

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.

Prefacio	v
Lista de figuras	ix
Lista de cuadros	xi
Resumen	xiii
Abstract	xv
1. Introduucción	1
2. Antecedentes	3
3. Justificación	5
4. Objetivos	7
5. Alcance	9
6. Marco teórico	11
7. Metodología	15
8. Prototipo de Mesa de Pruebas	17
9. Pruebas preliminares en Python y C++	21
10.Pruebas Algoritmo y OpenCV en C++	25
11.Pruebas Algoritmo y OpenCV en Python	27
12.Pruebas de generación de marcadores/códigos y detección en Python	29
13.Conclusiones	33

14.Recomendaciones	35
15.Bibliografía	37
16.Anexos	39
16.1. Planos de construcción	39

Lista de figuras

1.	Primer protitpo de mesa	17
2.	Primer prototipo de mesa y cámara	18
3.	Armado de base de cámara para el segundo prototipo	18
4.	Segundo prototipo de mesa de pruebas	19
5.	Colocación para el segundo armado de la mesa de pruebas	19
6.	Captura del primer archivo de texto para el ejemplo Multi-hilos	22
7.	Captura del segundo archivo de texto para el ejemplo Multi-hilos	22
8.	Captura del texto reconstruido para el ejemplo Multi-hilos	23
9.	Captura de la cámara web con OpenCV	24
10.	Intento 1 de calibración utilizando C++	25
11.	Intento 2 de calibración utilizando C++	26
12.	Colocación para el segundo armado de la mesa de pruebas	26
13.	Colocación para el segundo armado de la mesa de pruebas	26
14.	Colocación para el segundo armado de la mesa de pruebas	26
15.	Prueba calibración de la cámara con Python	28
16.	Segunda prueba calibración de la cámara con Python	28
17.	Calibración de la cámara para detección de los códigos utlizando Python . . .	30
18.	Prueba de generación de código utilizando Python	30
19.	Detección y reescalamiento de código generado para tamaño de 1x1 cm . . .	30
20.	Detección y reescalamiento de código generado para tamaño de 2x2 cm . . .	31
21.	Detección y reescalamiento de código generado para tamaño de 4x4 cm . . .	31
22.	Detección y reescalamiento de código generado para tamaño de 8x8 cm . . .	31

Lista de cuadros

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]

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]

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. [6]

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. [6]

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 [6]

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). [7]

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*. [8]

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. [9] [8]

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.).

Prototipo de Mesa de Pruebas

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 1 y 2. 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 4, además de tener un mejor base para la cámara como se muestra en la figura 3



Figura 1: Primer protitpo de mesa



Figura 2: Primer prototipo de mesa y cámara



Figura 3: Armado de base de cámara para el segundo prototipo



Figura 4: Segundo prototipo de mesa de pruebas



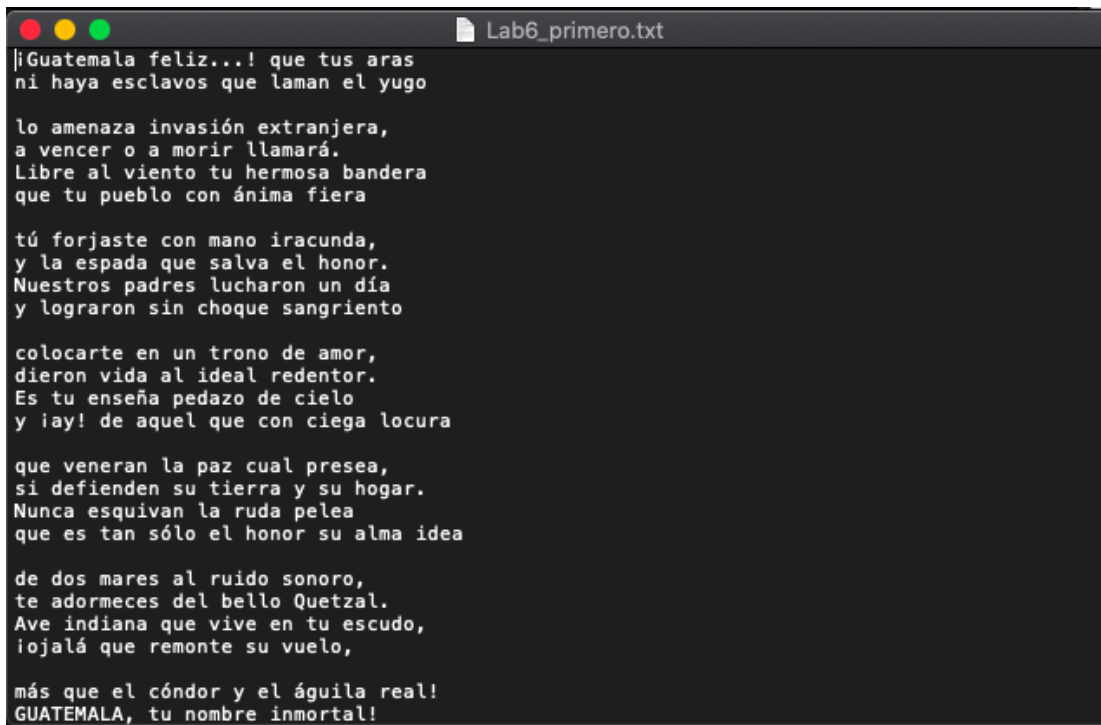
Figura 5: Colocación para el segundo armado de la mesa de pruebas

