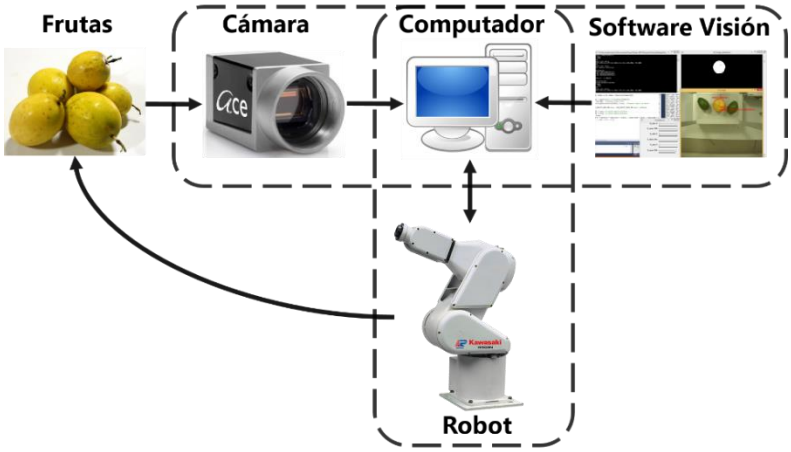

Figura 2.2: Diagrama de bloques del sistema

2.2 Condiciones del sistema

El sistema propuesto se localiza en el laboratorio de control de procesos de la Facultad de Ingeniería en Electricidad y Computación (FIEC). Las frutas elegidas para el proyecto son originalmente las que Ecuador mayormente exporta, pero además se consideran otros factores, como: el tamaño, la forma y la textura.

El fruto debe ser lo más uniforme posible debido a que no se cuenta con la medición de profundidad del objeto, por lo tanto, para este proyecto no puede haber frutos más grandes que otros ni variaciones considerables en su forma y textura ya que afectaría el agarre del robot. También se descartan aquellos frutos que sean de gran tamaño por motivo del reducido espacio libre en la zona donde se localiza el sistema de visión artificial.

Los frutos que se utilizan en el proyecto son: el maracuyá y el aguacate, los cuales estarán separados y ubicados inicialmente sobre una base. Cada uno de estos debe cumplir los criterios y condiciones expuestos en el párrafo anterior.

2.3 Sistema de visión artificial

2.3.1 Descripción

El sistema de visión artificial utiliza imágenes digitales para interpretar el entorno para el robot, establece un lazo de retroalimentación que otorga capacidades de regulación, aportado flexibilidad al proceso de manipulación. Poder tomar decisiones y estar en capacidad de modificar dinámicamente sus parámetros de agarre, vuelve al robot una máquina más inteligente. [4]

Esta aplicación considera que las frutas se encuentran aleatoriamente acomodadas en una base (véase figura 2.3), desde allí son captados por la cámara que envía esa información para ser procesada, obteniendo las características de posición y orientación de cada una de las frutas con respecto a una referencia base.

Figura 2.3: Entorno del sistema de visión artificial

2.3.2 Etapas del sistema de visión artificial

El sistema de visión artificial propuesto establece cuatro fases principales: captura, pre procesamiento, segmentación y reconocimiento (véase figura 2.4). Estas fases abarcan los dos segmentos generales de un sistema de visión que son la formación y procesamiento de imágenes. [5]

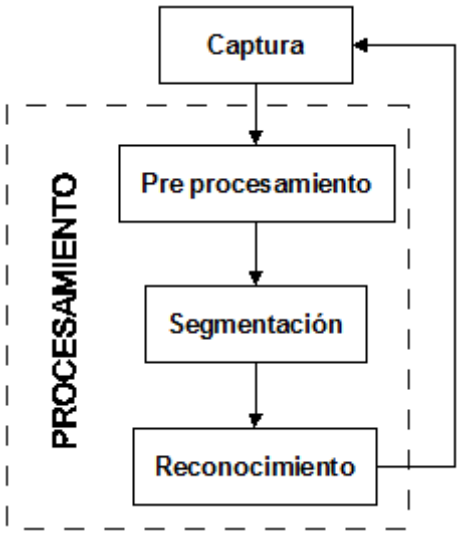

Figura 2.4: Diagrama de flujo de las etapas del sistema de visión artificial

2.4 Captura

El proceso de formación de imágenes comienza con la proyección de los rayos de luz en la escena sobre un sensor fotosensible, obteniendo una señal en representación de la imagen, luego esta información es digitalizada para poder ser procesada en un computador. [5]

2.4.1 Cámara Basler acA1300-75gc

La cámara es el principal componente de esta etapa, ya que en su interior aloja al elemento fotosensible, la cámara utilizada en este proyecto (véase figura 2.5) cuenta con un sensor de tecnología CMOS y una resolución de 1.3 Megapíxeles, aportando así con fotos de muy buena calidad. Cuenta además con una velocidad de toma de imágenes de 88

fps y una interfaz de comunicación gigabit ethernet, lo que permite capturar y enviar gran cantidad de datos en tiempos cortos. [6]

Figura 2.5: Parte frontal de la cámara Basler acA1300-75gc [6]

2.4.2 Óptica Fujinon DF6HA-1B

El objetivo de la óptica es captar los rayos luminosos de la escena y proyectarlos hacia el sensor de la cámara. Se dispone de una óptica (véase figura 2.6) con distancia focal de 6mm, logrando que las imágenes abarquen toda el área donde yacen las frutas. La óptica además posee dos perillas, una permite regular la apertura de diafragma (F1.2 – F16) lo que controla la cantidad de luz que llega al sensor mientras que la otra modifica el enfoque de la escena. [7]

Figura 2.6: Parte lateral de la óptica Fujinon DF6HA-1B [7]

2.5 Procesamiento de imágenes

La fase de procesamiento involucra la utilización de diversas técnicas implementadas por algoritmos computacionales para interpretar, extraer y modificar la información de las imágenes con el objetivo de resolver un problema en específico. Las técnicas de procesamiento varían de acuerdo a la aplicación y la consideración del desarrollador, para este proyecto se ha optado por escoger soluciones basadas en software libre.

2.5.1 OpenCV

Es una librería de visión por computador de código abierto, fue originalmente escrita en C y C++ y puede ser ejecutada en múltiples sistemas operativos. La librería contiene aproximadamente 500 funciones que abarcan muchas áreas de visión, incluyendo inspección de productos, seguridad, interfaces de usuario, calibración de cámaras, robótica, etc. [8]

2.5.2 Software

El software para el reconocimiento de frutas (véase figura 2.7) es implementado en OpenCV en su versión 3.0 debido a la solidez y amplio rango funciones de visión artificial que aporta esta librería. El entorno de desarrollo (IDE) escogido es Visual Studio 2013 utilizando el lenguaje de programación C++. Todo el sistema funciona en un computador con sistema operativo Windows 8 de 32 bits.

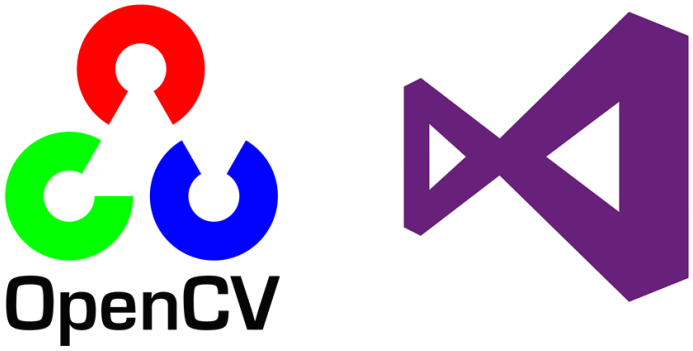

Figura 2.7: Logos de OpenCV y Visual Studio [9], [10]

2.6 Pre procesamiento

Aunque el acondicionamiento del entorno en la fase de captura es importante, la imagen digital obtenida en ocasiones debe ser previamente tratada con la finalidad de facilitar el trabajo en las etapas siguientes. Para mejorar la imagen se utilizan filtros y transformaciones geométricas que eliminan el ruido o destacan ciertas partes importantes de la misma. [5]

2.6.1 Filtro gaussiano

Se aplica un filtro gaussiano en la imagen con la finalidad de disminuir el ruido. [5] En la figura 2.8 la función ***GaussianBlur*** recibe como entrada la imagen original y la distribución gaussiana, esta última se aplica de acuerdo al tamaño del kernel, que para este caso es una matriz de 5x5. [9]

```
fc.Convert(image, ptrGrabResult);   //Convert the grabbed buffer to a pylon image
ori_img = cv::Mat(ptrGrabResult->GetHeight(), ptrGrabResult->GetWidth(), CV_8UC3, (uint8_t*)image.GetBuffer());
GaussianBlur(ori_img, fil_img, Size(5, 5), 0, 0, 4);    //Apply a filter to reduce the noise
```

Figura 2.8: Código en OpenCV para la fase de pre procesamiento

2.7 Segmentación

La segmentación consiste en separar una imagen en regiones bien definidas con la finalidad de distinguir y reconocer objetos, el método de segmentación utilizado es el basado en la umbralización. El proceso consiste en definir en la imagen un rango de pixeles que contienen el nivel de color que corresponde al objeto, mientras que el fondo posee un rango de valores diferentes. [5]

2.7.1 Espacio de color HSV

Un espacio de color es un método diseñado para definir cualquier color en base a los valores de ciertos parámetros. Para realizar la segmentación mediante umbrales se optó por trabajar en el espacio de color HSV, este modelo se basa en el modo de percibir los colores que tenemos los humanos, definiendo al color en función de los parámetros: tono, saturación y brillo. [5] En la figura 2.9 la función ***cvtColor*** se utiliza para realizar la conversión del espacio de color BGR a HSV, recibe como parámetro la imagen filtrada y el código de transformación de espacios CV_BGR2HSV. [9]

```
cvtColor(fil_img, fil_img, CV_BGR2HSV);
inRange(fil_img, Scalar(H_min, S_min, V_min), Scalar(H_max, S_max, V_max), cv_img);
morphTrans(cv_img);
```

Figura 2.9: Código en OpenCV para la fase de segmentación

2.7.2 Histograma

Es una herramienta visual que muestra la frecuencia de los valores de pixeles que se encuentran en una imagen. La gráfica establece en el eje de las abscisas los diferentes valores que pueden tomar los pixeles mientras que en el eje de las ordenadas se encuentran la cantidad de pixeles que toman dichos valores. [10] En la figura 2.10 la función ***calcHist*** se utiliza para calcular el histograma del objeto, recibiendo como parámetros la imagen en el espacio HSV, el número de canales requerido, el tamaño del histograma y el rango de intensidades HSV. [9]

```
Mat src, dst, h_hist, s_hist, v_hist;
vector<Mat> hsv_planes;
src = imread("maracuya_cut.jpg");
cvtColor(src, src, CV_BGR2HSV);
split(src, hsv_planes);

calcHist(&hsv_planes[0], 1, 0, Mat(), h_hist, 1, &hbins, &ranges[0], true, false);
calcHist(&hsv_planes[1], 1, 0, Mat(), s_hist, 1, &sbins, &ranges[1], true, false);
calcHist(&hsv_planes[2], 1, 0, Mat(), v_hist, 1, &vbins, &ranges[2], true, false);

int bin_w = cvRound((double)hist_w / hbins);
int bin_y = cvRound((double)hist_w / sbins);
int bin_z = cvRound((double)hist_w / vbins);

Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));
normalize(h_hist, h_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
normalize(s_hist, s_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
normalize(v_hist, v_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());

for (int h = 1; h < hbins; h++)
{
    int hval = h_hist.at<float>(h - 1);
    line(histImage, Point(bin_w*(h - 1), hist_h - cvRound(hval)),
        Point(bin_w*(h), hist_h - cvRound(h_hist.at<float>(h))),
        Scalar(255, 0, 0), 2, 8, 0);
    if (hval > maxhue)
    {
        maxhue = hval;
        hue = h;
    }
}
```

Figura 2.10: Código en OpenCV para la obtención del histograma

2.7.3 Binarización

Es el proceso donde las imágenes definidas por sus niveles de color son transformadas a una representación discreta, es decir dos intensidades de pixeles: mínimo y máximo. [5] La función ***inRange*** en la figura 2.9

recibe los valores de las regiones HSV que definen al objeto y convierte las imágenes a su forma binaria. [9]

2.7.4 Operaciones morfológicas

Son métodos que se aplican a las imágenes binarias para tratar problemas de formas. Dos operaciones morfológicas comunes que se utilizan son la erosión y dilatación, la primera tiene un efecto de disminuir el tamaño de los objetos y la segunda de aumentarlos. [5] En la figura 2.11 las funciones ***erode*** y ***dilate*** son utilizadas para lograr un efecto de separación de los objetos y eliminar huecos o aberturas que se puedan formar en ellos. [9]

```
void morphTrans(cv::Mat &rcv)
{
    cv::Mat erdodeImg = getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(7, 7));
    cv::Mat dilateImg = getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5));
    erode(rcv, rcv, erdodeImg);
    erode(rcv, rcv, erdodeImg);
    dilate(rcv, rcv, dilateImg);
    dilate(rcv, rcv, dilateImg);

}
```

Figura 2.11: Código en OpenCV para aplicar erosión y dilatación en la imagen

2.8 Reconocimiento

Es la última fase del sistema de visión y comprende la distinción de los objetos segmentados mediante la extracción de características. La posición y orientación son las propiedades que se deben describir en nuestro objeto, para ello se utilizan técnicas basadas en la detección de los bordes de la imagen. [5]

2.8.1 Extracción del contorno

Los bordes en una imagen son representados por una discontinuidad en los valores de los pixeles vecinos, esta información establece los límites entre los objetos y el fondo. El contorno se establece como un conjunto uniforme de bordes encontrados en una sección de la imagen binaria. [5] En la figura 2.12 la función ***findContours*** devuelve un vector con todos

los contornos de la imagen y otro vector que almacena la jerarquía con la que estos aparecen. [9]

```
bool trackObject(int &u, int &v, double &theta, cv::Mat rcv, cv::Mat &dst)
{
    cv::Mat obj_img;
    vector<vector<cv::Point>> contours;
    vector<cv::Vec4i> hierarchy;
    rcv.copyTo(obj_img);    //Copy to a new Mat object
    findContours(obj_img, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);   //Find contours
    cv::Mat drawing = cv::Mat::zeros(obj_img.size(), CV_8UC3);  //Draw contours
    for (int i = 0; i < contours.size(); i++)
    {
        drawContours(dst, contours, i, cv::Scalar(0, 0, 255), 1, 8, hierarchy, 0, cv::Point());
    }
    double refArea = 0;
    if (hierarchy.size() > 0)
    {
        int numObjects = hierarchy.size();
        if (numObjects < MAX_NUM_OBJECTS)
        {
            for (int index = 0; index >= 0; index = hierarchy[index][0])
            {
                cv::Moments mom = moments((cv::Mat)contours[index]);
                double area = mom.m00;  //Compute object area
                double per = arcLength(contours[index], true);  //Compute object perimeter
                if (area > MIN_OBJECT_AREA && area < MAX_OBJECT_AREA && area > refArea)
                {
                    u = mom.m10 / area; //x pixel object position
                    v = mom.m01 / area; //y pixel object position
                    refArea = area;
                    theta = (0.5 * atan2((2 * (mom.mu11 / area)) , ((mom.mu20 / area) - (mom.mu02 / area))));
                    drawOri(u, v, per, theta, dst);
                    return 1;
                }
            }
        }
    }
    return 0;
}
```

Figura 2.12: Código en OpenCV para la extracción del contorno y cálculo de momentos.

