ESP-CAM con AprilTag

ESP32CAM

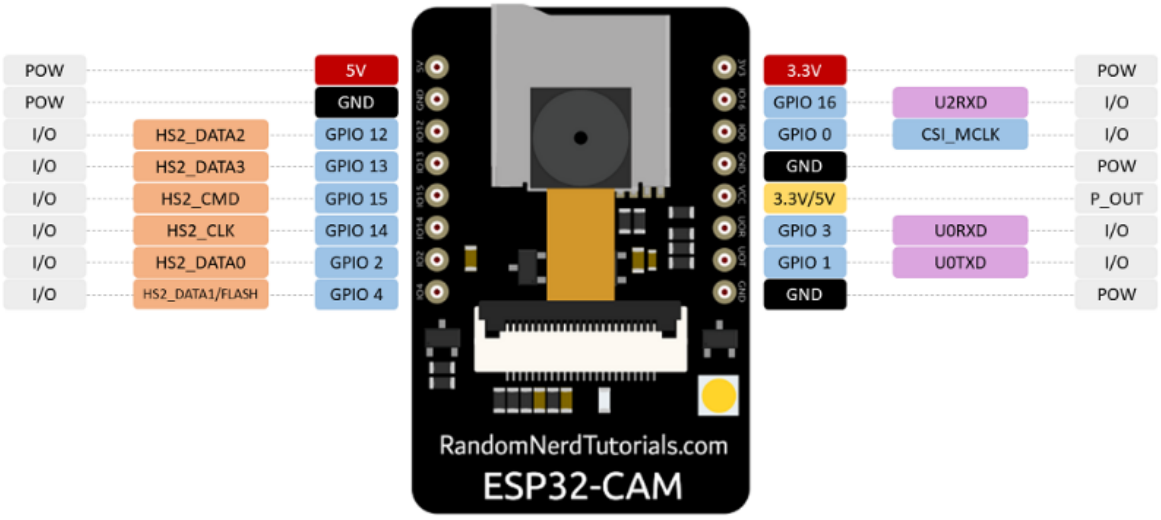

Fig. 1 Pin out ESP32-CAM

AprilTag

Las AprilTags son marcadores visuales bidimensionales diseñados para ser detectados y rastreados de manera eficiente por sistemas de visión artificial. Similares a los códigos QR, pero optimizados para aplicaciones de robótica y visión por computadora, las AprilTags permiten la identificación precisa de objetos y la estimación de su posición y orientación en el espacio tridimensional.

Una de las principales ventajas de las AprilTags es su robustez frente a variaciones en las condiciones de iluminación y su capacidad para ser detectadas desde diferentes ángulos y distancias. Esto las hace especialmente útiles en aplicaciones donde se requiere un seguimiento preciso, como en la navegación de robots móviles, la calibración de sistemas de cámaras y la realidad aumentada.

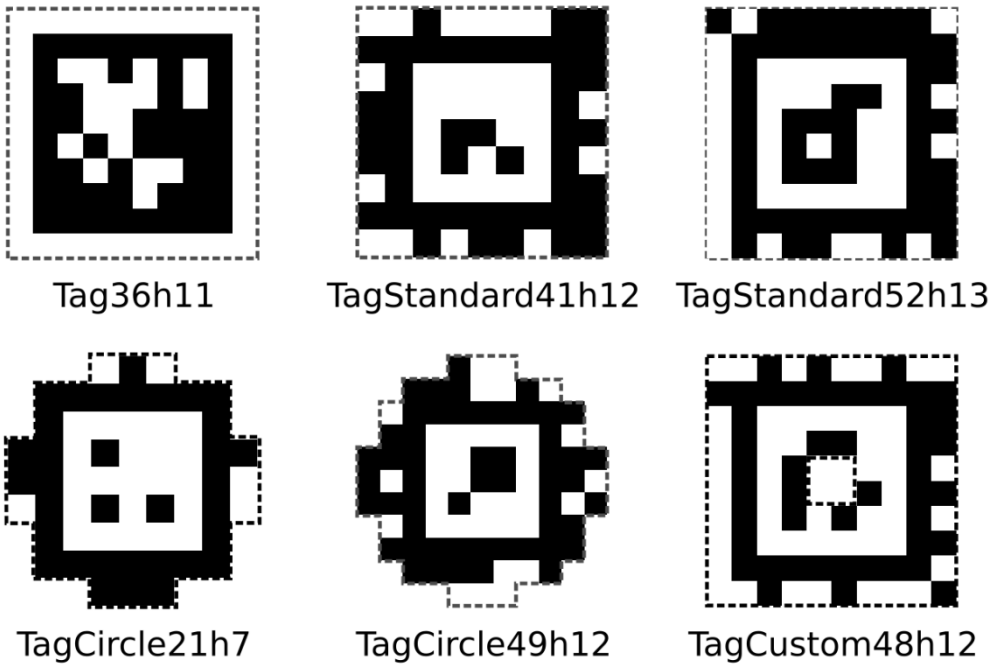

Fig. 2 Familia de Apriltags

apriltag-esp32

La librería **apriltag-esp32** constituye una herramienta robusta y versátil para la implementación de sistemas de visión artificial, desempeñando un papel fundamental en la identificación, posicionamiento y navegación dentro de sistemas mecatrónicos. A través de la integración del módulo **ESP32-CAM** y los marcadores **AprilTags**, es posible realizar la detección, identificación y cálculo de la pose de los marcadores en tiempo real.