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 6 se observa un primer archivo de texto del himno nacional de Guatemala pero con las líneas impares. Luego, en la figura 7, 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 8.



```
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

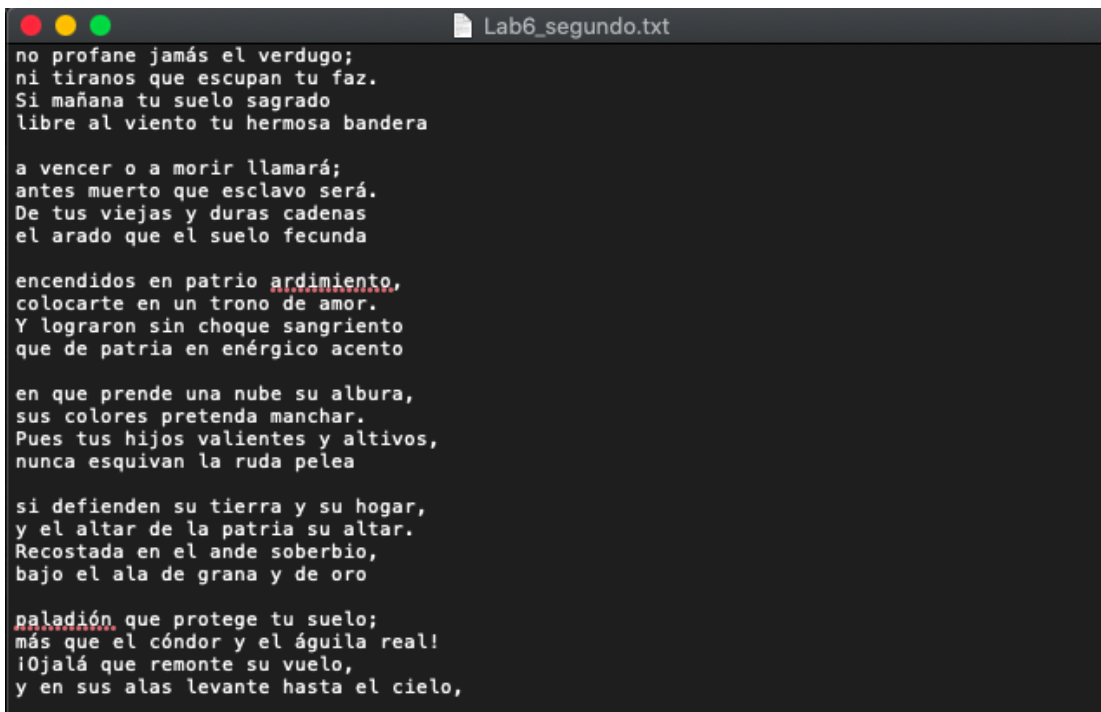
colocarte en un trono de amor,
dieron vida al ideal redentor.
Es tu enseña pedazo de cielo
y ¡ay! de aquel que con ciega locura

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 6: 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

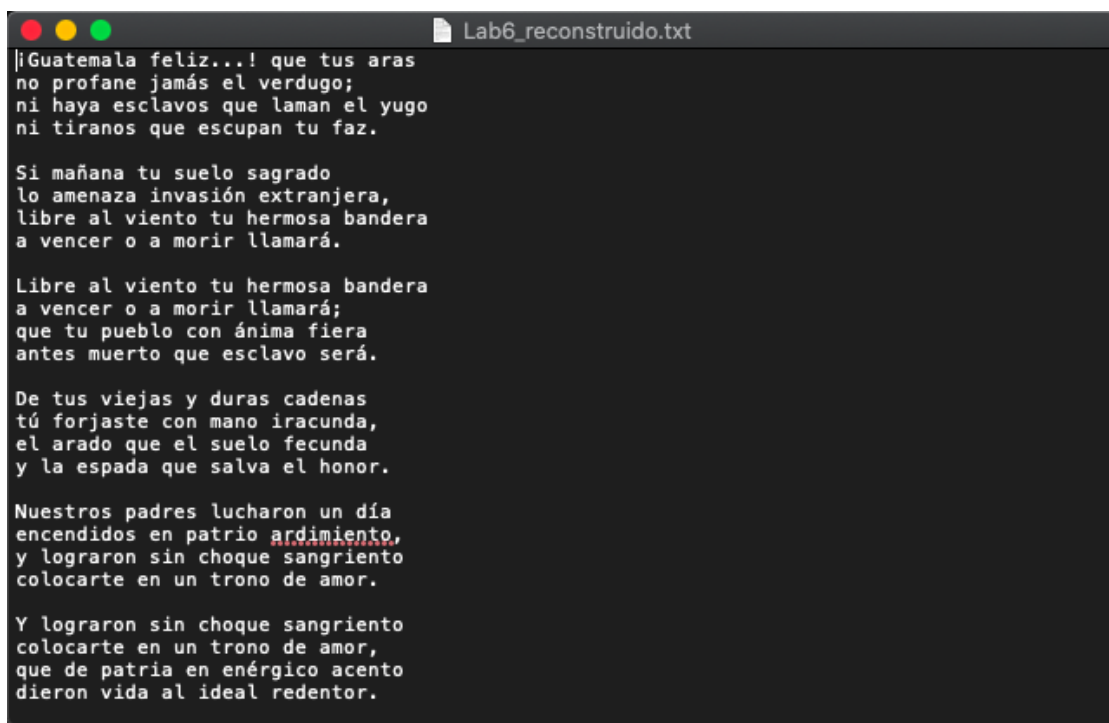
encendidos en patrio ardimiento,
colocarte en un trono de amor.
Y lograron sin choque sangriento
que de patria en enérgico acento

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

