Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores
(Computer Engineering Academic Area)

Documentación de UrbanPathGen
(Documentation of UrbanPathGen)

Allan Calderón Quirós

Índice general

Índice de figuras

Capítulo 1

UrbanPathGen

UrbanPathGen es un módulo de software diseñado para la generación autónoma de trayectorias en entornos urbanos desconocidos utilizando drones o UAV´s equipados con cámaras. Este módulo se basa en redes neuronales y técnicas de aprendizaje profundo para analizar imágenes capturadas en tiempo real y generar rutas seguras y eficientes para la navegación autónoma de drones en áreas urbanas.

UrbanPathGen presenta una interfaz agnóstica del hardware que enfatiza la flexibilidad y la compatibilidad con diversas plataformas de drones y hardware relacionado. Esta interfaz se compone de un conjunto de métodos que permiten una implementación de los controles de movimiento según la plataforma de hardware utilizada.

1.1 Descripción general

En los últimos años, el uso de drones o UAV´s(Unmanned Aerial Vehicle) ha experimentado un crecimiento exponencial debido a su versatilidad y capacidad para acceder a áreas de difícil acceso [1]. Estos dispositivos aéreos no tripulados se han convertido en herramientas indispensables en diversas aplicaciones, como inspecciones industriales, vigilancia, entrega de paquetes y cartografía, entre otros [2]. Uno de los desafíos clave en el campo de los UAV´s es la capacidad de navegar de manera autónoma en entornos urbanos desconocidos, donde la detección y reconocimiento de rutas se convierte en un factor crítico [11].

UrbanPathGen se centra en la generación de trayectorias en entornos urbanos no conocidos a través de imágenes mediante redes neuronales. Se plantea utilizar un conjunto de drones que explorarán un territorio desconocido de manera autónoma. La red neuronal se encargará de tomar las imágenes capturadas por la cámara de los drones y realizar un análisis de profundidad. Posteriormente, el resultado se analizará para determinar las posibles rutas que el dron puede tomar. El objetivo es que el UAV pueda reconocer y mapear el entorno urbano , generando trayectorias seguras y eficientes para su navegación autónoma.

El uso de redes neuronales para el reconocimiento de rutas en imágenes ha demostrado ser prometedor en investigaciones previas [14]. Estas redes utilizan técnicas de aprendizaje profundo para extraer características relevantes de las imágenes y realizar una clasificación precisa de las rutas [12]. Al entrenar la red neuronal con un conjunto de imágenes , se puede lograr un alto nivel de precisión en la detección y reconocimiento de patrones que permitan identificar rutas en entornos urbanos no conocidos.

Capítulo 2

Descripción del funcionamiento de UrbanPathGen

2.1 *Frameworks* y herramientas de desarrollo

En esta sección se describen las principales herramientas utilizadas para la implementación del sistema.

2.1.1 Ambiente de desarrollo

EL proyecto se desarrolla en el lenguaje de programación Python en el sistema operativo Windows 10 Home, en una computadora MSI GF63 con un procesador Intel® Core™ i7-10750H, una GPU GeForce® GTX 1650 Ti MAX Q y 8GB DDR4 [15].

2.1.2 CUDA

El NVIDIA® CUDA® Toolkit proporciona un entorno de desarrollo para la creación de aplicaciones de alto rendimiento aceleradas por GPU(*Graphics Processing Unit*) [16]. El *toolkit* incluye bibliotecas aceleradas por GPU, herramientas de depuración y optimización, un compilador C/C++ y una biblioteca de tiempo de ejecución para implementación de aplicaciones. En este proyecto, se aprovechará para explotar las capacidades de la GPU, dada su arquitectura SIMD, en la aceleración del tiempo de inferencia de la CNN que realiza la estimación de profundidad. La instalación de CUDA no es obligatoria para el funcionamiento del programa, sin embargo, se recomienda su instalación, ya que el tiempo de inferencia puede verse gravemente afectado, especialmente si el equipo donde se ejecuta no tiene un gran poder computacional. Para su instalación, se pueden seguir las instrucciones en su página oficial [16].

2.2 Diseño general del sistema

En esta sección se detalla el diseño de UrbanPathGen. Se inicia con el diseño general de la arquitectura, y posteriormente de especifica el diseño de cada módulo.

