Documentación de librería [**apriltag-esp32**](https://github.com/raspiduino/apriltag-esp32)

|  |  |  |
| --- | --- | --- |
| 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. |
|  |  |  |

**Preprocesamiento**

**Funciones Principales del Preprocesamiento**

1. **Conversión a Escala de Grises**:
   * Las imágenes capturadas suelen estar en formato RGB o similar (color).
   * Para simplificar el análisis, la imagen se convierte a escala de grises, donde cada píxel tiene un único valor de intensidad (0-255). Esto reduce el costo computacional.
2. **Reducción de Ruido**:
   * Se aplican filtros básicos (como un filtro de suavizado o mediana) para reducir el ruido en la imagen, lo que ayuda a eliminar píxeles no deseados que podrían interferir con la detección de bordes.
3. **Normalización**:
   * Ajusta los niveles de brillo y contraste para asegurar que las etiquetas sean detectables bajo distintas condiciones de iluminación.
4. **Conversión a Formato Específico**:
   * Los algoritmos de la biblioteca trabajan con estructuras de datos específicas (como image\_u8 o image\_u8x4), que representan imágenes en diferentes formatos (grises, 3 canales, 4 canales).

**Archivos que Implementan el Preprocesamiento**

1. **image\_u8.c y image\_u8x3.c**:
   * Estas bibliotecas manejan la estructura de datos de imágenes en escala de grises y en color (3 canales).
   * Funciones típicas incluyen:
     + **Cargar una imagen**: Inicializa la estructura con datos de la cámara.
     + **Acceso a píxeles**: Permite leer/escribir valores de intensidad de píxeles.
     + **Conversión de formato**: Transforma imágenes de color a escala de grises.
2. **image\_u8x4.c**:
   * Similar a image\_u8x3.c, pero trabaja con imágenes de 4 canales, que podrían incluir un canal adicional (como transparencia o un mapa de profundidad).
3. **apriltag\_quad\_thresh.c**:
   * Aunque principalmente realiza detección de bordes, este archivo también aplica operaciones de preprocesamiento como umbral adaptativo para resaltar las etiquetas en la imagen.

**Flujo del Preprocesamiento**

1. **Entrada**: La imagen cruda (RGB o similar) es capturada por la cámara ESP32.
2. **Conversión**: Se convierte la imagen a escala de grises (image\_u8).
3. **Normalización**: Se ajustan los niveles de brillo y contraste para mejorar la calidad.
4. **Formato Final**: La imagen resultante se convierte en una estructura image\_u8 que se pasa al siguiente paso del pipeline (detección de contornos).

**Detección de contornos**

**Pasos en la Detección de Contornos**

1. **Aplicación de un Umbral Adaptativo**:
   * La detección comienza destacando los bordes en la imagen.
   * El archivo apriltag\_quad\_thresh.c utiliza un **umbral adaptativo** para identificar áreas donde hay un cambio brusco de intensidad (bordes). Esto asegura que los bordes sean visibles incluso con iluminación no uniforme.
2. **Detección de Bordes**:
   * Una vez aplicados los umbrales, el algoritmo busca bordes continuos en la imagen, lo que genera una lista de segmentos de línea que delimitan posibles regiones cuadradas.
3. **Agrupamiento de Líneas**:
   * Los bordes detectados se agrupan para formar polígonos cerrados (o cuasi-cerrados) que puedan representar candidatos a etiquetas.
4. **Verificación de la Forma Cuadrada**:
   * Entre los polígonos encontrados, el algoritmo verifica cuáles tienen 4 lados (cuadrados o rectángulos).
   * También se evalúan características geométricas, como:
     + Ángulos cercanos a 90°.
     + Relación de aspecto consistente con las dimensiones de una etiqueta.
5. **Eliminación de Candidatos Inválidos**:
   * Polígonos que no cumplen con los criterios de una forma cuadrada son descartados.
   * Esto ayuda a reducir el número de falsos positivos en etapas posteriores.
6. **Salida de Contornos**:
   * Los polígonos válidos (cuadrados) se pasan a la siguiente etapa del pipeline, donde se analiza el contenido interior para decodificar el ID de la etiqueta.