A screenshot of a terminal window with a dark background. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left and a file icon followed by the text 'Lab6_reconstruido.txt' on the right. The terminal displays several lines of text in a monospaced font, organized into paragraphs. The text is a Spanish poem or song about Guatemala. The word 'ardimiento' in the fifth paragraph is highlighted in red. The text is as follows:

|¡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 fiera
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 8: 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.



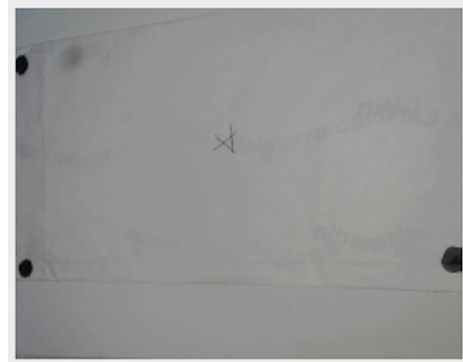
Figura 9: Captura de la cámara web con OpenCV

Pruebas Algoritmo y OpenCV en C++

Como se observa en las figuras siguientes, se realizaron pruebas del algoritmo de calibración en C++. La figura 10 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. Los píxeles como tal pueden variar de manera diversa según sea la resolución de la cámara, por lo que se procedió a eliminar dicha condición y se obtuvo una calibración adecuada, como se observa en la figura 11

