

Algoritmos de Visión por Computadora para el Reconocimiento de la Pose de Agentes Empleando Programación Orientada a Objetos y Multi-hilos

José Pablo Guerra Jordán

Ingeniería Electrónica

Índice

Resumen

Objetivos

Introducción

Diseño Experimental

Resultados

Conclusiones

Recomendaciones





Resumen

Se desarrolló una herramienta de software, cuyo objetivo principal es poder reconocer la pose de agentes (denominados así normalmente en el área de robótica de enjambre).

Se tomó el algoritmo desarrollado en la fase previa para encontrar puntos de mejora y eficientizarlo mediante el uso de programación orientada a objetos y multi-hilos. El lenguaje original es C++ y se realizó la migración hacia Python.

La herramienta es capaz de generar marcadores visuales para la identificación de los robots en la mesa, así como calibrar la cámara y la obtención de pose de los robots.



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.



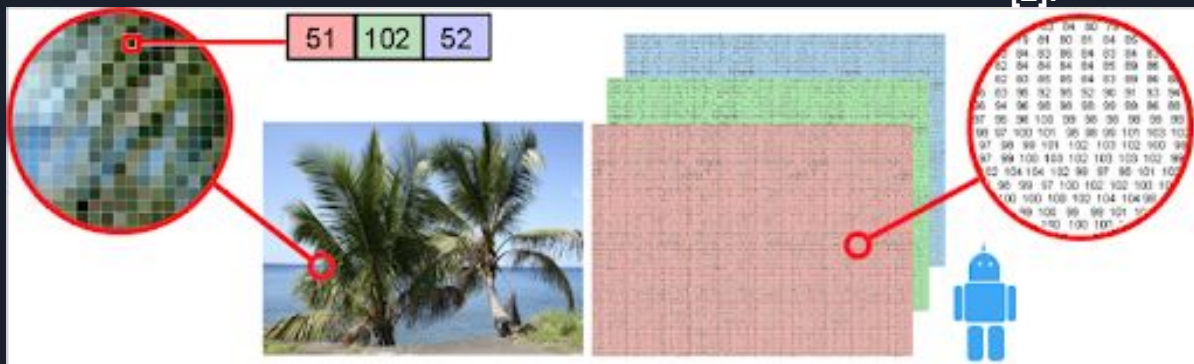
Objetivos Específicos

- 01 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.
- 02 Desarrollar algoritmos computacionalmente eficientes por medio de programación multi-hilos.
- 03 Diseñar e implementar una herramienta de software para aplicaciones de robótica de enjambre, usando los algoritmos desarrollados.
- 04 Validar la herramienta de software en la mesa de pruebas Robotat.

Introducción

Visión por Computadora

Consiste en obtener la información relevante, realizando un procesamiento a imágenes y vídeos, para que las computadoras puedan entender de mejor manera que es lo que hay en ellos [2].

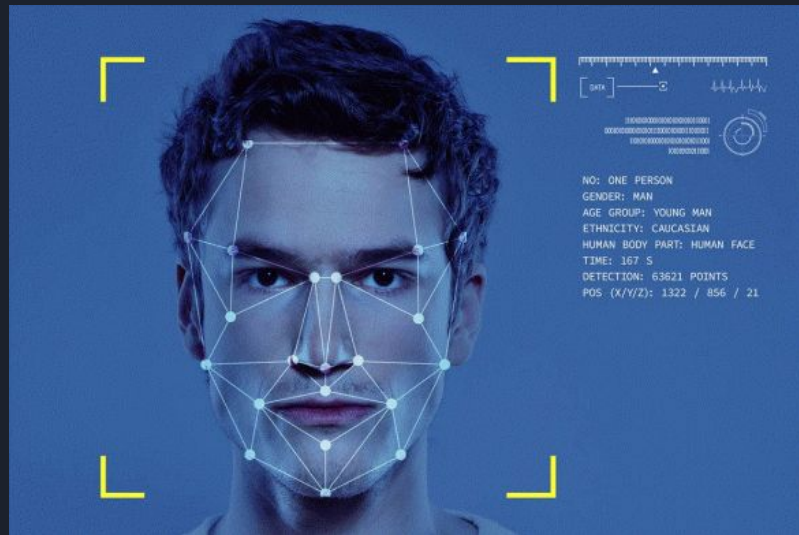


OpenCV

Esta es la librería de Open Source Computer Vision Library, es una librería para visión por computadora y machine learning.

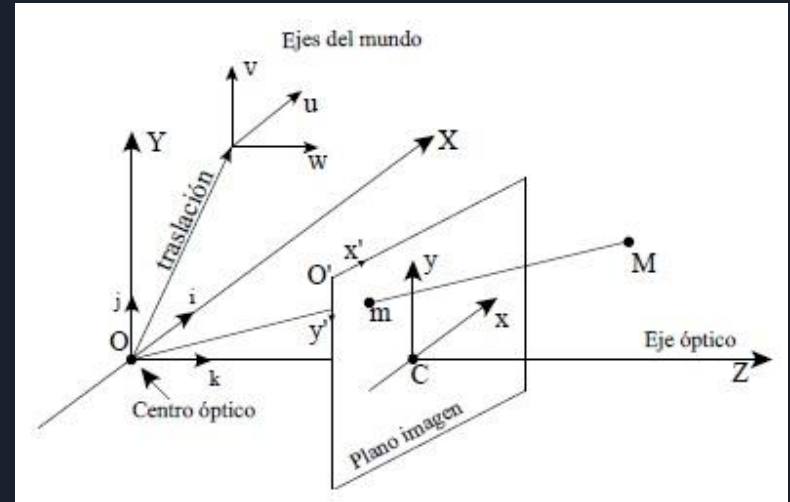
Incluye más de 2500 algoritmos utilizados para [3]:

- Reconocimiento facial
- Detección de objetos
- Detección de movimiento



Calibración de Cámaras

- Consiste en obtener información de los parámetros de la cámara para realizar un mapeo entre la imagen y el mundo físico [5].
- Existen técnicas de calibración como la calibración fotogramétrica [5].

