Universidad Rey Juan Carlos

URJC Máster Universitario en Visión Artificial

Documento de Diseño

6 de abril de 2022

Grupo E

Alumnos:

David Correas Oliver
Elena Giraldo del Viejo
Juan Manuel Molina Maza

Índice

1. Introducción

En este documento se da una descripción del diseño implementado para el programa *TrafficDetector*, que se puede encontrar en el repositorio `https://github.com/AIVA2022TeamE/AIVA_2022-ImagenesAereas`.

El programa *TrafficDetector* recibe una carpeta de imágenes de 5000x5000 y es capaz de marcar los coches que aparecen en cada imagen. Para realizar la detección se divide cada imagen en imágenes más pequeñas (Figura 1) de 250x250 píxeles y se aplica un filtro de color que segmenta las zonas pertenecientes a carreteras (Figura 2). De esta manera, quedan unas pequeñas manchas blancas que representan los coches que circulan por esa carretera. Después de tratar las imágenes, se filtran aquellas manchas blancas que es más probable que sean coches y se dibuja un rectángulo de color rojo a su alrededor (Figura 3).

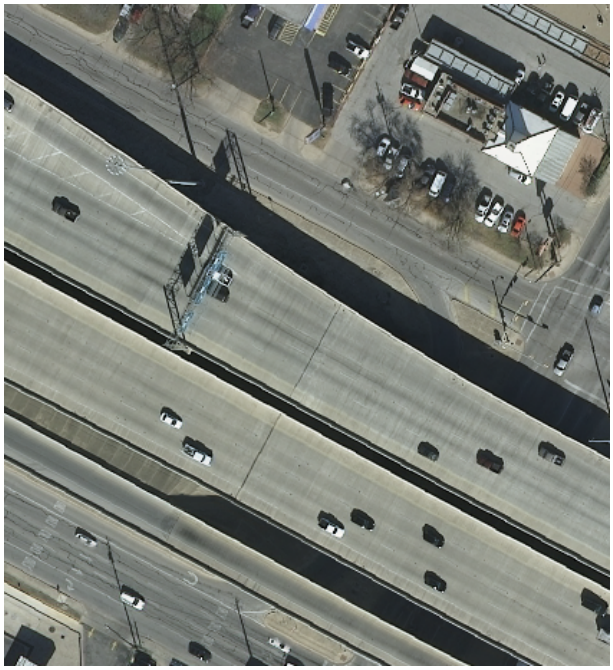

Figura 1: Imagen parte del input

La detección no es muy buena debido al pequeño tamaño de los coches que aparecen en la imagen. También hay bastante falsas detecciones cerca de sombras de gran tamaño como sombras de edificios o árboles, ya que tienen un color similar al gris de la carretera. También se encuentran problemas para detectar coches con poca separación entre ellos, como los coches que se encuentran aparcados. Como punto bueno, la detección es bastante rápida.