Esta librería permite procesar las imágenes capturadas por la cámara, transformándolas a formatos óptimos para su análisis. Posteriormente, implementa un procedimiento que incluye:

- Preprocesamiento: Convierte imágenes capturadas en formatos adecuados para análisis.

- Detección de contornos: Identifica bordes en la imagen para encontrar candidatos a etiquetas (quads).
- Verificación de cuadrados: Verifica que las regiones detectadas sean cuadradas, calculando transformaciones geométricas.
- Decodificación del interior: Extrae el patrón interno del cuadrado detectado para identificar la etiqueta.
- Validación del código: Confirma que el patrón interno pertenece a una familia soportada de etiquetas.
- Estimación de pose: Calcula la posición (x, y, z) y orientación de la etiqueta en el espacio.

Funcion	**Archivos Principales**	**Propósito/Detalles**
Preprocesamiento	image_u8.c, image_u8x3.c, image_u8x4.c	Convierte imágenes capturadas en formatos adecuados para análisis, como escala de grises.
Detección de contornos	apriltag_quad_thresh.c, g2d.c, g2d.h	Identifica bordes en la imagen para encontrar candidatos a etiquetas (quads).
Verificacion de cuadrados	homography.c, homography.h	Verifica que las regiones detectadas sean cuadradas, calculando transformaciones geométricas.
Decodificacion del interior	apriltag.c, tag16h5.c, tag36h11.c	Extrae el patrón interno del cuadrado detectado para identificar la etiqueta.
Validacion del código	apriltag.c	Confirma que el patrón interno pertenece a una familia soportada de etiquetas.

Estimacion de pose	apriltag_pose.c, matd.c, matd.h, math_util.h	Calcula la posición (x, y, z) y orientación de la etiqueta en el espacio tridimensional.

Fig.3 Tabla de archivos más relevantes del repositorio

Instalación

Para comenzar a implementar esta librería, debemos acceder a su repositorio en GitHub: *https://github.com/raspiduino/apriltag-esp32*. Una vez allí, hacemos clic en el botón azul que dice "Code", lo cual desplegará un menú con varias opciones para compartir el repositorio. En nuestro caso, seleccionamos la opción "Download ZIP". Esto descargará el repositorio en un archivo comprimido en formato ZIP.

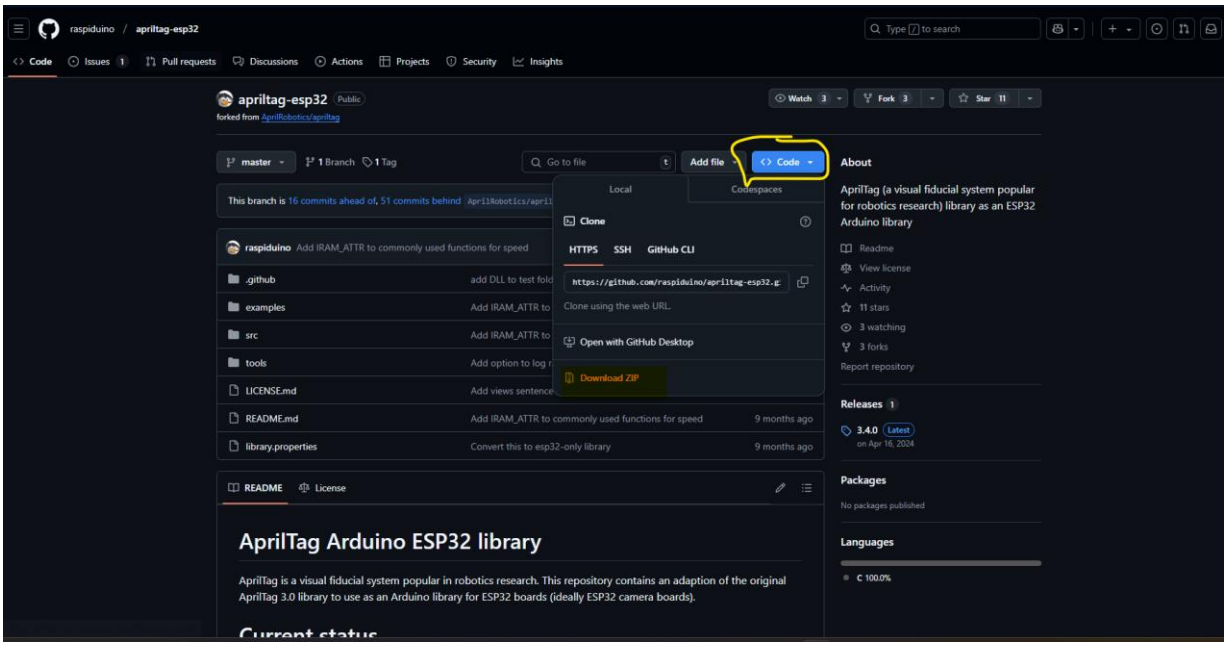

Fig.4 Github de la biblioteca

Abrimos el Arduino IDE y seleccionamos la opción **Sketch** en el menú principal. A continuación, hacemos clic en **Incluir biblioteca** y elegimos la opción **Añadir biblioteca**

.ZIP. Aparecerá una ventana donde debemos localizar el archivo .ZIP descargado del repositorio. Una vez seleccionado, hacemos clic en **Abrir**. Esto incorporará la biblioteca al Arduino IDE, permitiendo que podamos utilizarla en nuestros proyectos.