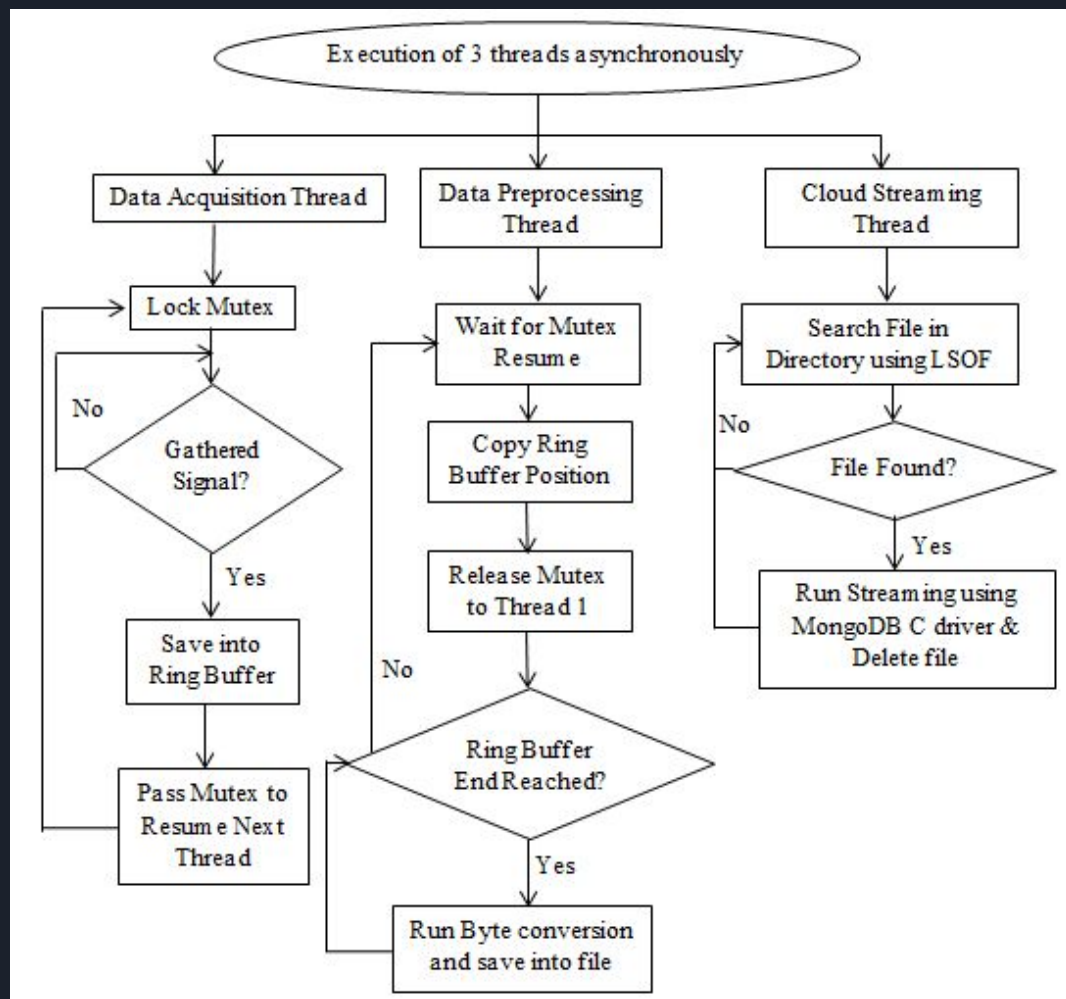


[6]



Programación Multi-hilos

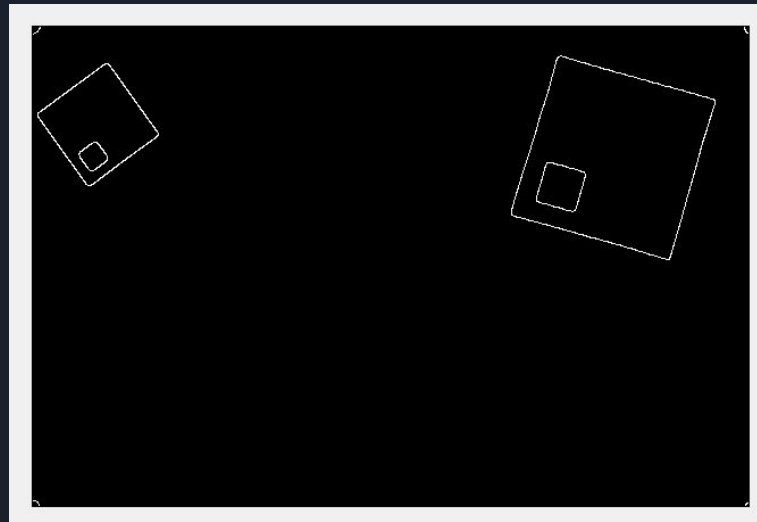
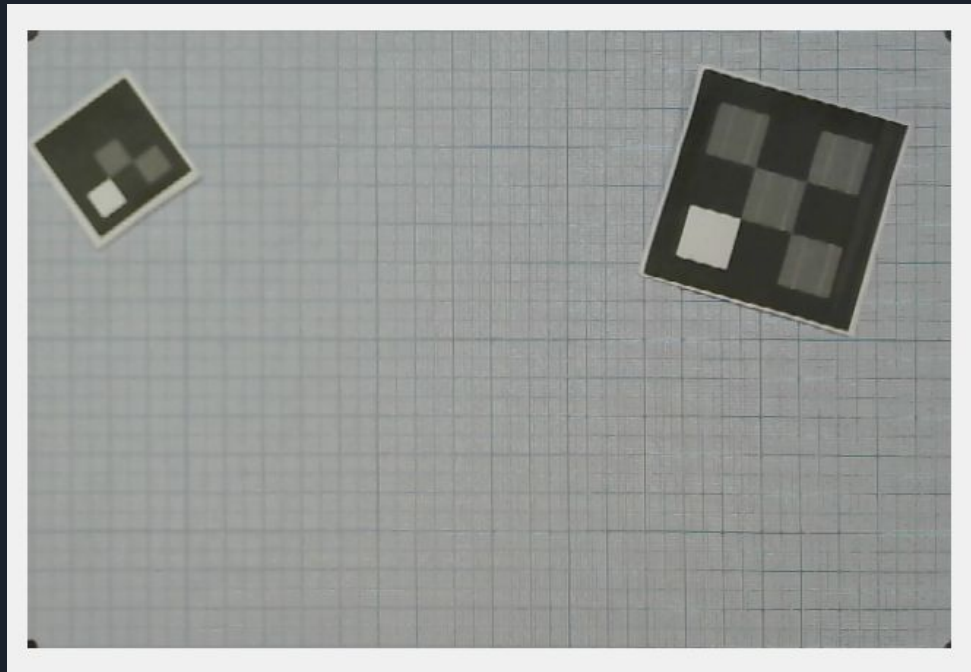
- Básicamente es orientar los programas para realizar varias tareas y en muchos casos realizar procesos más eficientes [7].
- El programa consta de un hilo principal encargado de las tareas principales y la creación de otros hilos [7].



[8]

Deteccción de bordes de Canny

Algoritmo de Canny es un operador desarrollado por John F. Canny en 1986 que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes [9].





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.

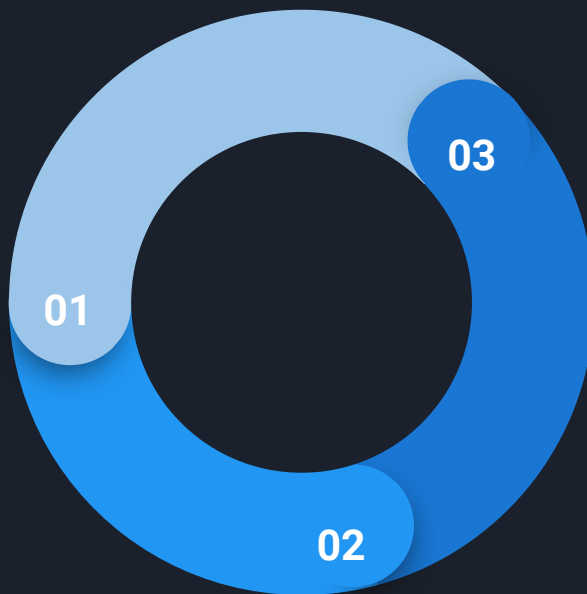
Diseño Experimental

Investigación

Se buscó información referente a los lenguajes de programación, OpenCV y su implementación, así como información sobre los multi-hilos en Python y C++

Implementación

Luego de haber realizado y comprendido el funcionamiento, se procedió a analizar los programas existentes (implementados en el lenguaje C++) para encontrar puntos de mejora y poder aplicar lo investigado e iniciar con la migración hacia Python.