Figura 2: Máscara de la imagen

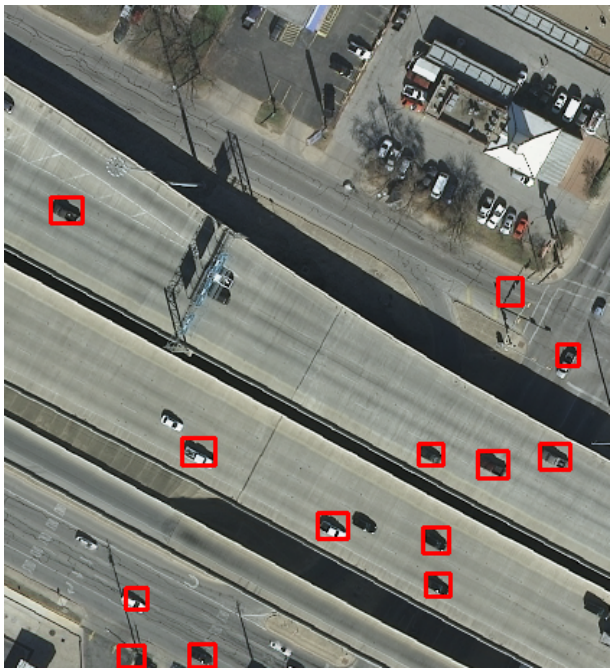

Figura 3: Detección de los coches

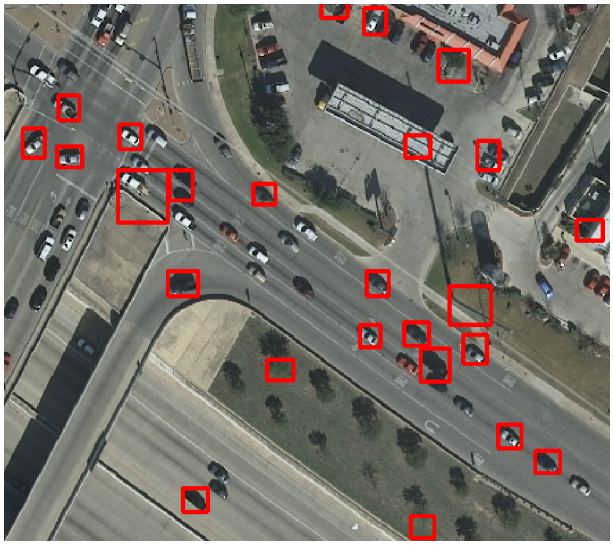

Figura 4: Ejemplo 2

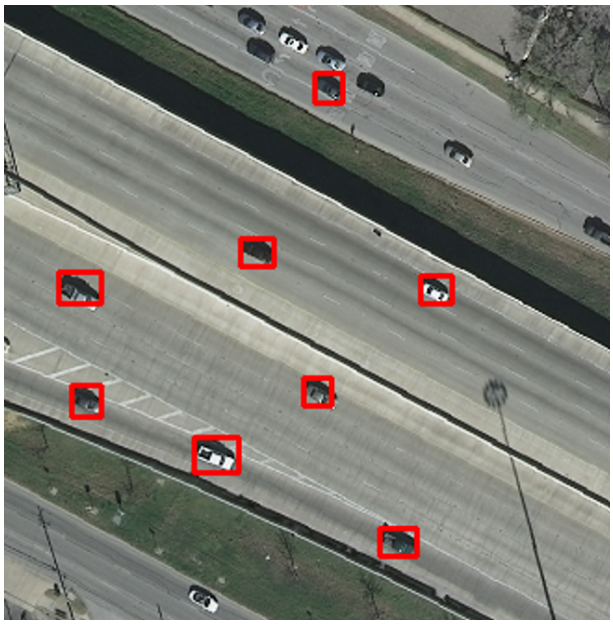

Figura 5: Ejemplo 3

2. Diseño de la solución

En esta sección se muestran algunos diagramas con el fin de mejorar la comprensión de la solución implementada.