**Archivos que Implementan la Detección de Contornos**

1. **apriltag\_quad\_thresh.c**:
   * Este archivo contiene el algoritmo principal para encontrar regiones cuadradas (quads) en la imagen.
   * Funciones clave:
     + **Detección de bordes**: Usa filtros para resaltar áreas con cambios de intensidad.
     + **Umbral adaptativo**: Segmenta la imagen en regiones relevantes.
     + **Agrupamiento de líneas**: Genera polígonos a partir de segmentos de línea.
2. **g2d.c y g2d.h**:
   * Manejan operaciones geométricas en 2D, como la detección y agrupamiento de bordes.
   * Incluyen funciones para calcular distancias, ángulos, y relaciones entre líneas.

**Flujo del Algoritmo de Detección de Contornos**

1. **Entrada**:
   * La imagen preprocesada (en escala de grises) es enviada al algoritmo.
2. **Detección de bordes**:
   * Se identifican bordes significativos mediante gradientes de intensidad o umbral adaptativo.
3. **Agrupamiento**:
   * Los bordes detectados se agrupan para formar posibles cuadrados.
4. **Filtrado de candidatos**:
   * Solo los polígonos que cumplen con los criterios geométricos se retienen.

**Verificación de Cuadrados**

**Pasos en la Verificación de Cuadrados**

1. **Entrada: Polígonos Detectados**:
   * Recibe una lista de polígonos generados en la etapa de detección de contornos. Estos polígonos son agrupaciones de bordes que podrían representar etiquetas.
2. **Verificación de la Forma**:
   * Se evalúa si el polígono tiene exactamente 4 lados.
   * Se comprueban propiedades geométricas como:
     + **Ángulos Internos**: Los ángulos entre los lados deben ser cercanos a 90°.
     + **Relación de Aspecto**: Los lados opuestos deben ser aproximadamente iguales en longitud.
3. **Homografía**:
   * Se aplica un cálculo de **homografía** para comprobar que el polígono puede transformarse en un cuadrado perfecto en perspectiva.
   * Esto implica evaluar si el polígono es un cuadrilátero proyectado en 2D.
4. **Eliminación de Polígonos Inválidos**:
   * Polígonos que no cumplen con las condiciones anteriores son descartados, ya que no tienen la forma esperada de una etiqueta válida.
5. **Salida: Cuadrados Verificados**:
   * Solo los polígonos que pasan las verificaciones geométricas avanzan al siguiente paso, donde se analiza el contenido interior.

**Archivos Clave para la Verificación de Cuadrados**

1. **homography.c y homography.h**:
   * Implementan funciones para calcular transformaciones geométricas (homografías) y evaluar la forma de los polígonos.
   * Utilizan matrices para mapear las coordenadas de un polígono detectado a las de un cuadrado ideal.
2. **apriltag\_quad\_thresh.c**:
   * Contiene funciones que verifican propiedades geométricas de los polígonos, como la cantidad de lados, ángulos internos y relación de aspecto.

**Propiedades Evaluadas en la Verificación**

1. **Cantidad de Lados**:
   * Solo se consideran polígonos con 4 lados, ya que las etiquetas AprilTag son cuadradas o rectangulares.
2. **Ángulos Internos**:
   * Se calculan los ángulos entre los vectores formados por lados adyacentes.
   * Los ángulos deben ser cercanos a 90° (± un margen de tolerancia).
3. **Lados Paralelos**:
   * Se verifica que los lados opuestos sean aproximadamente paralelos y de longitudes similares.
4. **Homografía**:
   * Se calcula una transformación homográfica para proyectar el cuadrilátero en un cuadrado.
   * Si los errores de proyección son pequeños, el polígono es válido.