Validación

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

Hardware

Laptop:

MacBook Air (13 pulgadas, principios de 2015) – Especificaciones técnicas



Almacenamiento¹

- 128 GB
128 GB de almacenamiento flash basado en PCIe
- 256 GB
256 GB de almacenamiento flash basado en PCIe
Configurable a 512 GB de almacenamiento flash

Procesador

- Intel Core i5 dual core de 1.6 GHz (Turbo Boost de hasta 2.7 GHz) con 3 MB de caché L3 compartida
Configurable a procesador Intel Core i7 dual core de 2.2 GHz (Turbo Boost de hasta 3.2 GHz) con 4 MB de caché L3 compartida

Memoria

- 4 GB u 8 GB de memoria integrada LPDDR3 de 1600 MHz (8 GB máximo).

Cámara:

Logitech



HD Pro Webcam C920



Software

Sistema operativo



Librería y Lenguajes de programación

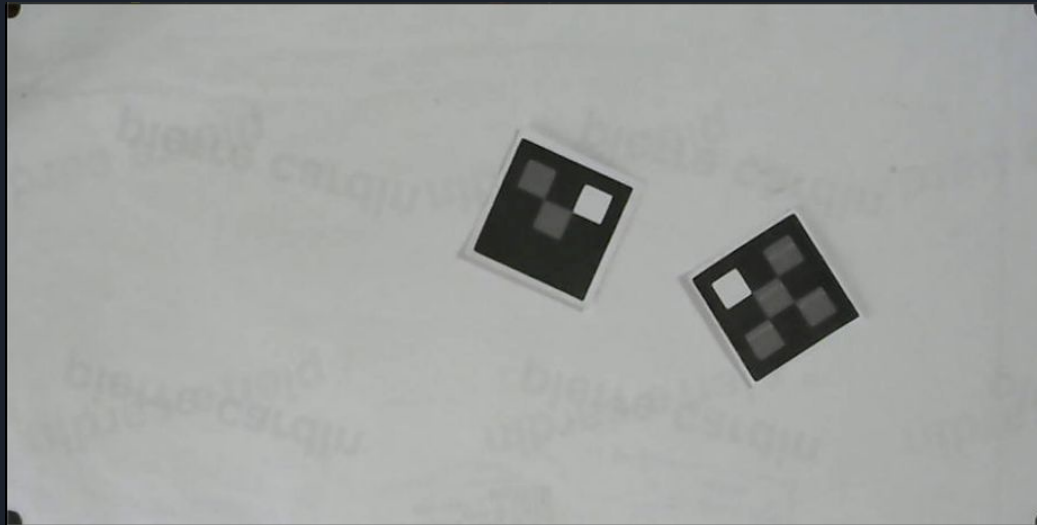





Resultados

Acerca de la herramienta

- Como se observa en la figura, el principal reto es poder identificar los robots que están colocados en la mesa.
- De aquí nace la necesidad de una herramienta que pueda ubicar su posición y rotación en la mesa.
- Además, es necesario un ID para saber a qué robot se hace referencia. Aquí la importancia de los marcadores visuales que le permitan a la herramienta obtener esta información.





Proceso general del funcionamiento de la herramienta.

Calibración de la cámara
(Se realiza solo la primera vez)



Detección de los agentes
mediante el uso de
Canny



Ubicar cada agente e
identificar su ID y su
respectiva posición y ángulo
de rotación. Repetir N veces,
donde N = número de
contornos detectados.



Migración del programa y uso de POO

Versión original en C++ (ejemplo)

```
void MainWindow::on_boton_ObtenerFirst_pressed()
{
    |
    cam.set(CV_CAP_PROP_FRAME_WIDTH, 960); //1280,640
    cam.set(CV_CAP_PROP_FRAME_HEIGHT, 720); //720,480
    cam.set(CAP_PROP_AUTOFOCUS, 0); // turn the autofocus off
    while (!cam.isOpened()) {
        std::cout << "Failed to make connection to cam" << std::endl;
        cam.open(0);
    }
    Snap = takePicture();
    CropSnap = getCroppedSnapshot(Snap);
    cout << "foto" << endl;
    //imshow("Cropped", CropSnap);
    cout << "A obtener el codigo" << endl;
    MyRobots = getRobotCodes(CropSnap, MyGlobalCannyInf, MyGlobalCannySup, MyMedidaCodigoCM);
    MyInitialRobots = MyRobots.Vrobots.size();
    cout << MyInitialRobots << endl;
```



Versión con POO C++ (1)

Clase Camara y sus respectivos métodos

```
//METODOS CLASE CAMARA
Mat Camara::Calibrar(Mat Snapshot,int calib_param)
{
    Mat CaliSnapshot;
    Point* Esqui = get_esquinas( Snapshot, calib_param, 0);
    lambda = getHomogenea(Esqui);
    MyWiHe = getWiHe(Esqui);
    warpPerspective(Snapshot, CaliSnapshot, lambda, { MyWiHe[0], MyWiHe[1] });
    return CaliSnapshot;
}

void Camara::Set_camara(int WIDTH, int HEIGHT){
    cam_libreria.set(CV_CAP_PROP_FRAME_WIDTH, WIDTH); //1280,640
    cam_libreria.set(CV_CAP_PROP_FRAME_HEIGHT, HEIGHT); //720,480
    cam_libreria.set(CAP_PROP_AUTOFOCUS, 0); // turn the autofocus off
    while (!cam_libreria.isOpened()) {
        std::cout << "Failed to make connection to cam" << std::endl;
        cam_libreria.open(0);
    }
}
```



Versión con POO C++ (2)

Se muestra como se utilizan los métodos de la clase cámara. El código se reduce en cuanto a líneas y lo hace más modular.

```
void MainWindow::on_boton_ObtenerFirst_pressed()  
{  
    camara.Set_camara(960, 720);
```

```
    Snap = camara.Take_picture();  
    //cout << "Tome la foto" << endl;  
    CropSnap = getCroppedSnapshot(Snap); //esto solo se hace la primera vez
```