2.2.1 Arquitectura general del sistema

En la figura 1 se muestra el diagrama de segundo nivel de la solución para el sistema de generación de trayectorias. Se observan las entradas y salidas del sistema, que se detallan a continuación:

- Cámaras: Son las entradas del sistema. Proveen la fuente de video para la cual el sistema tomará capturas periódicas y realizará un análisis de profundidad
- Instrucciones de movimiento: Son las salidas del sistema. Una vez realizado el análisis, se envían instrucciones al UAV sobre la dirección en la que debe moverse, según la trayectoria seleccionada.

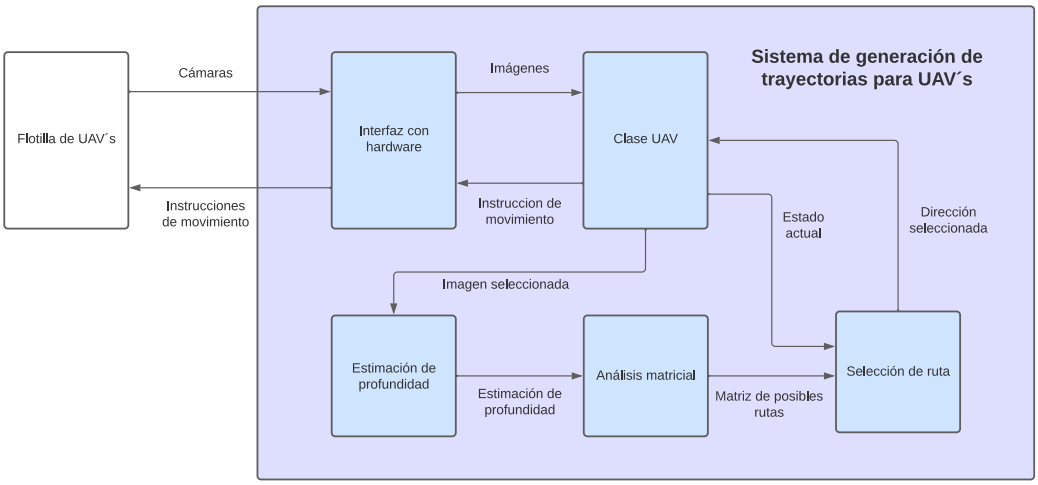

Figura 1: Diagrama de segundo nivel de la solución

2.2.2 Diseño de la clase UAV

La clase UAV corresponde a una abstracción de un dron o UAV en el sistema desarrollado en Python. Esta clase permite modelar las características y capacidades esenciales para el funcionamiento del sistema. El sistema se plantea para ser utilizado con una flotilla con un número variable de drones, por lo que se debe crear una instancia por cada UAV real disponible.

El diagrama de la clase UAV se muestra en la figura 2. En este diagrama, se muestran los siguientes atributos:

- ID: Corresponde al identificador del dron. Debe asignarse en orden ascendente, empezando desde 0.
- current_speed: Corresponde a la velocidad actual del UAV en metros por segundo. Es necesario conocer esta parámetro para la selección de rutas en las que el dron debe aumentar o disminuir su altura.
- field_of_view_x: Representa el campo de visión en grados del dron respecto al eje x. Es requerido para conocer el rango de grados que está siendo capturado por la cámara, para determinar cuantos grados puede debe girar el dron en una dirección.
- field_of_view_y: Representa el campo de visión en grados del dron respecto al eje y. Es requerido para determinar la altura que el dron debe subir o bajar en caso necesario.
- min_height: Es la altura mínima de operación del dron. Esta variable se define según el uso requerido.
- max_height: Corresponde a la altura máxima de operación del dron. Esta variable se define según el uso requerido y las limitaciones de rango del dron.
- current_height: Indica la altura actual a la que se encuentra el dron. Es necesaria para evitar que el dron se salga de los límites establecidos. En el caso de drones Crazyflie, el FlowDeck permite determinarla con su sensor óptico [6].
- resolution_x: Resolución en x de la cámara del UAV. Se requiere para realizar una estimación de la ubicación de las rutas posibles en la imagen.
- resolution_y: Resolución en y de la cámara del UAV. Se requiere para realizar una estimación de la ubicación de las rutas posibles en la imagen.
- video_source: Corresponde a la fuente del video a analizar. Utilizando OpenCV, se puede indicar el ID de la cámara correspondiente a la instancia del dron [9].
- video_type: Corresponde al tipo de video que se está capturando con el parámetro video_source. Se plantean dos tipos existentes, 'video', si la fuente es un archivo de video(para casos de prueba, o configuraciones preexistentes) o 'camera' si la fuente de video es directamente la cámara del UAV.