**Flujo del Algoritmo**

1. **Entrada**: Polígonos detectados de la etapa anterior.
2. **Verificación Geométrica**:
   * Evalúa lados, ángulos y relación de aspecto.
3. **Cálculo de Homografía**:
   * Verifica que el polígono puede representarse como un cuadrado proyectado.
4. **Filtrado**:
   * Descarta polígonos inválidos.
5. **Salida**: Cuadrados verificados para su posterior análisis.

**Homografía: Un Concepto Clave**

* **Definición**: Una homografía es una transformación geométrica que mapea puntos de un plano 2D a otro plano 2D. En este caso, mapea un cuadrilátero en perspectiva (la etiqueta) a un cuadrado ideal.
* **Uso**:
  + Ayuda a verificar si el polígono detectado puede ser una proyección de una etiqueta cuadrada.
  + Corrige la distorsión introducida por la perspectiva de la cámara.

**Decodificacion del interior**

**Pasos de Decodificación del Interior**

1. **Normalización del Polígono**:
   * Usando la transformación homográfica calculada previamente, el cuadrado detectado se "endereza" para que se vea como un cuadrado perfecto, eliminando cualquier distorsión por perspectiva.
2. **Segmentación en Celdas**:
   * El área interna del cuadrado se divide en una cuadrícula regular de celdas (por ejemplo, 6x6, 7x7), dependiendo de la familia de la etiqueta.
   * Cada celda representa un bit de información binaria.
3. **Lectura de Bits**:
   * Para cada celda, se calcula la intensidad promedio de los píxeles:
     + **Blanco**: Se interpreta como un 1.
     + **Negro**: Se interpreta como un 0.
4. **Validación de la Decodificación**:
   * Se verifica si el patrón binario extraído corresponde a un diseño válido de la familia de etiquetas (por ejemplo, 36h11, 16h5).
   * Esto incluye:
     + Validar un **código Hamming** incorporado en el diseño.
     + Corregir pequeños errores si es necesario (usando propiedades del código Hamming).
5. **Extracción del ID Único**:
   * Una vez validado, el patrón binario se traduce en el ID único de la etiqueta.
6. **Salida**:
   * El ID de la etiqueta, junto con información sobre su pose, se envía como resultado final del sistema.

**Archivos Clave para la Decodificación**

1. **apriltag.c**:
   * Contiene las funciones principales para procesar los datos del interior de los cuadrados.
   * Realiza tareas como la segmentación de celdas, lectura de bits y validación.
2. **Archivos de Familias de Etiquetas (tag16h5.c, tag36h11.c, etc.)**:
   * Estos archivos contienen los patrones binarios válidos y los códigos Hamming asociados a cada familia de etiquetas.
   * Se usan para validar y decodificar los patrones extraídos.

**Flujo del Algoritmo de Decodificación**

1. **Entrada**:
   * El cuadrado transformado (normalizado) con los píxeles de su área interna.
2. **Segmentación**:
   * Dividir el cuadrado en celdas correspondientes al diseño de la familia de etiquetas.
3. **Lectura de Bits**:
   * Interpretar cada celda como un 1 o 0 según la intensidad promedio de sus píxeles.
4. **Validación**:
   * Comparar el patrón binario con los patrones válidos de la familia de etiquetas.
   * Aplicar códigos Hamming para corregir errores menores.
5. **Extracción de ID**:
   * Traducir el patrón en el ID único de la etiqueta.
6. **Salida**:
   * El ID único de la etiqueta es devuelto al usuario o al sistema.

**Validación con Códigos Hamming**

* Los patrones de las etiquetas incluyen bits de redundancia diseñados para detectar y corregir errores menores (como un bit incorrecto debido a ruido o iluminación deficiente).
* El algoritmo compara el patrón leído con los patrones válidos y aplica correcciones si es posible.

**Relación con Familias de Etiquetas**

* **tag16h5, tag36h11, etc.**:
  + Cada familia define:
    - La cuadrícula (número de celdas por fila/columna).
    - Los patrones válidos.
    - La cantidad de redundancia (códigos Hamming) para corrección de errores.
  + Por ejemplo, 36h11 tiene 36 bits de datos, con 11 bits de corrección.