2.8.2 Momentos de imágenes

Los momentos en imágenes digitales son un tipo de descriptores de forma para los objetos y son calculados integrando a través de todos los pixeles de su contorno. [8] La función ***moments*** en la figura 2.12 utiliza el vector que contiene los contornos de los objetos para establecer los momentos de hasta tercer orden y mediante manipulación matemática se obtienen los parámetros descriptores: área, centro de gravedad (centroide) y orientación. [11], [9]

2.8.3 Representación visual

En la figura 2.13 se muestra el código para mostrar los datos de posición y orientación en la imagen, se utiliza las funciones de dibujo de OpenCV como ***rectangle*** para graficar un rectángulo sobre el centroide del objeto**,** ***putText*** para colocar la información de la posición en pixeles y ***arrowLine*** para mostrar la dirección en la orientación. [9]

```
void drawPos(int u, int v, double w, cv::Mat xyz, cv::Mat &rcv)
{
    string x, y, z;
    stringstream data;
    data << fixed << std::setprecision(3) << xyz.at<double>(0, 0) << " " << xyz.at<double>(1, 0) << " " << w ;
    data >> x >> y >> z;
    rectangle(rcv, cv::Point(u - 5, v - 5), cv::Point(u + 5, v + 5), cv::Scalar(0, 0, 255), 2);
    putText(rcv, "Object detected", cv::Point(u - 110, v - 110), 2, 1, cv::Scalar(0, 0, 255), 2);
    putText(rcv, to_string(u) + "," + to_string(v), cv::Point(u + 10, v + 40), 2, 1, cv::Scalar(0, 0, 255), 2);
    putText(rcv, x + "mm," + y + "mm," + z + "mm", cv::Point(u + 10, v + 80), 2, 1, cv::Scalar(0, 0, 255), 2);
}

void drawOri(int u, int v, double per, double theta, cv::Mat &rcv)
{
    int a = per*cos(theta) / 4;
    int b = per*sin(theta) / 4;
    int c = per*cos(theta + (PI / 2)) / 4;
    int d = per * sin(theta + (PI / 2)) / 4;
    arrowedLine(rcv, cv::Point(u, v), cv::Point(u + a, v + b), cv::Scalar(0, 0, 255), 2);
}
```

Figura 2.13: Código en OpenCV para mostrar la información de la fase de reconocimiento

2.9 Calibración de cámara

Los datos de posición obtenidos corresponden a proyecciones bidimensionales de la imagen dada en pixeles, es necesario conocer su representación tridimensional en milímetros para que el robot pueda interpretarla adecuadamente. Los parámetros de la cámara que realizan estas transformaciones se conocen como: intrínsecos (interiores a la cámara) y extrínsecos (dependen del entorno).

Para determinar los parámetros intrínsecos y extrínsecos se realiza un proceso de reconocimiento de puntos en una imagen patrón, es necesario recoger información de esta plantilla en varias posiciones en el espacio de visualización de la cámara para obtener bajo índice de error de calibración, además de tener un volumen de trabajo valido para los objetos. [8]

Es utilizado como patrón de calibración un tablero de ajedrez con dimensión de bloques 9x6 y 25mm como tamaño de cada cuadro. La figura 2.14 muestra el código en OpenCV para realizar el proceso de calibración, la función ***calcChessboardCorners*** define el patrón real del tablero y ***findChessboardCorners*** lo detecta, con esa información la función ***calibrateCamera*** determina los parámetros de la cámara. Se realizan 14 capturas a la plantilla, considerando la última como la posición donde estarán localizados los objetos, esta información es almacenada al final en un archivo .XML para luego cargarla al código principal del proyecto. [9]

```
if (successes < n_board)
{
    cvtColor(cv_img, gr_img, CV_BGR2GRAY);
    bool found = findChessboardCorners(gr_img,board_sz,corners2D,CALIB_CB_ADAPTIVE_THRESH+CALIB_CB_NORMALIZE_IMAGE+CALIB_CB_FAST_CHECK);
    if (found)
    {
        cornerSubPix(gr_img, corners2D, Size(11, 11), Size(-1, -1), CvTermCriteria(CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 30, 0.1));
        drawChessboardCorners(cv_img, board_sz, corners2D, found);
        coord2D.push_back(corners2D);
        coord3D.resize(n_board,corners3D);
        successes++;
        cout << "numero imagenes \n" << successes << endl;
    }
    if (successes == n_board)
    {
        cameraMatrix = initCameraMatrix2D(coord3D, coord2D, Size(FRAME_WIDTH, FRAME_HEIGHT));
        double rms = calibrateCamera(coord3D,coord2D,Size(FRAME_WIDTH,FRAME_HEIGHT),cameraMatrix,distCoeffs,rvecs,tvecs,CV_CALIB_USE_INTRINSIC_GUESS|CV_CALIB_RATIONAL_MODEL);
        cout << "Distortion coefficients: \n" << distCoeffs << endl;
        cout << "Camera Matrix: \n" << cameraMatrix << endl;
        writeParams("camcalib.xml", cameraMatrix, distCoeffs, rvecs, tvecs, rms);   //save camera calibration parameters
    }
}
```

Figura 2.14: Código en OpenCV para realizar la calibración de la cámara

2.10 Transformación de coordenadas

El paso final de la aplicación es convertir los datos de posición y orientación en las coordenadas de la cámara a las coordenadas del robot. Este proceso se logra utilizando operaciones de traslación y cambio de orientación entre estos sistemas.

2.10.1 Posición 2D a 3D

Se utilizan los parámetros intrínsecos y extrínsecos obtenidos en la calibración para transformar la posición del objeto de pixeles a milímetros, es decir pasar de un sistema bidimensional a uno tridimensional. El modelo pin-hole de la cámara nos permite describir matemáticamente las relaciones entre estos parámetros. [8]

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = A[R|t]\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} \quad (2.1)$$

O

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & 0 & C_x\\0 & f_y & C_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} \quad (2.2)$$

Reescrita

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & 0 & C_x\\0 & f_y & C_y\\0 & 0 & 1\end{bmatrix}\left(\begin{bmatrix}r_{11} & r_{12} & r_{13}\\r_{21} & r_{22} & r_{23}\\r_{31} & r_{32} & r_{33}\end{bmatrix}\begin{bmatrix}X\\Y\\Z\end{bmatrix} + \begin{bmatrix}t_1\\t_2\\t_3\end{bmatrix}\right) \quad (2.3)$$

O

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = A\left(R\begin{bmatrix}X\\Y\\Z\end{bmatrix} + t\right) \quad (2.4)$$

O

Donde:

- (X, Y, Z) son las coordenadas tridimensionales del objeto con respecto al sistema de coordenadas en la cámara.
- (u, v) son las coordenadas de proyección del objeto en pixeles.
- A es la matriz que contiene los parámetros intrínsecos de la cámara.
- R y t son las matrices de rotación y traslación de la cámara.

El parámetro s corresponde al factor de escalamiento y es necesario calcularlo inicialmente. Para ello se utiliza la ecuación 2.4 y se evalúa el parámetro del eje Z (altura del objeto), el cual es un dato previamente conocido para este proyecto. La solución quedaría expresada de la siguiente manera.

$$R^{-1}A^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix}\ s\ =\begin{bmatrix}X\\Y\\Z_h\end{bmatrix}+R^{-1}t \qquad (2.5)$$

Finalmente, para transformar la posición del objeto a las coordenadas tridimensionales de la cámara se aplica la misma ecuación 2.4 pero despejando los valores X, Y, Z como se muestra a continuación.

$$\begin{bmatrix}X\\Y\\Z\end{bmatrix}=R^{-1}\left(sA^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix}-t\right) \qquad (2.6)$$

2.10.2 Posición de la herramienta del robot

Los puntos X, Y, Z ahora definidos en las coordenadas tridimensional de la cámara deben ser transformados a las coordenadas del robot, para llevar a cabo esta tarea es necesario aplicar una rotación y traslación sobre el sistema original. La ecuación general que aplica este principio se muestra en la ecuación 2.7. [12]

$$XYZ_{world}=R_{eff}\left(\begin{bmatrix}X\\Y\\Z\end{bmatrix}-tcp_{off}\right)+\begin{bmatrix}X_0\\Y_0\\Z_0\end{bmatrix} \qquad (2.7)$$

Donde:

- XYZ_{world} son las coordenadas con respecto al sistema de coordenadas del mundo (base del robot).
- R_{eff} es la matriz de cambio de orientación de la cámara con respecto al robot.
- tcp_{off} es una matriz que contiene una desviación en la distancia del tcp del robot con respecto al punto X, Y, Z.
- X_0, Y_0, Z_0 es el vector de traslación de la cámara con respecto al robot.

El código que realiza las operaciones entre matrices se muestra en la figura 2.15, se han creado dos funciones ***TransCoord*** para obtener los valores XYZ y ***poseRobot*** para los valores XYZworld.

```
void transCoord(int u, int v, double hobj, cv::Mat &xyzcam, cv::Mat cameraMatrix, cv::Mat distCoeffs, cv::Mat rvecs, cv::Mat tvecs)
{
    cv::Mat uvPoint, tempMat, tempMat2, rotationMatrix;
    double s;
    uvPoint = cv::Mat::ones(3, 1, CV_64F);
    uvPoint.at<double>(0, 0) = u;
    uvPoint.at<double>(1, 0) = v;
    Rodrigues(rvecs, rotationMatrix);
    tempMat = rotationMatrix.inv() * cameraMatrix.inv() * uvPoint;
    tempMat2 = rotationMatrix.inv() * tvecs;
    s = hobj + tempMat2.at<double>(2, 0);
    s /= tempMat.at<double>(2, 0);
    xyzcam = rotationMatrix.inv() * ((s * cameraMatrix.inv() * uvPoint) - tvecs);
    cout << endl << "XYZ:" << endl << xyzcam << endl;
}

void poseRobot(cv::Mat xyzhome, cv::Mat &xyzcoord, double theta, const double offset)
{
    //offset matrix
    cv::Mat tcpoff = cv::Mat::zeros(3, 1, CV_64F);
    tcpoff.at<double>(0, 0) = offset * cos(theta);
    tcpoff.at<double>(1, 0) = offset * sin(theta);
    tcpoff.at<double>(2, 0) = 0;
    //rotation matrix patter to robot effector
    cv::Mat roteffector = cv::Mat::zeros(3, 3, CV_64F);
    roteffector.at<double>(0, 1) = 1;
    roteffector.at<double>(1, 0) = 1;
    roteffector.at<double>(2, 2) = 1;
    //object coordinate center at robot frame
    xyzcoord = (roteffector*(xyzcoord - tcpoff)) + xyzhome;
    cout << endl << "XYZworld:" << endl << xyzcoord << endl;
}
```

Figura 2.15: Código en OpenCV para la transformación de posición

2.10.3 Orientación de la herramienta del robot

El ángulo theta encontrado representa la orientación del objeto con respecto al sistema bidimensional de la cámara. Ya que la herramienta del robot debe posicionarse correctamente para capturar al objeto, la orientación debe estar definida en el sistema de coordenadas del robot. Para ello se aplican una serie de cambios de orientación entre los sistemas objeto-cámara-robot. [12]

$$R_{world} = R_{home}\, R_{matrix}\, R_{objetct} \qquad (2.8)$$

Donde:

- Rworld es la matriz de rotación con respecto a las coordenadas del mundo
- Robject es la matriz de rotación del objeto
- Rmatriz es la matriz de rotación del objeto con respecto a la cámara
- Rhome es la matriz de rotación de la cámara con respecto al robot

Una vez encontrada la matriz de rotación con respecto a las coordenadas del mundo se debe transformar esta representación de la orientación, ya

que el robot utiliza ángulos de Euler con la notación ZYZ. El código que realiza las operaciones entre matrices se muestra en la figura 2.16, se han creado dos funciones ***euler2Rotation*** transformar de ángulos de Euler a matriz de cambio de orientación y ***rotation2Euler*** para hacer el proceso contrario.

```
void orientRobot(cv::Mat rvecs, double theta, cv::Mat eulerhome, cv::Mat &eulerworld)
{
    cv::Mat rmatrix, inter, rothome;
    //Create object rotation matrix (camera frame)
    cv::Mat rotobject = cv::Mat::zeros(3, 3, CV_64F);
    rotobject.at<double>(0, 0) = cos(theta);
    rotobject.at<double>(0, 1) = sin(theta);
    rotobject.at<double>(1, 0) = -sin(theta);
    rotobject.at<double>(1, 1) = cos(theta);
    rotobject.at<double>(2, 2) = 1;
    //Compute rotation matrix from object to fixed frame chessboard
    Rodrigues(rvecs, rmatrix);
    rmatrix.at<double>(0, 2) = 0;   //force rotation matrix to z = 0
    rmatrix.at<double>(1, 2) = 0;
    rmatrix.at<double>(2, 2) = 1;
    rmatrix.at<double>(2, 0) = 0;
    rmatrix.at<double>(2, 1) = 0;
    inter = rmatrix * rotobject;
    //Transform from robot euler representation to rotation matrix
    euler2Rotation(eulerhome, rothome);
    cv::Mat rotworld = rothome * inter;
    rotation2Euler(eulerworld, rotworld);
    eulerworld = (eulerworld * 180) / PI;
    cout << endl << "theta: " << theta << endl;
    cout << endl << "rotobject: " << endl << rotobject << endl;
    cout << endl << "rmatrix: " << endl << rmatrix << endl;
    cout << endl << "rothome: " << endl << rothome << endl;
    cout << endl << "rotworld: " << endl << rotworld << endl;
    cout << endl << "eulerworld: " << endl << eulerworld << endl;
}
```

Figura 2.16: Código en OpenCV para la transformación de orientación

2.11 Robot industrial

En la figura 2.17 se muestra el modelo de robot que posee el laboratorio de control de procesos de la Facultad de Ingeniería en Electricidad y Computación (FIEC).