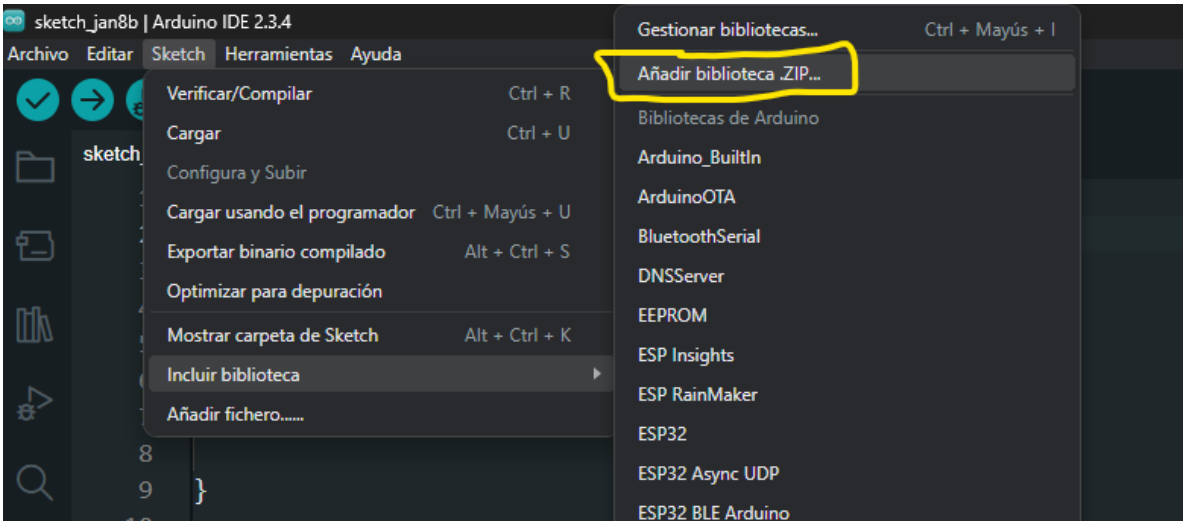

Fig. 5 menu de gestión de bibliotecas en Arduino IDE

Programación identificación ID

Para comenzar a programar, creamos un nuevo sketch en el Arduino IDE. A continuación, seleccionamos el modelo de cámara que vamos a utilizar. En este caso, como estamos trabajando con la **ESP32-CAM**, basta con incluir la siguiente línea en el código:

```
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
```

Además, es necesario declarar los headers correspondientes al hardware y a la librería de AprilTag. Para ello, debemos incluir las siguientes líneas en el código:

```
// Hardware-specific headers
#include "esp_camera.h"
#include "camera_pins.h"
```

```
// Apriltag headers
#include "apriltag.h"
#include "tag36h11.h"
#include "common/image_u8.h"
#include "common/zarray.h"
```

Debemos crear el archivo camera_pins.h. Para ello, hacemos clic en los tres puntos ubicados en la parte superior derecha o debajo del ícono del monitor serial (lupa), seleccionamos la opción **Nueva pestaña** y escribimos el nombre del archivo tal como se declaró.

En este proyecto utilizaremos la familia de AprilTag **tag36h11**, conocida por su estabilidad y que permite usar etiquetas desde la 0 hasta la 35. Por esta razón, declaramos los headers necesarios, siendo los dos últimos fundamentales para preprocesar las imágenes y detectar los tags.

Además, es necesario declarar los parámetros intrínsecos de la cámara. *Cada ESP32-CAM tiene valores únicos, por lo que es imprescindible realizar una calibración para obtener estos parámetros.* Los valores intrínsecos se deben agregar justo debajo de la declaración de los headers con el siguiente código:

```
// Parámetros de la cámara
#define TAG_SIZE 0.05
#define FX 370.1614
#define FY 379.2102
#define CX 160.5623
#define CY 128.9320
```

En la función setup(), debemos inicializar la PSRAM, para apoyarnos a saber si ya se realizó este paso, en el código escribimos :

```
  // Inicializar PSRAM
  psramInit();

  if (!psramInit()) {
    Serial.println("¡Error al inicializar PSRAM!");
  } else {
    Serial.println("PSRAM habilitada correctamente.");
  }
```

Y hacer la estructura de configuración de nuestra cámara, como se muestra a continuación:

```
  // Configuración de la cámara
  camera_config_t config;

 // Set camera pins (as defined in cam_pins.h)

  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  // Set clock frequency
  config.xclk_freq_hz = 20000000;

  // Set frame config
  config.frame_size = FRAMESIZE_VGA;
  config.pixel_format = PIXFORMAT_GRAYSCALE; // Required
for AprilTag processing
```

```
  config.grab_mode = CAMERA_GRAB_LATEST; // Has to be in
this mode, or detection will be lag
  config.fb_location = CAMERA_FB_IN_PSRAM;
  //config.jpeg_quality = 12;
  config.fb_count = 1; // Can't afford (and also not
needed) to have 2
```

Enel paso anterior, definimos los pines utilizados por nuestra ESP32-CAM, la frecuencia de operación, la resolución, y el formato de píxel. En este caso, hemos configurado el formato a escala de grises, ya que es necesario para identificar los tags. Finalmente, establecimos la calidad de la imagen capturada por la ESP32-CAM.

