

UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Algoritmos de Visión por computador para el  
reconocimiento de la pose de agentes empleando programación  
orientada a objetos y multi-hilos**

Trabajo de graduación presentado por José Pablo Guerra Jordán para  
optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2020







UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Algoritmos de Visión por computador para el  
reconocimiento de la pose de agentes empleando programación  
orientada a objetos y multi-hilos**

Trabajo de graduación presentado por José Pablo Guerra Jordán para  
optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2020



Vo.Bo.:

(f) \_\_\_\_\_  
Dr. Luis Rivera

Tribunal Examinador:

(f) \_\_\_\_\_  
Dr. Luis Rivera

(f) \_\_\_\_\_  
XXX

(f) \_\_\_\_\_  
XXX

Fecha de aprobación: Guatemala, DIA X de MES XXX de 2020.





Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vitae eleifend ipsum, ut mattis nunc. Pellentesque ac hendrerit lacus. Cras sollicitudin eget sem nec luctus. Vivamus aliquet lorem id elit venenatis pellentesque. Nam id orci iaculis, rutrum ipsum vel, porttitor magna. Etiam molestie vel elit sed suscipit. Proin dui risus, scelerisque porttitor cursus ac, tempor eget turpis. Aliquam ultricies congue ligula ac ornare. Duis id purus eu ex pharetra feugiat. Vivamus ac orci arcu. Nulla id diam quis erat rhoncus hendrerit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed vulputate, metus vel efficitur fringilla, orci ex ultricies augue, sit amet rhoncus ex purus ut massa. Nam pharetra ipsum consequat est blandit, sed commodo nunc scelerisque. Maecenas ut suscipit libero. Sed vel euismod tellus.

Proin elit tellus, finibus et metus et, vestibulum ullamcorper est. Nulla viverra nisl id libero sodales, a porttitor est congue. Maecenas semper, felis ut rhoncus cursus, leo magna convallis ligula, at vehicula neque quam at ipsum. Integer commodo mattis eros sit amet tristique. Cras eu maximus arcu. Morbi condimentum dignissim enim non hendrerit. Sed molestie erat sit amet porttitor sagittis. Maecenas porttitor tincidunt erat, ac lacinia lacus sodales faucibus. Integer nec laoreet massa. Proin a arcu lorem. Donec at tincidunt arcu, et sodales neque. Morbi rhoncus, ligula porta lobortis faucibus, magna diam aliquet felis, nec ultrices metus turpis et libero. Integer efficitur erat dolor, quis iaculis metus dignissim eu.



<b>Prefacio</b>	<b>v</b>
<b>Lista de figuras</b>	<b>IX</b>
<b>Lista de cuadros</b>	<b>XI</b>
<b>Resumen</b>	<b>XIII</b>
<b>Abstract</b>	<b>XV</b>
<b>1. Introducción</b>	<b>1</b>
<b>2. Antecedentes</b>	<b>3</b>
<b>3. Justificación</b>	<b>5</b>
<b>4. Objetivos</b>	<b>7</b>
<b>5. Alcance</b>	<b>9</b>
<b>6. Marco teórico</b>	<b>11</b>
<b>7. Metodología</b>	<b>17</b>
<b>8. Prototipos de Mesa de Pruebas</b>	<b>19</b>
8.1. Primer Prototipo . . . . .	19
8.2. Segundo Prototipo . . . . .	21
<b>9. Pruebas preliminares en Python y C++</b>	<b>23</b>
<b>10. Pruebas Algoritmo y OpenCV en C++</b>	<b>27</b>
10.1. Calibración en C++ . . . . .	27
10.2. Identificación de Códigos en C++ . . . . .	29

<b>11.Migración Código de C++ a Python</b>	<b>31</b>
11.1. Migración del Código para la Calibración de Cámara . . . . .	31
11.2. Migración del Código para Obtención de Pose e identificación . . . . .	33
11.2.1. Identificación del ID . . . . .	33
<b>12.Pruebas Algoritmo y OpenCV en Python</b>	<b>35</b>
<b>13.Pruebas de generación de marcadores/códigos y detección en Python</b>	<b>37</b>
<b>14.Comparación entre C++ y Python</b>	<b>41</b>
14.1. Pruebas de detección de pose . . . . .	41
14.2. Comparación de rendimiento . . . . .	42
<b>15.Conclusiones</b>	<b>43</b>
<b>16.Recomendaciones</b>	<b>45</b>
<b>17.Bibliografía</b>	<b>47</b>
<b>18.Anexos</b>	<b>49</b>
18.1. Planos de construcción . . . . .	49

---

## Lista de figuras

---

1.	Ejemplo 1 . . . . .	11
2.	Ejemplo 2 . . . . .	12
3.	Conversión entre el plano de imagen al mundo real . . . . .	13
4.	Patrón común para la calibración . . . . .	13
5.	Primer protitpo de mesa . . . . .	20
6.	Primer prototipo de mesa y cámara . . . . .	20
7.	Armado de base de cámara para el segundo prototipo . . . . .	21
8.	Segundo prototipo de mesa de pruebas . . . . .	22
9.	Colocación para el segundo armado de la mesa de pruebas . . . . .	22
10.	Captura del primer archivo de texto para el ejemplo Multi-hilos . . . . .	24
11.	Captura del segundo archivo de texto para el ejemplo Multi-hilos . . . . .	24
12.	Captura del texto reconstruido para el ejemplo Multi-hilos . . . . .	25
13.	Captura de la cámara web con OpenCV . . . . .	26
14.	Intento 1 de calibración utilizando C++ . . . . .	28
15.	Intento 2 de calibración utilizando C++ . . . . .	28
16.	Identificación de código . . . . .	29
17.	Primera prueba de identificación de código en la imagen capturada . . . . .	29
18.	Prueba calibración de la cámara con Python . . . . .	36
19.	Segunda prueba calibración de la cámara con Python . . . . .	36
20.	Calibración de la cámara para detección de los códigos utlizando Python . . . .	38
21.	Prueba de generación de código utilizando Python . . . . .	38
22.	Detección y reescalamiento de código generado para tamaño de 1x1 cm . . . .	38
23.	Detección y reescalamiento de código generado para tamaño de 2x2 cm . . . .	39
24.	Detección y reescalamiento de código generado para tamaño de 4x4 cm . . . .	39
25.	Detección y reescalamiento de código generado para tamaño de 8x8 cm . . . .	39



---

## Lista de cuadros

---

1. Comparación entre la detección de pose del algoritmo en Python-C++ y la posición real . . . . . 41





Este proyecto busca implementar un algoritmo de visión por computadora de una manera eficiente utilizando la Programación Orientada a Objetos (POO) y multi-hilos. Esto se busca mediante el análisis y mejora de los algoritmos desarrollados anteriormente en el lenguaje de programación C++ y hacer sus respectivas comparaciones con el lenguaje Python, mediante pruebas como la medición de tiempos de ejecución, uso del CPU, entre otras.

Se desarrollará una herramienta de *software*, cuyo objetivo principal es poder reconocer la pose de agentes (denominados así normalmente en el área de robótica de enjambre). La herramienta se combinará con una mesa de pruebas, la cual será armada para poder realizar pruebas en la calibración de la cámara y que permita identificar los puntos de mejora de los algoritmos.

La herramienta de *software* que se desarrollará será versátil y muy útil para futuros proyectos en la línea de investigación de robótica de enjambre.



---

## Abstract

---

This is an abstract of the study developed under the



La robótica de enjambre se ha convertido en un área de gran interés en los últimos años, sin embargo, aún tiene grandes retos que se deben abarcar. Para esto, se requiere tener herramientas versátiles que ayuden al investigador o usuario en el desarrollo de sus tareas, proyectos o investigaciones relacionadas con esta área.

Por lo tanto, se buscó encontrar el lenguaje orientado a objetos más adecuados que permita entregarle al usuario la versatilidad y utilidad que se busca. Además de esto, se pretende que estos algoritmos sean eficientes en temas computacionales (como procesamiento de imágenes y obtención de datos) por lo que se utilizó la programación multi-hilos como una herramienta útil para mejorar el rendimiento de estos procesos.

Se realizaron pruebas utilizando los algoritmos presentados, así como con las versiones de mejora, para validar su uso en la mesa de pruebas Robotat.

Este documento consta de una sección de objetivos, donde pretende introducir al lector a las metas que se plantean lograr con este trabajo, así como una sección de antecedentes que le informarán de otros proyectos similares o aplicaciones de esta área.

La metodología para este trabajo constituyó en una investigación previa para entender el funcionamiento de los lenguajes `Python` y `C++`, así como de la librería de `OpenCV`, con el objetivo de comprender y poder implementar sus componentes y funciones dentro de los algoritmos. Se procedió a implementar los algoritmos utilizando Programación Orientada a Objetos y se con esto, se buscó poder validar su funcionamiento y correcta aplicación dentro del área de robótica de enjambre.

