

1 BT2 - Estimación robusta de parámetros

En esta sección tratamos la detección de rectas y otras formas geométricas características en el ámbito de las señales de tráfico, candidatas idóneas sobre el papel para ser detectadas mediante técnicas de estimación robusta de parámetros vistas en la asignatura. Comenzamos presentando el subconjunto de imágenes seleccionado para exemplificar los métodos que aquí explicamos, mostrando el resultado la aplicación del pipeline propuesto en la sección anterior para la detección de líneas de borde. Además, proponemos otro método de detección de bordes muy enfocado a este tipo de imágenes basado en la conversión del espacio de color RGB a HSV y la posterior umbralización del canal de saturación. A continuación, exponemos nuestra implementación en C para la detección de líneas y circunferencias con RANSAC y mostramos los resultados obtenidos. Finalmente, presentamos una serie de conclusiones y posibles líneas de trabajo futuras.

1.1 Dataset y preprocessamiento

Del conjunto de imágenes expuesto [1] hemos seleccionado las siguientes 11 imágenes:



Figure 1: Imágenes seleccionadas del conjunto ZOD para la detección de líneas de borde.

Como vemos, son imágenes en entornos urbanos con múltiples señales de tráfico. Incluimos una imagen con señales de obras (Imagen 4), una sin edificios (Imagen 7), una con hojas y muchas sombras (Imagen 8) y una nocturna (Imagen 10) para contar con cierta variedad.

A estas imágenes les aplicamos el siguiente preprocesamiento:

- Corrección de la distorsión de Kannala Brandt.
- Reescalado por un factor de 0'5.
- Aplicación de un filtro de medianas de 5x5 para reducir el ruido.

El escalado se ha llevado a cabo para agilizar la salidas en el notebook. Ha sido llevado a cabo mediante la función `cv2.resize()` de OpenCV con el método de interpolación `INTER_AREA`, ideal para reducciones. El resultado de este preprocesamiento se muestra en la fig. 2. Las dimensiones finales de las imágenes son de 1924x1084 píxeles.



Figure 2: Imágenes preprocesadas del conjunto ZOD para la detección de líneas de borde.

1.2 Detección de bordes

Para la detección de bordes hemos implementado dos métodos diferentes. El primero de ellos es el ya explicado en la sección anterior basado en la detección de bordes de Canny. La segunda aproximación es una segmentación por color seguida por una dilatación. En ambos casos, vamos a hacer una transformación previa de la imagen al espacio de color HSV.

1.2.1 Espacio de color HSV

El espacio de color HSV (Hue, Saturation, Value) [2] es un modelo de color que representa los colores de una manera más intuitiva que el espacio RGB. En este modelo cada canal representa:

- Hue (Matiz): es una escala circular que representa el tipo de color (rojo, verde, azul, etc.) y se mide en grados (0-360°), aunque en OpenCV se escala a un rango de 0 a 179.
- Saturation (Saturación): representa la intensidad del color, donde 0 es un tono de gris y 255 es el color completo.
- Value (Valor): representa el brillo del color. Es el clásico concepto de luminosidad, donde 0 es negro y 255 es el color más brillante.

Las señales de tráfico, al tener que ser fácilmente detectables por el ojo humano en diversas condiciones de iluminación y clima, se diseñan con colores brillantes y contrastantes. Esto las hace ideales para ser segmentadas en el espacio de color HSV, donde los cambios en la iluminación afectan menos a la percepción del color. Identificamos que todas las principales señales cuentan con el color rojo o el azul, incluso aquellas que encontramos en obras viales que sustituyen el color blanco por amarillo.