El siguiente paso es inicializar la cámara. Para ello, debemos agregar el siguiente código en la función setup:

```
  if (esp_camera_init(&config) != ESP_OK) {
    Serial.println("¡Error al inicializar la cámara!");
    ESP.restart();
  }
  Serial.println("Cámara inicializada correctamente");
```

Este código asegura que la cámara se inicialice correctamente. En caso de que la inicialización falle, el sistema se reiniciará automáticamente para evitar desbordamientos de memoria o sobrecarga en el procesador.

En la sección principal del programa, es decir, en la función loop(), debemos implementar el flujo completo que realizará la ESP32-CAM. Esto incluye capturar una imagen, realizar el preprocesamiento, detectar los quads y decodificar su contenido. Para lograrlo, debemos agregar el siguiente código en la función loop():

```
// Captura de imagen
  camera_fb_t *fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Error al capturar imagen");
    return;
  }
// Convert our framebuffer to detector's input format

  image_u8_t im = {
    .width = static_cast<int32_t>(fb->width),
    .height = static_cast<int32_t>(fb->height),
    .stride = static_cast<int32_t>(fb->width),
    .buf = fb->buf
  };
  // Create tag family object
  apriltag_family_t *tf = tag36h11_create();

  // Create AprilTag detector object
  apriltag_detector_t *td = apriltag_detector_create();

  // Add tag family to the detector
  apriltag_detector_add_family(td, tf);

  // Tag detector configs
  td->quad_decimate = 2.0;
  td->nthreads = 1;

    // Detect
  zarray_t *detections = apriltag_detector_detect(td, &im);
  if (detections != nullptr) {
    int detections_count = zarray_size(detections);
    if (detections_count > 0) {
```

```
    for (int i = 0; i < detections_count; i++) {
      apriltag_detection_t *det;
      zarray_get(detections, i, &det);

      apriltag_detection_info_t info;
      info.det = det;
      info.tagsize = TAG_SIZE;
      info.fx = FX;
      info.fy = FY;
      info.cx = CX;
      info.cy = CY;


      // Construir mensaje con los datos de la etiqueta
      String message = "ID: " + String(det->id);

      // Depuración
      Serial.println("Enviando mensaje: " + message);
    }
  } else {
    Serial.println("No se detectaron etiquetas.");
  }

  apriltag_detections_destroy(detections);
}
  // Return camera framebuffer to the camera driver
esp_camera_fb_return(fb);

  // Free detection result object
apriltag_detector_destroy(td);

tag36h11_destroy(tf);
```

Programación de calculo de pose

Para calcular la pose, es necesario incluir algunos archivos headers adicionales. A continuación, se muestran los que debemos agregar:

```
#include "apriltag_pose.h"
#include "common/matd.h"
```

Para calcular únicamente la posición, debemos agregar el siguiente código dentro de la función loop():

```
        apriltag_pose_t pose;
        double err = estimate_tag_pose(&info, &pose);

        double posX = MATD_EL(pose.t, 0, 0);
        double posY = MATD_EL(pose.t, 1, 0);
        double posZ = MATD_EL(pose.t, 2, 0);

        // Construir mensaje con los datos de la etiqueta
        String message = "ID: " + String(det->id) +
                         ", X: " + String(posX, 4) +
                         ", Y: " + String(posY, 4) +
                         ", Z: " + String(posZ, 4);
```

Si también necesitamos calcular la orientación, debemos proceder de la siguiente manera:

```
      // Compute the yaw, pitch, and roll from the rotation
matrix (and convert to degree)
      double yaw = atan2(MATD_EL(pose.R, 1, 0),
MATD_EL(pose.R, 0, 0)) * RAD_TO_DEG;
      double pitch = atan2(-MATD_EL(pose.R, 2, 0),
sqrt(pow(MATD_EL(pose.R, 2, 1), 2) + pow(MATD_EL(pose.R, 2,
2), 2))) * RAD_TO_DEG;
```

```
      double roll = atan2(MATD_EL(pose.R, 2, 1),
MATD_EL(pose.R, 2, 2)) * RAD_TO_DEG;

      // Print the yaw, pitch, and roll of the camera
      printf("y,p,r: %15f, %15f, %15f\n", yaw, pitch,
roll);
```

Calibración de la cámara

Para llevar a cabo la calibración, es esencial disponer de un entorno con luz controlada, un tablero de ajedrez, Matlab, y varias fotografías tomadas desde diferentes ángulos. Utilizaremos Matlab, ya que cuenta con una aplicación llamada “Camera Calibrator”, la cual facilita enormemente este proceso.

Toma de fotos

Para la toma de fotos se debe seguir los siguientes pasos:

1. Totalmente paralelo a la lente de la cámara.
2. Realizar rotaciones en el sentido de las manecillas del reloj alrededor del eje x del plano (Cuadrícula).
3. Realizar rotaciones en sentido contrario a las manecillas del reloj alrededor del eje x del plano (Cuadrícula).
4. Mover la cuadrícula hacia la derecha sin exceder los 45º de ángulo (véase la figura 4).
5. Mover la cuadrícula hacia la izquierda sin sobrepasar los 45º de ángulo (véase la figura 4).
6. Mover la cuadrícula hacia abajo sin superar los 45º de ángulo (véase la figura 4).
7. Mover la cuadrícula hacia arriba sin exceder los 45º de ángulo (véase la figura 4).
8. Trasladar la cuadrícula a cierta distancia, ya sea más lejos o más cerca de la cámara.
9. Una vez trasladada, realizar los pasos desde el 3 hasta el 7.