2.1. Diagrama de clases

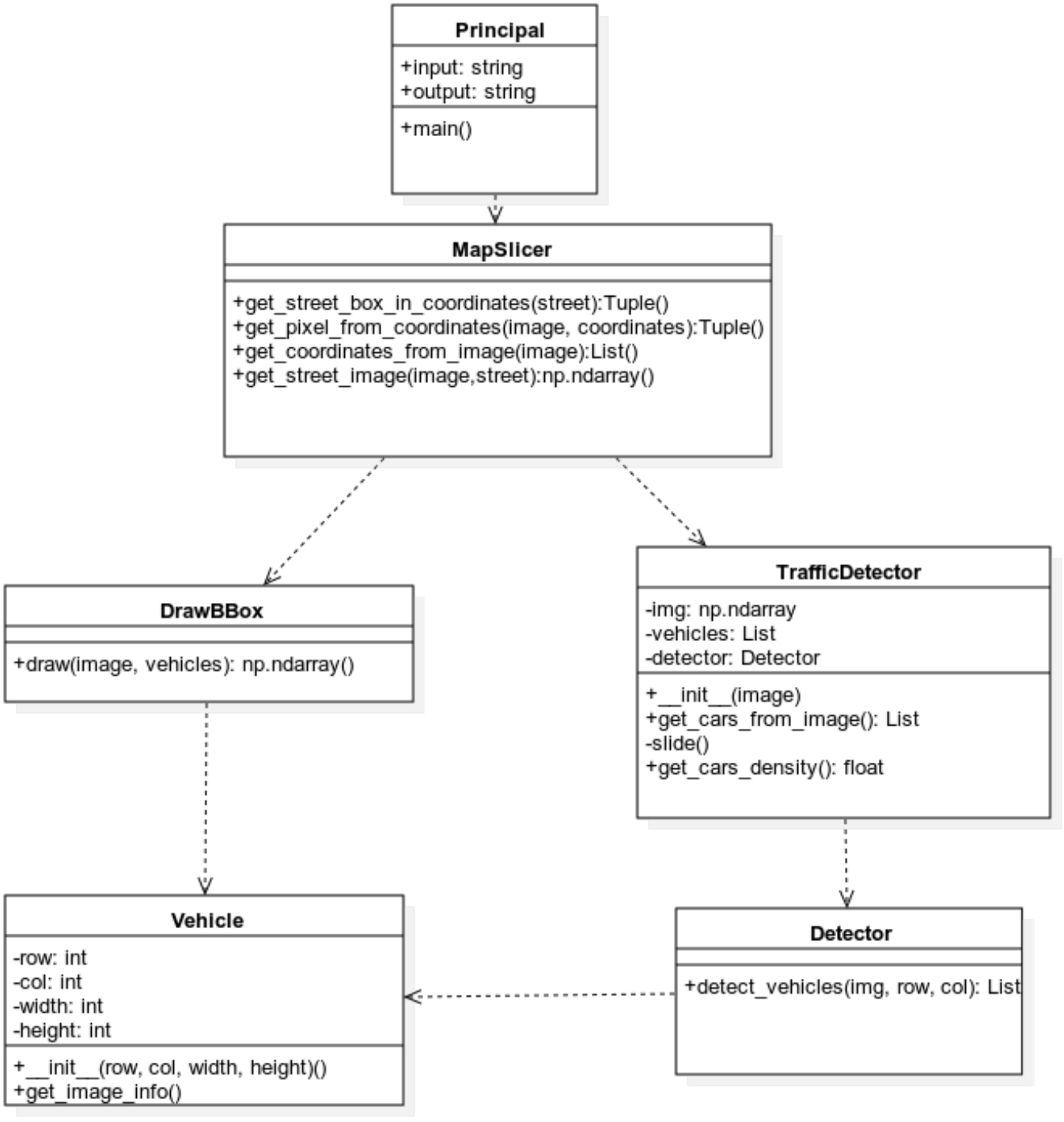

Figura 6: Diagrama UML estático de clases

2.1.1. Programa

La clase Programa contiene el programa principal y es el que se debe ejecutar. Tiene dos atributos:

- *input*: recibe la dirección de la carpeta donde se encuentran las imágenes.
- *output*: recibe la dirección de la carpeta donde se guardarán las imágenes con las detecciones.

Además, tiene una función: *main*. Esta función se encarga de, por cada imagen de la carpeta contenedora, realizar la detección y llamar a la clase TrafficDetector para obtener los coches detectados. Una vez se han obtenido los coches, se llama a la clase DrawBBox para dibujar una *bounding box* sobre cada coche detectado.