Finalmente, se presentan los resultados obtenidos a las pruebas, validando así el alcance que se buscaba obtener en esta tesis, además de presentar las recomendaciones para futuras aplicaciones y las respectivas conclusiones.



#### **Visión por Computadora Aplicado a Robótica de Enjambre**

En el documento de tesis de doctorado escrita por Luis Antón Canalís [1], titulada "Swarm Intelligence in Computer Vision: an Application to Object Tracking", describe cómo la visión por computadora se puede aplicar a la robótica de enjambre y a la orientación de sus agentes.

La robótica de enjambre va aplicada a la emulación del comportamiento de las colonias (por ejemplo de hormigas o abejas) con algoritmos por computadora. La mayoría de estos algoritmos han utilizado como herramientas diferentes algoritmos de visión por computadora. Esto significa que, por ejemplo, las imágenes emulan terrenos o espacios donde las colonias virtuales las recorren buscando información significativa basada en el procesamiento obtenido por visión por computadora.

Además de eso, en el documento "Visión artificial y comunicación en robots cooperativos omnidireccionales" [2] se detalla también la importancia del uso de (a lo que los autores se refieren como visión artificial) visión por computadora también para el reconocimiento de pose de agentes. Se denota el uso de Python como un lenguaje útil y simple para la programación, así como el uso de la librería OpenCV para la aplicación de algoritmos para visión por computadora.

#### **Continuación Fase II**

Este trabajo es la fase 2 del Proyecto de Graduación propuesto por André Rodas titulado "Desarrollo e implementación de algoritmo de visión por computador en una mesa de pruebas para la experimentación con micro-robots móviles en robótica de enjambre" [3].

La fase anterior consistió en la realización de un algoritmo que permitiera reconocer objetos dentro de una cama de pruebas para ser aplicada en robótica de enjambre. Dicha

implementación se realizó utilizando OpenCV y el lenguaje C++, con el objetivo de obtener el mayor rendimiento de procesamiento, aunque también se realizó un análisis de otros posibles candidatos de lenguajes para su uso.

El objetivo es poder identificar objetos dentro de las mesas de pruebas. Para esto, lo que se realizó fue una serie de pruebas para determinar el mejor identificador para el reconocimiento de objetos en dichas mesas. Agregado a esto, se realizaron pruebas con diferentes métodos de procesamiento de la librería OpenCV para obtener los mejores resultados y poder comparar con cuales de estos se obtenía el mejor margen de reconocimiento y procesamiento de la imagen en la mesa de pruebas.



Hoy en día, las herramientas computacionales son de gran ayuda para las distintas actividades que se realizan en diferentes áreas, tanto de la industria como en las ramas científicas.

Una de estas herramientas es la visión por computadora, que está enfocada en el uso de algoritmos para el procesamiento de imágenes, dándole la capacidad a la computadora de reconocer datos significativos que pueden ser orientados a diferentes aplicaciones. Su principal uso está basado en la resolución de problemas o toma de decisiones que requieran gran poder de cómputo.

Sin embargo, estas tareas pueden representar un costo alto en recursos computacionales. Para esto, la programación multi-hilos puede ser una herramienta muy útil. Al tener varios hilos de procesamiento, es posible capturar y procesar datos de forma más eficiente. Esto permite que las computadoras (y las diferentes aplicaciones que se puedan realizar en estas) puedan llegar a resultados de manera más rápida y eficiente.

Al combinar la programación multi-hilos y la visión por computadora, es posible obtener reconocimientos de imágenes o entornos (como mesas de pruebas, mapas, entre otros) de manera mucho más adecuada, permitiendo utilizar estos datos en posteriores proyectos, como en la robótica de enjambre, por ejemplo.

La POO agrega muchas otras ventajas a los procesos de cómputo. Primero, ofrece una reutilización del código, es decir, permite utilizar distintos métodos en diversas partes de un código y en variedad de proyectos y aplicaciones. Segundo, permite modificaciones de manera sencilla y práctica ya que se puede añadir o eliminar objetos según sea la necesidad de la aplicación, así como también fiabilidad, ya que es posible reducir códigos grandes en partes más pequeñas, permitiendo encontrar errores de manera más rápida y precisa.

Como se ha dicho, el área de investigación de robótica de enjambre ha crecido y tomado relevancia en la actualidad.

Con este trabajo, y las herramientas propuestas, se busca tener versatilidad en dichas herramientas para que puedan ser aplicadas en diferentes tipos de proyectos o áreas de investigación, así como expandir sus diferentes aplicaciones.

Además, la importancia recae en ofrecer mejoras a algoritmos ya propuestos, que buscan hacer más eficiente los diferentes procesos que se utilizan en la visión por computadora y la robótica de enjambre. Con esto, se busca ayudar a los investigadores a obtener resultados de manera más rápida y precisa en las diferentes ramas de aplicación.

#### **Objetivo General**

Mejorar el algoritmo de visión por computadora desarrollado para la mesa de pruebas Robotat para experimentación de robótica de enjambre, usando programación orientada a objetos, la librería OpenCV, y programación multi-hilos.

#### **Objetivos Específicos**

- Seleccionar el lenguaje de programación orientado a objetos más adecuado para la implementación de algoritmos de visión por computador para la mesa de pruebas Robotat.
- Desarrollar algoritmos computacionalmente eficientes por medio de programación multi-hilos.
- Diseñar e implementar una herramienta de software para aplicaciones de robótica de enjambre, usando los algoritmos desarrollados.
- Validar la herramienta de software en la mesa de pruebas Robotat.



Este algoritmo esta adoptado para funcionar en la mesa de pruebas Robotat con una cámara montada en la parte superior para la captura e identificación de los códigos de cada robot.

Mientras la cámara enfoque la mesa completamente, y tenga claramente figuras circulares en sus esquinas, se podrá calibrar automáticamente las imágenes capturadas. La interfaz gráfica en `Python` ofrece la posibilidad de calibrar la cámara y guardar sus parámetros, generar un código identificador para cada robot, así como la toma de imágenes, procesamiento e identificación de los robots de manera paralela mediante el uso de multi-hilos. Los códigos además no podrán exceder el valor de 255 y tendrán hasta un mínimo de 0.

La identificación de los códigos va desde tamaños de 3x3 cm hasta 10x10cm mientras las condiciones de luz sean las adecuadas (que la mesa este bien iluminada, con luz blanca preferiblemente) para alto contraste de los objetos sobre la mesa)



## Visión por Computadora

Visión por computadora se refiere al uso de cámaras o cualquier otro dispositivo de toma de fotografías o vídeos, para recolectar información para su posterior análisis, desarrollando algoritmos para hacer entender a la computadora que es lo que hay (en cuanto a datos o información significativa) en este tipo de archivos. [4]

En otras palabras, consiste en obtener la información relevante, realizando un procesamiento a imágenes y vídeos, para que los seres humanos puedan entender de mejor manera que es lo que hay en ellos. Es decir, poder visualizar lo que una computadora hace en este tipo de procesamientos. Normalmente, este tipo de procesamiento de datos es utilizado para obtener información del medio o entorno (mapas, carreteras, imágenes de todo tipo, etc.) y poder ser utilizado en resolución de problemas o toma de decisiones por parte de una computadora, basado en su entorno o aplicación [4]

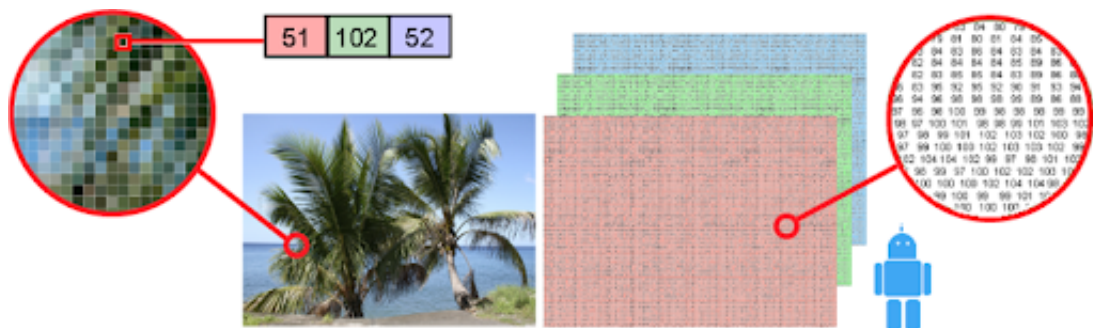


Figura 1: Ejemplo 1

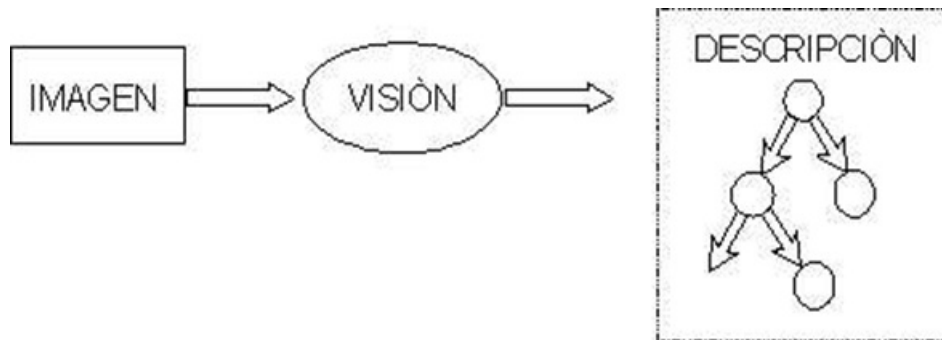


Figura 2: Ejemplo 2

## OpenCV

Esta es la librería de **Open Source Computer Vision Library** y está disponible para Windows, MacOS y Linux. Es una librería para visión por computadora y machine learning. Incluye mas de 2500 algoritmos que permiten ser utilizados para reconocimiento de rostros, identificación de objetos, clasificación del comportamiento humano, rastreo del movimiento de los ojos entre otros.