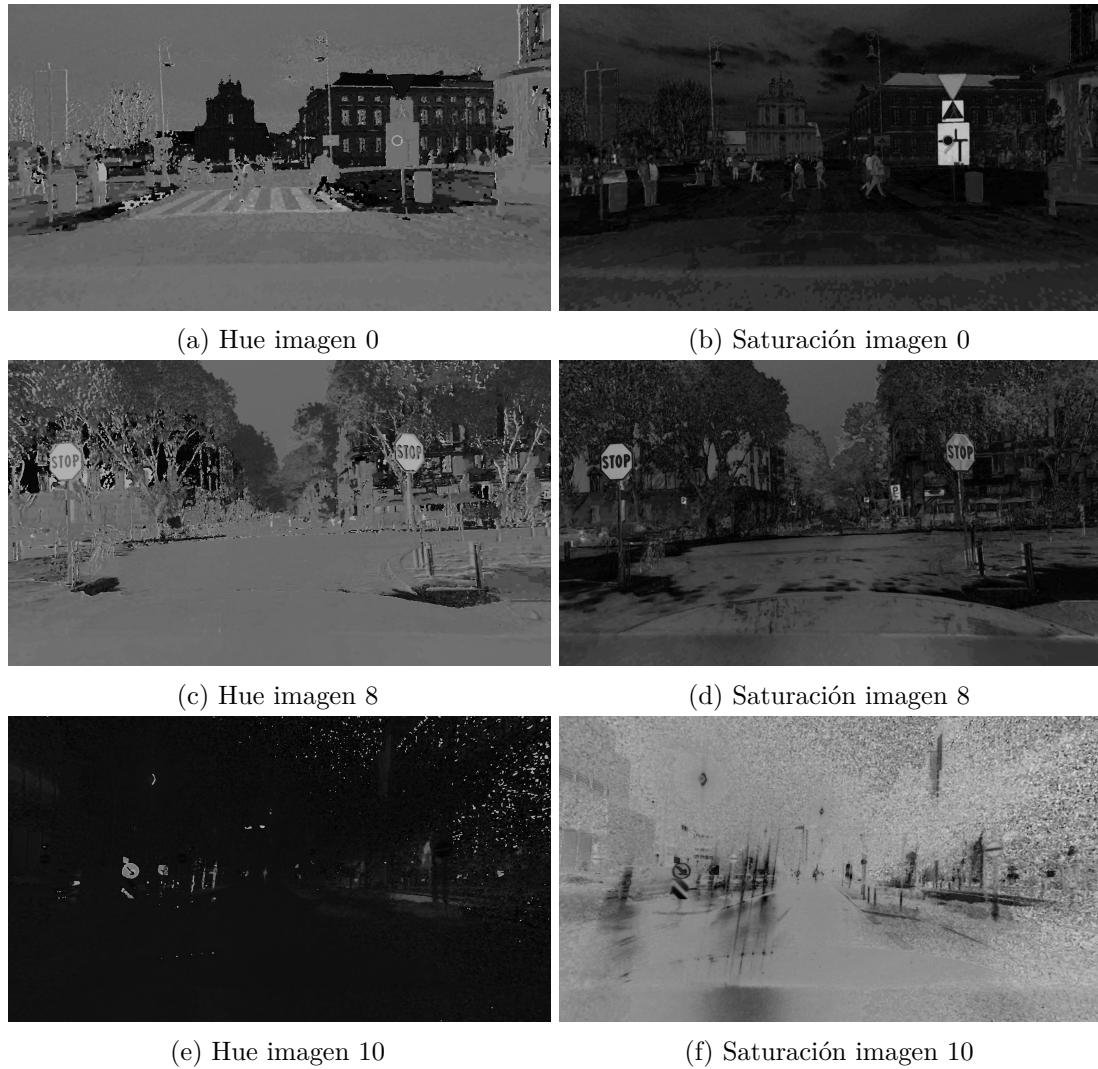


Figure 3: Canales de Hue y Saturación para las imágenes 0, 8 y 10.

En la escala de matiz, el rojo se corresponde con valores cercanos a 0 y el azul a 120. Como vemos en la fig. 3, el canal de saturación resalta mucho más las señales de tráfico que el canal de matiz. Es una excepción la imagen nocturna (Imagen 10), donde el canal de saturación apenas resalta las señales debido a la baja iluminación y a la presencia de luces artificiales que

distorsionan los colores. El canal de matiz, por otro lado, es prometedor en la imagen 10 al resaltar las señales de la zona propiamente iluminada ya que estas están fabricadas con un material que permite reflejar la luz de manera más efectiva durante la noche.