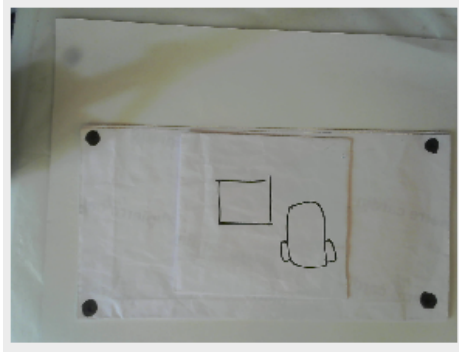


(a) 1a

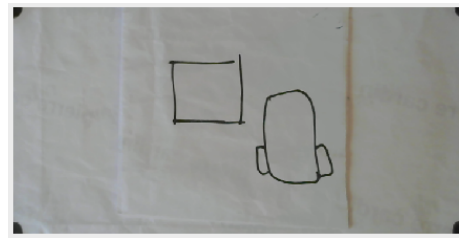


(b) 1b

Figura 10: Intento 1 de calibración utilizando C++



(a) 1a



(b) 1b

Figura 11: Intento 2 de calibración utilizando C++

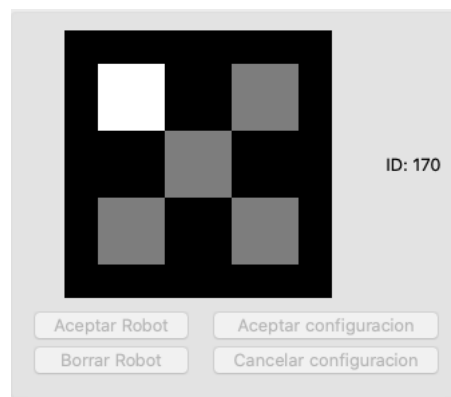


Figura 12: Colocación para el segundo armado de la mesa de pruebas

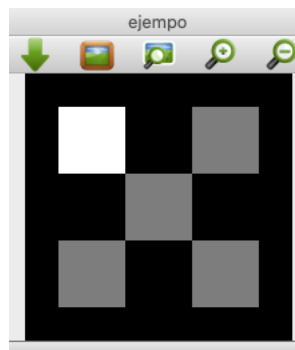


Figura 13: Colocación para el segundo armado de la mesa de pruebas

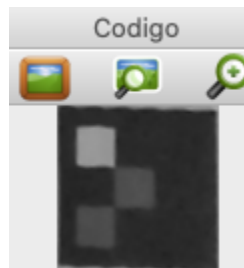


Figura 14: Colocación para el segundo armado de la mesa de pruebas

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 15) 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 16 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.

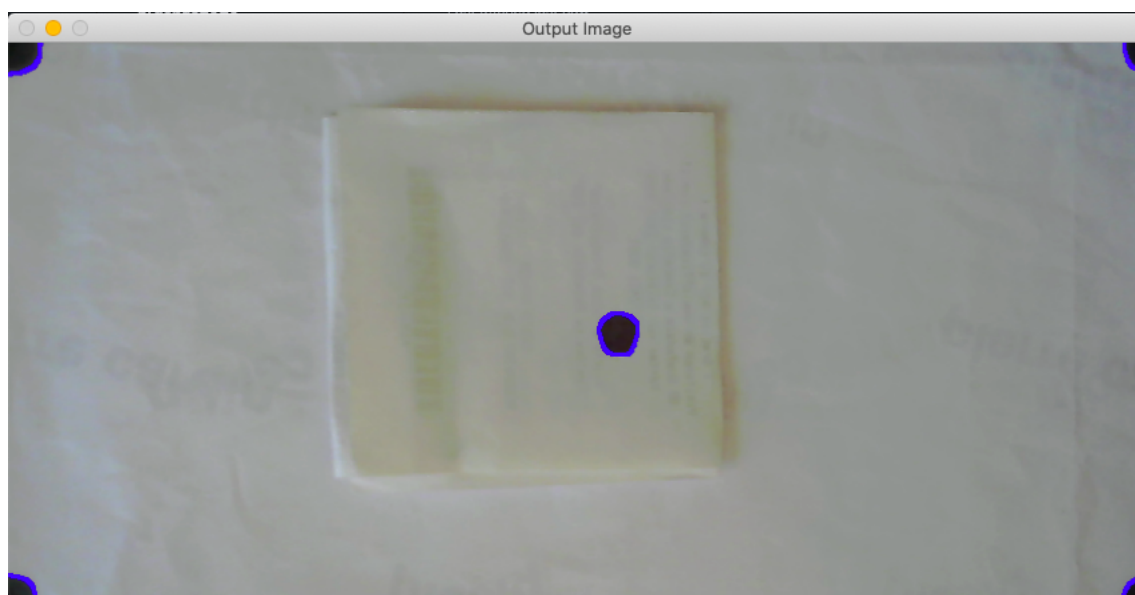


Figura 15: Prueba calibración de la cámara con Python