Su implementación puede realizarse en C++, Python, Java y Matlab para las diferentes aplicaciones mencionadas. [5]

## Calibración de Cámaras

La calibración de cámaras es una herramienta útil al momento de querer obtener información con respecto a una imagen. La calibración permite que la cámara pueda ser utilizada en aplicaciones como Realidad Aumentada, seguimiento y reconstrucción 3D entre otros[6].

Básicamente, consiste en obtener los parámetros internos de la cámara (como píxeles, tamaño de imagen, etcétera) y con esto, se pretende establecer un marco referencial con respecto al mundo real, esto con el objetivo de unir el marco de referencia del mundo real con el de la imagen. Para esto, existen técnicas de calibración como la Calibración fotogramétrica, que es la observación de objetos 3D con geometría conocida y buena precisión, utilizando los planos ortogonales es posible, mediante configuraciones elaboradas, obtener una calibración y resultados eficientes; así como la Autocalibración, que consiste en tomar una serie de fotos fijas para obtener los parámetros intrínsecos y extrínsecos de la cámara.[6]



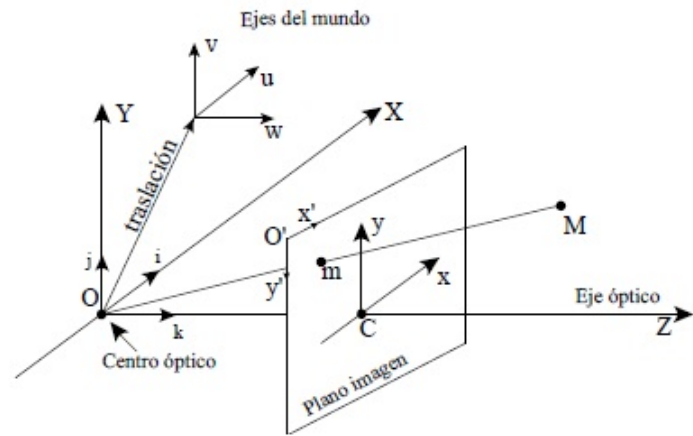


Figura 3: Conversión entre el plano de imagen al mundo real

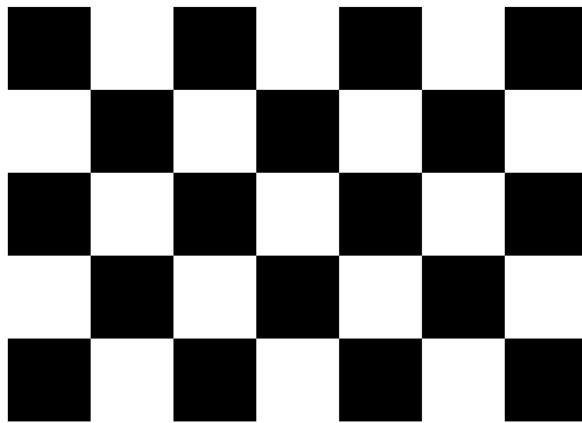


Figura 4: Patrón común para la calibración

## Programación Multi-hilos

Los procesadores orientados a multiprocesos, permiten realizar diferentes tareas al mismo tiempo. Estos a su vez, son responsables de manejar los recursos que se le asignen a cada uno de estos.

Más específicamente, cuando un programa es ejecutado, la computadora crea algo llamado **proceso** que contiene toda la información relevante del programa, como su identificador por ejemplo, hasta que dichos los programas terminan su ejecución.

Los sistemas operativos multiprocesos, permiten la ejecución de varios programas de manera simultánea y es la computadora la encargada de asignar los recursos a los diferentes programas que los necesiten. El objetivo principal es obtener un uso adecuado de los recursos del CPU. [7]

Un hilo, por tanto, es una unidad de control dentro de un proceso. El programa corre un hilo principal o **main thread** encargado de crear otros hilos según sea su programación, siendo este el principio básico del multi-hilos. Básicamente, orientar los programas ejecutados para realizar varias tareas y en muchos casos realizar procesos más eficientes. [7]

Existen ventajas en el uso de multi-hilos. El primero, es la ventaja de tener un programa realizando captura o procesamiento de información, mientras otro está esperando estas salidas. Es decir, en lugar de tener un programa que espera una imagen o dato, para luego procesarla y dar un resultado, se pueden tener dos procesos corriendo, uno capturando datos y el otro procesando. El resultado de esto será mucho más rápido.

Además, mencionar que la comunicación multi-hilos es mucho más eficiente que la comunicación entre procesos [7]

## Programación Orientada a Objetos

Es un enfoque para la organización y el desarrollo de programas que intenta incorporar características más potentes y estructuradas para la programación. Es una nueva forma de organizar y desarrollar programas y no tiene nada que ver con ningún lenguaje de programación específico. Sin embargo, no todos los lenguajes son adecuados para implementar fácilmente los conceptos de la Programación orientada a objetos (POO). [8]

### Objetos

Un objeto en POO cumple básicamente las mismas funciones que un objeto de la vida real. Un objeto puede guardar atributos o características y ser utilizadas mediante métodos. Esto es útil porque permite de alguna forma esconder esta parte interna para solo enfocarse en su función principal (aquella para la cual fueron diseñados). A esto se le conoce como *encapsulación*. [9]

## **Clase**

Una clase es el plano a partir del cual se crean métodos o atributos para los diferentes objetos individuales que pertenecen a esta clase. Es, lo que se podría llamar, como un molde o modelo para la creación de objetos. [10] [9]

## **Encapsulación**

Esta es una de las características más importantes dentro de la POO. Básicamente se refiere a que todo lo referente a un objeto quede dentro de él, es decir, que solo se pueda acceder a sus propiedades mediante el uso de los métodos y propiedades que se proporcionen en la clase. [11]

## **Abstracción**

Otra propiedad importante dentro de la POO es la abstracción. Como su nombre lo indica, es la capacidad de poder mostrar las características del objeto hacia el mundo exterior (en un programa, por ejemplo) pero quitándole la complejidad de dichos métodos. Es decir, que el usuario se ocupe únicamente de entender que funciones tiene más allá de entender como funcionen a lo interno[11]



#### Investigación

Se buscará información referente a los lenguajes de programación para alcanzar los objetivos. El primer punto es investigar el funcionamiento de la programación multi-hilos y la sintaxis para los lenguajes seleccionados (funciones, restricciones de uso, diferencias entre ellos, etc.), así como todo lo referente a la librería de OpenCV para comprender su uso y realizar las pruebas que se requieren con la cámara.

Entre este paso también se puede incluir investigar sobre la programación orientada a objetos como una herramienta para el desarrollo de un algoritmo eficiente para aplicaciones de robótica de enjambre.

#### Implementación

Una vez investigado lo referente a la programación multi-hilos y OpenCV, se procederá a realizar códigos de prueba para validar el funcionamiento de las diferentes aplicaciones que se necesiten implementar. Esto implica programas ejemplo de aplicación multi-hilos, así como el uso de una cámara y OpenCV. Para esto, se tiene estipulado el uso de los lenguajes de C++ y Python

Luego de haber realizado y comprendido el funcionamiento, se procederá a analizar los programas existentes (implementados en el lenguaje C++) para intentar encontrar puntos de mejora y poder aplicar lo investigado. Finalmente, se realizará una migración hacia el lenguaje Python para tener puntos de comparación.

Además de esto, se requiere implementar una mesa de pruebas que busca ser una herramienta para validar el uso de OpenCV y sus aplicaciones en el procesamiento de imagen. Por

lo que, con la finalidad de reconocer el entorno para la pose de agentes y realizar calibraciones a la cámara, se utilizará esta mesa para realizar las pruebas necesarias.