1.2.2 Operador de Canny en espacio HSV

Aplicando el pipeline de detección de bordes basado en Canny explicado en la sección anterior, pero esta vez en el espacio HSV, acabamos concluyendo que una ponderación de los canales H y S con pesos 0.25 y 0.75 respectivamente daba los mejores bordes. El proceso completo es el siguiente:

1. Convertir la imagen de RGB a HSV.
2. Calcular la imagen ponderada: $I_{weighted} = 0.25 \cdot I_H + 0.75 \cdot I_S$.
3. Suavizar $I_{weighted}$ con un filtro Gaussiano de $\sigma = 0.4$.
4. Aplicar el operador de Canny a $I_{weighted}$ con umbrales 150 y 250.

El resultado de este proceso se muestra en la fig. 4.

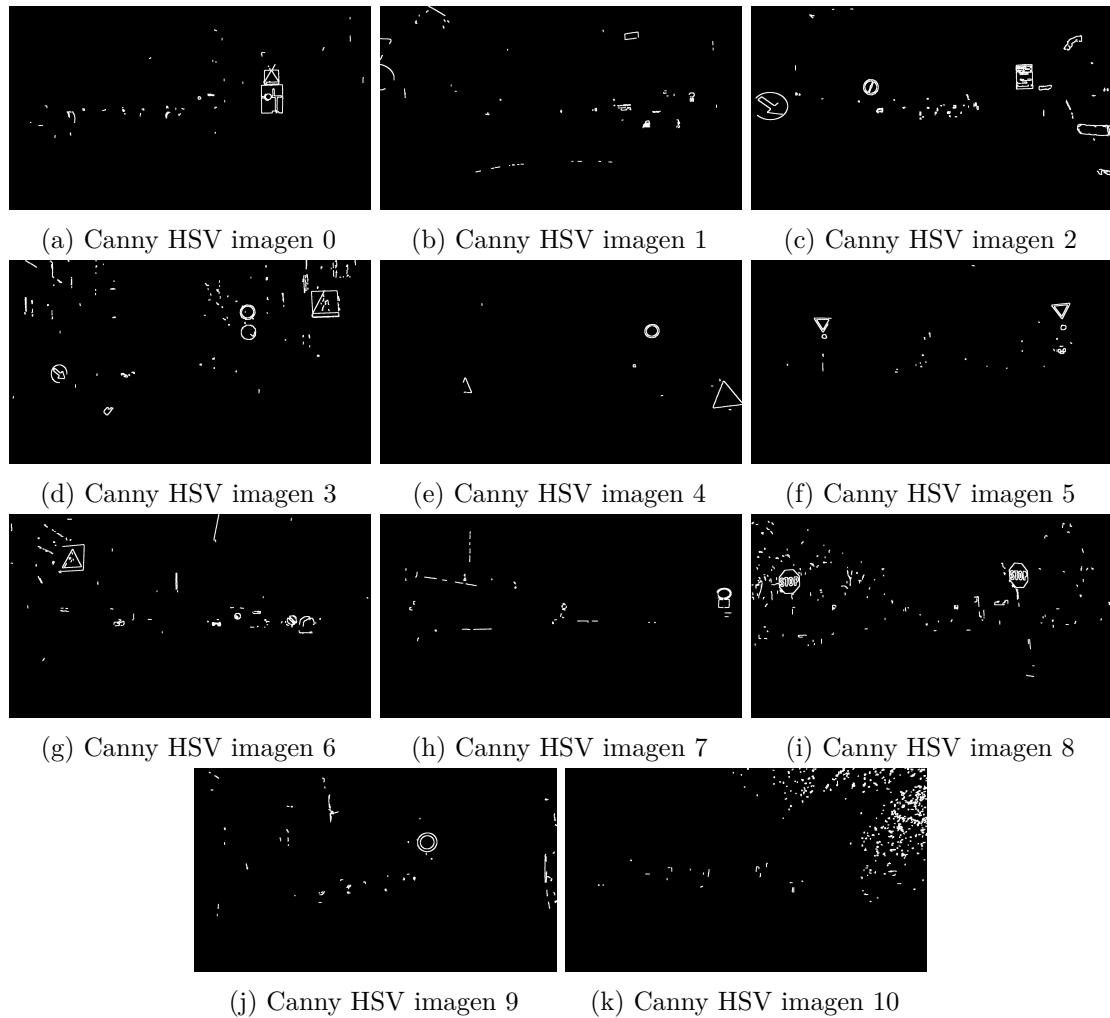


Figure 4: Resultados del operador de Canny en espacio HSV para las imágenes seleccionadas. Los bordes se muestran dilatados para mayor visibilidad.