El manipulador antropomórfico es de 6 grados de libertad de la serie R del fabricante Kawasaki, esta familia pertenece a los robots de propósito general,

aunque es común utilizarlo en aplicaciones como ensamblaje, desensamblaje, inspección, manejo de material, etc. [13]

Figura 2.17: Robot industrial Kawasaki RS03N

2.11.1 Características generales

En la tabla 1, se muestran las especificaciones técnicas de mayor importancia del manipulador robótico.

Carga	3 Kg
Alcance Horizontal	620 mm
Alcance Vertical	967 mm
Repetitividad	±0.02mm
Velocidad máxima	6000 mm/s
Peso	20 Kg
Grado de Protección	IP54

Tabla 1. Especificaciones del robot Kawasaki RS03N. [13]

2.11.2 Características mecánicas

El robot posee seis articulaciones de tipo rotacional, nombradas como JT1,… JT6 de acuerdo a su incorporación en el manipulador. Estas articulaciones se presentan en la figura 2.18 y en la tabla 2 se muestran sus características mecánicas. [14]

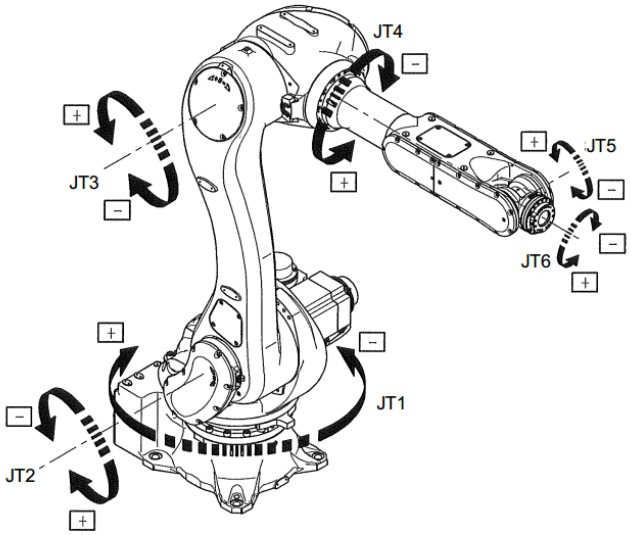

Figura 2.18: Representación de las articulaciones del robot Kawasaki RS03N [15]

Rango de movimiento	JT1	±160°
	JT2	+150° y -60°
	JT3	+120° y -150°
	JT4	±360°
	JT5	±135°
	JT6	±360°
Velocidad máxima	JT1	360°/s
	JT2	250°/s
	JT3	2250°/s
	JT4	540°/s
	JT5	225°/s
	JT6	540°/s
Torque máximo	JT4	5.8 N·m
	JT5	5.8 N·m
	JT6	2.9 N·m

Tabla 2. Características mecánicas del robot Kawasaki RS03N [13]

2.11.3 Zona de trabajo

En la figura 2.19 el área celeste representa la zona de trabajo, esta región indica las posiciones donde el punto P puede moverse libremente. Todas las distancias mostradas son las dimensiones del robot y están medidas en milímetros. [13]

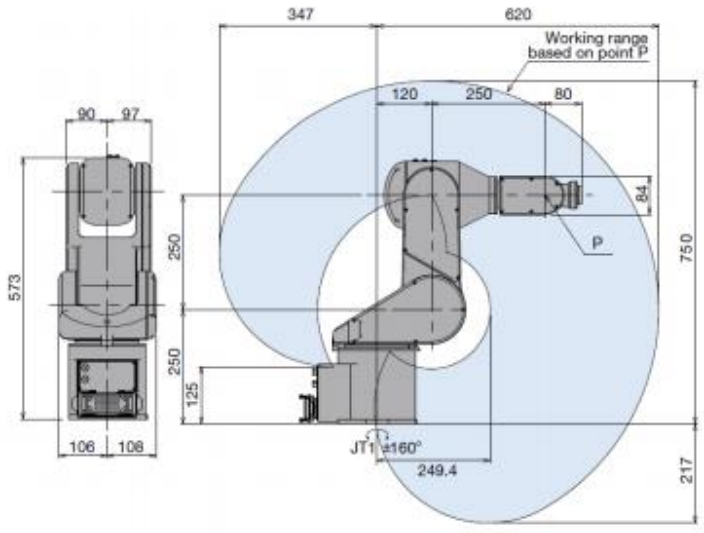

Figura 2.19: Área de trabajo del robot Kawasaki RS03N [13]

2.11.4 Componentes del manipulador

Las partes que constituyen al robot son: estructura mecánica, controlador, unidad de servicio manual o “teach pendant” y el efector. Donde:

- **Estructura mecánica:** Es el cuerpo del robot y está conformado por sensores y actuadores que permiten realizar algún tipo de movimiento. [14]
- **Controlador Serie E70:** Su función es de regular cada uno de los movimientos del manipulador, las acciones, cálculos y el procesamiento de la información. El controlador puede recibir y enviar señales a otros dispositivos (por medio de señales de entrada/salida) y es capaz de almacenar programas. [16]
- **Teach Pendant:** Es un tipo de interfaz HMI que permite diseñar, modificar y consultar programas que se cargan al manipulador industrial. [17]
- **Efector:** Herramienta que permite manipular objetos. El robot está equipado con ventosas neumáticas.

2.12 Sistema de manipulación de frutas

2.12.1 Descripción

El objetivo del robot (véase figura 2.20) es la manipulación de frutas localizadas en una base fija. El método que se emplea para esta aplicación es conocido como una tarea de “pick and place”, en la cual es fundamental conocer la posición y orientación de los elementos. La tarea consiste en desplazar la herramienta del robot a la localización de las frutas, transportarlas hacia otra ubicación y como acción final ordenarlas dentro de una caja. Estas acciones son diseñadas en un programa, escrito en el lenguaje de programación del robot Kawasaki RS03N llamado “lenguaje AS”.

Para la aplicación del “pick and place”, se establece una posición intermedia entre la posición inicial y final de las frutas, la cual llamaremos “home”. El robot se ubica en la posición home en estos casos: Antes de manipular algún objeto, durante la ejecución de la aplicación y después de ubicar el objeto dentro de la caja.

Figura 2.20: Robot industrial Kawasaki RS03N del laboratorio de control de procesos

2.12.2 Algoritmo

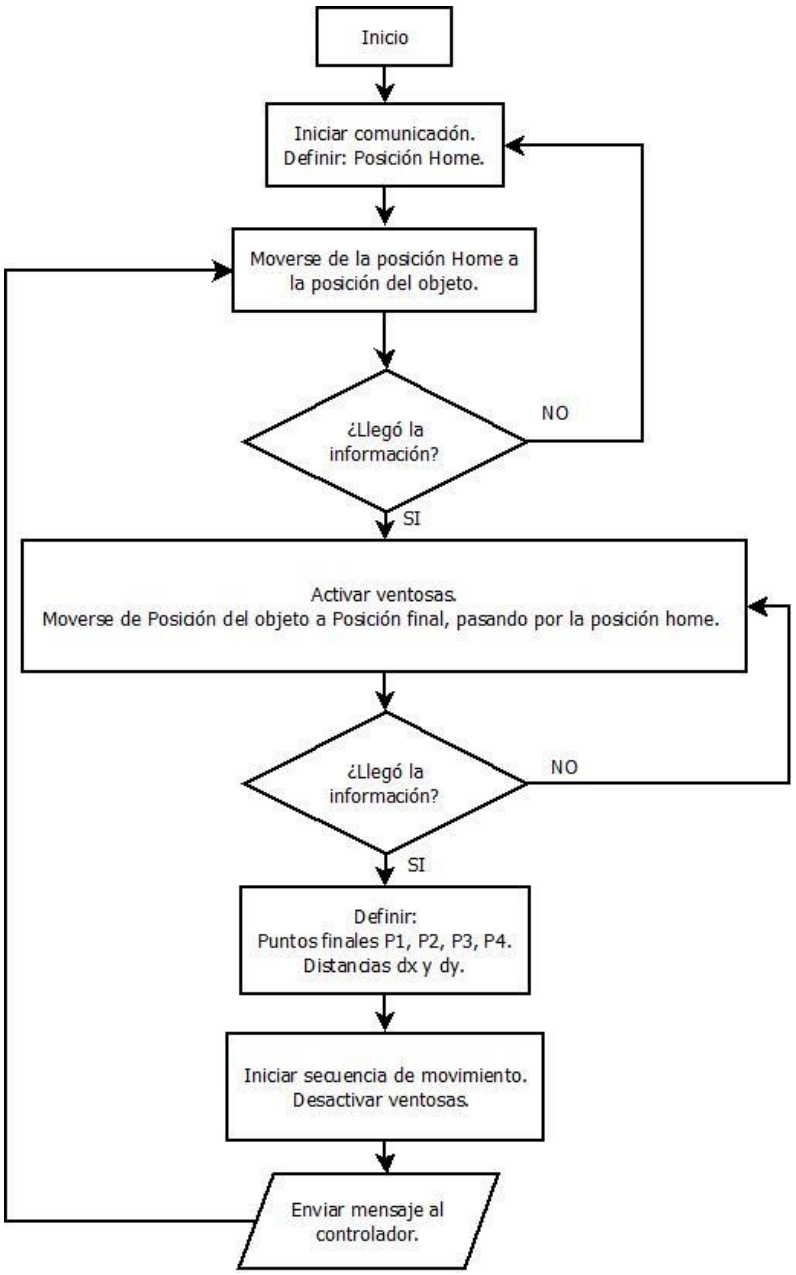

Figura 2.21: Diagrama de flujo para el algoritmo del sistema de manipulación [18]

El diagrama de flujo de la figura 2.21 describe el comportamiento del robot, lo primero que hace el algoritmo es validar la comunicación entre el computador y la tarjeta de red del controlador, una vez establecida la comunicación, el robot recibe instrucciones del controlador para ejecutar los siguientes movimientos: El robot se traslada desde la posición home hacia la posición del objeto, aquí tiene la orden de activar las ventosas para recoger el objeto, y desde la posición del objeto hacia la posición final, transporta el objeto hacia la canasta pasando por la posición home. [18]

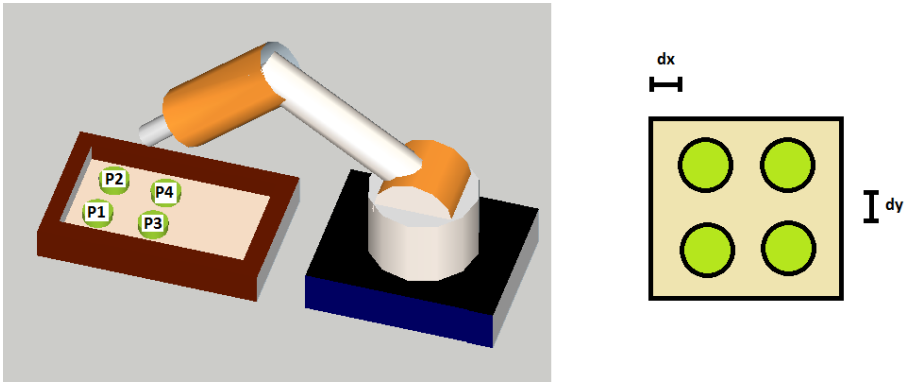

Figura 2.22: Ubicación de las posiciones finales (P1, P2, P3 y P4) de las frutas

Una vez que el objeto llega a la canasta, se tienen cuatro posiciones finales (P1, P2, P3 y P4) y se definen las distancias (dx, dy) (véase figura 2.22), estos parámetros definen el ordenamiento matricial. Cuando el robot llega a la posición final, el controlador recibe una instrucción para que se desactiven las ventosas y así ubicar el objeto en la respectiva posición. [18]

2.13 Comunicación

Todos los dispositivos del sistema están conectados a través de una red TCP/IP, tal como se muestra en la figura 2.23. El computador inicialmente establece la comunicación con la cámara en el proceso de captura y luego la computadora con el robot en la parte final del reconocimiento. El kit de desarrollo de software de la cámara es utilizado en la primera parte del sistema, mientras que para la segunda se crea una comunicación socket entre la computadora y el robot, realizando una programación de un cliente y servidor respectivamente.

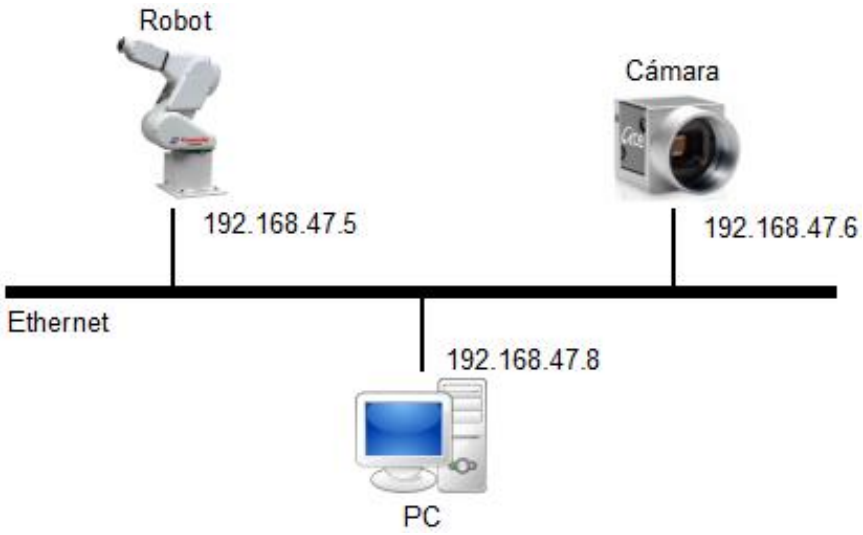

Figura 2.23: Diagrama de red ethernet del sistema

2.13.1 Cámara - PC

Para efectuar la comunicación entre la cámara y el computador es necesario instalar el paquete de software pylon 5.0, el cual en su versión de desarrollador permite agregar el SDK para el lenguaje C++. Con estas herramientas instaladas se puede incluir en el proyecto de Visual Studio las nuevas librerías y utilizar las funciones del API para realizar la captura de imágenes. [6]

2.13.2 Robot - PC

La comunicación entre el robot y el computador se establece mediante una red socket, un mecanismo que se basa en la arquitectura cliente-servidor. Para esto se agrega un programa de servidor en el controlador del robot y uno de cliente en el entorno de Visual Studio, el proceso de comunicación del sistema se muestra en la figura 2.24. [19]