## Validación

Finalmente, luego de tener dos elementos de comparación, se procederá a realizar pruebas para obtener parámetros que permitan medir y validar cual de las dos opciones es la mejor, o si ambas aportan de igual forma.

Uno de las primeras pruebas es realizar comparaciones entre la programación multi-hilos de ambos lenguajes definidos. Esto con el fin de medir su rendimiento y comparar el resultado de las tareas que se dispongan en esta primera prueba.

La segunda prueba consiste en la calibración de la cámara y el procesamiento de imágenes. Utilizando OpenCV y la mesa de pruebas diseñada, se pretende que de manera autónoma la computadora pueda reconocer la mesa y tomar acciones de calibración con la cámara. Para esto se tiene ya implementado un código en `C++` (fase anterior de esta tesis) el cual se comparará con los resultados en `Python` realizando el mismo procedimiento, comparando además la sintaxis y facilidad de programación como primer punto comparativo entre estos dos lenguajes.

Al final, las métricas de validación pueden ir desde el rendimiento de la computadora, tiempos de ejecución, entre otros, así como que los resultados para los cuales el programa fue diseñado sean los adecuados (correcta calibración de la cámara, obtención de resultados del procesamiento de imágenes adecuado, etc.).

---

### Prototipos de Mesa de Pruebas

---

#### 8.1. Primer Prototipo

Con el objetivo de poder simular a escala la mesa de pruebas que se encuentra en el laboratorio de la UVG, se realizaron dos prototipos de dicha mesa. El primero se realizó con un tablero o pizarrón pequeño y una base robusta para colocarla cámara, como se observa en las figuras 5 y 6. Como se puede observar, se colocaron asteriscos en las esquinas de la mesa que representar los puntos donde se deseaba calibrar y la base estaba montada sobre ella y la cámara colocada para tener la visión superior. Sin embargo esto presentaba ciertos problemas en cuanto a la iluminación y captura correcta de las imágenes. Analizando esto, se realizó un segundo prototipo de dicha mesa, como se muestra en la figura 8, además de tener un mejor base para la cámara como se muestra en la figura 7



Figura 5: Primer protitpo de mesa



Figura 6: Primer prototipo de mesa y cámara



## 8.2. Segundo Prototipo

Como se mencionó, el primer prototipo de mesa tenía problemas, ya que la base de la cámara creaba una sombra sobre el tablero, lo cual era un problema para las pruebas, ya que en Visión por Computadora, la iluminación juega un papel importante. Por lo que, para este segundo prototipo se procedió a conseguir un trípode en el cual montar la cámara (figura 7). Esto daba una mejor colocación de la cámara, así como evitar el problema de la sombra sobre la mesa. Finalmente, se utilizó un cartón de dimensiones de 14 cm x 28 cm para simular la mesa y de igual forma se le colocaron puntos en las esquinas para referencia en la calibración (figura 8)



Figura 7: Armado de base de cámara para el segundo prototipo



Figura 8: Segundo prototipo de mesa de pruebas



Figura 9: Colocación para el segundo armado de la mesa de pruebas

---

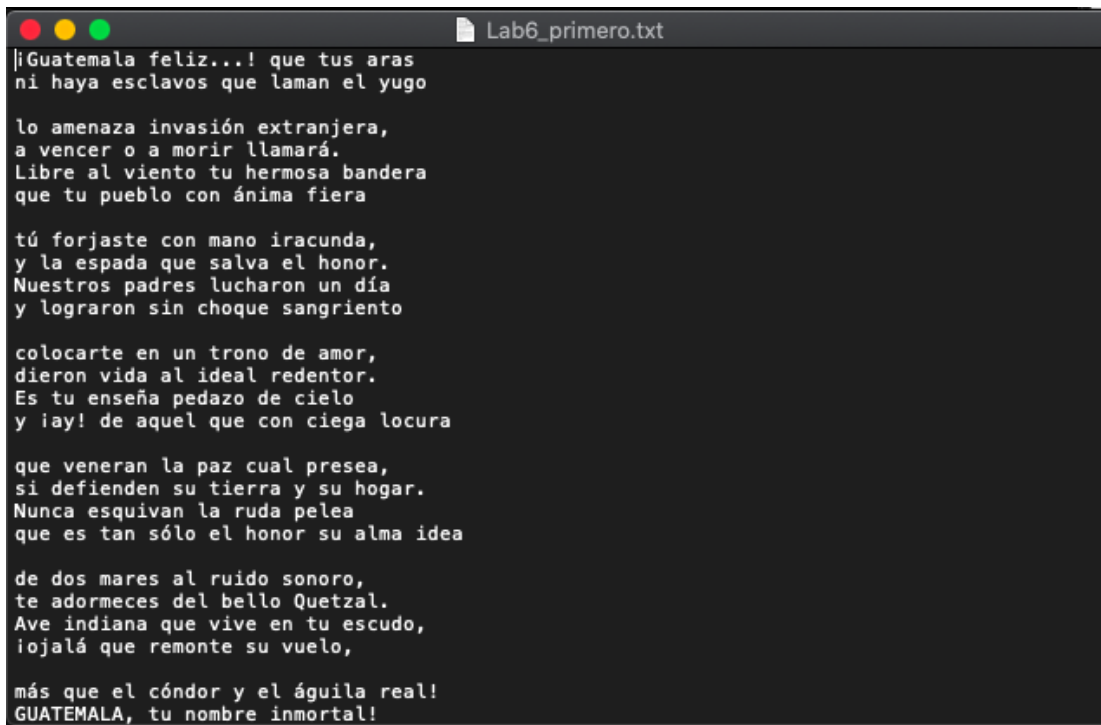
### Pruebas preliminares en Python y C++

---

Con el objetivo de validar el correcto funcionamiento de los lenguajes a utilizar (Python y C++) se realizaron algunas pruebas para entender su funcionamiento. La primera prueba fue realizada en Python para verificar el funcionamiento de los multi-hilos e ilustrarlos de igual forma.

En la figura 10 se observa un primer archivo de texto del himno nacional de Guatemala pero con las líneas impares. Luego, en la figura 11, se encuentra la otra parte del himno, pero con las líneas pares.

El objetivo era desarrollar un programa en lenguaje Python, con multi-hilos, que tomara ordenadamente cada línea de cada archivo y lo ordenara en uno nuevo, como se muestra en la figura 12.



```
Lab6_primer.txt
¡Guatemala feliz...! que tus aras
ni haya esclavos que laman el yugo

lo amenaza invasión extranjera,
a vencer o a morir llamará.
Libre al viento tu hermosa bandera
que tu pueblo con ánima fiera

tú forjaste con mano iracunda,
y la espada que salva el honor.
Nuestros padres lucharon un día
y lograron sin choque sangriento

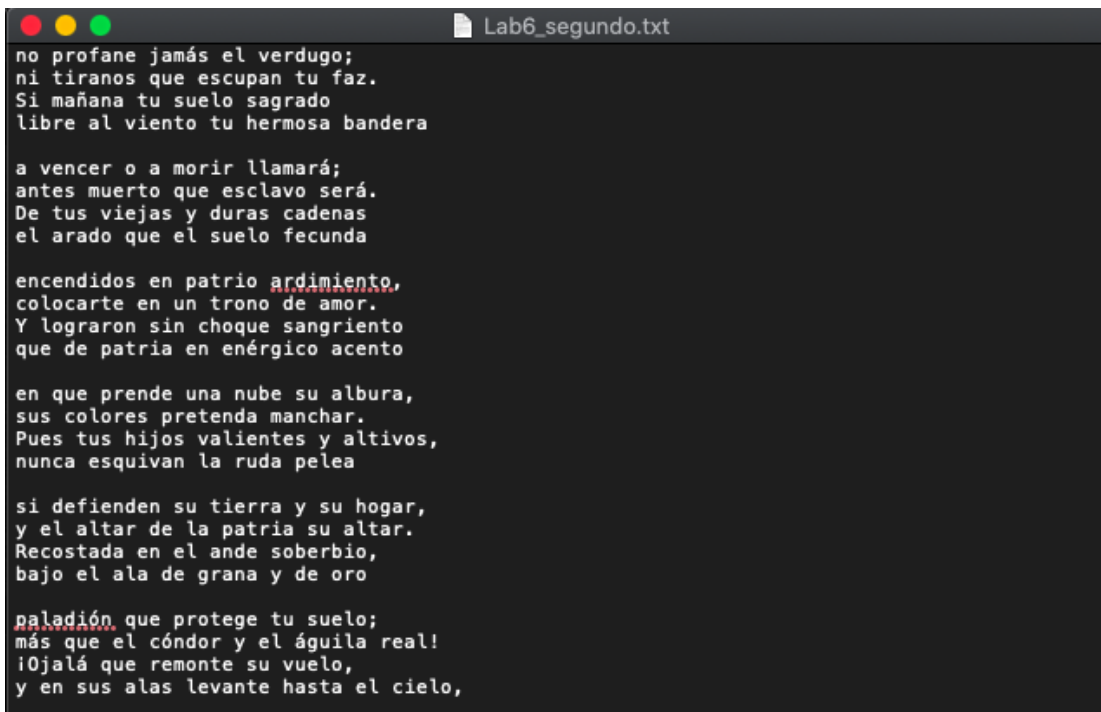
colocarte en un trono de amor,
dieron vida al ideal redentor.
Es tu enseña pedazo de cielo
y ¡ay! de aquel que con ciega locura

que veneran la paz cual presea,
si defienden su tierra y su hogar.
Nunca esquivan la ruda pelea
que es tan sólo el honor su alma idea

de dos mares al ruido sonoro,
te adormeces del bello Quetzal.
Ave indiana que vive en tu escudo,
¡ojalá que remonte su vuelo,

más que el cóndor y el águila real!
GUATEMALA, tu nombre inmortal!
```