2.1.2. MapSlicer

La clase MapSlicer se encarga de recortar las imágenes aéreas del cliente. No contiene atributos y los métodos son los siguientes:

- *get_street_box_in_coordinates*: Método que recibe el nombre de una calle y devuelve las coordenadas en latitud y longitud del límite inferior izquierdo y el límite superior derecho. Usa la API de *Nominatim* para obtener la información.
- *get_pixel_from_coordinates*: Método que recibe unas coordenadas en latitud y longitud y una imagen(con sus coordenadas en los metadatos *.tif*) y devuelve el píxel en altura y en anchura relativa a dicha imagen.
- *get_coordenates_from_image*: Método que recibe una imagen *.tif* y devuelve las coordenadas en latitud y longitud descritas en sus metadatos.
- *get_street_image*: Método que recibe una imagen y un nombre de una calle en dicha imagen y devuelve el segmento de la calle en dicha imagen.
- *get_street_surface*: Método que recibe una imagen y devuelve el número de píxeles que pertenece a una calle.

2.1.3. TrafficDetector

La clase TrafficDetector se encarga de realizar la detección de vehículos sobre una imagen. Tiene tres atributos privados:

- *img*: un numpy array que contiene la imagen donde se va a realizar la detección.
- *vehicles*: una lista (al principio vacía) con elementos de tipo Vehículo que contiene todos los

vehículos detectados.

- *detector*: un elemento de tipo Detector que será el encargado de hacer la detección.

Además, cuenta con tres funciones además de la función *__init__* para inicializar un objeto:

- *get_cars_from_image*: Función pública que devuelve una lista con los vehículos detectados sobre una imagen.
- *slide*: Función privada que se encarga de dividir la imagen recibida en imágenes más pequeñas de 250x250 píxeles y de llamar a la clase Detector para que detecte en donde se encuentran los vehículos de esa imagen más pequeña.
- *get_cars_density*: Función pública que calcula el número de vehículos entre el tamaño de la imagen.

2.1.4. Detector

La clase Detector se encarga de realizar la detección de coches sobre la imagen de 250x250 píxeles que recibe de la función *slide()*. Tiene una única función pública llamada *detect_vehicles* que recibe la imagen de 250x250 píxeles y la primera fila y primera columna correspondiente en la imagen grande. Es decir, si la mini imagen que se pasa es la zona que va de la fila 500 a la 750 (250 píxeles de alto) y de la columna 250 a la 500 (250 píxeles de ancho) el valor de *row* será 500 y el de *col* será 250. Esto se hace así ya que la función *detect_vehicles* devolverá una lista con elementos de la clase Vehículos, para los que se necesita saber las coordenadas en las que se encuentra cada uno en la imagen original, y no las coordenadas en la mini imagen.

2.1.5. Vehicle

La clase Vehicle crea un objeto de la clase vehículo. Para ello, cuenta con cuatro atributos privados:

- *row*: un entero que indica la fila mínima en la que se encuentra el vehículo en la imagen.
- *col*: un entero que indica la columna mínima en la que se encuentra el vehículo en la imagen.
- *width*: un entero que indica la anchura del vehículo.
- *height*: un entero que indica la altura del vehículo.

Además, tiene una función pública *get_image_info()* que devuelve el valor de todos los atributos y la función *__init__* para inicializar un objeto.

2.1.6. DrawBBox

La clase DrawBBox se encarga de dibujar una *bounding box* sobre cada vehículo detectado en la imagen. tiene una única función, *draw*, que recibe una imagen y una lista de vehículos y devuelve la misma imagen con rectángulos de color rojo dibujados sobre cada vehículo.