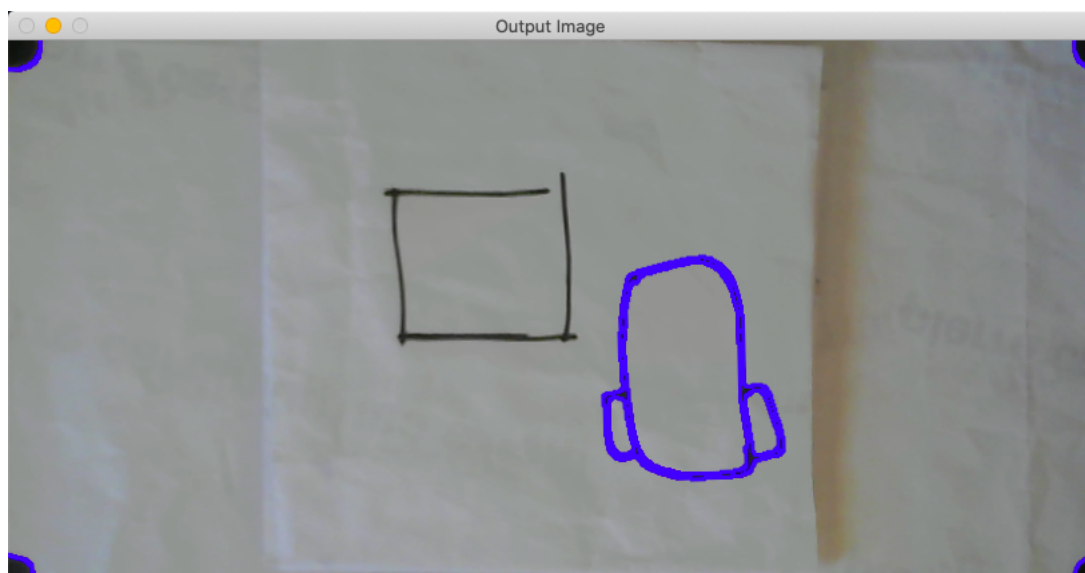


Figura 16: Segunda prueba calibración de la cámara con Python

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 18, 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 19, 20, 21 y 22 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 imagenes 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

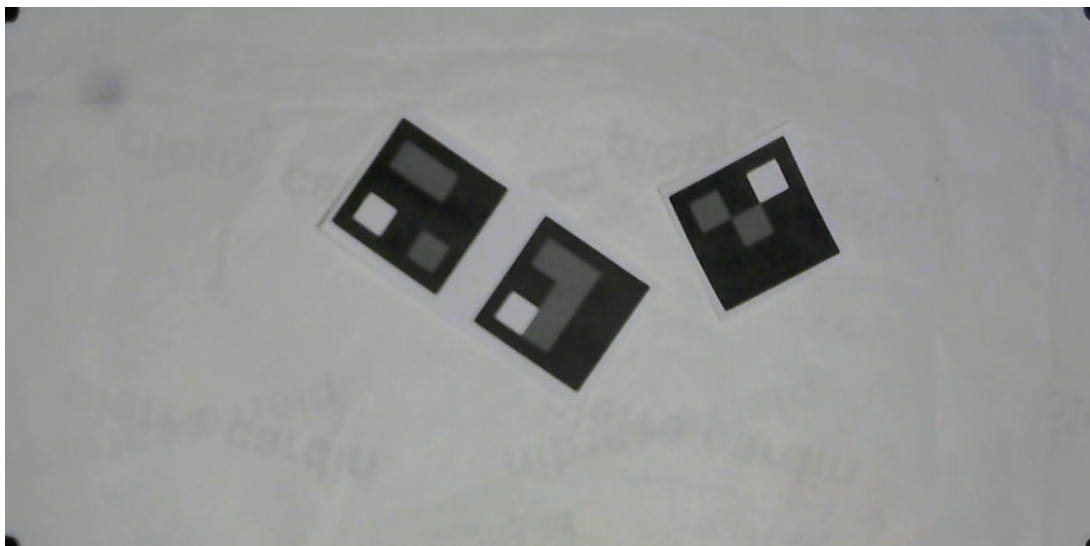


Figura 17: Calibración de la cámara para detección de los códigos utilizando Python

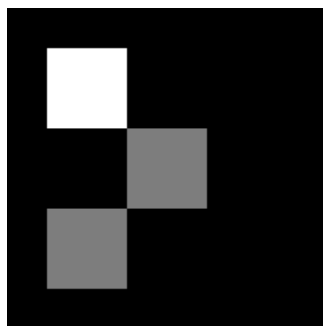
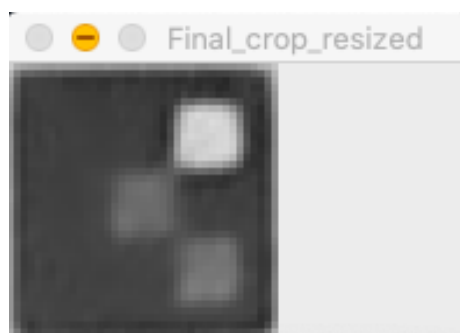


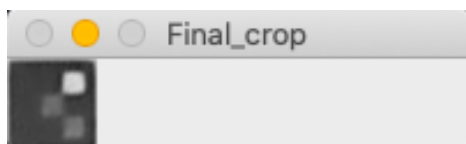
Figura 18: Prueba de generación de código utilizando Python



(a) Código rotado para la detección

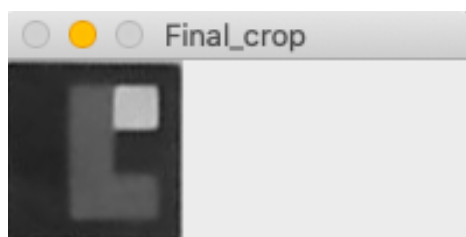


(b) Imagen aplicándole un reescalamiento



(c) Imagen original

Figura 19: 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 20: Detección y reescalamiento de código generado para tamaño de 2x2 cm