Figura 10: Captura del primer archivo de texto para el ejemplo Multi-hilos



```
Lab6_segundo.txt
no profane jamás el verdugo;
ni tiranos que escupan tu faz.
Si mañana tu suelo sagrado
libre al viento tu hermosa bandera

a vencer o a morir llamará;
antes muerto que esclavo será.
De tus viejas y duras cadenas
el arado que el suelo fecunda

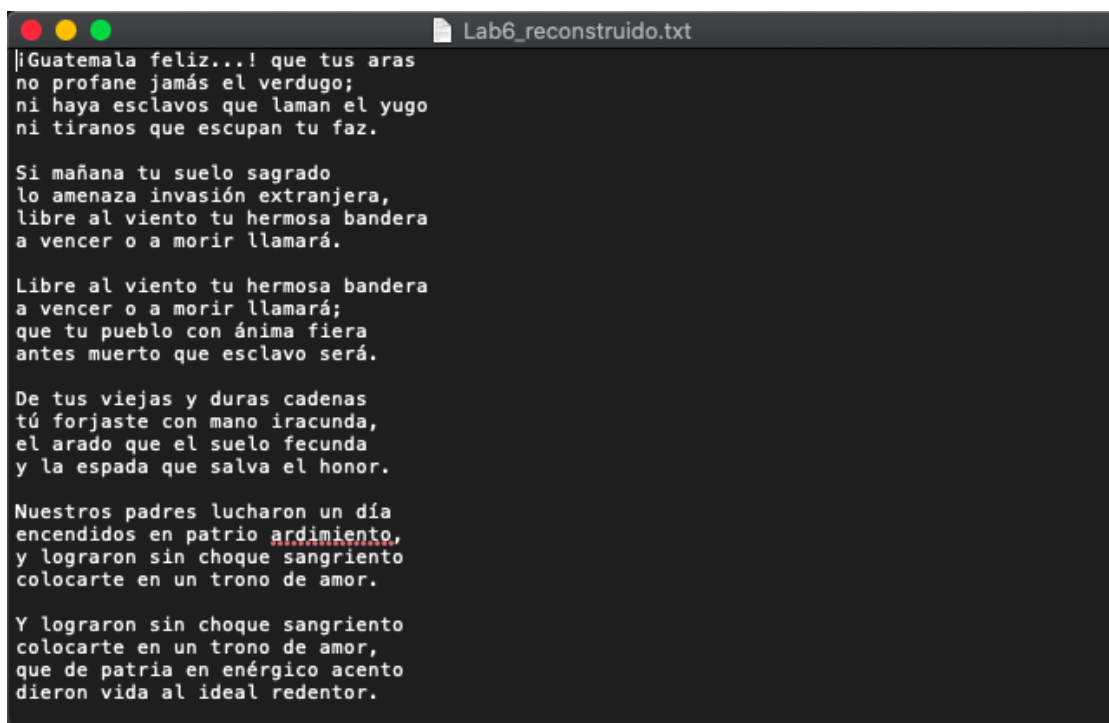
encendidos en patrio ardimiento,
colocarte en un trono de amor.
Y lograron sin choque sangriento
que de patria en enérgico acento

en que prende una nube su albura,
sus colores pretenda manchar.
Pues tus hijos valientes y altivos,
nunca esquivan la ruda pelea

si defienden su tierra y su hogar,
y el altar de la patria su altar.
Recostada en el ande soberbio,
bajo el ala de grana y de oro

paladión que protege tu suelo;
más que el cóndor y el águila real!
¡Ojalá que remonte su vuelo,
y en sus alas levante hasta el cielo,
```

Figura 11: Captura del segundo archivo de texto para el ejemplo Multi-hilos



```
|¡Guatemala feliz...! que tus aras  
no profane jamás el verdugo;  
ni haya esclavos que laman el yugo  
ni tiranos que escupan tu faz.  
  
Si mañana tu suelo sagrado  
lo amenaza invasión extranjera,  
libre al viento tu hermosa bandera  
a vencer o a morir llamará.  
  
Libre al viento tu hermosa bandera  
a vencer o a morir llamará;  
que tu pueblo con ánimo fiero  
antes muerto que esclavo será.  
  
De tus viejas y duras cadenas  
tú forjaste con mano iracunda,  
el arado que el suelo fecunda  
y la espada que salva el honor.  
  
Nuestros padres lucharon un día  
encendidos en patrio ardimiento,  
y lograron sin choque sangriento  
colocarte en un trono de amor.  
  
Y lograron sin choque sangriento  
colocarte en un trono de amor,  
que de patria en enérgico acento  
dieron vida al ideal redentor.
```

Figura 12: Captura del texto reconstruido para el ejemplo Multi-hilos

Otras de las pruebas realizadas en Python fue probar la librería de OpenCV para entender su funcionamiento y aplicar ciertas funciones que se usaron en los algoritmos propuestos. Para esta prueba se realizo un vídeo, es decir, la cámara estaba tomando un vídeo en tiempo real y se aplicaba un filtro de grises para luego mostrar la imagen en blanco y negro, como lo ilustra la imagen siguiente.



Figura 13: Captura de la cámara web con OpenCV

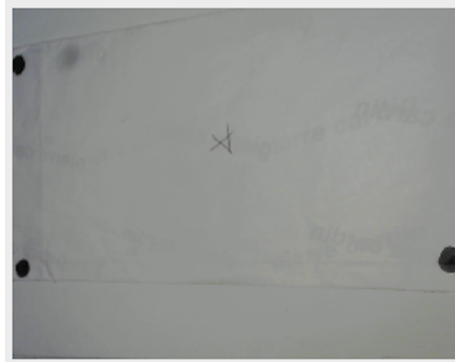
### 10.1. Calibración en C++

Como se observa en las figuras siguientes, se realizaron pruebas del algoritmo de calibración en C++. La figura 14 se tuvo una calibración fallida de la mesa (los puntos identificados no fueron colocados en la esquina de la imagen). Esto debido a unos parámetros de identificación basados en píxeles. Sin embargo, los píxeles pueden variar de manera diversa según sea la resolución de la cámara, la posición, iluminación, entre otros factores. El objetivo de esta condicional basada en los píxeles era evitar que figuras o contornos no deseados dentro o fuera de la mesa fueran tomadas como parámetros o puntos de calibración.

Por lo que, para hacerlo de manera más generalizada a cualquier escenario, se procedió a eliminar esta condición y que se basará únicamente en la ubicación de los puntos en la imagen. Los puntos más cercanos (los marcados en negro en la figura 14a u 15a) serán tomados como las esquinas de la mesa y por tanto, la imagen se calibraría, como se puede observar en la figura 15b (y como a pesar de tener objetos aún se logró el resultado deseado).