La clase UAV posee cuatro métodos principales:

- turn(direction): Elige la dirección en la que el dron debe girar para tomar la ruta seleccionada. El rango en el que el dron puede girar corresponde al campo de visión de la cámara del dron, y se define desde $\frac{-field\_of\_view\_x}{2}$ hasta $\frac{field\_of\_view\_x}{2}$, donde $field\_of\_view\_x$ corresponde al campo de visión de la cámara del UAV en x.

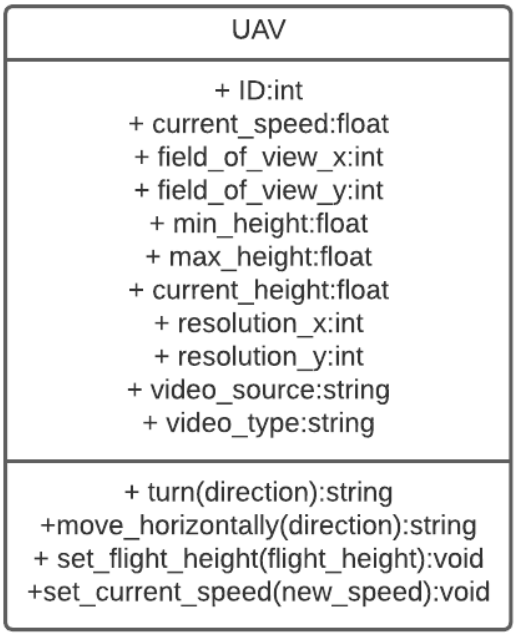

Figura 2: Diagrama de la clase UAV

- move_horizontally(direction): Elige la dirección en la que el dron se moverá (izquierda o derecha) en caso de que no se logre detectar una ruta a tomar.
- set_flight_height(flight_height): Este método permite definir la altura del dron, dentro de los límites establecidos.
- set_current_speed(new_speed): Este método permite definir la velocidad a la que el dron se está moviendo. En caso de que no se pueda determinar mediante hardware, se puede tomar la velocidad media o la velocidad máxima que el dron puede alcanzar. En caso de los drones Crazyflie corresponde a 1 m/s [7]. Esta modificación del parámetro permite calcular el *Worse Case Scenario* de distancia hacia una ruta u obstáculo.

Esta clase posee todos los parámetros necesarios para que el sistema de generación de trayectorias funcione correctamente y se pueda comunicar con el hardware.

2.2.3 Diseño del módulo de estimación de profundidad

El módulo de estimación de profundidad es esencial para el funcionamiento del sistema, ya que permite determinar la profundidad de objetos en una imagen a partir de una sola imagen monocular. Esto permite que a partir de una imagen de entrada, se identifiquen las áreas más profundas, las cuales corresponden a las posibles rutas que el dron puede tomar.

Para lograr la estimación de profundidad, se utiliza MiDaS. MiDaS Depth Estimation es un modelo de aprendizaje automático de Intel Labs para la estimación monocular de la profundidad [19]. Se entrenó con 12 *datasets* y cubre tanto escenarios interiores como exteriores.

MiDaS calcula la profundidad inversa relativa a partir de una sola imagen. Para esto, proporciona múltiples modelos que cubren diferentes casos de uso, desde un modelo pequeño de alta velocidad hasta un modelo muy grande que proporciona la máxima precisión. Los modelos se entrenaron utilizando optimización multiobjetivo para garantizar una alta calidad en una amplia gama de entradas [19].

La selección del modelo MiDaS que permita un mejor ajuste requiere un compromiso ingenieril entre el tiempo de inferencia de la estimación de profundidad, y la precisión obtenida. Un modelo poco preciso causaría que la estimación no sea segura, provocando el riesgo de posibles colisiones. Por el contrario, una estimación precisa pero que requiera mucho tiempo causaría que el UAV no tenga suficiente tiempo de reacción, lo que de igual manera podría ocasionar colisiones.