2.2. Diagrama de secuencia

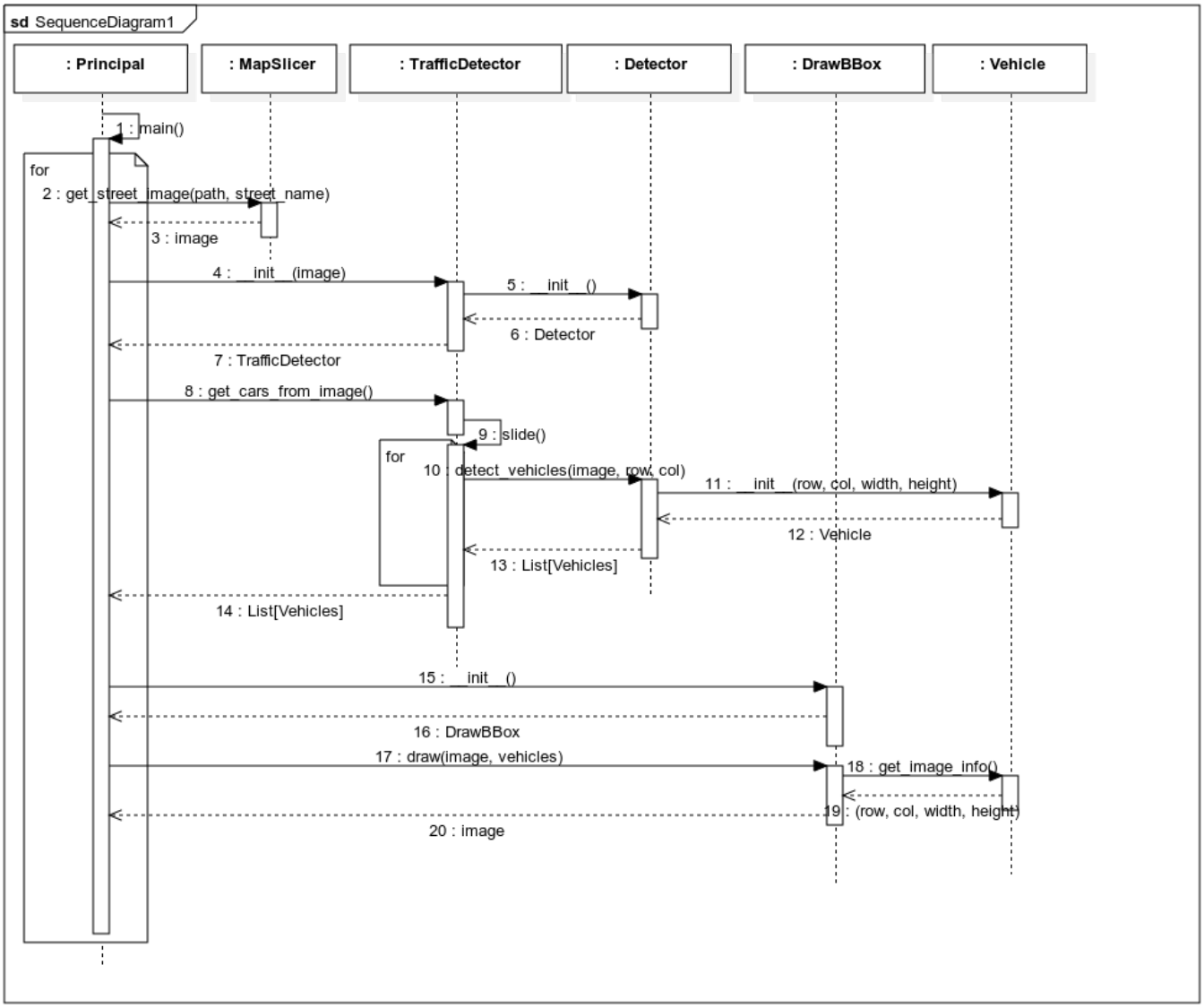

Figura 7: Diagrama UML de Secuencia

El funcionamiento del programa es el siguiente. Primero, se recibe la dirección de la carpeta que

contiene las imágenes y de la carpeta de salida donde se guardarán las detecciones. A continuación, se crea un objeto de la clase Principal con las dos carpetas y se llama a la función *main()*.

La función *main()* realizará un bucle por cada imagen y opcionalmente el nombre de la calle. La imagen se leerá con la función *get_street_image* y devuelve la calle segmentada. A continuación, se crea un objeto de la clase TrafficDetector, que recibe la imagen de la calle (y crea un objeto de la clase Detector) y se llama a la función *get_cars_from_image()*.

La función *get_cars_from_image()* llama a la función *slide()*, que divide la imagen en mini imágenes y por cada una llama a la función *detect_vehicles(image, row, col)*.