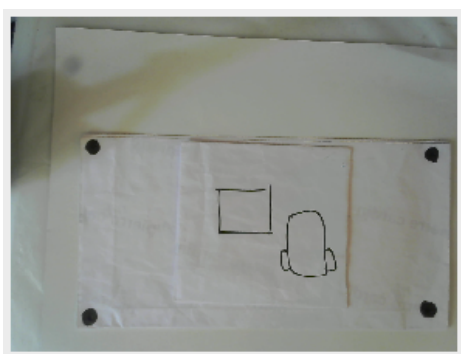


(a) 1a

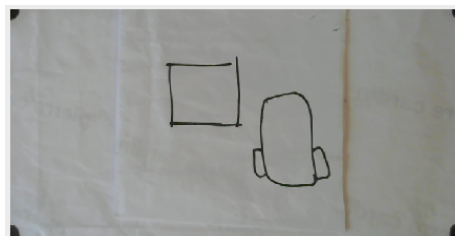


(b) 1b

Figura 14: Intento 1 de calibración utilizando C++



(a) 1a



(b) 1b

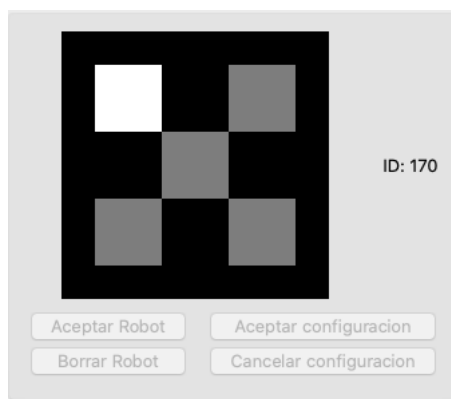
Figura 15: Intento 2 de calibración utilizando C++



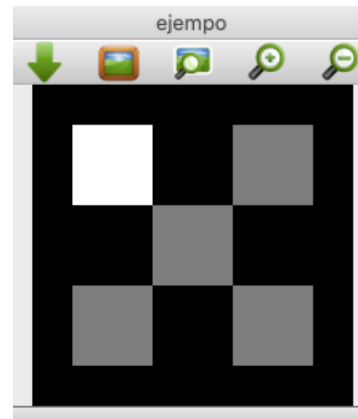
## 10.2. Identificación de Códigos en C++

Además de las pruebas de calibración, también se realizaron pruebas de identificación de marcadores. Estas pruebas consistían en capturar imágenes de la mesa con los marcadores (los códigos mostrados en las figuras 16a, 16b y 17) para poder saber que ID representan y la posición en la mesa. Como se observa en la figura 17, luego de identificar al respectivo marcador, se procede a recortarlo de la imagen completa, rotarlo y poner el pivote (el cuadro blanco) en la esquina superior izquierda. Posterior a esto, se realiza la identificación.

El resultado es un cuadro de imagen que se llama **ejemplo** como se muestra en la figura 16b y en la figura 16a donde ya se muestra el valor del ID. Para el caso de las dos figuras anteriores, el código era 170 y para la figura 17 era 40.



(a) Identificación de Código 170



(b) Código generado después de ser identificado

Figura 16: Identificación de código

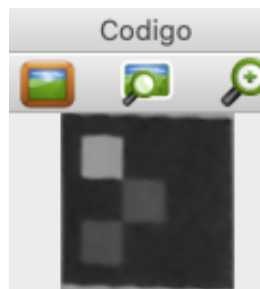


Figura 17: Primera prueba de identificación de código en la imagen capturada



## 11.1. Migración del Código para la Calibración de Cámara

Como es posible observar en es posible observar en el código 11.1 para obtener las esquinas se utiliza la función `minAreaRect` (línea 6), esto encierra las figuras encontradas en pequeños cuadros (los más pequeños que los puedan encerrar) y con esto, se calcula el centro de dicho cuadro para compararlo con las esquinas iniciales y determinar si esta esquina está más cerca del borde de la imagen. Sin embargo, para Python se modifico dicho procedimiento.

Lo que se hizo en el código 11.2 fue utilizar la función `cv.approxPolyDP()`. Esta función busca obtener una aproximación de los contornos. Finalmente, para detectar contornos lo más circular posible, se procedió a comparar el resultado de la aproximación y el área del contorno, si este estaba dentro del rango entonces se añadía a una lista que guardaba dichos contornos (líneas 2 a la 6). Con esto, se obtienen los centros de cada contorno en la lista (línea 17 y 18) y se comparan de igual forma con los esquinas predefinidas (línea 11) para finalmente determinar cual esta más cerca del borde de la imagen y tomarlo como la nueva esquina de la imagen calibrada.

En ambos casos, la calibración es correcta como se ha mostrado en figuras anteriores, dando validez a ambos métodos para la detección de las esquinas. El objetivo de hacerlo así en Python era poder enfocarse en los contornos circulares únicamente, olvidando otros elementos que pudieran interferir en su detección.

```

1 Point bordesMAX[4] = { Point(0, 0) , Point(0, drawing.size().height) , Point
  (drawing.size().width, 0) , Point(drawing.size().width, drawing.size().
    height) };
2
3 for (int i = 0; i < contours.size(); i++)
4 {
5     RotatedRect cod;
6     cod = minAreaRect(contours[i]);
7     if (a) {
8         for (int i = 0; i < 4; i++)
9             esquina[i] = cod.center;
10        a = false;
11    }
12    else {
13        for (int i = 0; i < 4; i++)
14            if (distancia2puntos(bordesMAX[i], cod.center) <
15                distancia2puntos(bordesMAX[i], esquina[i]))
16                esquina[i] = cod.center;
17    }
18 }

```

Listing 11.1: Detección de esquinas para calibrar cámara en C++

```

1 for con in contour:
2     approx = cv.approxPolyDP(con, 0.01*cv.arcLength(con, True), True) #utiliza
    el metodo de aproximacion approxPolyDP para encontrar los contornos
    circulares.
3     area = cv.contourArea(con) #calcula el area de este contorno.
4
5     if ((len(approx) > 8) & (area > 3) ): #Treshold aproximado, puede variar
    si se desea para detectar circulos
6         contour_list.append(con) #si cumple, lo agrega a la lista
7
8 Cx = 0 #coordenada en x
9 Cy = 0 #coordenada en y
10
11 esquinas_final = [[1,1], [1,2], [2,1], [2,2]] #un valor inicial para la
    comparativa
12 a = True
13 for c in contour_list: #recorre la lista de contornos para buscar el centro
    # calcula el centro
14     M = cv.moments(c)
15 #ver documentacion para obtener mas informacion de como se calcula la
    coordenada (x,y)
16     Cx = int(M["m10"] / M["m00"])
17     Cy = int(M["m01"] / M["m00"])
18 #agrega la primera esquina en la posicion inicial.
19     if (a):
20         esquinas_final[0] = [Cx, Cy]
21         a = False
22
23
24     for i in range (0,4):
25         if (distancia2puntos(boardMax[i], (Cx, Cy)) < distancia2puntos(boardMax
    [i], esquinas_final[i])):
26             esquinas_final[i] = [Cx, Cy]

```

Listing 11.2: Detección de esquinas para calibrar cámara en Python

## 11.2. Migración del Código para Obtención de Pose e identificación

### 11.2.1. Identificación del ID

Para la detección del ID, en el código 11.4 de Python se ubican manualmente los cuadros dentro del marcador/identificador que representan el número con el que será identificado el robot. Lo que se hace es cortar manualmente la imagen del marcador para obtener cada cuadro que conforma el código identificador. Estos finalmente se añaden a un array de 8 posiciones y se compara mediante una condición. Dicha condición evalúa si el número está dentro del rango establecido (que puede variar dependiendo la luz que se tenga presente en el lugar) y si cumple lo marca como un 1, sino, lo marca como un 0. Finalmente se convierte en un número entero para obtener el ID respectivo (línea 31). Como se puede ver, convertir el binario a un entero resulta más simple en la implementación de Python que en C++

```
1  cout << "Encontrando los valores de la matriz" << endl;
2  for (int u = 1; u <= 3; ++u) {
3      for (int v = 1; v <= 3; ++v) {
4          int ValColorTemp = finalCropCodRotated.at<uchar>((
5              finalCropCodRotated.size().height * u / 4), (finalCropCodRotated.size().
6              width * v / 4));
7          MatrizValColores[(u - 1)][(v - 1)] = ValColorTemp;
8          if ((ValColorTemp < EscalaColores[2] - GlobalColorDifThreshold) && (
9              ValColorTemp > EscalaColores[0] + GlobalColorDifThreshold)) {
10              EscalaColores[1] = ValColorTemp;
11          }
12      }
13  }
14  //Extraemos el codigo binario
15  cout << "Extraemos el codigo binario" << endl;
16  string CodigoBinString = "";
17  for (int u = 0; u < 3; ++u) {
18      for (int v = 0; v < 3; ++v) {
19          if ((u == 0) && (v == 0))
20              CodigoBinString = CodigoBinString;
21          else if ((MatrizValColores[u][v] > EscalaColores[1] -
22              GlobalColorDifThreshold) && (MatrizValColores[u][v] < EscalaColores[1] +
23              GlobalColorDifThreshold))
24              CodigoBinString = CodigoBinString + "1";
25          else
26              CodigoBinString = CodigoBinString + "0";
27      }
28  }
```