Versión en Python usando POO (1)

```
class camara():
    """
    Definición de la clase de camara, incluye los siguientes metodos:
    set_camara(): para setear configuracion inicial de la camara
    tomar_foto(): para capturar el frame.
    Calibrar(): Calibracion de la camara.
    """

    def __init__(self, cam_num = 0, WIDTH = 960, HEIGHT = 720):
        """
        Parameters
        -----
        cam_num : TYPE, optional
            DESCRIPTION. The default is 0. Setea el valor de la camara en 0, normalmente una camara
            adicional (no la que la computadora trae) por default es 0
        WIDTH : TYPE, optional
            DESCRIPTION. The default is 960. El ancho del marco de captura
        HEIGHT : TYPE, optional
            DESCRIPTION. The default is 720. El alto del marco de captura


        Returns
        -----
        None.

        """
        self.cap = cv.VideoCapture(cam_num)
        self.cap.set(cv.CAP_PROP_FRAME_WIDTH, WIDTH)
        self.cap.set(cv.CAP_PROP_FRAME_HEIGHT, HEIGHT)
        self.cam_num = cam_num
        self.img_counter_code = 0
```

Versión en Python usando POO (2)

Nuevamente, se muestra como se utilizan los métodos de la clase cámara. El código se reduce en cuanto a líneas y lo hace más modular.

```
def capture(self):  
    foto = camara.get_frame()  
    CaliSnapshot = camara.Calibrar(foto, Calib_param, Treshold)  
    #self.label_img.show()  
    #self.image = QImage(self.image.data, self.image.shape[1], s  
    #self.image_frame.setPixmap(QPixmap.fromImage(self.image))  
  
    #cv.imshow("Output Image", CaliSnapshot)  
    #cv.waitKey(2000)  
    #camara.destroy_window()  
    self.bToma_pose.setEnabled(True)  
    self.size_codigo.setEnabled(True)  
    self.bcapturar.setEnabled(False)
```

- 
- Ambas implementaciones (C++ y Python) cuentan con su respectiva versión utilizando programación orientada a objetos.
 - En ambos casos la POO aporta modularidad para el código.
 - Ayuda a la organización de las diversas funciones del programa y permite su reutilización de manera sencilla para el usuario.

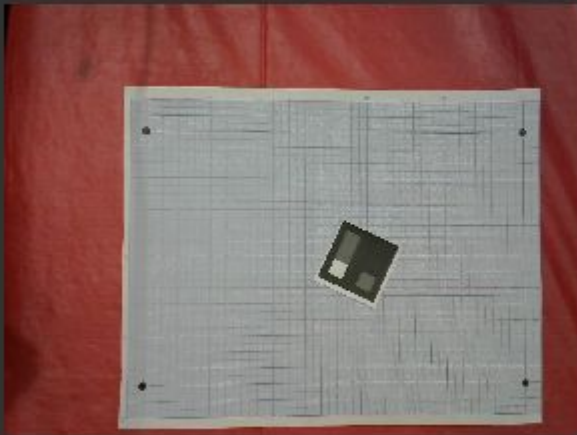


Objetivos Específicos

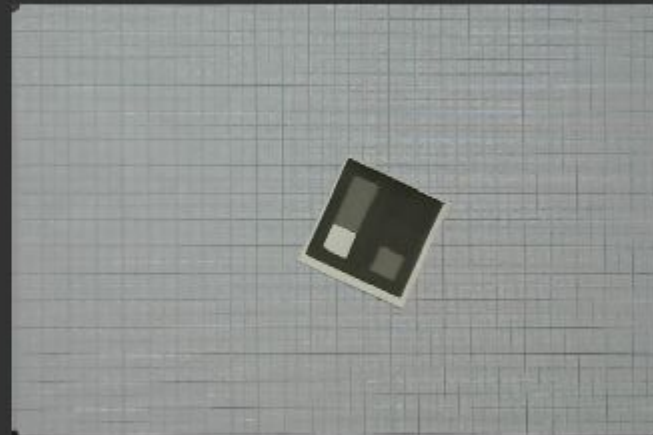
- 01 Seleccionar el lenguaje de programación orientado a objetos más adecuado para la implementación de algoritmos de visión por computador para la mesa de pruebas Robotat.
- 02 Desarrollar algoritmos computacionalmente eficientes por medio de programación multi-hilos.
- 03 Diseñar e implementar una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