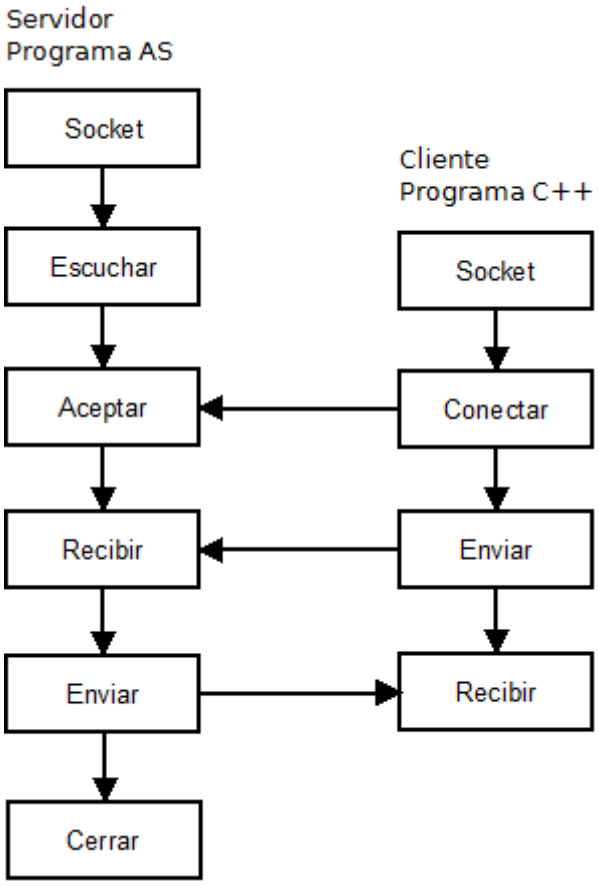

Figura 2.24: Diagrama de bloques del proceso de comunicación socket entre cliente y servidor [20]

En el programa del servidor se inicia con la instrucción en lenguaje AS ***TCP_LISTEN***, la cual se encarga de crear el socket, asignar el respectivo puerto y esperar por la solicitud del cliente, cuando esto ocurre ***TCP_ACCEPT*** permite dar acceso a la conexión. Para realizar la recepción y envió de datos se utiliza ***TCP_RECV*** y ***TCP_SEND*** respectivamente, y ***TCP_CLOSE*** se encarga de cortar la comunicación y cerrar el socket. [20]

En el programa del cliente ocurre algo similar, pero utilizando las funciones definidas en la librería WinSock para C++. La diferencia radica en que el cliente se encarga de solicitar la conexión mientras que el servidor es quien escucha los requerimientos. Las funciones que realizan el proceso de comunicación por parte del cliente son: ***socket***, ***connect***, ***recv*** y ***send***. [21] [22]

CAPÍTULO 3

3. RESULTADOS DEL SISTEMA DE RECONOCIMIENTO Y MANIPULACIÓN DE FRUTAS

En este capítulo se muestra los resultados obtenidos en la implementación del sistema de reconocimiento y manipulación de frutas y se analiza su desempeño en función de los criterios establecidos.

3.1 Funcionamiento del sistema de reconocimiento de frutas

3.1.1 Captura

El sistema de visión parte en la fase de captura, la figura 3.1 corresponde a la imagen obtenida al realizar la adquisición y transformación al tipo de dato manejado en OpenCV. Se observa que la imagen mantiene los colores y nitidez que permite la resolución de la cámara, también se aprecia en la adquisición continua que no existe ningún retardo que afecte significativamente este proceso.

Figura 3.1: Captura de imagen

3.1.2 Pre-procesamiento

La figura 3.2 muestra la imagen luego de la etapa de pre procesamiento, en la cual se realizó un filtrado de la señal. Como resultado se observan

menos imperfecciones, pero aparece un ligero efecto de desenfoque, una característica común en el filtro gaussiano.

Figura 3.2: Imagen luego de la etapa de pre procesamiento.

3.1.3 Segmentación

La figura 3.3 muestra la imagen en el inicio del proceso de segmentación, es decir la transformación al espacio de color HSV. Aunque para el ojo humano esta representación parezca algo extraña, se logra observar como resalta con mayor intensidad las regiones donde se encuentran las frutas.

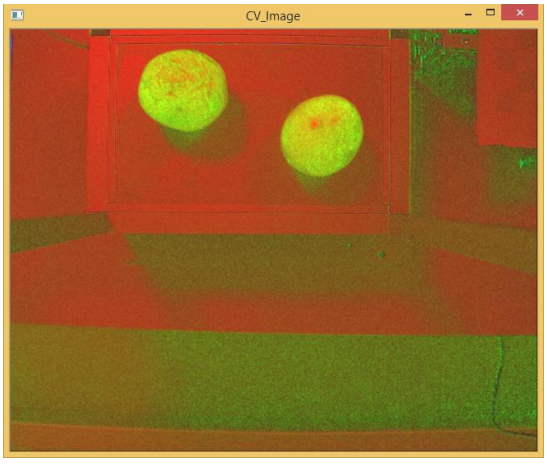

Figura 3.3: Imagen en el espacio de color HSV

El histograma de la imagen se muestra en la figura 3.4, donde las gráficas azul, verde y rojo representan los canales HSV. Con estas gráficas se determinan los rangos de valores HSV que definen al objeto, los cuales se muestran en la tabla 3.

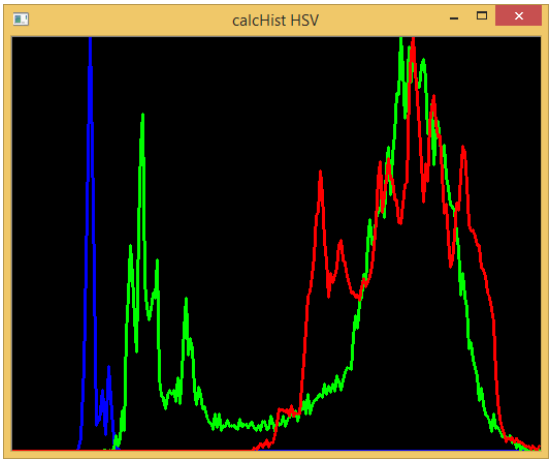

Figura 3.4: Gráfica del Histograma para el maracuyá

		Maracuyá	Aguacate
H	**Max**	30	49
	Min	23	30
S	**Max**	256	256
	Min	120	90
V	**Max**	256	130
	Min	120	40

Tabla 3. Rangos de valores HSV de las frutas

La imagen en su representación binaria se muestra en la figura 3.5, los colores que se observan son el negro y blanco (0 y 255 pixeles) que corresponden al fondo y los objetos respectivamente.

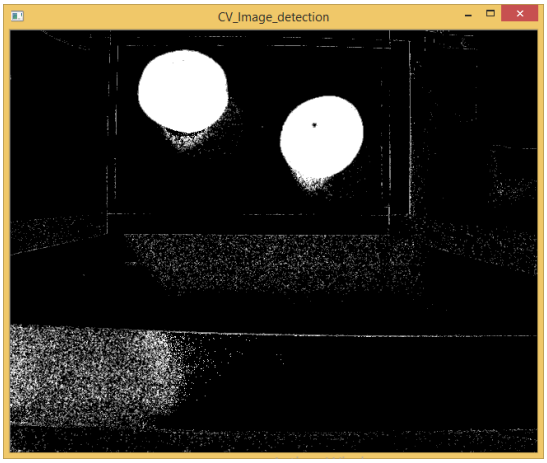

Figura 3.5: Imagen en su forma binaria

La figura 3.6 muestra el resultado de la aplicación de erosión y dilatación en la imagen, se logran eliminar las pequeñas manchas que no corresponden al objeto y se rellenan los agujeros que podrían aparecen en estos. También se consigue diferenciar mejor a los objetos a pesar de estar muy juntos, esto es importante para que el software no los interprete como un solo elemento de mayor tamaño.

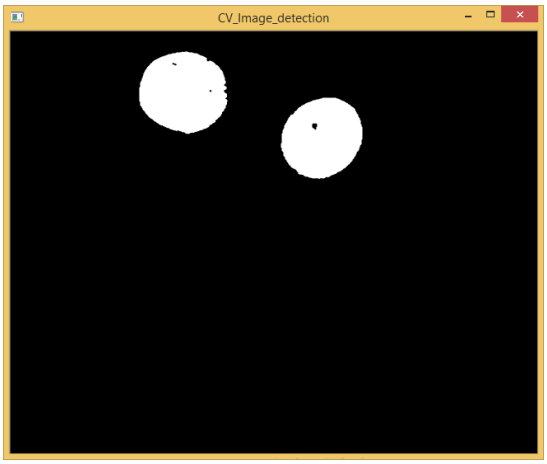

Figura 3.6: Imagen después del proceso de erosión y dilatación

3.1.4 Reconocimiento

Se observa en la figura 3.7 el dibujo de los contornos en los objetos detectados, los cuales presentan buen seguimiento a la forma de la fruta

y no muestran discontinuidad, esta información puede ser utilizada sin problemas para calcular el centroide y los momentos.

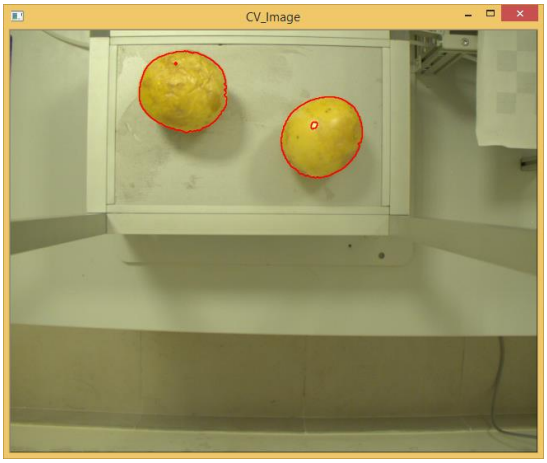

Figura 3.7: Detección de los contornos en los objetos

En la figura 3.8 se observa la ubicación del centroide en el objeto y el valor en pixeles de ese punto. La orientación se muestra a través de una semirrecta con respecto al eje de coordenadas, ubicado en la parte superior izquierda de la imagen.

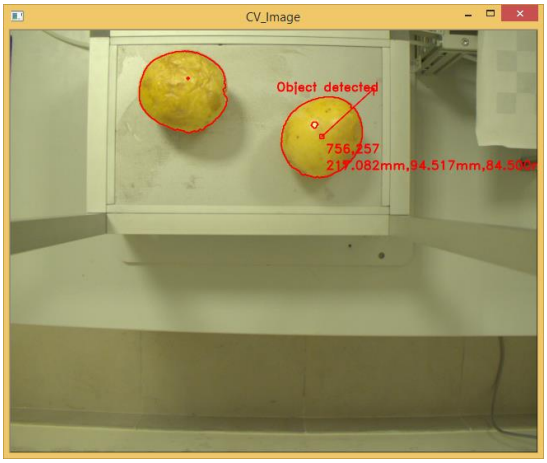

Figura 3.8: Ubicación del centroide y orientación del objeto en la imagen

La información de posición que se envía al robot XYZworld se muestra del lado izquierdo de la figura 3.9, estos datos son el resultado de la aplicación de las ecuaciones 2.6 y 2.7. También se observa la posición

en milímetros XYZ del centroide del objeto con respecto a las coordenadas tridimensionales de la cámara.

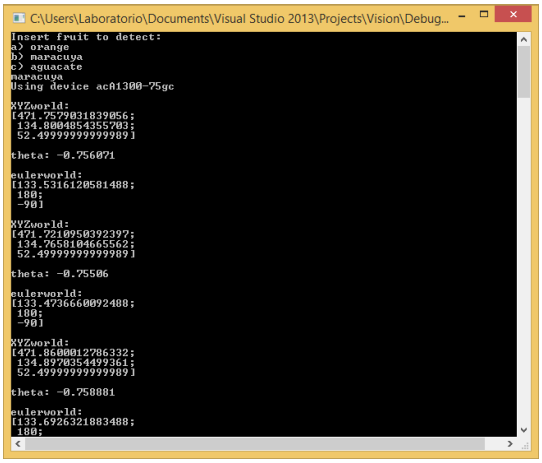

Figura 3.9: Valores de posición y orientación del objeto detectado

Los ángulos de Euler que definen la orientación del efector en el robot con respecto al objeto se muestran en la figura 3.9. Estos valores equivalen a la transformación del ángulo theta con respecto a las coordenadas de la cámara.

3.2 Funcionamiento del sistema de manipulación de frutas

El proceso de manipulacion de la fruta por parte del robot sigue el algoritmo de la figura 2.22 mostrado en el capitulo 2. A continuación se presenta la puesta en marcha del sistema y las acciones del robot para esta aplicación.

3.2.5 Selección de la fruta

En la figura 3.10, se muestra la zona de agarre del robot, que cuenta con unas dimensiones de 27x40 cm, dentro de esta área yacen las frutas válidas que se utilizan en el proyecto. El software de visión se encarga de elegir que fruta será detectada, por lo tanto, a pesar que existan otras frutas en la base, el robot únicamente manipula las que pertenezcan al tipo escogido.

Figura 3.10: Maracuyás y aguacates en la zona de agarre del robot

3.2.6 Agarre y traslado de la fruta

El robot parte desde su posición home y se desplaza hacia la fruta, alineando su herramienta con respecto a esta y sujetandola activando las ventosas. En la figura 3.11 del lado izquiero se observa como las ventosas recaen en la posición central de la fruta detectada, mientras que en el lado derecho se muestra como se modifica la orientación de la herramienta para adaptarse a la forma del otro tipo de fruta.

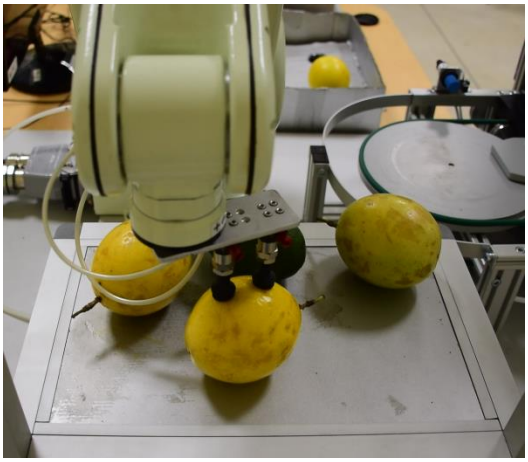

Figura 3.11: Posición y orientación de la herramienta durante el agarre de las frutas

Una vez sostenida la fruta por parte de la herramienta del robot, este se aleja de la zona de agarre y se traslada hacia el punto de descarga. En la figura 3.12 se observa al robot realizando este proceso.

Figura 3.12: Robot en proceso de traslado de la fruta

3.2.7 Empaquetado de las frutas

Luego de pasar por la selección, agarre y el posterior traslado de la fruta se tiene como fase final el empaquetado, que consiste en llenar una caja con el producto. La caja aloja un máximo de 4 frutos distribuidos de forma matricial, en donde cada uno de estos es colocado ordenadamente en una posición disponible.