Listing 11.3: Reconocimiento del ID en C++

```

1 """
2 Luego, se ubica cada cuadro manualmente, es decir, desde a0 hasta a7, se
3 busca dentro de la imagen. Esto se hace manual para tener un
4 mejor control de la ubicacion y que sea mas facil cambiarlo si se desea.
5 Al igual que con las esquinas, se toma el valor medio del cuadro (tratando
6 de buscar justo el centro) para tener un valor
7 de referencia de que color se esta identificando (en este caso, negro o gris
8 para 0 o 1 respectivamente).
9 """
10 temp_a1 = resized[int(height_Final_Rotated*1/8):40, 65:100]
11 temp_a5 = resized[int(height_Final_Rotated*1/4 + 42):int(
12     height_Final_Rotated*1/2 + 40), int(height_Final_Rotated*1/8):int(
13     height_Final_Rotated*1/8 + 23)]
14 temp_a7 = resized[int(height_Final_Rotated*1/2) + 15:int(
15     height_Final_Rotated*1/2) + 45, int(width_Final_Rotated*1/2)+20 :int(
16     width_Final_Rotated*1/2) + 45]
17
18 ...
19
20 """
21 La extraccion del codigo se hace mas facil utilizando las funciones dentro
22 de python para el manejo de numeros binarios
23 y decimales. Se define un TRESHOLD_DETECT_MAX y TRESHOLD_DETECT_MIN para que
24 el valor que tome como gris se adecuado.
25 Estos valores pueden variar segun la luz (normalmente varia
26 TRESHOLD_DETECT_MIN)
27 """
28 i = 0 #para evitar alguna sobrescritura de esta variable.
29 for i in range (0, len(code)):
30
31     #con los thresholds establecidos, busca que valores sean grises y los
32     cataloga como 1, sino, los cataloga como 0.
33     if code[i] > TRESHOLD_DETECT_MIN and code[i]< TRESHOLD_DETECT_MAX:
34
35         CodigoBinString = CodigoBinString + "1"
36     else:
37         CodigoBinString = CodigoBinString + "0"
38
39     ...
40
41 #esta funcion pasa el string de bits a formato de numero int.
42 tempID =int(CodigoBinString, 2)

```

Listing 11.4: Reconocimiento del ID en Python

---

### Pruebas Algoritmo y OpenCV en Python

---

Para las pruebas en `Python`, primero se realizó la migración del código (o algoritmo) planetado en `C++`. Una vez hecha dicha migración, se realizaron pruebas para validar su funcionamiento.

Una de las primeras pruebas son las que se muestran en las figuras siguientes. El objetivo principal del algoritmo en `Python` era identificar figuras o contornos circulares (como lo muestra la figura 18) y al final, cuando estos contornos fueran identificados, se procedía a ubicar los más cercanos a los bordes de la imagen y estos eran tomados como las nuevas esquinas (como se ejemplifica en la figura mencionada)

Otra de las pruebas fue realizar la misma calibración simulando teniendo objetos en la mesa. La figura 19 muestra un dibujo que simula ser un robot en la mesa. Como se observa, el algoritmo lo identifica, pero al no estar cerca de los bordes de la imagen, no es tomado como una esquina para la calibración.

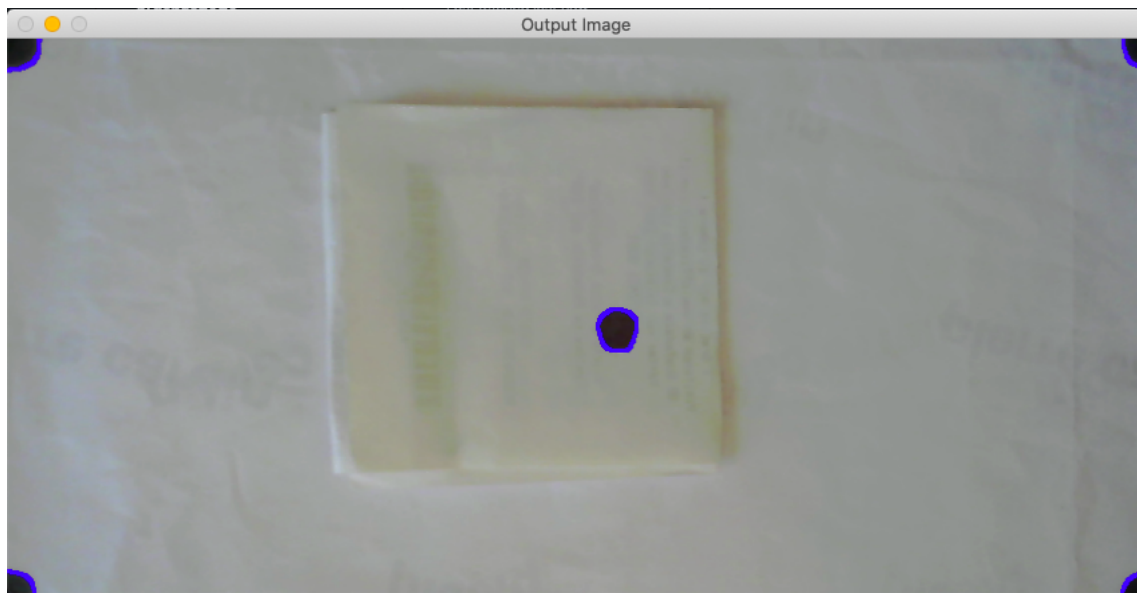


Figura 18: Prueba calibración de la cámara con Python

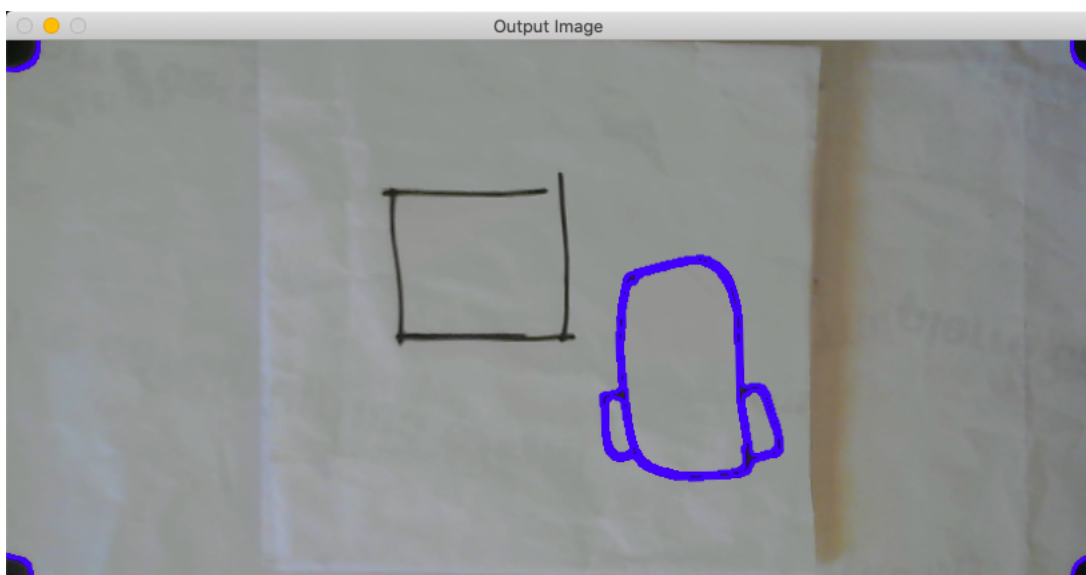


Figura 19: Segunda prueba calibración de la cámara con Python



---

### Pruebas de generación de marcadores/códigos y detección en Python

---

La creación de los códigos o marcadores fue realizado también migrando el código en `C++`. Este genera un código entre 0 y 255 que posteriormente es mapeado a una imagen como se muestra en la figura 21, que representa el número 40. Luego de eso, la impresión y colocación de ese marcador queda a discreción del usuario (tanto en tamaño como en posición).

Finalmente, las figuras 22, 23, 24 y 25 muestran como el algoritmo modificado en `Python`, toma a la imagen original y la reescala a un tamaño estándar para su identificación.