Calibración de cámara en C++

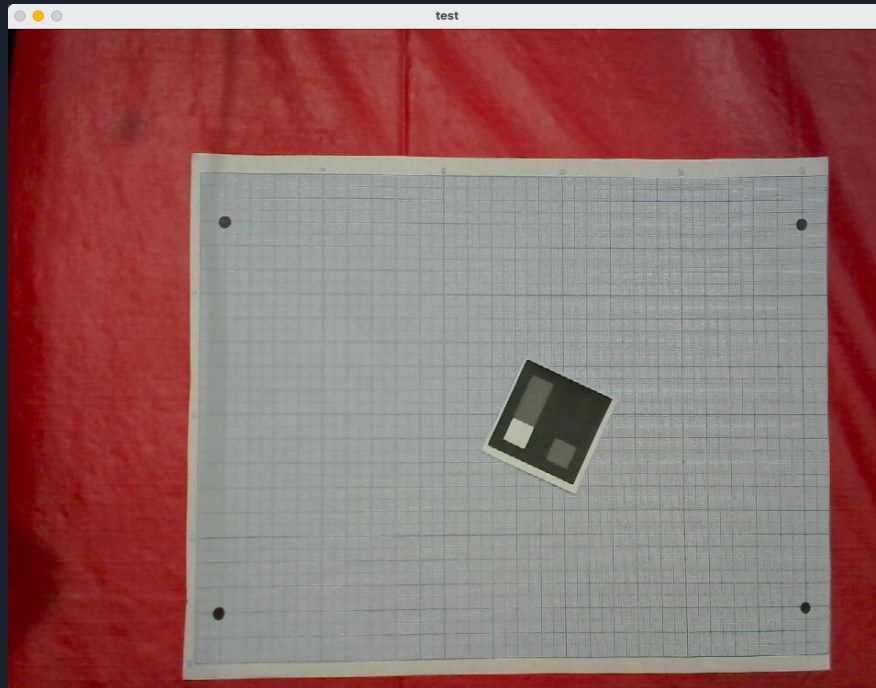
Tomar Foto



Calibrar



Calibración de cámara en Python



Sistema Swarm - Mesa Robotat

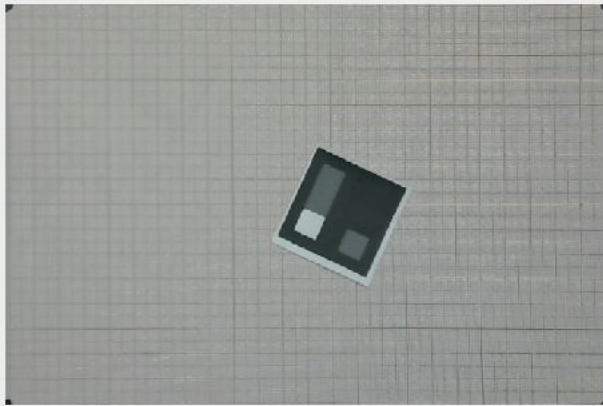
Calibracion* Generacion Identificador

Calibrar Reiniciar calibracion GenerarCodigo Ingrese número

Obtencion de Pose**

Tomar Pose Tamaño del código

Detener Procesamiento

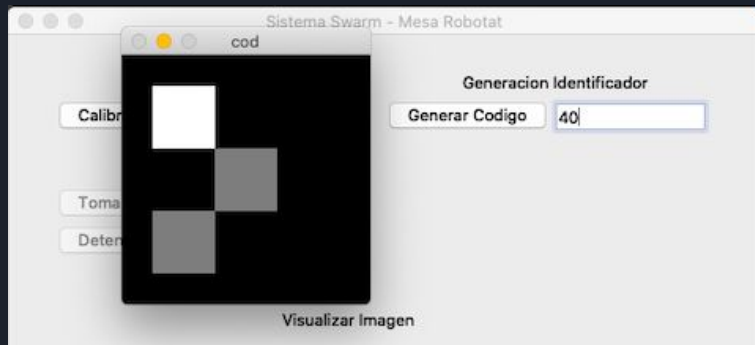
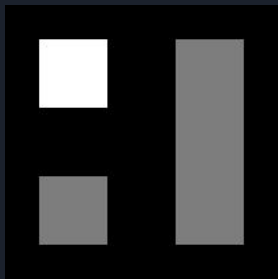


NOTA 1(*): Calibrar hasta ver las 4 esquinas del tablero (Presionar Reiniciar Calibracion)

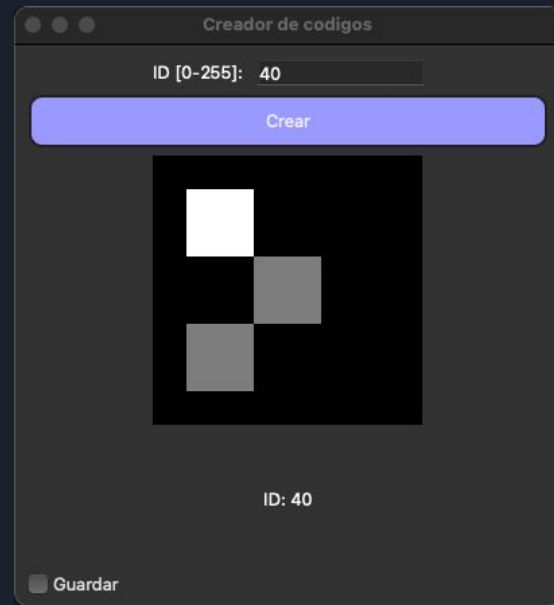
NOTA 2(**): Mejores resultados se obtienen con iluminacion directa sobre la mesa

Creación e identificación de marcadores

Generación de Marcadores



Generación de Marcador en Python



Generación de Marcador en C++

Identificación de Marcadores en Python (1)

Identificación Marcadores 1x1 cm en Python



Identificación Marcadores 5x5 cm en Python

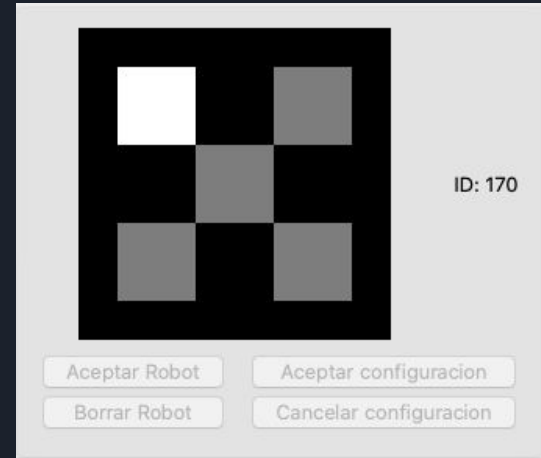
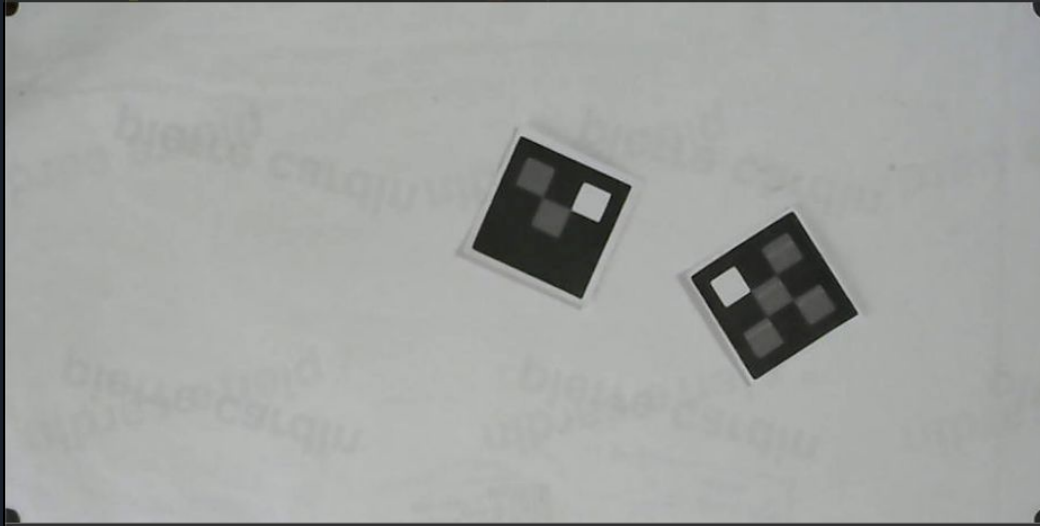


Identificación de Marcadores en Python (2)



```
opencv_CalibSnapshot_0.png Canny Guardado!  
Soy el hilo: 1  
Entre a process image  
ID del robot 40
```