1.2.3 Segmentación por color y dilatación

El segundo método de detección de bordes que hemos implementado se basa en un segmentación por color en el espacio HSV. Después de identificar que las señales de tráfico se caracterizan por colores azules y rojos muy saturados, hemos optado por umbralizar segmentar la imagen a partir de los rangos de matiz [165, 180] y [0, 15] para el rojo y [90, 130] para el azul. Además, aplicamos una umbralización en el canal de saturación para eliminar colores poco saturados, estableciendo un umbral en 90. El proceso completo es:

1. Convertir la imagen de RGB a HSV.
2. Crear una máscara binaria donde se cumplan las siguientes condiciones:
 - El canal de saturación sea mayor que 90.
 - El canal de matiz esté en los rangos [165, 180] o [0, 15] (rojo) o [90, 130] (azul).
3. Eliminar las regiones pequeñas mediante un filtrado de componentes conexas con un umbral de 50 píxeles.
4. Aplicar una dilatación circular de un pixel de grosor y sustraer la máscara original para obtener los bordes.

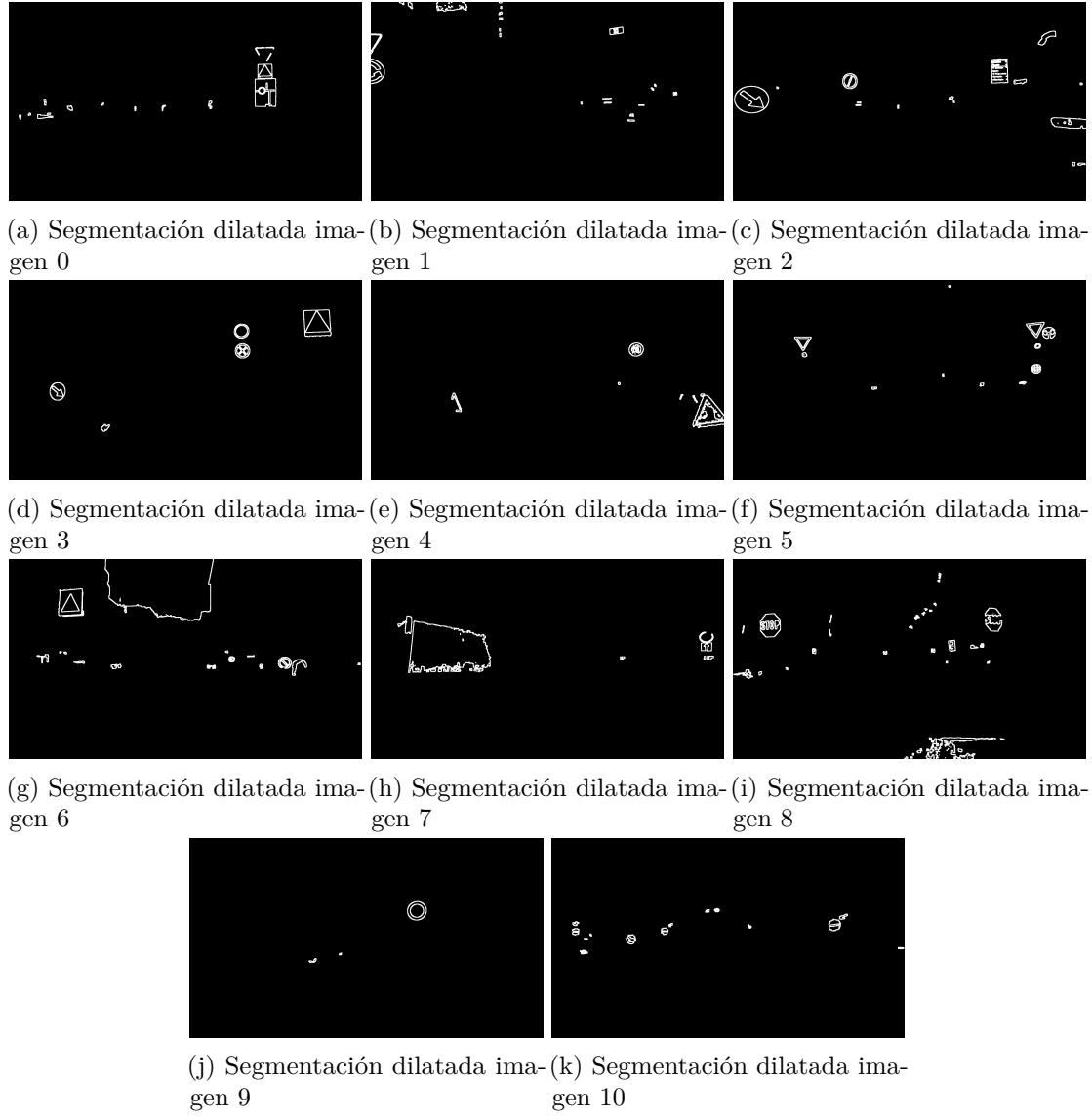


Figure 5: Resultados de la segmentación por color y dilatación para las imágenes seleccionadas. Los bordes se muestran dilatados para mayor visibilidad.

1.3 Comparación de resultados

Aunque ambos métodos ofrecen resultados aceptables, hemos observado que la segmentación por color y dilatación tiende a ser más robusta en condiciones de iluminación variables y en presencia de ruido. Lo que nos hace decantarnos por este método es el buen funcionamiento en la imagen nocturna (Imagen 10), donde el operador de Canny en espacio HSV no logra detectar los bordes de las señales de tráfico debido a la baja iluminación y al ruido introducido por las luces artificiales. Esta segunda técnica también parece generar menos falsos positivos en ciertas imágenes con fondos complejos.

1.4 Detección de features con RANSAC

