# ESP-CAM con AprilTag

## ESP32CAM

La ESP32-CAM es un módulo de desarrollo basado en el microcontrolador ESP32-S que integra una cámara OV2640 y soporte para tarjetas microSD, lo que la convierte en una solución económica y compacta para proyectos de visión artificial, videovigilancia, reconocimiento de imágenes y aplicaciones IoT. Además de contar con conectividad Wi-Fi y Bluetooth, ofrece suficiente capacidad de procesamiento para ejecutar algoritmos básicos de visión y transmitir video en tiempo real. Su bajo costo y versatilidad la han vuelto muy popular en proyectos de domótica, robótica y sistemas de seguridad. Su Pin out, se muestra en la figura 1.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 1. Pin out ESP32-CAM [4]

## 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 (figura 2).

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.

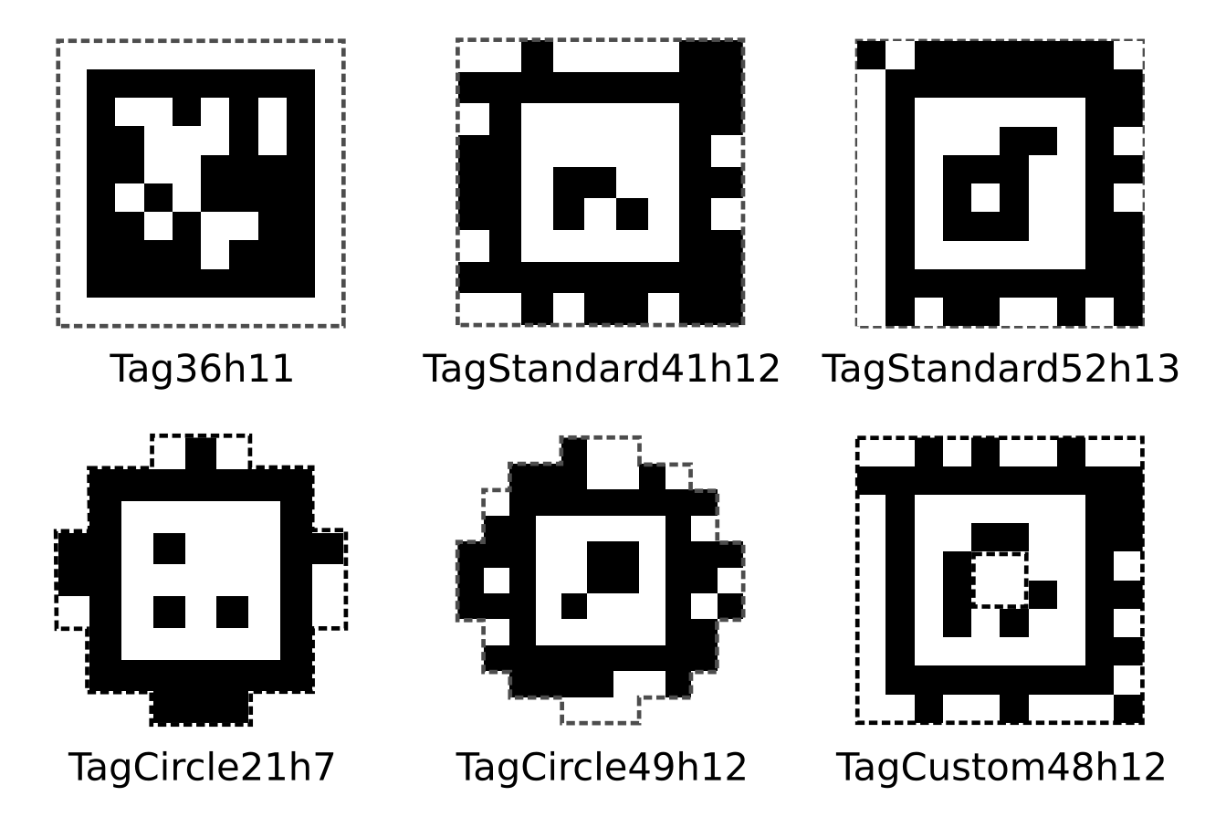


Figura 2. Familia de Apriltags [1]

## [apriltag-esp32](https://github.com/raspiduino/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.

Tabla 1. Tabla de etapas del proceso de detección de etiquetas AprilTag [1]

|  |  |  |
| --- | --- | --- |
| Función | 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*). |
| Verificación de cuadrados | homography.c, homography.h | Verifica que las regiones detectadas sean cuadradas, calculando transformaciones geométricas. |
| Decodificación del interior | apriltag.c, tag16h5.c, tag36h11.c | Extrae el patrón interno del cuadrado detectado para identificar la etiqueta. |
| Validación del código | apriltag.c | Confirma que el patrón interno pertenece a una familia soportada de etiquetas. |
| Estimación 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. |

## Instalación

Para comenzar a implementar esta librería, debemos acceder a su repositorio en GitHub: <https://github.com/raspiduino/apriltag-esp32> [1]. Una vez allí, hacemos clic en el botón azul que dice "Code", como se ve en la figura 3, 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.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 3. 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** (figura 4). 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.

Captura de pantalla con la imagen de una pantalla

Descripción generada automáticamente

Figura 4. Menú 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 [1,3]:

#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

En el 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 [2]: <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, como se ve en la figura 5.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 5. Menú de Apps en Matlab

Después de unos segundos, se cargará esta ventana donde nos pedirá cargar las fotos, se recomienda que tomemos varias fotos para tener más datos, presionamos click en *add files* y *from file,* como se muestra en la figura 6.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Figura 6. Menú de Camera Calibrator

Seleccionamos todas las fotos que deseamos y de ahí presionamos en abrir, como se ve en la figura 7.

Tabla

Descripción generada automáticamente

Figura 7 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. Mostrado en la figura 8.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 8. 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 (Figura 9).

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Figura 9 Visualización de puntos

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 10. 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* (Figura 10). Posteriormente, presionamos la opción *Calibrate*. Después empezara a procesar las imágenes como se muestra en la figura 11.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 11. 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. Como se muestra en la figura 12.Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 12. Resultados y errores de reproyección

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 en la figura 13.

Tabla

El contenido generado por IA puede ser incorrecto.

Fig. 13 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*.

## Referencias

[1] raspiduino, "apriltag-esp32," GitHub repository, 2025. [En línea]. Disponible en: <https://github.com/raspiduino/apriltag-esp32>

[2] UNAM, "Manual de calibración," Centro de Ciencias de Sisbem, México. [En línea]. Disponible en: https://cc.sisal.unam.mx/AnalisisMedicion/Manuales/Manual%20de%20calibracion.pdf

[3] J. Lopez Pob, "SS," GitHub repository. [En línea]. Disponible en: https://github.com/JesusLopezPob/SS

[4] Random Nerd Tutorials, "ESP32-CAM," Random Nerd Tutorials. [En línea]. Disponible en: https://randomnerdtutorials.com/?s=ESP32-CAM