En la figura 3 se muestra el diseño del módulo de estimación de profundidad. Al principio de la ejecución, se debe seleccionar el modelo de MiDaS que se cargará. Para la versión 3.0, disponible via PyTorch, existen 4 modelos de MiDaS recomendados, con distintas profundidades [13]:

- DPT_Large
- DPT_Hybrid
- MiDas_large
- MiDaS_small

El tiempo de inferencia de cada modelo depende del poder y disponibilidad de procesamiento de la computadora donde se realice la ejecución. A mayor tiempo de inferencia, mayor precisión de la estimación. Sin embargo, una de las principales formas de reducir el tiempo de inferencia es aprovechar el poder de procesamiento del GPU. Para esto, el módulo de selección de dispositivo identifica si el *wrapper* CUDA está disponible, y lo selecciona . En caso de no estar disponible, se selecciona el CPU.

Al realizar pruebas con los distintos modelos recomendados (en un ambiente con CUDA disponible), se obtiene que el modelo MiDaS_small realiza una inferencia rápida (cerca de 0.1 segundos) pero poco precisa, que no permite identificar las posibles rutas en la imagen. Por el contrario, los modelos MiDaS_large y DPT_Large ofrecen estimaciones precisas, donde los limites de los elementos de la imagen se delinean claramente en la estimación de profundidad. Sin embargo, estos modelos podían llegar a un tiempo de inferencia de hasta 14 segundos, lo cual en tiempo de vuelo no permitiría que el UAV tenga suficiente margen de maniobra.

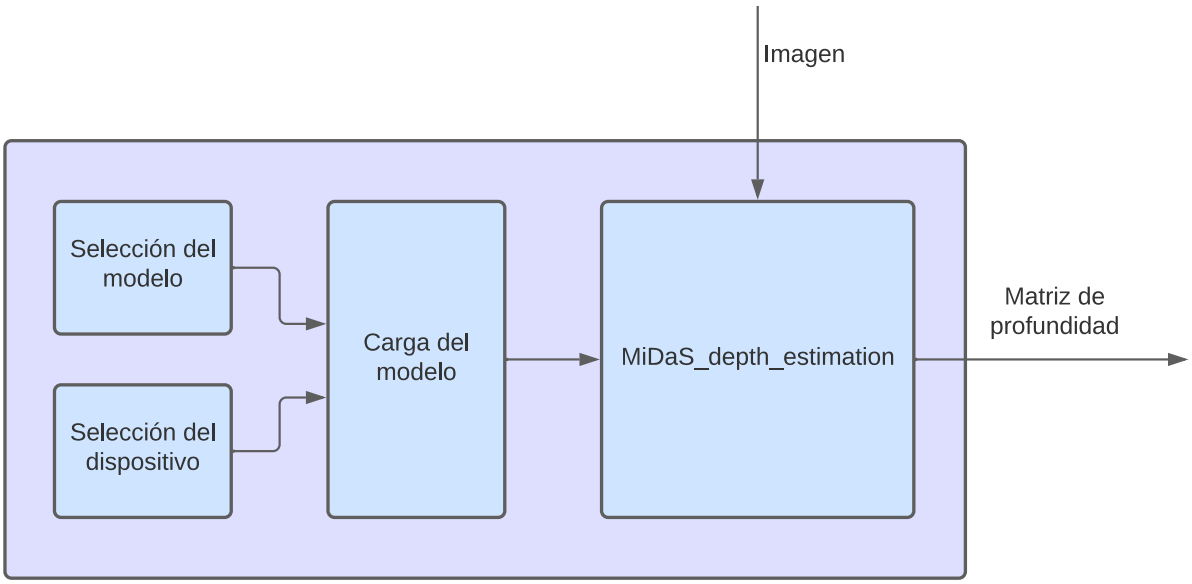

Figura 3: Diagrama del módulo de estimación de profundidad

El modelo DPT_Hybrid proporciona una estimación precisa en un tiempo promedio cercano a los 0.3 segundos, con casos extremos en valores cercanos a 1 segundo, por lo que se considera el modelo más óptimo para la implementación. Por lo que se define DPT_Hybrid como el modelo predeterminado que se carga en el módulo de selección del modelo.