**Validacion del código**

**Pasos de Validación del Código**

1. **Comparación con Patrones Válidos**:
   * El patrón binario decodificado del interior de la etiqueta se compara con los patrones predefinidos de la familia correspondiente (por ejemplo, 36h11, 16h5).
   * Si el patrón coincide exactamente con uno válido, se considera correcto.
2. **Detección de Errores (Códigos Hamming)**:
   * Si el patrón leído no coincide exactamente con ningún patrón válido, se calcula la distancia Hamming entre el patrón leído y los patrones válidos.
   * La **distancia Hamming** mide el número de bits diferentes entre dos patrones binarios.
3. **Corrección de Errores**:
   * Si la distancia Hamming es pequeña (por ejemplo, 1 o 2 bits), el sistema puede corregir el patrón reemplazando los bits incorrectos para coincidir con un patrón válido.
   * Esto se basa en la redundancia incorporada en los diseños de las familias de etiquetas (por ejemplo, la familia 36h11 tiene 36 bits de datos con 11 bits dedicados a la corrección de errores).
4. **Validación Final**:
   * Después de la corrección (si fue necesaria), el patrón resultante se verifica nuevamente para asegurarse de que es válido.
5. **Rechazo de Patrones Inválidos**:
   * Si la distancia Hamming es demasiado grande (es decir, demasiados bits diferentes), el patrón se considera inválido y la etiqueta es descartada.

**Archivos Clave para la Validación del Código**

1. **apriltag.c**:
   * Contiene las funciones principales para comparar patrones, calcular distancias Hamming y realizar la corrección de errores.
2. **Archivos de Familias de Etiquetas (tag16h5.c, tag36h11.c, etc.)**:
   * Estos archivos contienen los patrones binarios válidos y las tablas de corrección de errores específicas para cada familia de etiquetas.

**Códigos Hamming: Cómo Funciona**

* **Bits de Datos y Bits de Redundancia**:
  + Cada etiqueta codifica información en bits de datos (representan el ID de la etiqueta) y bits de redundancia (utilizados para detectar y corregir errores).
  + Por ejemplo:
    - La familia 36h11 tiene 36 bits de datos y 11 bits de redundancia.
* **Detección de Errores**:
  + El sistema calcula una "suma de verificación" basada en los bits de redundancia. Si esta suma no coincide con lo esperado, se detecta que hay errores.
* **Corrección de Errores**:
  + Si la distancia Hamming es pequeña, el sistema puede identificar y corregir los bits erróneos basándose en las propiedades del código Hamming.

**Flujo del Algoritmo de Validación**

1. **Entrada**:
   * El patrón binario decodificado de la etiqueta.
2. **Comparación con Patrones Válidos**:
   * Comparar el patrón leído con los patrones válidos almacenados en las tablas de la familia de etiquetas.
3. **Cálculo de Distancia Hamming**:
   * Si el patrón no coincide exactamente, calcular cuántos bits difieren del patrón más cercano.
4. **Corrección de Errores**:
   * Si la distancia Hamming es pequeña, corregir los bits erróneos.
5. **Validación Final**:
   * Asegurarse de que el patrón corregido sea válido.
6. **Rechazo de Patrones Inválidos**:
   * Si el patrón no puede corregirse o la distancia Hamming es demasiado grande, rechazarlo.

**Optimización para ESP32**

1. **Comparación Rápida**:
   * El sistema utiliza tablas predefinidas de patrones válidos para minimizar el tiempo de búsqueda.
2. **Uso de Códigos Hamming**:
   * Permite corregir errores menores, reduciendo la probabilidad de falsos negativos sin aumentar significativamente el costo computacional.

**Relación con Familias de Etiquetas**

* Cada familia tiene su propio conjunto de patrones válidos y su esquema de codificación (incluyendo el número de bits de redundancia).
* Por ejemplo:
  + **16h5**: Etiquetas con 16 bits de datos y 5 bits de redundancia.
  + **36h11**: Etiquetas con 36 bits de datos y 11 bits de redundancia.