10. Si es necesario, repetir los pasos 8 y 9 una vez más hasta obtener suficientes fotografías.

Adicionalmente se recomienda consultar el siguiente documento:

https://cc.sisal.unam.mx/AnalisisMedicion/Manuales/Manual%20de%20calibracion.pdf

MatLab

Para acceder, debemos iniciar Matlab, tener el add-on de *image processing and computer vision instalado* y presionar en el icono que se muestra a continuación.

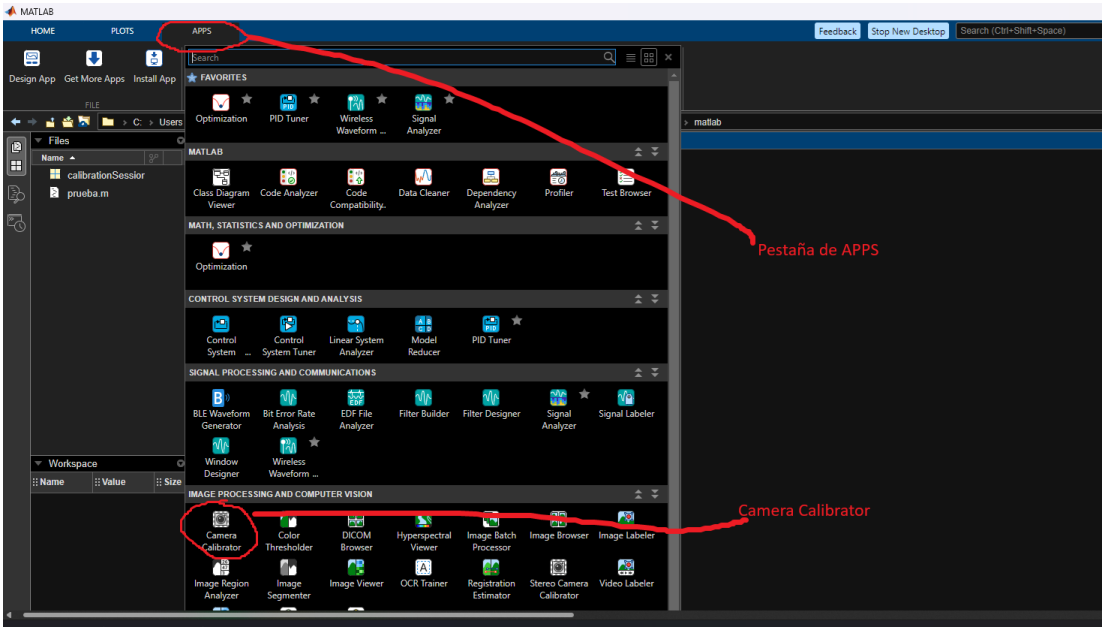

Figura 6 Menu de Apps en Matlab

Después de unos segundos, se cargara esta ventana donde nos pedirá cargar las fotos, se recomienda que tomemos varias fotos para tener mas datos, presionamos click en *add files* y *from file*