Para la detección de features en imágenes hemos creado una clase en Python llamada `FeatureExtractor`. En particular, la clase sirve de interfaz para la detección de líneas, segmentos y circunferencias mediante las ideas de RANSAC.

1.4.1 Detección de rectas

Una vez contamos con una imagen de bordes, la propuesta de RANSAC es la siguiente:

1. Seleccionar aleatoriamente dos puntos de borde para definir una línea candidata.
2. Calcular la distancia perpendicular de todos los puntos de borde a esta línea.
3. Contar el número de inliers, es decir, puntos cuya distancia a la línea es menor que un umbral predefinido.
4. Repetir los pasos 1-3 un número fijo de veces o hasta que se alcance un número suficiente de inliers.
5. Seleccionar la línea con el mayor número de inliers como la mejor estimación.
6. Opcionalmente, refinar la línea utilizando todos los inliers mediante un ajuste por mínimos cuadrados.

Nosotros hemos decidido implementar nuestra versión en C, la cual puede ser consultada en `src/dgst/ffi/ransac.c` bajo el nombre de `ransac_line_fitting`. El procedimiento implementado es el recién descrito. Los parámetros que necesita son:

- input: Puntero a un array de puntos de borde.
- width, height: Dimensiones de la imagen.
- distance_threshold: Umbral de distancia para considerar un punto como inlier.
- max_iterations: Número de iteraciones de RANSAC.
- max_lsq_iterations: Número de iteraciones para el ajuste por mínimos cuadrados.
- min_inlier_count: Número mínimo de inliers para aceptar una línea.

El resultado es una recta representada en forma $ax + by + c = 0$ mediante los parámetros a, b y c (en caso de que se haya encontrado alguna línea). Adicionalmente, como funcionalidad en python incluimos la posibilidad de eliminar los inliers de la recta detectada para facilitar la detección de otras líneas en iteraciones posteriores.