**Estimacion de pose**

**Qué es la Pose**

La pose se refiere a:

1. **Posición (x, y, z)**: La ubicación de la etiqueta en el espacio tridimensional con respecto a la cámara.
2. **Orientación (roll, pitch, yaw)**: Los ángulos que describen cómo está rotada la etiqueta en los ejes tridimensionales.

**Pasos en la Estimación de la Pose**

1. **Transformación de Homografía**:
   * La homografía calculada previamente (en la verificación de cuadrados) proporciona una correspondencia entre los puntos en 2D de la imagen (coordenadas de píxeles) y los puntos en 3D de la etiqueta en el espacio del mundo real.
   * Esta transformación se utiliza como base para calcular la pose.
2. **Modelo de Cámara**:
   * Se utiliza un modelo de cámara conocido (intrínsecos), que incluye:
     + Longitud focal (fx, fy).
     + Punto principal (cx, cy).
     + Distorsión radial y tangencial (en algunos casos).
   * Estos parámetros son necesarios para mapear las coordenadas de la imagen (2D) al espacio 3D.
3. **Resolución del Problema PnP (Perspective-n-Point)**:
   * El problema de PnP consiste en determinar la posición y orientación de un objeto 3D conocido (la etiqueta) a partir de sus proyecciones 2D en una imagen.
   * Se resuelve estableciendo correspondencias entre:
     + Los puntos en el espacio 3D de la etiqueta (corners físicos de la etiqueta).
     + Los puntos proyectados en la imagen 2D (corners detectados).
4. **Cálculo de la Pose**:
   * Se calculan dos matrices principales:
     + **Matriz de Rotación (R)**: Describe cómo está orientada la etiqueta.
     + **Vector de Translación (T)**: Indica la posición de la etiqueta respecto a la cámara.
5. **Validación**:
   * Se valida la pose calculada verificando la coherencia geométrica entre los puntos detectados en 2D y los puntos proyectados desde el espacio 3D.
6. **Salida**:
   * La pose se devuelve como:
     + **Posición (x, y, z)** en unidades reales (por ejemplo, metros).
     + **Orientación (roll, pitch, yaw)** en grados o radianes.

**Archivos Clave para la Estimación de la Pose**

1. **apriltag\_pose.c**:
   * Implementa las funciones principales para calcular la pose.
   * Contiene los algoritmos que resuelven el problema de PnP.
2. **matd.c y matd.h**:
   * Proveen herramientas matemáticas para manejar matrices (rotación, transformación, etc.).
   * Son utilizadas para realizar cálculos necesarios en la estimación de pose.
3. **math\_util.h**:
   * Contiene funciones auxiliares para cálculos matemáticos, como proyecciones y transformaciones geométricas.

**Modelo Matemático del Proceso**

1. **Homografía**:
   * La relación entre puntos 3D y su proyección en 2D está dada por:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. **Resolución del Problema PnP**:
   * Dados varios puntos 3D y sus proyecciones 2D , se resuelve para R y T.
2. **Proyección de Puntos**:
   * Una vez conocida la pose, cualquier punto en el espacio 3D puede proyectarse a la imagen 2D para verificar la coherencia.

**Flujo del Algoritmo**

1. **Entrada**:
   * Puntos detectados en la imagen (2D).
   * Coordenadas conocidas de la etiqueta (3D).
   * Parámetros intrínsecos de la cámara.
2. **Calcular Homografía**:
   * Usar correspondencias 2D-3D para encontrar la transformación inicial.
3. **Resolver PnP**:
   * Usar algoritmos como el de **Levenberg-Marquardt** para optimizar RRR y TTT.
4. **Validar Pose**:
   * Verificar si los puntos proyectados en la imagen coinciden con los detectados.
5. **Salida**:
   * Pose final (x,y,z,roll,pitch,yawx, y, z, roll, pitch, yawx,y,z,roll,pitch,yaw).

Repositorio:

<https://github.com/raspiduino/apriltag-esp32> [Repositorio]

<https://chaitanyantr.github.io/apriltag.html> [generador AprilTag]