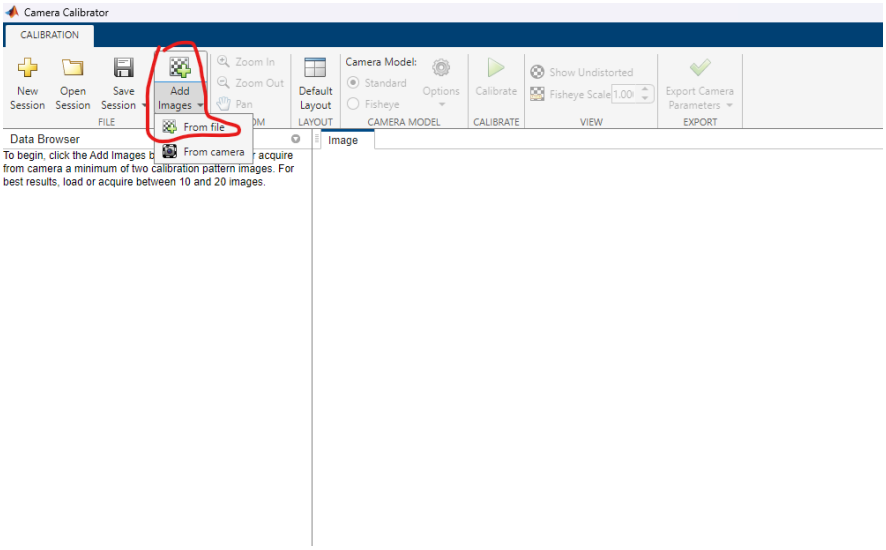

Fig. 7 menú de Camera Calibrator

Seleccionamos todas las fotos que deseamos y de ahí presionamos en abrir

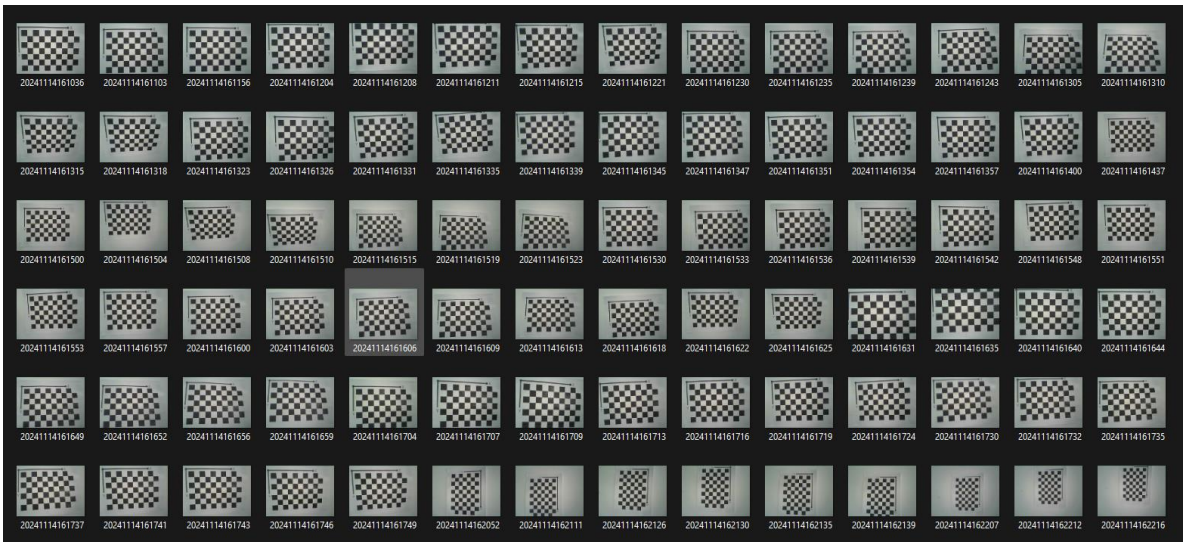

Fig. 8 selección de imagenes

Después nos saldrá una ventana, la cual nos pedirá la forma de nuestro patrón, cual es el largo de los cuadrados y presionamos en low distorsion. Y de ahí presionamos en ok.

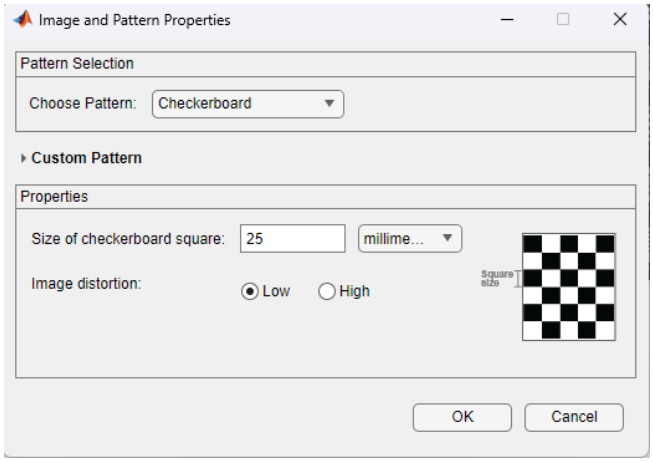

Fig. 9 Definición de parámetros de la cuadricula

Se tardará un momento en lo que procesan las imágenes, dependerá de la cantidad de las imágenes que nosotros le cargamos, una vez finalizada la cargada de imágenes podemos visualizar los puntos de referencia y el origen del patrón (Fig. 10).