Aplicando este método a nuestras imágenes de bordes exigiendo un mínimo de 200 inliers en la primera iteración y un mínimo de 50 tras aplicar RANSAC hasta 150 veces obtenemos imágenes como las mostradas en la fig. 7.

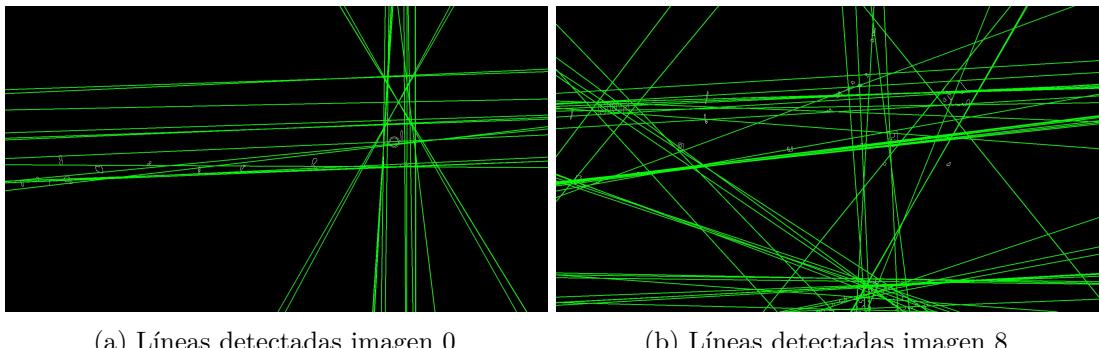


Figure 6: Resultados de la detección de líneas mediante RANSAC en las imágenes 0 y 8.

El algoritmo funciona bien detectando las rectas, pero resulta ser muy sensible al ruido. En cuanto hay una cantidad no demasiado alta de nubes de puntos alineados las rectas encontradas

se ajustan a este ruido. Para solucionar esto, podríamos aumentar el número mínimo de inliers requerido, pero a costa de sacrificar algunas rectas más pequeñas.

Debido a que las señales de tráfico ocupan una sección relativamente pequeña de la imagen, proponemos aplicar RANSAC mediante un algoritmo de ventana deslizante. De esta forma, permitimos eliminar esas rectas que se ajustan al ruido y detectar bordes más pequeños al poder reducir el número mínimo de inliers necesarios para determinar una recta. Para ello, utilizamos un nuevo método extractor de características llamado `windowed_ransac_line_fitting`.

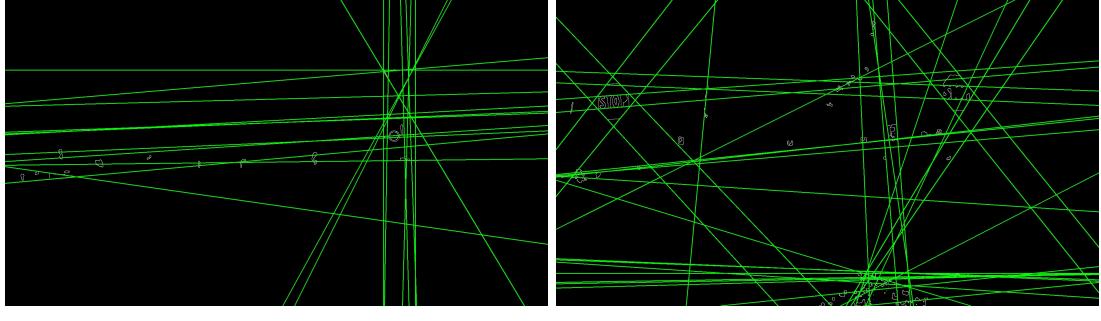


Figure 7: Resultados de la detección de líneas mediante RANSAC en ventana deslizante en las imágenes 0 y 8.

Como vemos, a pesar de utilizar en este caso un umbral de detección de inliers mayor (1.4 frente a 0.7 en el caso anterior) y un requerimiento menor de inliers por iteración, el método de ventana deslizante consigue detectar las rectas relevantes de la imagen sin verse tan afectado por el ruido.