La función *detect_vehicles(image, row, col)* de la clase Detector convierte la imagen al espacio de color HSV y establece unos rangos para cada canal. Con estos rangos, se crea una máscara de la imagen. Una vez obtenida la máscara, se realizan una serie de operaciones morfológicas y se detectan posibles líneas rectas (que pertenecen a bordes de la carretera o casas) que se eliminan de la imagen, ya que no son necesarias para la detección. A continuación se obtienen los contornos de la imagen (*findContours*) y aquellos que tengan un tamaño parecido al que tiene un coche en la imagen, se decidirá que es un vehículo y se llama a la función de OpenCV *boundingRect* para obtener las coordenadas y tamaño de ese vehículo.

Por último, todos los vehículos se añaden a una lista que será la que se reciba *main()*.

Una vez obtenida la lista de vehículos, desde la función *main()* se crea un objeto de la clase DrawBBox y se llama a la función *draw(image, vehicles)*. Esta función recorre la lista de vehículos, llama a la función *get_image_info()* de la clase Vehículo para obtener las coordenadas de ese vehículo y, utilizando la función *rectangle* de OpenCV, dibuja en la imagen un rectángulo de color rojo.

Por último, *main()* recibe la imagen modificada con *draw()* y la guarda en la carpeta de salida.

2.3. Algoritmos desarrollados

De la implementación realizada, cabe destacar dos algoritmos: la función *slide* y la función *detect_vehicles*.

2.3.1. Slide

El algoritmo de slide es un algoritmo implementado en la clase TrafficDetector que se encarga de dividir la imagen de entrada en lotes de imágenes más pequeñas. En concreto, se ha establecido que sean de tamaño 250x250 píxeles, ya que es un tamaño de imagen razonable y porque sale un número exacto de imágenes del mismo tamaño.

Como se puede observar en la figura, el primer paso es obtener la anchura y altura de la imagen de entrada. A continuación, se obtiene la primera imagen de 250x250, se detectan los vehículos con