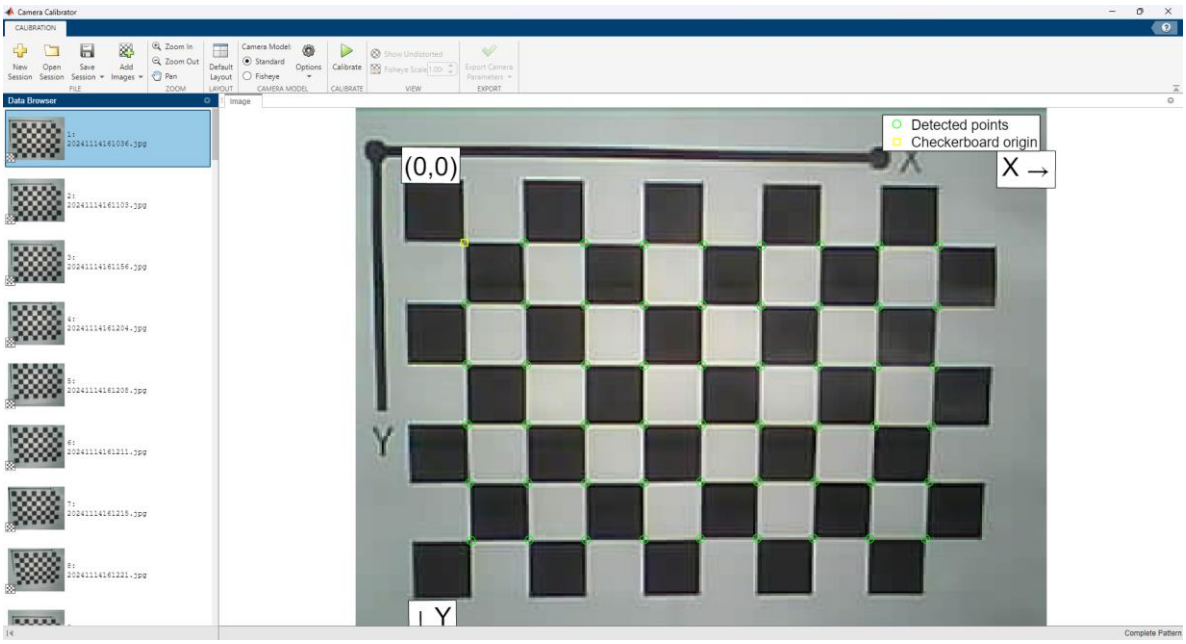

Fig. 10 Visualización de puntos

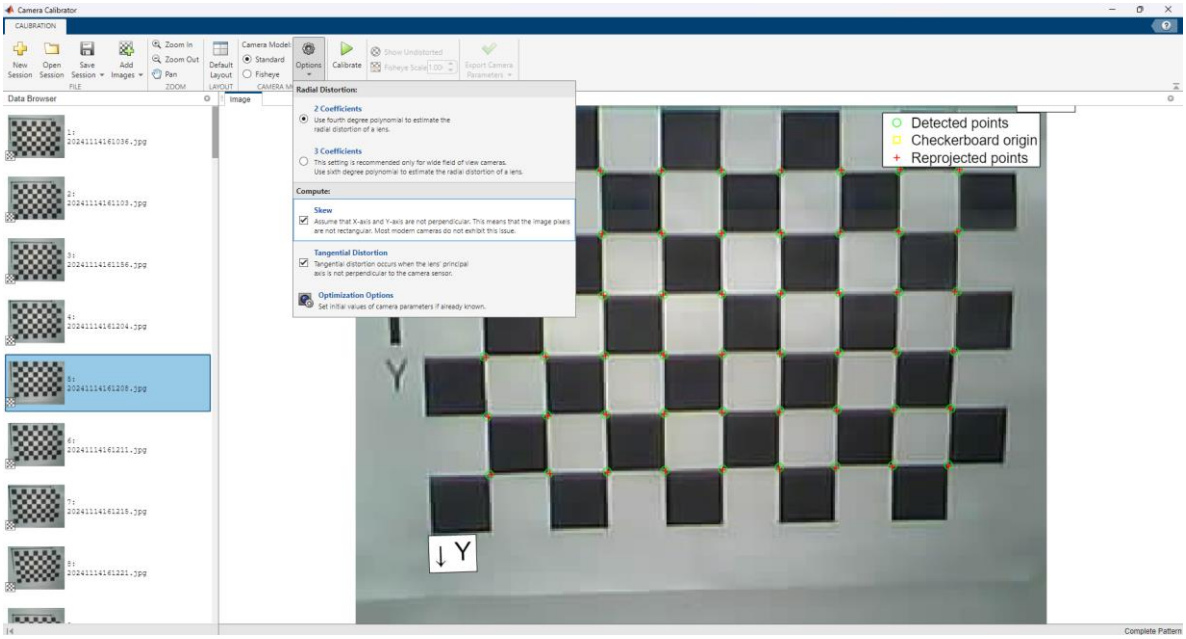

Fig. 11 Declaración de parámetros de camera

Adicionalmente, es necesario especificar si la cámara es estándar o si presenta un efecto de ojo de pez. También se debe definir la cantidad de parámetros intrínsecos de la cámara. En nuestro caso, seleccionaremos dos coeficientes para distorsión radial y en compute: *skew* y tangencial distortion (Fig. 11). Posteriormente, presionamos la opción *Calibrate*.

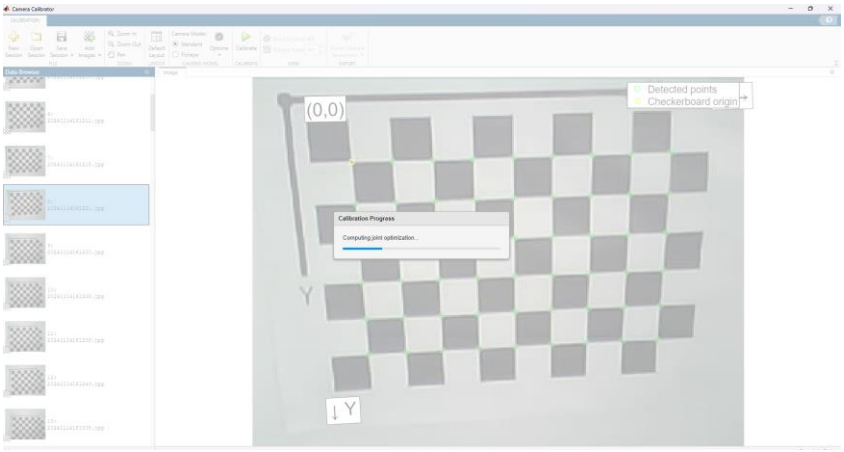

Fig. 12 Procesamiento de imágenes

Se presentarán las imágenes con los puntos reproyectados, junto con una gráfica que muestra los errores de reproyección correspondientes a cada fotografía. En esta etapa, será posible determinar el margen de error tolerable. De manera arbitraria, se establecerá que aquellos errores que superen el 50% serán descartados.