En la figura 3.13 se observa como el robot realiza el proceso de empaque, ubicando las frutas en las posiciones disponibles P1, P2, P3 y P4. Finalmente en la figura 3.14 se muestra la caja de empaque completa con las cuatro frutas colocadas.

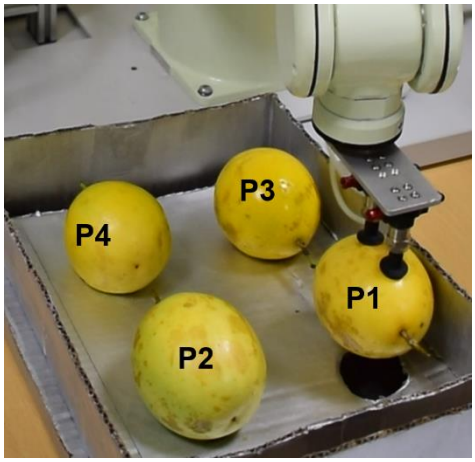

Figura 3.13: Ordenamiento en forma matricial de las frutas

Figura 3.14: Caja llena con el producto final

3.3 Pruebas de agarre de las frutas

Para este análisis se ubicaron las frutas de tal manera que abarcaron gran parte de la zona de trabajo de la cámara (véase figura 3.15). Con esto se pretende verificar que el robot este en capacidad de colocar su herramienta adecuadamente en cualquiera que sea la ubicación de las frutas en la base y realice su trabajo de manipulación eficientemente.

Figura 3.15: Frutos distribuidos en la zona de trabajo

En la figura 3.16 se muestra como la herramienta se posiciona y orienta justo en el centro de los frutos en las cuatro ubicaciones establecidas. En todos estos casos el robot fue capaz de atrapar al maracuyá con las ventosas y trasladarlo hacia la zona de empaque, dando como resultado que la herramienta del robot puede atrapar cualquier fruto que se encuentre dentro de la zona de trabajo.

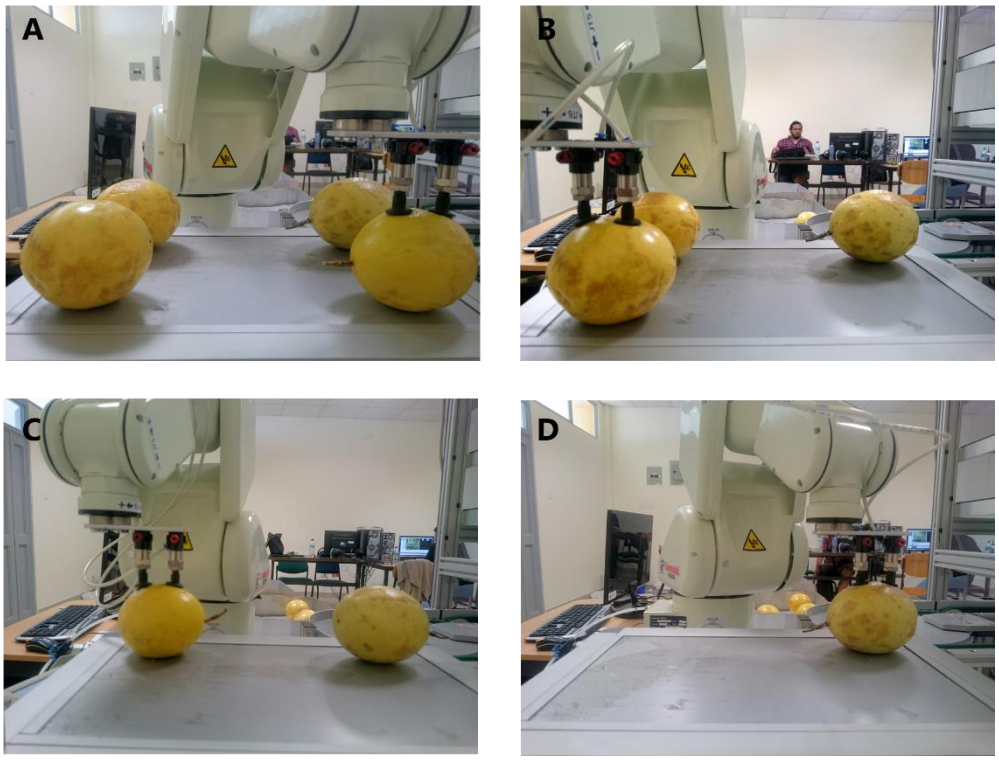

Figura 3.16: Agarre de la herramienta del robot para las cuatro ubicaciones de las frutas

Si existiese en algún momento una variación en el punto de agarre de la herramienta se debe realizar una nueva calibración de la cámara y definir un nuevo punto referencia del TCP del robot con respecto a la base de las frutas. Luego se verifica el correcto funcionamiento del agarre repitiendo las pruebas antes mencionadas.

3.4 Pruebas de velocidad de agarre y traslado de las frutas

Para esta prueba el robot realizó el agarre del maracuyá en una única posición en la zona de trabajo (véase figura 3.17). Los parámetros que se modifican son la velocidad de monitoreo y programación. Estos parámetros están relacionados, y su producto establece la velocidad final de movimiento del robot como porcentaje de la velocidad máxima. [23]

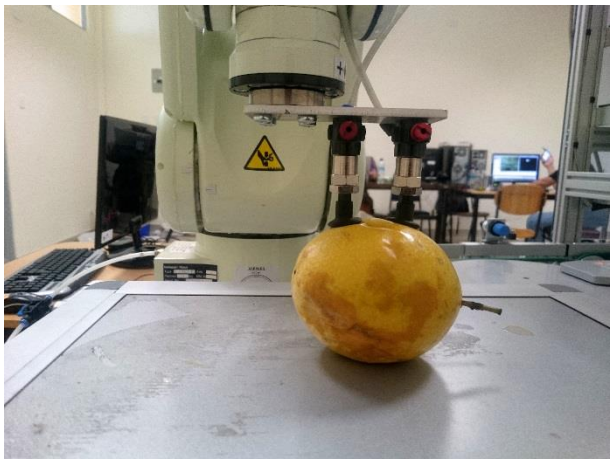

Figura 3.17: Maracuyá en la posición de agarre para la prueba de velocidad

En la tabla 4 se muestran las columnas con los valores de las velocidades escogidos. Para la velocidad de programación se ha tomado el 100% en todos los casos, siendo así la velocidad de monitoreo la que se modifica partiendo desde el 10%, en la siguiente columna se presenta su producto y como se mencionó anteriormente establece la velocidad final de ejecución.

Velocidad Programación[%]	Velocidad Monitoreo[%]	Velocidad Final[%]	Tiempo[s]	Agarre
100	10	10	17.7	✓
100	20	20	9	✓
100	30	30	6	✓
100	40	40	5	✓
100	**50**	**50**	**4**	✓
100	55	55	3.9	✓
100	60	60	3.6	✓
100	65	65	3.5	✓
100	70	70	-	✘

Tabla 4. Velocidades y tiempos de manipulación de las frutas

Para cada prueba realizada se ha tomado el tiempo que tarda el robot en agarrar el fruto y llevarlo hasta la caja de empaque exitosamente. En la primera prueba se aprecia un tiempo de ejecución muy largo de 17.7 segundos, pero este se reduce considerablemente a unos 4 segundos a partir de la velocidad de 50%. El sistema falla al 70% de la velocidad, la herramienta balancea demasiado la fruta y la suelta bruscamente.

Se debe evitar trabajar con velocidades superiores el 50%, ya que al aumentar la velocidad no se logran significativos cambios en el tiempo de manipulación, pero en cambio la fruta si puede llegar a ser afectada por los movimientos bruscos y los constantes balanceos que se generan en las altas velocidades con este tipo de herramienta de agarre.

CONCLUSIONES Y RECOMENDACIONES

Se comprobó que el sistema robotizado incorporado con visión artificial puede realizar adecuadamente el trabajo de empaquetamiento de frutas, produciendo tiempos de trabajo de alrededor de 4 segundos por fruta, optimizando de esta forma el proceso. En la etapa de manipulación se verificó que el robot no provoque maltratos o golpes en las frutas, con el fin de mantener la calidad y evitar pérdidas de producción, es recomendable el uso de otro tipo de herramienta de agarre para el robot, ya que las ventosas no son la mejor opción para manipular frutas, debido a sus variadas formas y delicadas texturas que se podrían ver afectadas por las altas velocidad de movimiento. Para la trayectoria del robot se estableció un punto inicial, intermedio y final, además se determinaron restricciones de velocidad y precisión para cada movimiento del robot de tal manera que se asegure la suavidad de la trayectoria.

En la etapa de visión artificial se desarrolló una aplicación que fue capaz de realizar un reconocimiento en tiempo real y extraer las características de posición y orientación de las frutas y adaptar esa información a la cinemática del robot, el uso de herramientas de software libre nos permitió crear una aplicación muy sólida, veloz y escalable, con capacidad de agregar mejoras e integrarla en cualquier robot industrial con relativa facilidad, es aconsejable incluir en el sistema de visión artificial la medición de la profundidad, de tal forma que se convierta en una aplicación 3D y los frutos puedan ser manipulados sin condiciones de similitud de dimensiones entre ellos. Se sugiere una mejora en el proceso de segmentación de imágenes, con el fin de poder incluir frutos que posean distintas tonalidades en su color, como es el caso del mango de exportación.

BIBLIOGRAFÍA

[1] Corporación Mexicana de Investigación en Materiales, «Centros Públicos de Investigación CONACYT,» [En línea]. Available: http://centrosconacyt.mx/objeto/robotica/. [Último acceso: 30 Julio 2016].

[2] Interempresas, «Intermpresas: Ferias Virtuales y eMagazines para la industria y la empresa,» [En línea]. Available: http://www.interempresas.net/Robotica/Articulos/100711-Aplicaciones-y-tendencias-en-robotica-para-la-industria-de-la-alimentacion.html. [Último acceso: 30 Julio 2016].

[3] PRO ECUADOR, «Análisis Sector Frutas No Tradicionales | PRO ECUADOR,» [En línea]. Available: http://www.proecuador.gob.ec/wp-content/uploads/2013/11/PROEC_AS2012_FRUTAS.pdf. [Último acceso: 30 Julio 2016].

[4] E. Á. Sobrado Malpartida , «Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot,» Creative Commons, Lima, 2003.

[5] J. F. Vélez Serrano, A. B. Moreno Díaz, Á. Sánchez Calle y J. L. Esteban Sánchez-Marín, Visión por computador, Madrid: Creative Commons , 2003.

[6] Basler AG, «Basler AG - Industrial Camera Manufacter,» [En línea]. Available: http://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca1300-75gc. [Último acceso: 26 Mayo 2016].

[7] Fujifilm Holdings Corporation, «Fujifilm USA,» [En línea]. Available: http://www.fujifilmusa.com/products/optical_devices/machine-vision/1-2-15/df6ha-1b/. [Último acceso: 26 Mayo 2016].

[8] G. Brandski y A. Kaehler, Learning OpenCV Computer Vision with the OpenCV Library, Estados Unidos: O'Reilly Media, 2008.

[9] OpenCV, «OpenCV | OpenCV,» [En línea]. Available: http://docs.opencv.org/2.4/modules/refman.html. [Último acceso: 11 Junio 2016].

[10] A. De La Escalera, Visión por Computador Fundamentos y Métodos, España: Pearson Educación, 2001.

[11] J. Kilian, «Toby Breckon Reader, Computer Vision and Image Processing,» [En línea]. Available: http://breckon.eu/toby/teaching/dip/opencv/SimpleImageAnalysisbyMoments.pdf. [Último acceso: 26 Junio 2016].

[12] J. J. Craig, Introduction to Robotics Mechanics and Control, Upper Saddle River, New Jersey: Pearson Education, Inc, 2005.

[13] Kawasaki Heavy Industries, Ltd, «RS003 robot datasheet,» 2013. [En línea]. Available: https://robotics.kawasaki.com/userAssets1/productPDF/RS003N.pdf. [Último acceso: 15 Mayo 2016].

[14] A. Barrientos, Fundamentos de Robótica, España: S.A. MCGRAW-HILL / Interamericana de España, 2007.

[15] Kawasaki Heavy Industries, Ltd, «Kawasaki Robot Controller E Series Operation Manual,» Estados Unidos, 2013.

[16] V. R. González, «Estructura de un robot industrial,» 2002-2003. [En línea]. Available: http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/sistema/morfologia.htm. [Último acceso: 25 Mayo 2016].

[17] P. Gonsa y A. Granollers, «Diseño y Automatización Industrial,» [En línea]. Available: http://www.epsevg.upc.edu/hcd/material/lecturas/interfaz.pdf. [Último acceso: 22 Mayo 2016].

[18] T. I. Gómez Adaros, «Sistema de visión artificial para un brazo robótico industrial,» Valparaíso-Chile, 2012.

[19] M. Gaub, A. Bürkle, T. Längle y H. Wörn, «An Architecture and Communication Protocol for Interaction of Industrial Robots and Vision Systems,» de *The 11th International Conference on Advance Robotics*, Coimbra, Portugal, 2003.

[20] Kawasaki Heavy Industries, Ltd, «Kawasaki Robot Controller E Series TCP/IP Communication Manual,» Estados Unidos, 2013.

[21] M. J. Donahoo y K. L. Calvert, TCP/IP Sockets in C Practical Guide for Programmers, San Francisco, California: Morgan Kaufmann Publishers, 2001.

[22] Microsoft, «Aprende a desarrollar con Microsoft Developer Network | MSDN,» [En línea]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms741394(v=vs.85).aspx. [Último acceso: 26 Mayo 2016].

[23] Kawasaki Heavy Industries, Ltd, «Kawasaki Robot Controller E Series AS language Reference Manual,» Estados Unidos, 2010.

[24] Stack Overflow, «Stack Overflow,» Agosto 2008. [En línea]. Available: http://stackoverflow.com/. [Último acceso: 11 Junio 2016].

[25] H. M. Deitel y P. J. Deitel, Cómo Programar en C/C++ y Java, México: PEARSON EDUCACIÓN, 2004.

ANEXOS

Código del robot

El programa está desarrollado en el lenguaje AS de Kawasaki, el cual utiliza la extensión .as. En este código se encuentran las variables definidas, funciones de comunicación y de movimiento del robot.