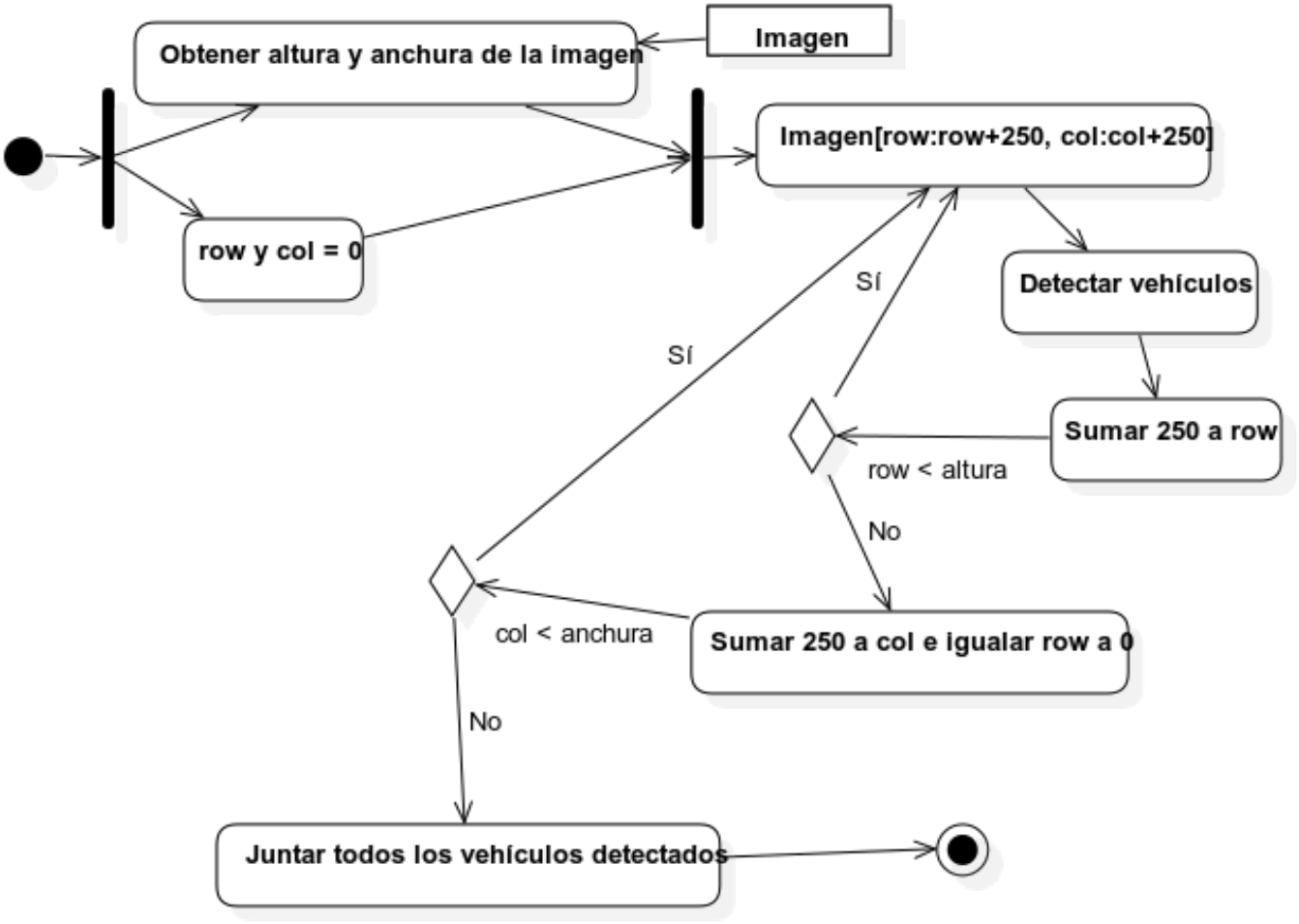

Figura 8: Diagrama de Actividad de *slide*

el algoritmo de detección de vehículos y se pasa a la siguiente imagen, es decir, la que se encuentra a su derecha. En el caso de que no haya más espacio a la derecha, se pasará a obtener la imagen de la siguiente fila. Este bucle anidado se repite hasta que ya no quedan más imágenes en el lote de imágenes.

Por último, todos los vehículos detectados en cada imagen del lote se unen en una sola lista que será la lista de vehículos detectados de la imagen de entrada.

2.3.2. Detectar vehículos

El algoritmo de detección de vehículos está implementado en la clase Detector en la función *detect_vehicles*.