1.4.2 Detección de segmentos

Otro problema de la baja ocupancia de las señales es que las rectas cubren innecesariamente toda la imagen. Sería ideal limitar la expresión de las mismas a los segmentos donde realmente hay señales. Es por esto que proponemos un método para inferir segmentos a partir de las rectas detectadas. El procedimiento, implementado mediante el método `get_line_support` de la clase FeatureExtractor, es el siguiente:

1. Se proporciona una recta en forma $ax + by + c = 0$, desactivando la eliminación de inliers tras la detección.
2. Se reobtienen los inliers de la línea en la imagen de bordes.
3. Los inliers se proyectan ortogonalmente sobre la recta para obtener sus coordenadas 1D.
4. Se calcula la densidad de inliers a lo largo de la línea.
5. Se identifica el segmento maximal con densidad que supere un umbral dado.

Para llevar a cabo el último paso, hemos utilizado una versión adaptada del Algoritmo de Kadane [kadane], un algoritmo de programación dinámica que resuelve el problema **Maximum Subarray Problem** de encontrar un subarray de suma máxima. Los pasos de esta adaptación son, partiendo de una serie de puntos en 1D (proyecciones sobre la recta):

1. Se establece un umbral de densidad μ (número de inliers por unidad de longitud).
2. Se construye un array auxiliar A , de forma que $A[i]$ representa la contribución de densidad del i -ésimo intervalo $A_i = 1 - \mu \times (p_{i+1} - p_i)$

3. Se inicializan las variables:

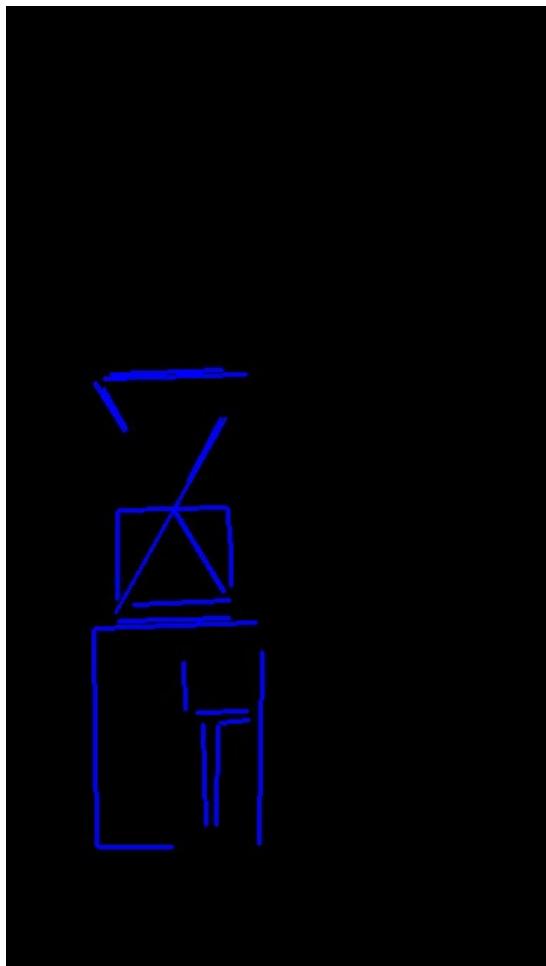
- `max_sum = 0`
- `current_sum = 0`
- `start_index = 0`
- `best_start = -1`
- `best_end = -1`

4. Se recorren los elementos del array A guardando las sumas acumuladas:

- `current_sum += A[i]`
- Si `current_sum > max_sum`:
 - `max_sum = current_sum`
 - `best_start = start_index`
 - `best_end = i`
- Si `current_sum < 0`:
 - `current_sum = 0`
 - `start_index = i + 1`

5. Se devuelve el segmento `[best_start, best_end]`.

Una de las bondades de este algoritmo es que funciona en tiempo lineal con respecto al número de inliers. Podemos verlo en acción sobre una región de la imagen 0 en la fig. 8.



(a) Segmentos detectados imagen 0

Figure 8: Resultados de la detección de segmentos mediante RANSAC en las imagen 0.