Archivo "kserver.as"

```
;-------------------------Main program-----------------------------
.PROGRAM main()

    ;ROBOT PARAMETER
    ACCURACY 100 ALWAYS

    ; SOCKET COMMUNICATION
    port=49197
    max_length=255
    tout_open=60
    tout_rec=60

    CALL open_socket            ;Connecting communication
    IF sock_id<0 THEN
        GOTO exit_end
    END
    text_id=0
    tout=60
    eret=0
    rret=0

    cond=2001
    boxstart=2002
    $sdata[1]="wait for data"

    row.max=2
    col.max=2
    xs= -155
    ys= -130
    IF  (SIG(boxstart)==-1) THEN
        row=lastrow
        col=lastcol
        JMOVE REST1
        POINT put=putstart
        OPENI
        OPENI 2
        GOTO com
    END
    SIGNAL cond                         ;Pick and place application
```

```
    WHILE (SIG(cond)==-1) DO
                                        ;vision communication to pos data
        JMOVE REST1
        POINT put=posini
        OPENI
        OPENI 2
    FOR row=1 TO row.max
        FOR col=1 TO col.max
com:
        CALL send(eret,$sdata[1])           ;Instructing processing
        IF eret<0 THEN
            PRINT "CODE 001 ERROR END CODE=",eret
            GOTO exit
        END
        CALL recv               ;Reveiving the result of processing1
        IF rret<0 THEN
            PRINT "CODE 001 RECV ERROR END CODE=",eret
            GOTO exit
        END
        IF ($recv_buf[1]=="no object detected") THEN
            GOTO com
        END
        CALL transf($recv_buf[1])           ;Transformation variable

                JAPPRO PICK,100
                TWAIT(0.1)
                SPEED 10
                LMOVE PICK
                CLOSEI
                CLOSEI 2
                DELAY(0.1)
                SPEED 10
                LDEPART 100
                ACCURACY 20
                JMOVE REST1
                ACCURACY 100
                JAPPRO put,180
                TWAIT(0.1)
                SPEED 10
                LMOVE put
                DELAY(0.1)
                OPENI
                OPENI 2
                LDEPART 300
                ACCURACY 20
                JMOVE REST1
                ACCURACY 100
                POINT put =SHIFT(put BY xs,0,0)
            END
                POINT put= SHIFT(posini BY 0,ys*row,0)
        END
    END
exit:
```

```
    CALL close_socket                    ;Closing communication
exit_end:
lastrow=row
lastcol=col
POINT putstart=put
.END

;-----------------------Starting communication---------------------
.PROGRAM open_socket()
    er_count=0
listen:
    TCP_LISTEN retl,port
    IF retl<0 THEN
        IF er_count>=5 THEN
            PRINT "Connection with PC is failed(LISTEN). Program is
stopped."
            sock_id=-1
            goto exit
        ELSE
            er_count=er_count+1
            PRINT "TCP_LISTEN error=",retl," error count=",er_count
            GOTO listen
        END
    ELSE
        PRINT "TCP_LISTEN OK",retl
    END
    er_count=001
accept:
    TCP_ACCEPT sock_id,port,tout_open,ip[1]
    IF sock_id<0 THEN
        IF er_count>=5 THEN
            PRINT "Connection with PC is failed(ACCEPT). Program is
stopped."
            TCP_END_LISTEN ret,port
            sock_id=-1
        ELSE
            er_count=er_count+1
            PRINT "TCP_ACCEPT error id=",socket_id," error
count=",er_count
            GOTO accept
        END
    ELSE
        PRINT "TCP_ACCEPT OK id=",sock_id
    END
exit:
.END

;-------------------Communication sending data---------------------
.PROGRAM send(.ret,.$data)
    $send_buf[1]=.$data
    buf_n=1
    .ret=1
    TCP_SEND sret,sock_id,$send_buf[1],buf_n,tout
    IF sret<0 THEN
```

```
        .ret=-1
        PRINT "TCP_SEND error in SEND",sret
        PRINT $send_buf[1]
    ELSE
        PRINT "TCP_SEND OK in SEND",sret
        PRINT $send_buf[1]
    END
.END

;--------------------Communication receiving data------------------
.PROGRAM recv()
    num=001
    TCP_RECV rret,sock_id,$recv_buf[1],.num,tout_rec,max_length
    IF rret<0 THEN
        PRINT "TCP_RECV error in RECV",rret
        PRINT $recv_buf[1]
        $recv_buf[1]="000"
    ELSE
        IF .num>0 THEN
            PRINT "TCP_RECV OK in RECV",rret
            PRINT $recv_buf[1]
        ELSE
        PRINT $recv_buf[1]
            $recv_buf[1]="000"
        END
    END
.END

;------------------------Closing communication---------------------
.PROGRAM close_socket()
    TCP_CLOSE ret,sock_id                   ;Normal socket closure
    IF ret<0 THEN
        PRINT "TCP_CLOSE error ERROE=(",ret,")",$ERROR(ret)
        TCP_CLOSE ret1,sock_id
        IF ret1<0 THEN
            PRINT "TCP_CLOSE error id=",sock_id
        END
    ELSE
        PRINT "TCP_CLOSE OK id=",sock_id
    END
    TCP_END_LISTEN ret,port
    IF ret<0 THEN
        PRINT "TCP_CLOSE error id",sock_id
    ELSE
        PRINT "TCP_CLOSE OK id=",sock_id
    END
.END

;------------------------Transfer pos variable---------------------
.PROGRAM transf(.$strdata)
    $recv_dat[1]=.$strdata
    i=0
    DO
        $nstr=$DECODE($recv_dat[1],",",0)
```

```
        d[i]=VAL($nstr)
        if $recv_dat[1]=="" GOTO end1
        $nstr=$DECODE($recv_dat[1],",",1)
        i=i+1
    UNTIL $recv_dat[1]==""
end1:
POINT PICK=TRANS(d[0],d[1],d[2],d[3],d[4],d[5])
.END
```

Código del sistema de visión

El programa está desarrollado en C++ en el entorno Visual Studio 2013. Se incluye inicialmente un código de cabecera .h donde se agregan las librerías de OpenCV, se definen constantes y se crean funciones adicionales.

Archivo "vdetect.h"

```
#include <sstream>
#include <string>
#include <iostream>
#include <cmath>
#include <iomanip>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/calib3d/calib3d.hpp"

const double PI = 3.141592653589793;    //PI value
const int FRAME_WIDTH = 1280;   //default capture width and height
const int FRAME_HEIGHT = 1024;
const int MAX_NUM_OBJECTS = 50; //max number of objects to be
detected in frame
const int MIN_OBJECT_AREA = 125 * 125;  //minimum and maximum object
area
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH / 30;

using namespace std;

void onTrackbar(int, void*)
{

}

void createTrackbars(int &H_min, int &H_max, int &S_min, int &S_max,
int &V_min, int &V_max)
{
    string trackbarWindowName = "TrackBars50";
    cv::namedWindow(trackbarWindowName, cv::WINDOW_AUTOSIZE);
    cv::createTrackbar("H_min", trackbarWindowName, &H_min, H_max,
onTrackbar);
    cv::createTrackbar("H_max", trackbarWindowName, &H_max, H_max,
onTrackbar);
    cv::createTrackbar("S_min", trackbarWindowName, &S_min, S_max,
onTrackbar);
    cv::createTrackbar("S_max", trackbarWindowName, &S_max, S_max,
onTrackbar);
    cv::createTrackbar("V_min", trackbarWindowName, &V_min, V_max,
onTrackbar);
    cv::createTrackbar("V_max", trackbarWindowName, &V_max, V_max,
onTrackbar);
```

```
}

void drawPos(int u, int v, double w, cv::Mat xyz, cv::Mat &rcv)
{
    string x, y, z;
    stringstream data;
    data << fixed << std::setprecision(3) << xyz.at<double>(0, 0) <<
" " << xyz.at<double>(1, 0) << " " << w ;
    data >> x >> y >> z;
    rectangle(rcv, cv::Point(u - 5, v - 5), cv::Point(u + 5, v + 5),
cv::Scalar(0, 0, 255), 2);
    putText(rcv, "Object detected", cv::Point(u - 110, v - 110), 2,
1, cv::Scalar(0, 0, 255), 2);               //Draw object location
on screen
    putText(rcv, to_string(u) + "," + to_string(v), cv::Point(u +
10, v + 40), 2, 1, cv::Scalar(0, 0, 255), 2); //Print uv pixel
coordinates
    putText(rcv, x + "mm," + y + "mm," + z + "mm", cv::Point(u + 10,
v + 80), 2, 1, cv::Scalar(0, 0, 255), 2);  //Print xyz coordinates
in mm
}

void drawOri(int u, int v, double per, double theta, cv::Mat &rcv)
{
    int a = per*cos(theta) / 4;
    int b = per*sin(theta) / 4;
    int c = per*cos(theta + (PI / 2)) / 4;
    int d = per * sin(theta + (PI / 2)) / 4;
    arrowedLine(rcv, cv::Point(u, v), cv::Point(u + a, v + b),
cv::Scalar(0, 0, 255), 2);
}

void mat2String(cv::Mat xyz, cv::Mat euler, string &posvariable)
{
    stringstream data;
    data << fixed << std::setprecision(3) << xyz.at<double>(0, 0) <<
"," << xyz.at<double>(1, 0) << "," << xyz.at<double>(2, 0) << "," <<
euler.at<double>(0, 0) << "," << euler.at<double>(1, 0) << "," <<
euler.at<double>(2, 0);
    data >> posvariable;
}

bool trackObject(int &u, int &v, double &theta, cv::Mat rcv, cv::Mat
&dst)
{
    cv::Mat obj_img;
    vector<vector<cv::Point>> contours;
    vector<cv::Vec4i> hierarchy;
    rcv.copyTo(obj_img);    //Copy to a new Mat object
    findContours(obj_img, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE);   //Find contours
    cv::Mat drawing = cv::Mat::zeros(obj_img.size(), CV_8UC3);
//Draw contours
    for (int i = 0; i < contours.size(); i++)
```

```
    {
        drawContours(dst, contours, i, cv::Scalar(0, 0, 255), 2, 8,
hierarchy, 0, cv::Point());
    }
    double refArea = 0;
    if (hierarchy.size() > 0)
    {
        int numObjects = hierarchy.size();
        if (numObjects < MAX_NUM_OBJECTS)
        {
            for (int index = 0; index >= 0; index =
hierarchy[index][0])
            {
                cv::Moments mom = moments((cv::Mat)contours[index]);
                double area = mom.m00;  //Compute object area
                double per = arcLength(contours[index], true);
//Compute object perimeter
                if (area > MIN_OBJECT_AREA && area < MAX_OBJECT_AREA
&& area >= refArea)
                {
                    u = mom.m10 / area; //x pixel object position
                    v = mom.m01 / area; //y pixel object position
                    refArea = area;
                    cout << "area: " << refArea << endl;
                    theta = (0.5 * atan2((2 * (mom.mu11 / area)) ,
((mom.mu20 / area) - (mom.mu02 / area))));   //Angle of the
orientation
                    drawOri(u, v, per, theta, dst);
                    return 1;
                }
            }
        }
    }
    return 0;
}

void morphTrans(cv::Mat &rcv)
{
    cv::Mat erdodeImg = getStructuringElement(cv::MORPH_ELLIPSE,
cv::Size(7, 7));
    cv::Mat dilateImg = getStructuringElement(cv::MORPH_ELLIPSE,
cv::Size(11, 11));
    cv::Mat erdodeImg1 = getStructuringElement(cv::MORPH_ELLIPSE,
cv::Size(3, 3));
    erode(rcv, rcv, erdodeImg);
    dilate(rcv, rcv, dilateImg);
    //erode(rcv, rcv, erdodeImg1);
}

int readParams(string &filename, cv::Mat &cameraMatrix, cv::Mat
&distCoeffs, vector<cv::Mat> &rvecs, vector<cv::Mat> &tvecs, double
&rms)
{
    cv::FileStorage fs;
```

```
    fs.open(filename, cv::FileStorage::READ);
    if (!fs.isOpened())
    {
        cerr << "Failed to open" << filename << std::endl;
        return 1;
    }
    fs["Calibrate_Accuracy"] >> rms;
    fs["Camera_Matrix"] >> cameraMatrix;
    fs["Distortion_Coeffs"] >> distCoeffs;
    fs["Rotation_Vector"] >> rvecs;
    fs["Translation_Vector"] >> tvecs;

    //cout << "Distortion coefficients: \n" << distCoeffs << endl;
    //cout << "Camera Matrix: \n" << cameraMatrix << endl;
}

void transCoord(int u, int v, cv::Mat &xyzcam, cv::Mat cameraMatrix,
cv::Mat distCoeffs, cv::Mat rvecs, cv::Mat tvecs)
{
    cv::Mat uvPoint, tempMat, tempMat2, rotationMatrix;
    double s;

    uvPoint = cv::Mat::ones(3, 1, CV_64F);
    uvPoint.at<double>(0, 0) = u;
    uvPoint.at<double>(1, 0) = v;

    Rodrigues(rvecs, rotationMatrix);

    tempMat = rotationMatrix.inv() * cameraMatrix.inv() * uvPoint;
    tempMat2 = rotationMatrix.inv() * tvecs;

    s = 0 + tempMat2.at<double>(2, 0);
    s /= tempMat.at<double>(2, 0);

    xyzcam = rotationMatrix.inv() * ((s * cameraMatrix.inv() *
uvPoint) - tvecs);
}

void poseRobot(cv::Mat xyzhome, cv::Mat &xyzcoord, double theta,
const double offset)
{
    //offset matrix
    cv::Mat tcpoff = cv::Mat::zeros(3, 1, CV_64F);
    tcpoff.at<double>(0, 0) = offset * cos(theta);
    tcpoff.at<double>(1, 0) = offset * sin(theta);
    tcpoff.at<double>(2, 0) = 0;

    //rotation matrix patter to robot effector
    cv::Mat roteffector = cv::Mat::zeros(3, 3, CV_64F);
    roteffector.at<double>(0, 1) = 1;
    roteffector.at<double>(1, 0) = 1;
    roteffector.at<double>(2, 2) = 1;

    //object coordinate center at robot frame
```

```
    xyzcoord = (roteffector*(xyzcoord - tcpoff)) + xyzhome;
}

void euler2Rotation(cv::Mat &euler, cv::Mat &rotationmatrix)
{
    double O, A, T;
    rotationmatrix = cv::Mat::eye(3, 3, CV_64F);

    O = euler.at<double>(0, 0);
    A = euler.at<double>(1, 0);
    T = euler.at<double>(2, 0);
    rotationmatrix.at<double>(0, 0) = (cos(O)*cos(A)*cos(T)) -
(sin(O)*sin(T));
    rotationmatrix.at<double>(1, 0) = (sin(O)*cos(A)*cos(T)) +
(cos(O)*sin(T));
    rotationmatrix.at<double>(2, 0) = (-sin(A)*cos(T));
    rotationmatrix.at<double>(0, 1) = (-cos(O)*cos(A)*sin(T)) -
(sin(O)*cos(T));
    rotationmatrix.at<double>(1, 1) = (-sin(O)*cos(A)*sin(T)) +
(cos(O)*cos(T));
    rotationmatrix.at<double>(1, 2) = sin(A)*sin(T);
    rotationmatrix.at<double>(2, 0) = cos(O)*sin(A);
    rotationmatrix.at<double>(2, 1) = sin(O)*sin(A);
    rotationmatrix.at<double>(2, 2) = cos(A);
}

void rotation2Euler(cv::Mat &euler, cv::Mat &rotationmatrix)
{
    double O, A, T;
    euler = cv::Mat::eye(3, 1, CV_64F);

    double sy = sqrt((rotationmatrix.at<double>(0,
2)*rotationmatrix.at<double>(0, 2)) + (rotationmatrix.at<double>(1,
2)*rotationmatrix.at<double>(1, 2)));
    bool singular = sy < 1e-1;

    if (!singular)
    {
        O = atan2(rotationmatrix.at<double>(1, 2),
rotationmatrix.at<double>(0, 2));
        A = atan2(sy, rotationmatrix.at<double>(2, 2));
        T = atan2(rotationmatrix.at<double>(2, 1), -
rotationmatrix.at<double>(2, 0));
    }
    else
    {
        T = -PI/2;
        A = PI;
        O = -atan2(rotationmatrix.at<double>(1,
0),rotationmatrix.at<double>(0, 0));
    }
    euler.at<double>(0, 0) = O;
    euler.at<double>(1, 0) = A;
    euler.at<double>(2, 0) = T;
```

```
}

void orientRobot(cv::Mat rvecs, double theta, cv::Mat eulerhome,
cv::Mat &eulerworld)
{
    cv::Mat rmatrix, inter, rothome;

    //Create object rotation matrix (camera frame)
    cv::Mat rotobject = cv::Mat::zeros(3, 3, CV_64F);
    rotobject.at<double>(0, 0) = cos(theta);
    rotobject.at<double>(0, 1) = sin(theta);
    rotobject.at<double>(1, 0) = -sin(theta);
    rotobject.at<double>(1, 1) = cos(theta);
    rotobject.at<double>(2, 2) = 1;

    //Compute rotation matrix from object to fixed frame chessboard
    Rodrigues(rvecs, rmatrix);
    rmatrix.at<double>(0, 2) = 0;   //force rotation matrix to z = 0
    rmatrix.at<double>(1, 2) = 0;
    rmatrix.at<double>(2, 2) = 1;
    rmatrix.at<double>(2, 0) = 0;
    rmatrix.at<double>(2, 1) = 0;
    inter = rmatrix * rotobject;

    //Transform from robot euler representation to rotation matrix
    euler2Rotation(eulerhome, rothome);
    cv::Mat rotworld = rothome * inter;
    rotation2Euler(eulerworld, rotworld);
    eulerworld = (eulerworld * 180) / PI;
}
```