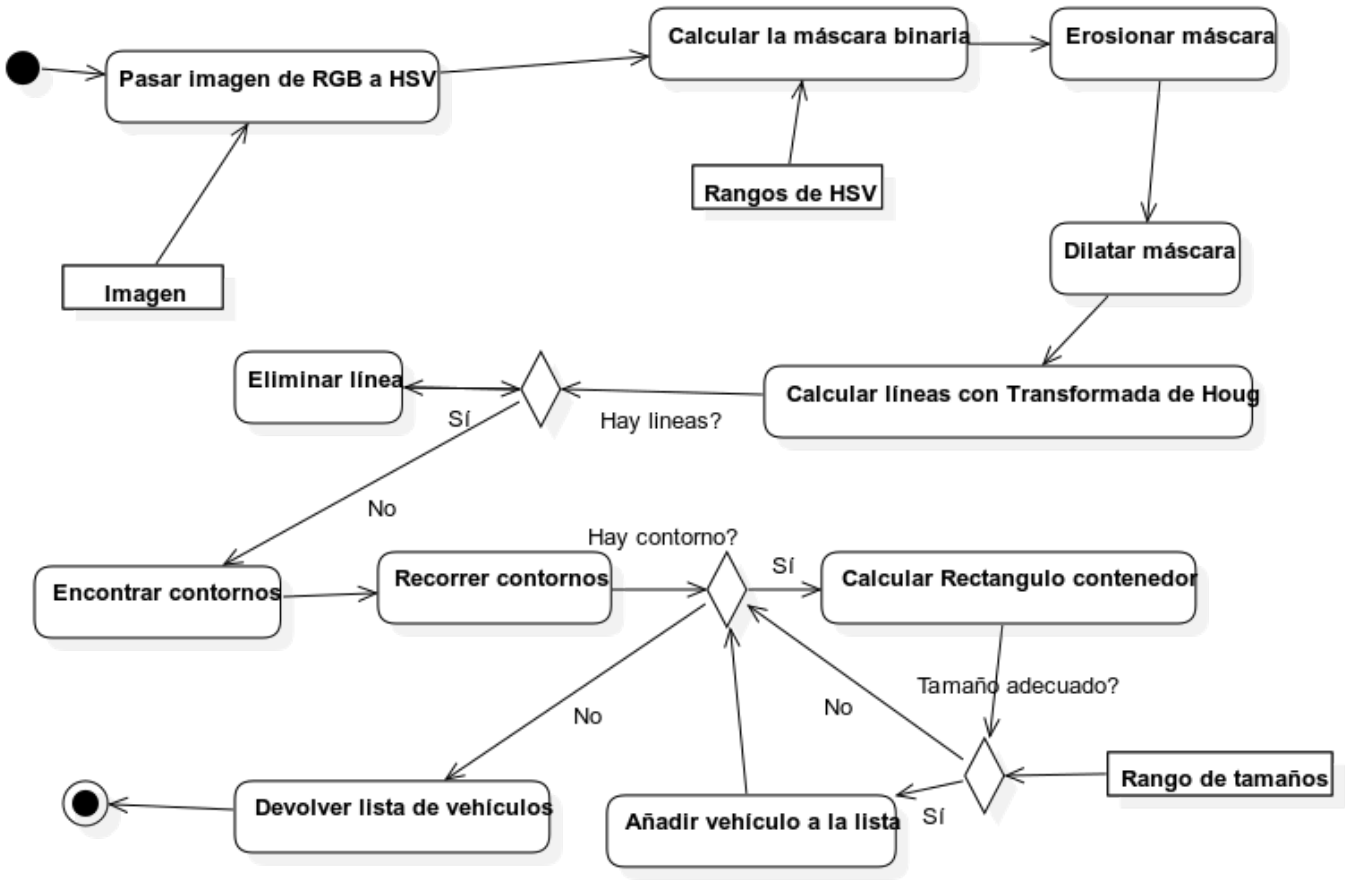

Figura 9: Diagrama de Actividad de *detect_vehicles*

Se ha optado por usar un filtro de color para segmentar los vehículos de la imagen, por lo que el primer paso es convertir la imagen RGB al espacio de color HSV y calcular la máscara binaria estableciendo unos rangos mínimos y máximos para cada canal. Estos rangos están calculados para que aparezcan en color negro las zonas oscuras y grises de la imagen, por lo que las carreteras aparecen en negro, y los vehículos que son de un color más claro, aparecen en blanco. Para mejorar el resultado de la binarización se aplican unas operaciones de erosión y dilatación que eliminan el posible ruido que se ha podido obtener. Sin embargo, aparecen muchas líneas de color blanco que representan los bordes de la carretera o edificios, por lo que además se utiliza la Transformada de Hough para detectar estas líneas y así ir eliminándolas todas coloreándolas de negro.

Una vez que se ha terminado de tratar la máscara, se utiliza la función *findContours* de OpenCV para calcular los contornos que aparecen en la máscara. Por cada contorno, se calcula el rectángulo que contiene a ese contorno (*BoundingRect*) y, si la altura y anchura del rectángulo están en un rango aceptable (es decir, tiene un tamaño que podría coincidir con el tamaño de un coche) se crea un objeto de la clase Vehículo con los datos obtenidos del rectángulo y se añade a la lista de

vehículos. Una vez que ya se han evaluado todos los contornos, se devuelve la lista de vehículos.