Identificación en C++



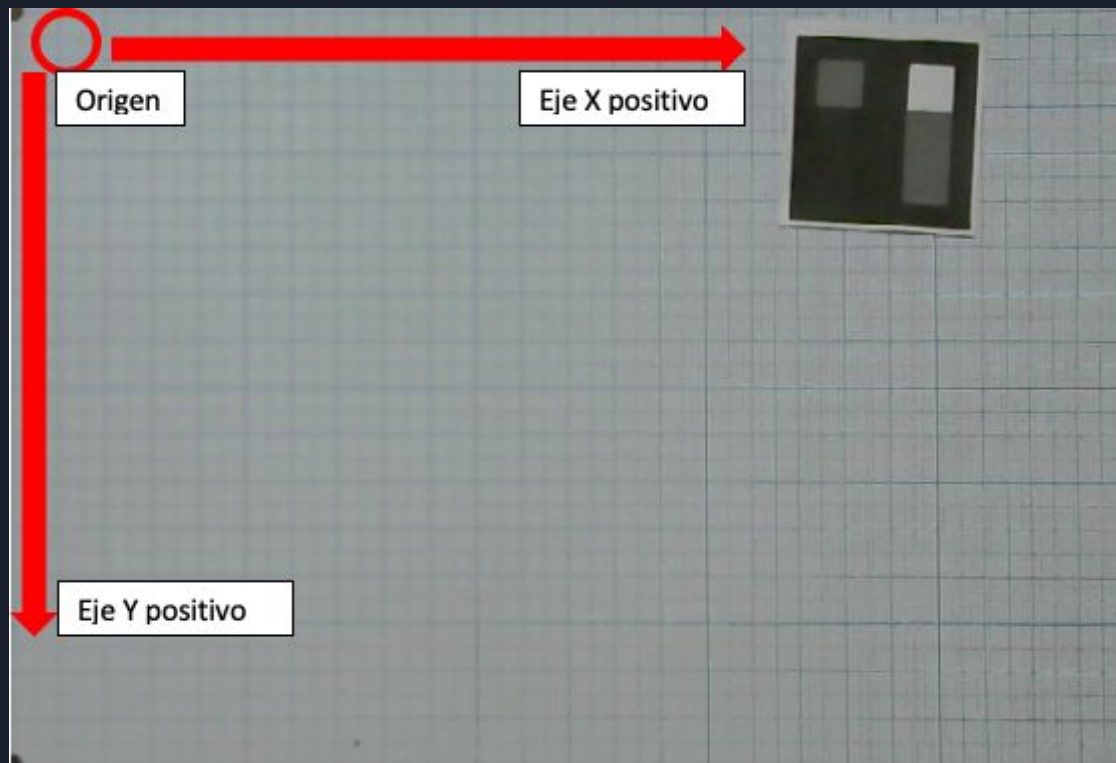


Objetivos Específicos

- 01 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.
- 02 Desarrollar algoritmos computacionalmente eficientes por medio de programación multi-hilos.
- 03 Diseñar e implementar una herramienta de software para aplicaciones de robótica de enjambre, usando los algoritmos desarrollados.
- 04 Validar la herramienta de software en la mesa de pruebas Robotat.

Detección de la pose de los robots

Ejes de referencia para la toma de posición





Toma de pose y ubicación en la mesa de pruebas(1)

Parámetro	Media	Moda	Desviación Estándar
Posición (cm)	0	0	0
Ángulo (grados)	0.467	0	0.6399

Cuadro 1: Media y moda para la posición con aproximación a enteros entre los algoritmos de Python y C++

Parámetro	Media	Moda	Desviación Estándar
Posición (cm)	0.1157	0.085	0.0526
Ángulo (grados)	0.2	0	0.5606

Cuadro 2: Media y moda para la posición sin aproximación a enteros entre los algoritmos de Python y C++



Toma de pose y ubicación en la mesa de pruebas(2)

Parámetro	Media	Moda	Desviación Estándar
Posición (cm)	0.3758	0.5	0.2314

Cuadro 3: Media y moda para la diferencia con respecto a la posición real del algoritmo en Python (ambos ejes)

Parámetro	Media	Moda	Desviación Estándar
Posición (cm)	0.2736	NA	0.1968

Cuadro 4: Media y moda para la diferencia con respecto a la posición real del algoritmo en C++ (ambos ejes)



Objetivos Específicos

- 01 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.
- 02 Desarrollar algoritmos computacionalmente eficientes por medio de programación multi-hilos.
- 03 Diseñar e implementar una herramienta de software para aplicaciones de robótica de enjambre, usando los algoritmos desarrollados.
- 04 Validar la herramienta de software en la mesa de pruebas Robotat.



Comparación de tiempos de ejecución (1)

Lenguaje	Media	Moda	Desviación Estándar	T. Máximo (s)	Repeticiones
Python	0.2519 s	0.29 s	0.031	0.31	45
C++	0.1083 s	0.1073 s	0.00597	0.1402	45

Cuadro 5: Media de tiempos de ejecución para los algoritmos usando 4 hilos en Python y 2 hilos en C++

Lenguaje	Media	Moda	Desviación Estándar	T. Máximo (s)	Repeticiones
Python	0.0636 s	0.0047 s	0.02659	0.119	45

Cuadro 6: Media de tiempos de ejecución para los algoritmos en Python usando 2 hilos



Comparación de tiempos de ejecución (2)

Lenguaje	Media	Moda	Desviación Estándar	T. Máximo (s)	Repeticiones
Python	0.0551 s	0.085 s	0.02808	0.100	45
C++	0.1076 s	0.107 s	0.0063	0.1388	45

Cuadro 7: Media de tiempos de ejecución para los algoritmos en Python y C++ sin hilos



Objetivos Específicos

- 01 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.
- 02 Desarrollar algoritmos computacionalmente eficientes por medio de programación multi-hilos.
- 03 Diseñar e implementar una herramienta de software para aplicaciones de robótica de enjambre, usando los algoritmos desarrollados.
- 04 Validar la herramienta de software en la mesa de pruebas Robotat.



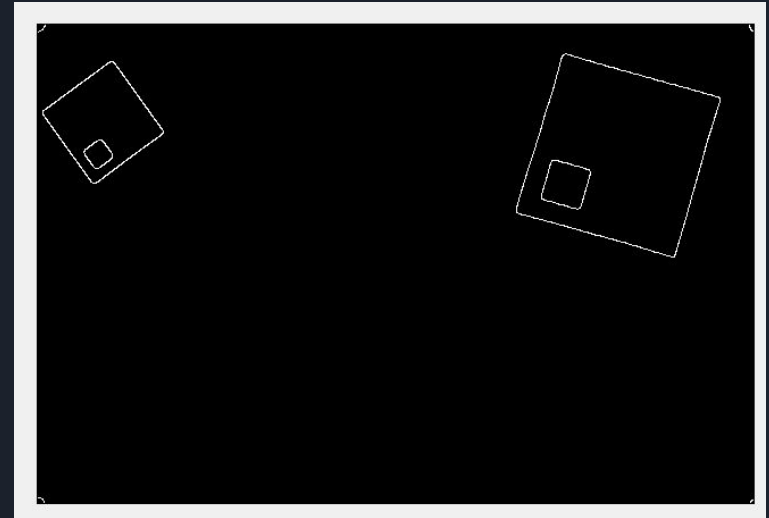
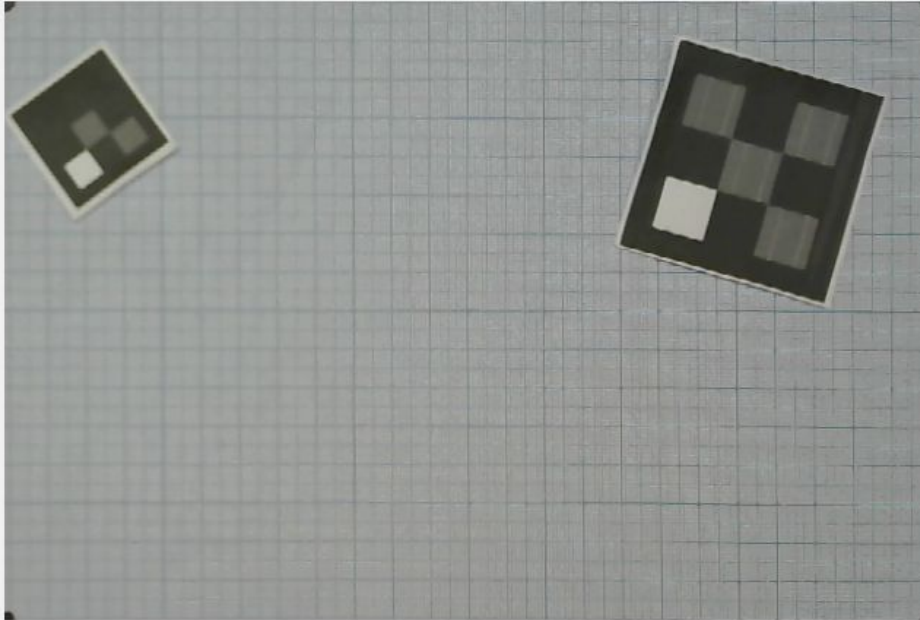
Resultados adicionales

El alcance de este trabajo consistía en crear una herramienta utilizando los lenguajes de C++ y Python aplicando lo explicado anteriormente. Sin embargo, se inició con una fase adicional, que consiste en realizar la migración hacia el lenguaje de Matlab. A continuación se presentan esos resultados.