Se incluye también inicialmente otro código de cabecera .h donde se agregan las librerías de Winsock, se definen constantes y se crean funciones adicionales.

Archivo "tcpcom.h"

```
#define WIN_32_LEAN_AND_MEAN
#define _WINSOCK_DEPRECATED_NO_WARNINGS

#include <iostream>
#include <string>
#include <WinSock2.h>
#include <WS2tcpip.h>

#pragma comment(lib,"ws2_32.lib")

using namespace std;

const int buffer_sz = 255;
char msg[buffer_sz];
int timeout = 200;  //Waits for data from server
SOCKET Connection;

int tcpstart()
{
    //Winsock Startup
    WSAData wsaData;
    WORD DllVersion = MAKEWORD(2, 1);
    if (WSAStartup(DllVersion, &wsaData) != 0) //If WSAStartup
returns anything other than 0, then that means an error has occured
in the WinSock Startup.
    {
        MessageBoxA(NULL, "Winsock startup failed", "Error", MB_OK |
MB_ICONERROR);
        return(0);
    }
    return 1;
}


SOCKET tcpConnect(int &retl, const int port, const char*
ipdirection)
{
    if (retl > 0)
    {
        SOCKADDR_IN addr; //Address to be binded to our Connection
socket
        int sizeofaddr = sizeof(addr); //Need sizeofaddr for the
connect function
        addr.sin_addr.s_addr = inet_addr(ipdirection);
        addr.sin_port = htons(port);
        addr.sin_family = AF_INET; //IPv4 Socket
        SOCKET connection = socket(AF_INET, SOCK_STREAM, NULL);
//Set Connection socket
```

```
        if (connect(connection, (SOCKADDR*)&addr, sizeofaddr) != 0)
//If we are unable to connect...
        {
            MessageBoxA(NULL, "Failed to Connect", "Error", MB_OK |
MB_ICONERROR);
            retl = 0; //Failed to Connect
            return NULL;
        }
        cout << "Connected!" << endl;
        return connection;
    }
}

void tcpSend(int &retl, string message, SOCKET connection)
{
    if (retl > 0)
    {
        //send data from client
        send(connection, message.c_str(), message.length(), NULL);
        cout << message << "\n" << flush;
    }
}

string tcpRecv(int &retl, SOCKET connection)
{
    if (retl > 0)
    {
        fd_set set;
        struct timeval timeout;
        //set up the file desciptor set
        FD_ZERO(&set);
        FD_SET(connection, &set);
        //check tout for data
        timeout.tv_sec = 60.;
        timeout.tv_usec = 0.;
        //receive data from server
        int iResult = select(connection, &set, NULL, NULL,
&timeout);
        if (iResult > 0)
        {
            int length = recv(connection, msg, buffer_sz, NULL);
            for (int q = length; q < buffer_sz; q++)
            {
                msg[q] = '\0';
            }
            cout << length << endl;
            cout << msg << endl << flush;
            string str(msg);
            return str;
        }
        return "";
    }
}
```

El programa principal posee una extensión .cpp. Se han incluido las librerías pylon para capturar las imágenes con la cámara, además de las funciones para realizar el procesamiento de imágenes y la comunicación con el robot.

Archivo "openCV.cpp"

```
#include "vdetect.h"
#include "tcpcom.h"
#include <time.h>

#include <pylon/PylonIncludes.h>
#ifdef PYLON_WIN_BUILD
#include <pylon/PylonGUI.h>
#endif

using namespace std;
using namespace cv;     //Namespace for opencv objects
using namespace Pylon;  // Namespace for using pylon objects

static const uint32_t c_countOfImagesToGrab = 100;  // Number of
images to be grabbed
int H_min = 0, S_min = 0, V_min = 0;    //Declare min and max HSV
values
int H_max = 180, S_max = 256, V_max = 256;
const double offset = 50.0; //tcp measure to center the robot
effector
const double z_ori = -32.0; //Z value for base pose robot
const char *ipdirection = "192.168.47.5";   //Address = localhost
(this pc)
const int port = 49197;
const string key = "wait for data"; //message from robot server for
sending pos an orientation data
string message;

int main(int argc, char* argv[])
{
    int exitCode = 0;   // The exit code of the sample application
    int retl = 1; // Communication error flag
    int camflag = -1;   //camera object flag detector
    int comflag = -1;   //communication flag detector
    string fruit;
    Scalar hsv_max, hsv_min, max_man, min_man, max_mar, min_mar,
max_ag, min_ag;
    double h_fruit, h_man, h_mar, h_ag;
    //value for maracuya
    max_mar = Scalar(30, 256, 256);
    min_mar = Scalar(23, 120, 120);
    h_mar = 67.500;
    //value for aguacate
    max_ag = Scalar(48, 256, 120);
    min_ag = Scalar(23, 90, 50);
    h_ag = 61.500;
```

```
    //value for mandarina
    max_man = Scalar(42, 256, 180);
    min_man = Scalar(28, 110, 20);
    h_man = 55.500;
    while (1)
    {
        cout << "Insert fruit to detect: " << endl << "a) maracuya"
<< endl << "b) aguacate" << endl << "c) mandarina" << endl;
        getline(cin, fruit);
        if (fruit == "a"){
            hsv_max = max_mar;
            hsv_min = min_mar;
            h_fruit = h_mar;
            break;
        }
        else if (fruit == "b"){
            hsv_max = max_ag;
            hsv_min = min_ag;
            h_fruit = h_ag;
            break;
        }
        else if (fruit == "c")
        {
            hsv_max = max_man;
            hsv_min = min_man;
            h_fruit = h_man;
            break;
        }
    }
    time_t timer1;  //variable to storage program time value
    time_t timer2;
    time(&timer2);

    createTrackbars(H_min, H_max, S_min, S_max, V_min, V_max);
//Call initial functions trackbars
    onTrackbar(0, 0);
    Pylon::PylonAutoInitTerm autoInitTerm;  // Automagically call
PylonInitialize and PylonTerminate to ensure the pylon runtime
system is initialized during the lifetime of this object
    CGrabResultPtr ptrGrabResult;   //This smart pointer will
receive the grab result data
    namedWindow("CV_Image", WINDOW_AUTOSIZE);   //Create an opencv
display window
    try
    {
        CInstantCamera
camera(CTlFactory::GetInstance().CreateFirstDevice());   //Create an
instant camera object with the camera device found first
        cout << "Using device " <<
camera.GetDeviceInfo().GetModelName() << endl;   //Print the model
name of the camera
        camera.Open();  //Open the camera before accesing any
parameters
```

```
        GenApi::CIntegerPtr
width(camera.GetNodeMap().GetNode("Width"));
        GenApi::CIntegerPtr
height(camera.GetNodeMap().GetNode("Height"));
        camera.MaxNumBuffer = 5;    //Count of buffers allocated for
grabbing (max 10)
        camera.StartGrabbing();
        CImageFormatConverter fc;   //Create a pylon
ImageFormatConverter object
        fc.OutputPixelFormat = PixelType_BGR8packed;    //Specify
the output pixel format
        CPylonImage image;  //Create a pylonImage that will be used
to create a opencv images later
        //Adquition image initialitation
        Mat ori_img, fil_img, ori_rz, cv_rz, cv_img, ej;
        int u = 0, v = 0;
        double theta = 0;
        //xyz mm robot world frame
        Mat xyzhome = Mat::ones(3, 1, CV_64F);
        xyzhome.at<double>(0, 0) = 337.937;
        xyzhome.at<double>(1, 0) = -50.905;
        xyzhome.at<double>(2, 0) = h_fruit + z_ori;

        //euler angles robot and world frame
        Mat eulerworld;
        Mat eulerhome = Mat::ones(3, 1, CV_64F);
        eulerhome.at<double>(0, 0) = 0. * (PI / 180);
        eulerhome.at<double>(1, 0) = 180. * (PI / 180);
        eulerhome.at<double>(2, 0) = -90. * (PI / 180);
        //Calibration parameters initialitation
        Mat cameraMatrix(3, 3, CV_64F);
        Mat distCoeffs(8, 1, CV_64F);
        vector<Mat> rvecs, tvecs;
        Mat xyzcoord;
        double rms;
        string filename = "camcalib.xml";

        //Load camera calibration parameters from xml file
        readParams(filename, cameraMatrix, distCoeffs, rvecs, tvecs,
rms);
        //Create tcp communication with kawasaki robot
        retl = tcpstart();
        SOCKET connection = tcpConnect(retl, port, ipdirection);
        if (retl == 0)
        {
            exitCode = 1;
        }

        while (camera.IsGrabbing()){
            camera.RetrieveResult(5000, ptrGrabResult,
TimeoutHandling_ThrowException);
//Wait for an image an then retrieve it. A timeout of 5000ms is used
            if (ptrGrabResult->GrabSucceeded()){
```

```
                fc.Convert(image, ptrGrabResult);   //Convert the
grabbed buffer to a pylon image
                ori_img = cv::Mat(ptrGrabResult->GetHeight(),
ptrGrabResult->GetWidth(), CV_8UC3, (uint8_t*)image.GetBuffer());
//Create an opencv image from a pylon image
                GaussianBlur(ori_img, ej, Size(11, 11), 0, 0);
//Apply a filter to reduce the noise
                cvtColor(ori_img, fil_img, CV_BGR2HSV); //Convert to
HSV space color
                //inRange(fil_img, Scalar(H_min, S_min, V_min),
Scalar(H_max, S_max, V_max), cv_img);   //Modify HSV image
parameters
                inRange(fil_img, hsv_min, hsv_max, cv_img); //Modify
HSV image parameters
                morphTrans(cv_img); //Apply expantion to the image
by morphology
                bool found = trackObject(u, v,theta, cv_img,
ori_img);  //Detect contour and the object position
                if (found)
                {
                    transCoord(u, v, xyzcoord, cameraMatrix,
distCoeffs, rvecs[13], tvecs[13]); //Transform from camera to object
coordinates
                    drawPos(u, v, h_fruit, xyzcoord, ori_img);
                    orientRobot(rvecs[13], theta, eulerhome,
eulerworld);   //euler format ZYZ in degree for robot orientation
                    poseRobot(xyzhome, xyzcoord, theta, offset);
//xyz mm format for robot pose
                    mat2String(xyzcoord, eulerworld, message);
                }
                //TCP communication to robot
                time(&timer1);  //get actuall time value at this
point
                if (waitKey(30) == 32)
                {
                    while (1)
                    {
                        cout << "Insert fruit to detect: " << endl
<< "a) maracuya" << endl << "b) aguacate" << endl << "c) mandarina"
<< endl;
                        getline(cin, fruit);
                        if (fruit == "a"){
                            hsv_max = max_man;
                            hsv_min = min_man;
                            h_fruit = h_man;
                            hsv_max = max_mar;
                            hsv_min = min_mar;
                            h_fruit = h_mar;
                            break;
                        }
                        else if (fruit == "b"){
                            hsv_max = max_ag;
                            hsv_min = min_ag;
                            h_fruit = h_ag;
```

```
                            break;
                        }
                        else if (fruit == "c"){
                            hsv_max = max_man;
                            hsv_min = min_man;
                            h_fruit = h_man;
                            break;
                        }
                    }
                    xyzhome.at<double>(2, 0) = h_fruit + z_ori;
                    time(&timer2);
                }
                if ((retl == 1) & (difftime(timer1,timer2) >= 10.0))
                {
                    if (found)
                    {
                        cout << endl << "XYZworld: " << endl <<
xyzcoord << endl;
                        cout << endl << "theta: " << theta << endl;
                        cout << endl << "eulerworld: " << endl <<
eulerworld << endl;
                        tcpRecv(retl, connection);
                        tcpSend(retl, message, connection);
                    }
                    else
                    {
                        tcpRecv(retl, connection);
                        tcpSend(retl, "no object detected",
connection);
                    }
                    time(&timer2);
                }
                resize(ori_img, ori_rz, Size(FRAME_WIDTH/2,
FRAME_HEIGHT/2));   //Modify screen size
                resize(cv_img, cv_rz, Size(FRAME_WIDTH/2,
FRAME_HEIGHT/2)); //Modify screen size
                imshow("CV_Image", ori_rz); //Display the current
image in the opencv display window
                imshow("CV_Image_detection", cv_rz);
                waitKey(1);
                if (waitKey(20) == 27){
                    camera.StopGrabbing();
                }
            }
        }
    }
    catch (GenICam::GenericException &e)
    {
        cerr << "An exception occurred." << endl
            << e.GetDescription() << endl;
        exitCode = 1;
    }
    return exitCode;
}
```