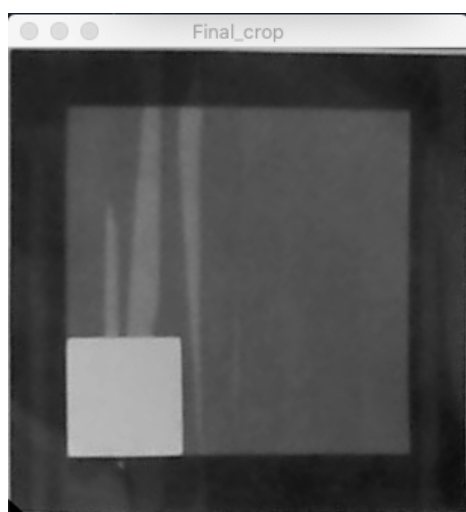


(a) Imagen rotada y reescalada para la detección del código



(b) Imagen original

Figura 21: 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 22: Detección y reescalamiento de código generado para tamaño de 8x8 cm

CAPÍTULO 13

Conclusiones

CAPÍTULO 14

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] R. H. Carver y K.-C. Tai, *Modern multithreading: implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs*. John Wiley & Sons, 2005.
- [7] P. Chawla, *OOP with C++*, <http://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf>, visitado el 05/06/2020.
- [8] Oracle, *Lesson: Object-Oriented Programming Concepts*, <https://docs.oracle.com/javase/tutorial/java/concepts/index.html>, visitado el 05/06/2020.
- [9] 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, visitado el 05/06/2020.

CAPÍTULO 16

Anexos

16.1. Planos de construcción