Resultados adicionales

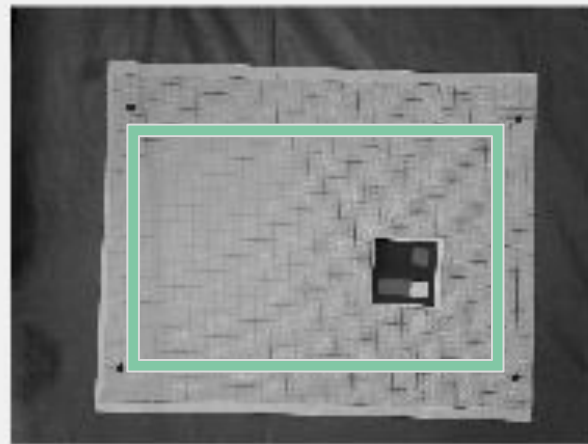
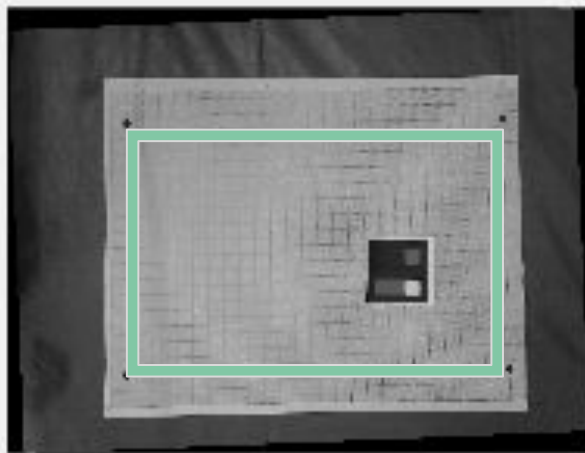
Resultados preliminares en Matlab

Detección de los contornos mediante el uso de Canny



Calibración de la cámara

Corrección de perspectiva



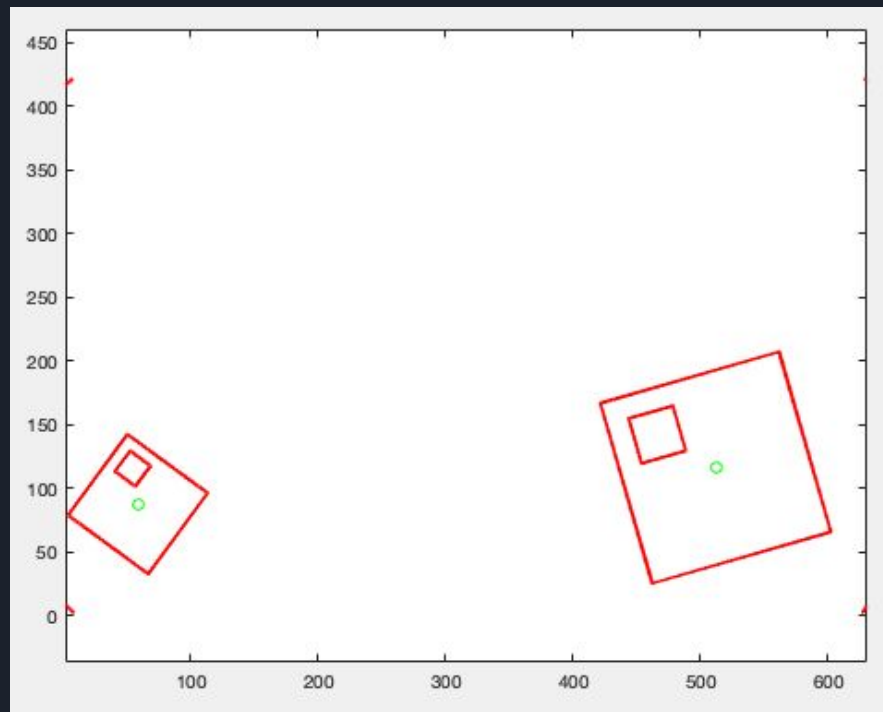
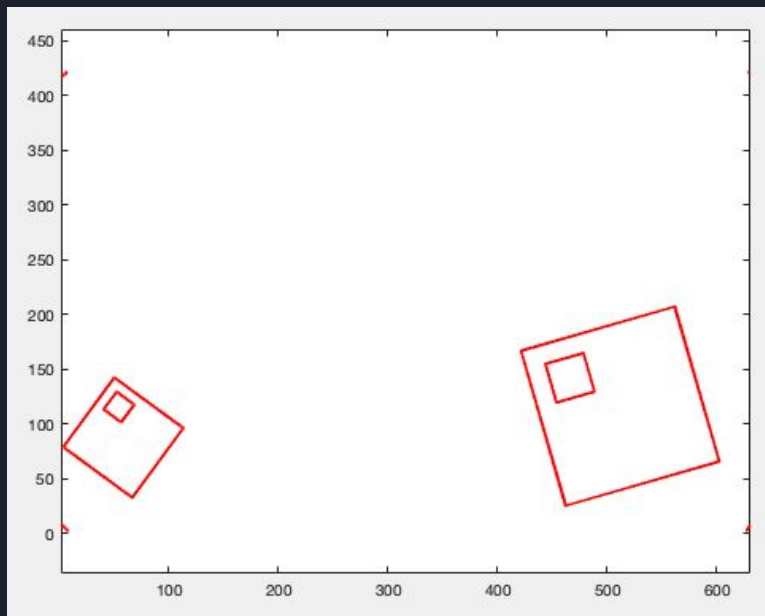


Código para la corrección de perspectiva y calibración.

```
initial_points=[0 0;size(I,1) 0;size(I,1) size(I,2);0 size(I,2)];  
  
tform = fitgeotrans(final_points,initial_points,'NonreflectiveSimilarity');  
  
out = imwarp(I,tform);  
  
figure(60);  
subplot(1,2,1)  
imshow(out);  
subplot(1,2,2)  
imshow(I);
```

DetECCIÓN Y UBICACIÓN DE LOS MARCADORES EN LA MESA

El algoritmo es capaz de detectar los contornos que Canny identifico, y con esto ubicar sus centros (círculos verdes). Estos centros serán usados como la posición de los robots en la mesa física.