Cabe mencionar que la identificación en códigos menores a tamaño de 3x3 cm la identificación se hace complicada debido a que al reescalarlo, se pierde definición de la imagen. Aunque es posible que en condiciones de luz adecuada (luz directa sobre la mesa y los códigos, de preferencia un color blanco) las imágenes de menor tamaño pueden ser mejor identificadas. Aunque estas mismas condiciones también ayudan a una mejor identificación de los marcadores en tamaños mayores a 3x3 cm

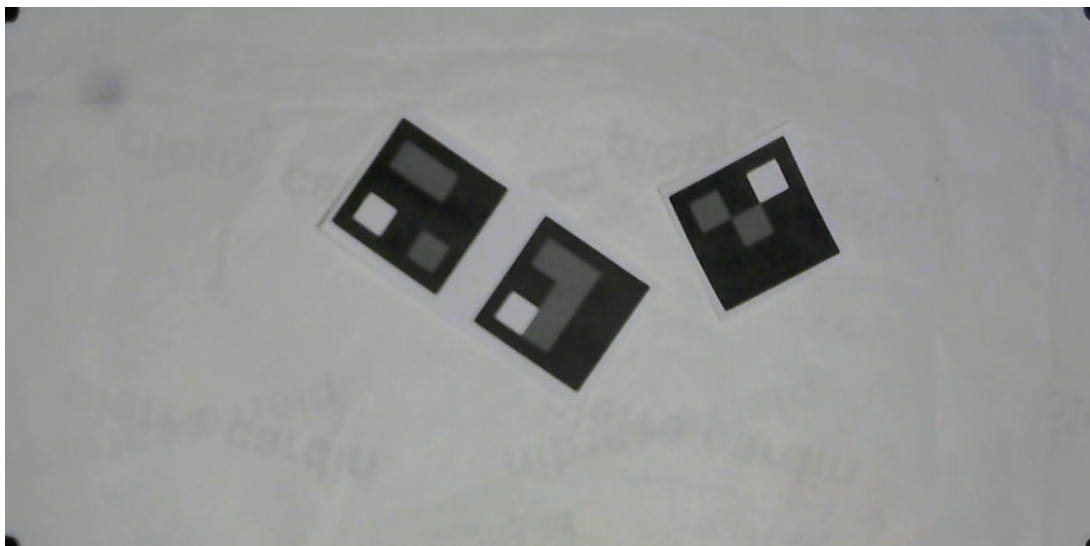


Figura 20: Calibración de la cámara para detección de los códigos utilizando Python

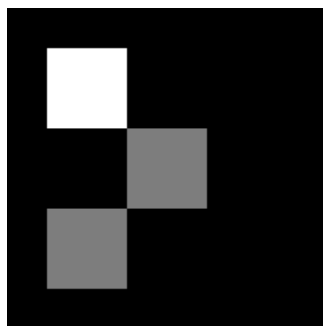
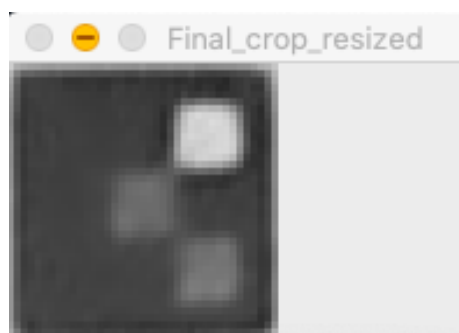


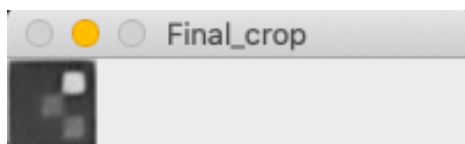
Figura 21: Prueba de generación de código utilizando Python



(a) Código rotado para la detección

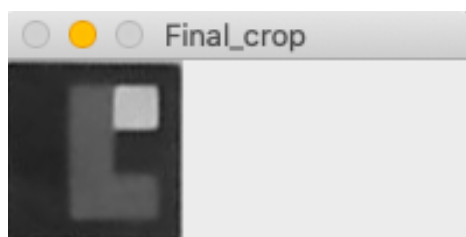


(b) Imagen aplicándole un reescalamiento



(c) Imagen original

Figura 22: Detección y reescalamiento de código generado para tamaño de 1x1 cm



(a) Imagen original



(b) Imagen rotada y reescalada para la detección del código

Figura 23: Detección y reescalamiento de código generado para tamaño de 2x2 cm

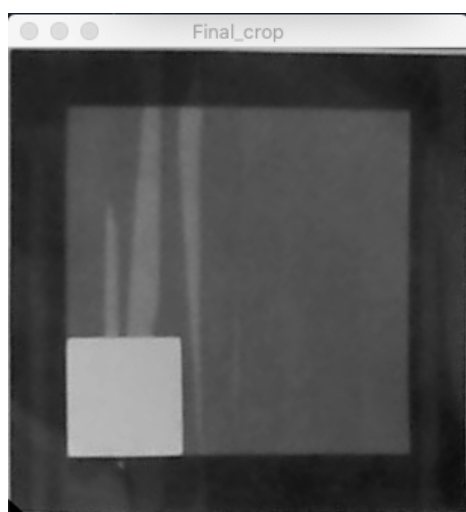


(a) Imagen rotada y reescalada para la detección del código



(b) Imagen original

Figura 24: Detección y reescalamiento de código generado para tamaño de 4x4 cm



(a) Imagen original



(b) Imagen rotada y reescalada para la detección del código

Figura 25: Detección y reescalamiento de código generado para tamaño de 8x8 cm



### 14.1. Pruebas de detección de pose

Las primeras pruebas de comparación entre ambos algoritmos fue la de validar la posición detectada de cada robot en la mesa. Para esto, se utilizó un papel milimetrado (con divisiones entre centímetros y milímetros) para saber cual era la posición exacta físicamente en la mesa (Posición Real). El algoritmo se corrió en ambos lenguajes y como es posible observar en el cuadro 1, la posición es igual en ambos casos (tanto Python como C++) y de manera similar con las posiciones físicas en la mesa.

Python			C++			Posición Real	
X	Y	theta	X	Y	Theta	X	Y
9	7	85	9	7	85	9.5	7.5
5	3	90	5	3	91	5	3
19	13	90	19	13	90	20	13
11	4	91	11	4	90	12	4.7
14	8	147	14	8	147	14.5	8
19	13	25	19	13	25	19.5	13.5
3	13	-121	3	13	-121	3.5	13.5
10	8	-68	10	8	-68	10.5	8
19	7	114	19	7	114	20	7.5
9	7	177	9	7	176	9	7.5

Cuadro 1: Comparación entre la detección de pose del algoritmo en Python-C++ y la posición real

## 14.2. Comparación de rendimiento

## CAPÍTULO 15

---

Conclusiones

---





## CAPÍTULO 16

---

Recomendaciones

---



- [1] Luis Antón Canalís, “Swarm Intelligence in Computer Vision: an Application to Object Tracking”, Univesidad de las Palmas de Gran Canaria, mar. de 2010.
- [2] Javier Andrés Lizarazo Zambrano y Mario Alberto Ramos Velandia, “Visión artificial y comunicación en robots cooperativos omnidireccionales”, Universidad Central, Facultad de Ingeniería y Ciencias Básicas, ene. de 2016.
- [3] André Josué Rodas Hernández, “Desarrollo e implementación de algoritmo de visión por computador en una mesa de pruebas para la experimentación con micro-robots móviles en robótica de enjambre”, Departamento de ingeniería electrónica, mecatrónica y biomédica, Universidad del Valle de Guatemala, ene. de 2019.
- [4] Raúl E. López Briega, *Visión por computadora*, <https://iaarbook.github.io/vision-por-computadora/>, visitado el 10/05/2020.
- [5] OpenCV, *About*, <https://opencv.org/about/>, visitado el 05/04/2020.
- [6] D. A. Pizarro, P. Campos y C. L. Tozzi, “COMPARACIÓN DE TÉCNICAS DE CALIBRACIÓN DE CÁMARAS DIGITALES”, *Universidad Tarapacá*, vol. 13, n.º 1, págs. 58-67, 2005. DOI: <https://scielo.conicyt.cl/pdf/rfacing/v13n1/art07.pdf>.
- [7] R. H. Carver y K.-C. Tai, *Modern multithreading: implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs*. John Wiley & Sons, 2005.
- [8] P. Chawla, *OOP with C++*, <http://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf>, visitado el 05/06/2020.
- [9] Oracle, *Lesson: Object-Oriented Programming Concepts*, <https://docs.oracle.com/javase/tutorial/java/concepts/index.html>, visitado el 05/06/2020.
- [10] E. Doherty, *What is Object Oriented Programming? OOP Explained in Depth*, [https://www.educative.io/blog/object-oriented-programming?aid=5082902844932096&utm\\_source=google&utm\\_medium=cpc&utm\\_campaign=blog-dynamic&gclid=Cj0KCQjwoPL2BRDxARIsAEMm9y\\_jZ7Mu3oH1wRJc5uHdWIeMdhGHbLeR22kkNxokV1-YZS-isYFjZKIaArzfEALw\\_wcB](https://www.educative.io/blog/object-oriented-programming?aid=5082902844932096&utm_source=google&utm_medium=cpc&utm_campaign=blog-dynamic&gclid=Cj0KCQjwoPL2BRDxARIsAEMm9y_jZ7Mu3oH1wRJc5uHdWIeMdhGHbLeR22kkNxokV1-YZS-isYFjZKIaArzfEALw_wcB), visitado el 05/06/2020.

- [11] J. M. Alarcón, *Los conceptos fundamentales sobre Programación Orientada Objetos explicados de manera simple*, <https://www.campusmvp.es/recursos/post/los-conceptos-fundamentales-sobre-programacion-orientada-objetos-explicados-de-manera-simple.aspx>, visitado el 19/08/2020.

## CAPÍTULO 18

---

Anexos

---

### 18.1. Planos de construcción