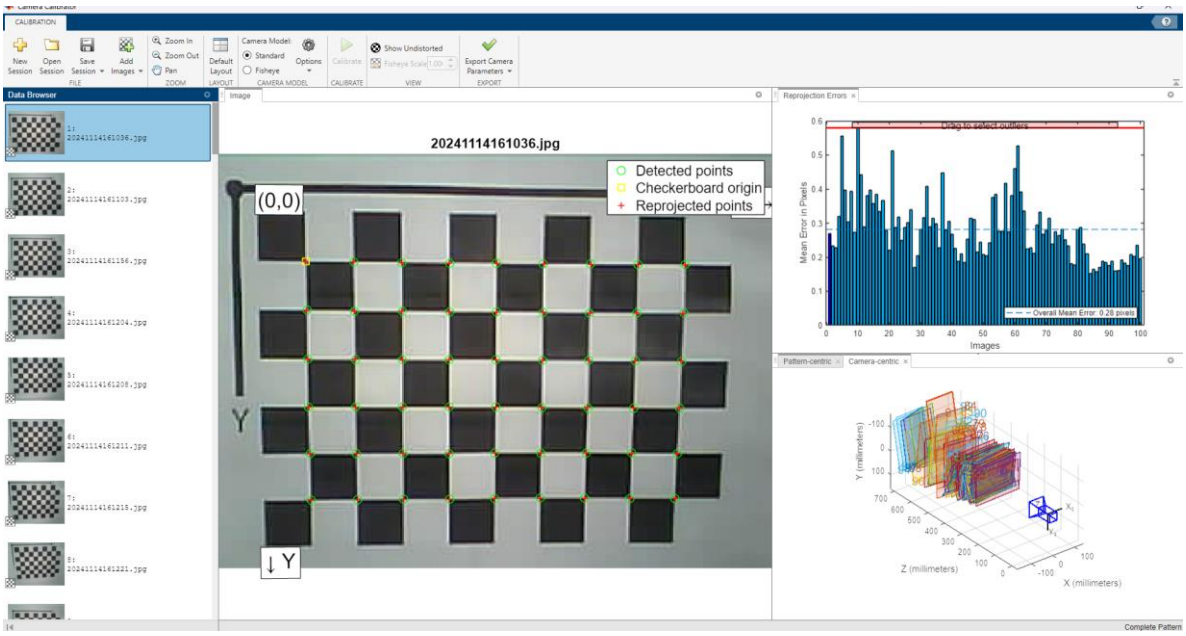

Fig. 13 Resultados y errores de reproyeccion

Finalmente, seleccionamos la opción *Export Camera Parameters*, lo que desplegará la herramienta *Generate MATLAB Script*. Al generar el script, este deberá ejecutarse para obtener los parámetros necesarios, tal como se muestra a continuación.

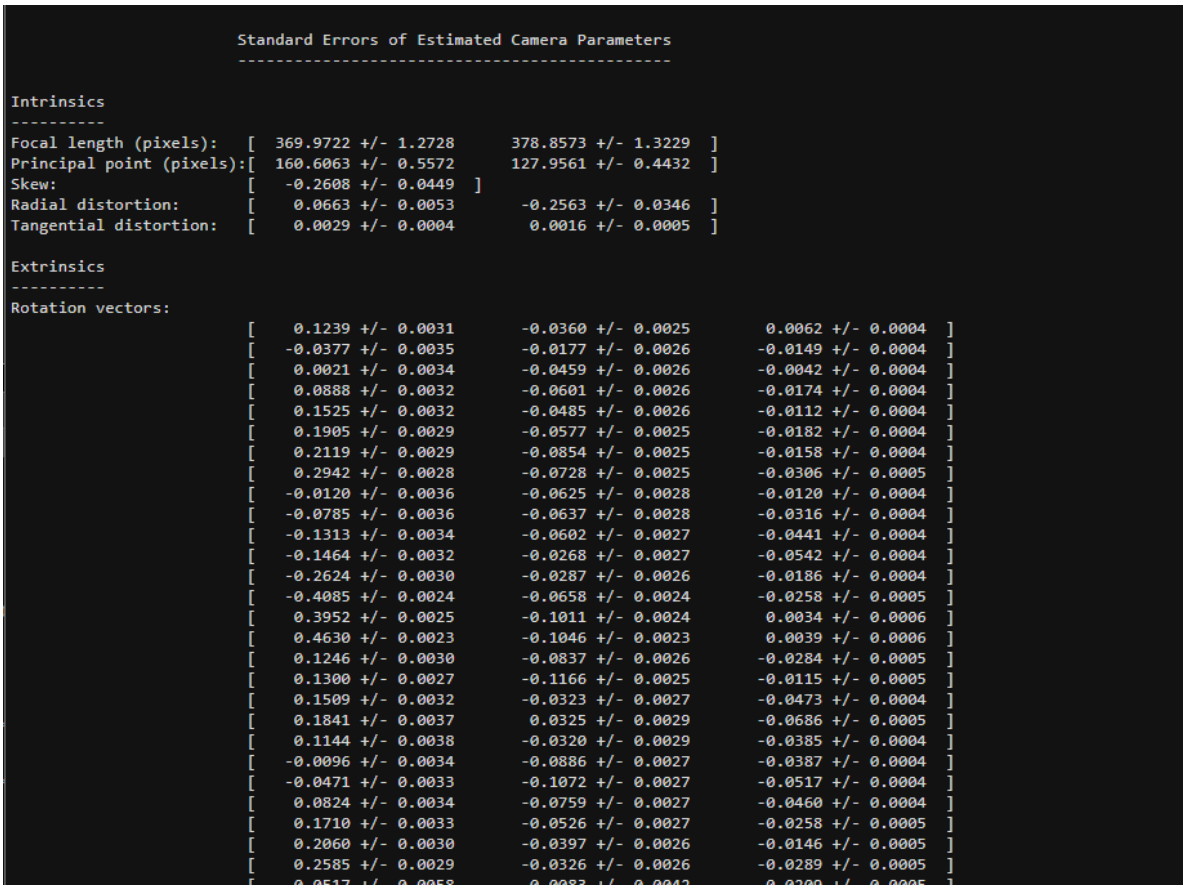

Fig. 14 Parámetros intrínsecos

Los valores de Fx e Fy se obtendrán de la sección denominada *Focal Length*, correspondiendo el primer valor a Fx y el segundo a Fy . De manera similar, los valores de Cx y Cy se derivarán de la sección *Principal Point*.