Código para la detección y ubicación de los marcadores en la mesa(1)

```
%% Para la detección de bordes y ubicación de los marcadores en la mesa
clear;
clf;
close all; % Close all figures (except those of imtool.)
anchoMesa = 16.0;
largoMesa = 24.0;

I = imread('Calibsnapshot.png');
[rows, columns, numberOfColorChannels] = size(I);
GlobalHeight = columns;
GlobalWidth = rows;

figure(3);
imshow(I);
I=rgb2gray(I);

BW1 = edge(I, 'Canny', 0.7);
B = bwboundaries(BW1, 'noholes');

max_size = 110;
center_count = 1;
```

Código para la detección y ubicación de los marcadores en la mesa(2)

```
for i=1:length(B)
    C = cell2mat(B(i));
    Cont = C';
    boundingBox = minBoundingBox(Cont);
    C2 = boundingBox(:,2);
    C4 = boundingBox(:,4);
    C3 = boundingBox(:,4);
    sizeX = abs(C2(1) - C4(1));
    sizeY = abs(C2(2) - C4(2));

    C1 = boundingBox(:,1);
    figure(42);
    plot(boundingBox(2,[1:end 1]),boundingBox(1,[1:end 1]),'r');
    hold on;
    axis equal
    if sizeX > max_size || sizeY > max_size

        Point2 = [(C2(1)+C4(1))/2,(C2(2)+C4(2))/2];

        Cx = Point2(1);
        Cy = Point2(2);
        X = abs(C1(1) - C2(1));
        Y = abs(C1(2) - C4(2));
```

```
dummy_center = [Point2(1),Point2(2)];

centers(center_count,:) = dummy_center;

plot(Point2(2),Point2(1),'go');
axis equal

I2 = imcrop(I,[C2(1)-X, C2(2)-Y, (Cx+X), (Cy+Y)]);
figure(105);
hold on;
imshow(I2);

pause(5);

center_count=center_count+1;

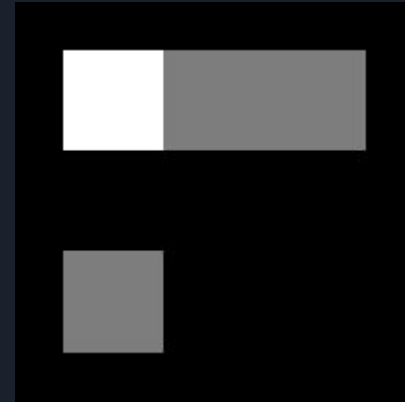
end

%pause(2);
end

tempFloatX = (anchoMesa/GlobalWidth) * centers(length(centers),2);
tempFloatY = (largoMesa/GlobalHeight) * centers(length(centers),1);
```

Generación de marcadores

```
for u = 0:2
    for v = 0:2
        %para generar el pivote (ver la tesis de Andre)
        %El pivote sirve para saber que cuadro debe estar alineado.
        if k == 0
            for i = u*control3+control1:u*control3+control2 %25:75
                for i2 = v*control3+control1:v*control3+control2
                    Cod(i,i2) = 255; %llena el pivote, 255 = blanco
                end
            end
        else
            %genera los otros cuadros en escala de grises, 125 = gris
            count = 9-k;
            if count == 0
                break
            end
            t = binStr(9-k);
            n = str2num(t);
            for i3 = u*control3+control1:u*control3+control2
                for i4 = v*control3+control1:v*control3+control2
                    Cod(i3,i4) = n * 125;
                end
            end
        end
    end
end
```





Conclusiones

01

Trabajar con una resolución de 920×760 permite identificar códigos de un tamaño mínimo de 3×3 cm hasta hasta 10×10 cm, de lo contrario, la detección podría ser errónea.

02

En promedio, la diferencia de losl algoritmo de Python y C++ con respecto a las medias reales es de 0.595 centímetros. Esto es una diferencia baja tomando en cuenta las dimensiones de la mesa

03

Si las medidas son trabajadas en números enteros, estadísticamente no hay diferencia entre la versión de C++ y la versión de Python. Sin embargo, si no se aproxima a números enteros, la diferencia entre los algoritmos es en promedio 1 milímetro. En ambos casos se denota exactitud y precisión entre los algoritmos.

04

El tiempo de ejecución en la implementación con mult-hilos, en el lenguaje Python, es mayor que su versión sin hilos. Esto puede deberse a la cantidad de hilos implementados y a otros factores como que estos se estén procesando en un solo core o el overhead que puede existir entre hilos



Recomendaciones

- 01 Es posible identificar correctamente los marcadores en condiciones adecuadas de luz, ya que esto ayuda a mejorar el contraste para el uso de Canny.
- 02 Para tener una correcta calibración en Python se requiere que las esquinas estén correctamente identificadas con formas lo más circulares posible.
- 03 Es posible utilizar verdadera paralelización en los multi-hilos. Esto se puede lograr utilizando varios cores de una computadora.
- 04 Es posible encontrar una mejor forma en cómo el algoritmo de Python identifica el ID. En este trabajo se plantea recortar la imagen en cada cuadro que conforma el marcador, pero puede existir una forma más eficiente
- 05 El uso de Matlab puede ser una opción para explorar como otra alternativa para la implementación de esta herramienta de software. Esto brindará mayor versatilidad a esta herramienta.



Referencias

- [1] J. C. Hernández, J. Rodríguez y M. F. Santiago, VISIÓN POR COMPUTADORA, <http://graficacionporcomputadora.blogspot.com/2013/05/52-vision-por-computadora.html>, visitado el 14/09/2020
- [2] R. L. Briega, Visión por computadora, <https://iaarbook.github.io/vision-por-computadora/>, visitado el 10/05/2020.
- [3] OpenCV, About, <https://opencv.org/about/>, visitado el 05/04/2020.
- [4] R. Díaz, “Tus fotos online tiene un propósito que no esperabas”, Viatec.do. <https://viatec.do/tus-fotos-online-sirven-un-proposito-que-quizas-no-esperabas/>, visitado el 11/12/2020
- [5] D. A. Pizarro, P. Campos y C. L. Tozzi, “COMPARACIÓN DE TÉCNICAS DE CALIBRACIÓN DE CÁMARAS DIGITALES”, Universidad Tarapacá, vol. 13, no. 1, págs. 58-67, 2005. doi: <http://dx.doi.org/10.4067/S0718-13372005000100007>.
- [6] F. Lanzaro y M. Fuentes, Calibración y Posicionamiento 3D, <https://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2014/candombe/calibracion.html>, visitado el 14/09/2020.



Referencias

- [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. Yong y E. T. W. Ho, “Streaming brain and physiological signal acquisition system for IoT neuroscience application”, págs. 752-757, dic. de 2016. doi:10.1109/IECBES.2016.7843551.
- [9] Wikipedia, “Algoritmo de Canny”, https://es.wikipedia.org/wiki/Algoritmo_de_Canny, visitado el 11/12/2020.



Gracias por su atención