Se presenta un programa con extensión .cpp donde se ha desarrollado el proceso de calibración de la cámara

Archivo "calib.cpp"

```
#include <sstream>
#include <string>
#include <iostream>
#include <cmath>

#include <pylon/PylonIncludes.h>
#ifdef PYLON_WIN_BUILD
#include <pylon/PylonGUI.h>
#endif

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/calib3d/calib3d.hpp"

using namespace std;
using namespace cv;      //Namespace for opencv objects
using namespace Pylon;  // Namespace for using pylon objects

const int n_board = 14; //Number of image to take
const float side = 25.4;

const int FRAME_WIDTH = 1280;   //default capture width and height
const int FRAME_HEIGHT = 1024;

void calcChessboardCorners(Size boardSize, float SquareSize,
vector<Point3f> &corners);
void writeParams(const string &filename, const Mat &cameraMatrix,
const Mat &distCoeffs, const vector<Mat> &rvecs, const vector<Mat>
&tvecs, const double &rms);

int main(int argc, char* argv[])
{
    int exitCode = 0;   // The exit code of the sample application

    Pylon::PylonAutoInitTerm autoInitTerm;  // Automagically call
PylonInitialize and PylonTerminate to ensure the pylon runtime
system is initialized during the lifetime of this object
    CGrabResultPtr ptrGrabResult;   //This smart pointer will
receive the grab result data
    namedWindow("CV_Image", WINDOW_AUTOSIZE);   //Create an opencv
display window
    try
    {
        CInstantCamera
camera(CTlFactory::GetInstance().CreateFirstDevice());   //Create an
instant camera object with the camera device found first
```

```
        cout << "Using device " <<
camera.GetDeviceInfo().GetModelName() << endl;   //Print the model
name of the camera
        camera.Open();  //Open the camera before accesing any
parameters

        GenApi::CIntegerPtr
width(camera.GetNodeMap().GetNode("Width"));
        GenApi::CIntegerPtr
height(camera.GetNodeMap().GetNode("Height"));
        camera.MaxNumBuffer = 5;    //Count of buffers allocated for
grabbing (max 10)

        int successes = 0;

        vector<Point2f> corners2D;
        vector<Point3f> corners3D;
        vector<vector<Point2f>> coord2D;
        vector<vector<Point3f>> coord3D;
        vector<Mat> rvecs, tvecs;

        Size board_sz(9, 6);    //Define dimension of the board

        Mat cv_img(height->GetValue(), width->GetValue(), CV_8UC3);
//Declare opencv Mat variables
        Mat gr_img, cameraMatrix, distCoeffs;

        camera.StartGrabbing();
        CImageFormatConverter fc;   //Create a pylon
ImageFormatConverter object
        fc.OutputPixelFormat = PixelType_BGR8packed;    //Specify
the output pixel format
        CPylonImage image;  //Create a pylonImage that will be used
to create a opencv images later

        calcChessboardCorners(board_sz, side, corners3D);
//Compute real chessboard pattern position

        while (camera.IsGrabbing())
        {
            camera.RetrieveResult(5000, ptrGrabResult,
TimeoutHandling_ThrowException); //Wait for an image an then
retrieve it. A timeout of 5000ms is used
            if (ptrGrabResult->GrabSucceeded())
            {
                fc.Convert(image, ptrGrabResult);   //Convert the
grabbed buffer to a pylon image
                cv_img = Mat(ptrGrabResult->GetHeight(),
ptrGrabResult->GetWidth(), CV_8UC3, (uint8_t*)image.GetBuffer());
//Create an opencv image from a pylon image
                //resize(cv_img, cv_img, Size(FRAME_WIDTH,
FRAME_HEIGHT));  //Modify screen size
                if (waitKey(50) == 32)
                {
```

```
                    if (successes < n_board)
                    {
                        cvtColor(cv_img, gr_img, CV_BGR2GRAY);
                        bool found = findChessboardCorners(gr_img,
board_sz, corners2D, CALIB_CB_ADAPTIVE_THRESH +
CALIB_CB_NORMALIZE_IMAGE + CALIB_CB_FAST_CHECK); //True is return if
find the pattern
                        if (found)
                        {
                            cornerSubPix(gr_img, corners2D, Size(11,
11), Size(-1, -1), CvTermCriteria(CV_TERMCRIT_EPS +
CV_TERMCRIT_ITER, 30, 0.1));
                            drawChessboardCorners(cv_img, board_sz,
corners2D, found);  //Show the result
                            coord2D.push_back(corners2D);
                            coord3D.resize(n_board,corners3D);
                            successes++;
                            cout << "numero imagenes \n" <<
successes << endl;
                        }
                        if (successes == n_board)
                        {
                            cameraMatrix =
initCameraMatrix2D(coord3D, coord2D, Size(FRAME_WIDTH,
FRAME_HEIGHT));
                            double rms = calibrateCamera(coord3D,
coord2D, Size(FRAME_WIDTH, FRAME_HEIGHT), cameraMatrix, distCoeffs,
rvecs, tvecs, CV_CALIB_USE_INTRINSIC_GUESS |
CV_CALIB_RATIONAL_MODEL);
cout << "rms: " << rms << endl;
                            cout << "Distortion coefficients: \n" <<
distCoeffs << endl;
                            cout << "Camera Matrix: \n" <<
cameraMatrix << endl;
                            writeParams("camcalib.xml",
cameraMatrix, distCoeffs, rvecs, tvecs, rms);   //save camera
calibration parameters
                        }
                    }

                }
                imshow("CV_Image", cv_img); //Display the current
image in the opencv display window
                if (waitKey(30) == 27){
                    camera.StopGrabbing();
                }
            }
        }

    }

catch (GenICam::GenericException &e)
        {
            // Error handling.
```

```
            cerr << "An exception occurred." << endl
                << e.GetDescription() << endl;
            exitCode = 1;
        }

        // Comment the following two lines to disable waiting on
exit
        //cerr << endl << "Press Enter to exit." << endl;
        //while (cin.get() != '\n');

        return exitCode;
}

void calcChessboardCorners(Size boardSize, float SquareSize,
vector<Point3f> &corners)
{
    corners.clear();
    for (int i = 0; i < boardSize.height; i++)
        for (int j = 0; j < boardSize.width; j++)
            corners.push_back(Point3f(float(j*side), float(i*side),
0));
}

void writeParams(const string &filename, const Mat &cameraMatrix,
const Mat &distCoeffs, const vector<Mat> &rvecs, const vector<Mat>
&tvecs, const double &rms)
{
    FileStorage fs(filename, FileStorage::WRITE);
    fs << "Calibrate_Accuracy" << rms;
    fs << "Camera_Matrix" << cameraMatrix;
    fs << "Distortion_Coeffs" << distCoeffs;
    fs << "Rotation_Vector" << rvecs;
    fs << "Translation_Vector" << tvecs;
    fs.release();
}
```

Se presenta un código con extensión .xml que almacena los parámetros obtenidos del proceso de calibración de la cámara

Archivo "camcalib.xml"

```
<?xml version="1.0"?>
<opencv_storage>
<Calibrate_Accuracy>5.8112529314876982e-001</Calibrate_Accuracy>
<Camera_Matrix type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    1.3034686136283381e+003 0. 6.4097639156581386e+002 0.
    1.3000918790478404e+003 4.1685728412997543e+002 0. 0.
1.</data></Camera_Matrix>
<Distortion_Coeffs type_id="opencv-matrix">
  <rows>1</rows>
  <cols>12</cols>
  <dt>d</dt>
  <data>
    -2.2647711196693471e+001 1.2508438231509902e+002
    -7.6589124674885682e-003 -1.8427898761195589e-003
    7.5921037273275189e+001 -2.2428573165470137e+001
    1.1999723942466235e+002 1.0619803811820977e+002 0. 0. 0.
0.</data></Distortion_Coeffs>
<Rotation_Vector>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      3.8378083326756535e-001 -5.0563388442361627e-002
      8.2221759136946945e-002</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -2.5141924841299945e-001 3.4506241374074753e-001
      7.2721907066268565e-001</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -8.7692637526664341e-002 -5.1928495586307211e-001
      -5.3327442400083758e-001</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
```

```
    -5.3921095672195407e-001 8.0048349735154101e-001
    -2.5592382610444903e-001</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    1.1379699297581725e+000 -5.6982280615529901e-001
    -1.9686000149299823e+000</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -9.7232724803526172e-001 -8.7105448851883283e-001
    -1.5736982701215656e+000</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -1.0628474273524104e+000 4.9570178577351715e-001
    -7.1330321135665209e-001</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -5.0416924394839102e-001 -4.3337778621102319e-001
    -1.3798125561384036e+000</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -1.9224384318553046e-002 3.7104488018333959e-001
    -1.1011538003375114e+000</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -7.8772441121448555e-001 -3.8509663676404918e-002
    -3.1341057504634895e-001</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -4.3978807372699391e-001 1.0065863228524602e-001
    1.2140769869713166e+000</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
```

```
    <dt>d</dt>
    <data>
      4.1119823248685633e-001 8.1991814974008070e-001
      1.6738343339880881e+000</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -3.9809847154498523e-001 1.6782930623792866e-001
      1.7135283166592805e+000</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -1.6513781537305613e-003 -3.0670466876044602e-002
      3.4630937754633381e-003</data></_></Rotation_Vector>
<Translation_Vector>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -1.5612661601000315e+002 -1.6539157175902420e+002
      5.6099404421316103e+002</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -7.1695066836787419e+001 -1.5235166399322216e+002
      5.9246485321706496e+002</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -1.4908563129510634e+002 -3.2961785353112781e+001
      4.6413195008604811e+002</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -6.7364475733640504e+001 -2.2038720584467328e+001
      5.2378266711584524e+002</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -8.9082484473239546e+001 1.0641771354881236e+002
      4.3762469236148303e+002</data></_>
```

```
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -9.056518455882599e+001 7.150240738748306e+001
    2.949090840362663e+002</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -1.098536248981176e+002 1.054133510903397e+002
    5.087062774472740e+002</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -1.264517961286089e+002 4.694204413985522e+001
    3.532189057492143e+002</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -1.359397178966456e+002 4.423402359148131e+001
    5.557800919572693e+002</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -1.253751228034610e+002 -8.617699232038316e+001
    5.809454802150919e+002</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -7.302968018359883e+001 -1.895417021763384e+002
    6.268937604777783e+002</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    6.099939881149557e+001 -7.449458363743578e+001
    4.799575431560294e+002</data></_>
<_ type_id="opencv-matrix">
  <rows>3</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
```

```
      4.5883704410066741e+001 -8.3161321991534351e+001
      5.5779957855649093e+002</data></_>
  <_ type_id="opencv-matrix">
    <rows>3</rows>
    <cols>1</cols>
    <dt>d</dt>
    <data>
      -1.5954894672389875e+002 -1.7484352478151115e+002
      6.4063046926133813e+002</data></_></Translation_Vector>
</opencv_storage>
```

Finalmente se presenta un código .cpp para calcular el histograma de las frutas utilizado en el proceso de segmentación del sistema de visión artificial.

Archivo "histogram.cpp"

```
#include <string>
#include <iostream>
#include <cmath>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"

using namespace std;
using namespace cv;     //Namespace for opencv objects

int hist_w = 512; int hist_h = 400;
int hbins = 180, sbins = 256, vbins = 256;
int maxhue = 0, maxsat = 0, maxval = 0;
int hue = 0, sat = 0, val = 0;
float hranges[] = { 0, 180 };
float sranges[] = { 0, 256 };
float vranges[] = { 0, 256 };
const float* ranges[] = { hranges, sranges, vranges };

int main(int argc, char* argv[])
{

    Mat src, dst, h_hist, s_hist, v_hist;
    vector<Mat> hsv_planes;
    src = imread("maracuya_cut.jpg");
    cvtColor(src, src, CV_BGR2HSV);
    split(src, hsv_planes);

    calcHist(&hsv_planes[0], 1, 0, Mat(), h_hist, 1, &hbins,
&ranges[0], true, false);
    calcHist(&hsv_planes[1], 1, 0, Mat(), s_hist, 1, &sbins,
&ranges[1], true, false);
    calcHist(&hsv_planes[2], 1, 0, Mat(), v_hist, 1, &vbins,
&ranges[2], true, false);

    int bin_w = cvRound((double)hist_w / hbins);
    int bin_y = cvRound((double)hist_w / sbins);
    int bin_z = cvRound((double)hist_w / vbins);

    Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));
    normalize(h_hist, h_hist, 0, histImage.rows, NORM_MINMAX, -1,
Mat());
    normalize(s_hist, s_hist, 0, histImage.rows, NORM_MINMAX, -1,
Mat());
    normalize(v_hist, v_hist, 0, histImage.rows, NORM_MINMAX, -1,
Mat());
```

```
    for (int h = 1; h < hbins; h++)
    {
        int hval = h_hist.at<float>(h - 1);
        line(histImage, Point(bin_w*(h - 1), hist_h -
cvRound(hval)),
             Point(bin_w*(h), hist_h - cvRound(h_hist.at<float>(h))),
             Scalar(255, 0, 0), 2, 8, 0);
        if (hval > maxhue)
        {
            maxhue = hval;
            hue = h;
        }
    }
    for (int s = 1; s < sbins; s++)
    {
        int sval = s_hist.at<float>(s - 1);
        line(histImage, Point(bin_y*(s - 1), hist_h -
cvRound(sval)),
             Point(bin_y*(s), hist_h - cvRound(s_hist.at<float>(s))),
             Scalar(0, 255, 0), 2, 8, 0);
        if (sval > maxsat)
        {
            maxsat = sval;
            sat = s;
        }
    }
    for (int v = 1; v < vbins; v++)
    {
        int vval = v_hist.at<float>(v - 1);
        line(histImage, Point(bin_z*(v - 1), hist_h -
cvRound(vval)),
             Point(bin_z*(v), hist_h - cvRound(v_hist.at<float>(v))),
             Scalar(0, 0, 255), 2, 8, 0);
        if (vval > maxval)
        {
            maxval = vval;
            val = v;
        }
    }
    namedWindow("calcHist HSV", CV_WINDOW_AUTOSIZE);
    imshow("calcHist HSV", histImage);
    cout << "hue: " << hue << endl;
    cout << "Saturation: " << sat << endl;
    cout << "Value: " << val << endl;
    waitKey(0);
}
```