La carga del modelo se realiza al iniciar la ejecución del programa. Además de seleccionar el modelo, se debe seleccionar el tipo de transformación que se aplicará en la red neuronal. Estas transformaciones controlan factores como el tamaño de entrada de la matriz que representa la imagen, la normalización de la imagen de entrada, conversiones de tipos de datos y transposiciones. Para los modelos anteriormente seleccionados, se recomiendan dos transformaciones:

- dpt_transform
- small_transform

Utilizando la *AI-deck color camera module* del Crazyflie 2.1, se considera una resolución de 324 x 324 pixeles, por lo que, siguiendo las resoluciones definidas en [13], se puede utilizar la transformación small_transform.

Una vez cargado el modelo, se provee al módulo MiDaS_Depth_Estimation con dicho modelo, además de la imagen capturada a la que se le debe realizar la estimación de profundidad. Una vez realizada la estimación de profundidad, se pasa al módulo de análisis matricial.

2.2.4 Diseño del módulo de de análisis matricial

El módulo de estimación de profundidad proporciona como salida una matriz, donde cada entrada corresponde al valor de profundidad del pixel correspondiente. Al estar realizando una estimación de profundidad inversa relativa, las entradas que representan áreas más alejadas tienen valores más cercanos a cero, mientras que las entradas de los objetos más próximos tienen valores más cercanos al valor máximo proximidad. Al ser una estimación relativa, los valores de las entradas no representan un dato real, sino que son relativos a los valores de las demás entradas. Utilizando el modelo DPT_Hybrid, el valor máximo de proximidad es cercano a 3000. Esto se debe a la normalización que se le realiza a la imagen antes del proceso de estimación de profundidad.

El primer paso del algoritmo de selección de rutas consiste en determinar las áreas más profundas, para poder determinar las posibles rutas que el UAV puede tomar. Para esto, se debe definir un valor umbral o *threshold* que delimite el nivel de profundidad al que se desea dirigir el dron. Como se menciona anteriormente, los valores de la matriz resultante de la estimación de profundidad no corresponden a una cantidad física real, por lo que el *threshold* se determina empíricamente basado en imágenes de prueba donde un humano identificaría cierta cantidad de rutas, y ajustando el valor para que las rutas coincidan. Para determinar las áreas de la imagen que corresponden a posibles trayectorias, se utiliza el algoritmo descrito en 1. Este algoritmo recibe como entradas la matriz obtenida de la estimación de profundidad y el valor de *threshold* definido, y devuelve una matriz binaria que indica si una entrada se mantiene por debajo del *threshold* definido.

Algorithm 1: Generación de una matriz binaria a partir de valores de profundidad

Entrada: Matriz de valores de profundidad, umbral
Salida : Matriz binaria resultante

```
foreach elemento element en Matriz de valores de profundidad do
    if element cumple la condición element < umbral then
        Agregar 1 a Matriz binaria resultante en la posición de element;
    else
        Agregar 0 a Matriz binaria resultante en la posición de element;
```

Una vez ejecutado el algoritmo 1, se tiene una matriz binaria donde las entradas con un valor de 1 corresponden a los pixeles de las áreas de las posibles rutas que el dron puede tomar. Sin embargo, pueden existir áreas pequeñas de unos pocos pixeles, que no corresponden a rutas que el dron puede tomar, sino a desviaciones debidas a la estimación de profundidad. Por lo que se debe definir un tamaño mínimo de área (submatriz) para que sea considerada una ruta válida. Este tamaño mínimo se define en 2.1, donde res_x y res_y corresponden a la resolución x y y de la imagen respectivamente, $field\_of\_view\_x$ corresponde al campo de visión en x y *accuracy* a la precisión deseada, como un parámetro que se define en función de la cantidad de grados de giro deseados para el UAV. Esto

establece una relación entre el tamaño de la matriz y la precisión deseada para el dron.

$$tamaño\_minimo = \frac{res_x \cdot res_y}{\frac{field\_of\_view\_x}{accuracy}} \tag{2.1}$$

Tras la definición de este parámetro, se aplica el algoritmo descrito en 2. Este algoritmo recibe como entrada la matriz binaria, y el tamaño mínimo de la submatriz, y retorna una lista con las coordenadas del centro en x de cada una de las áreas identificadas que tienen el tamaño mínimo para ser consideradas una ruta.

Algorithm 2: Obtener Coordenadas Horizontales Medias

Entrada: Matriz binaria, Tamaño mínimo
Salida : Lista de coordenadas horizontales medias

```
# Etiquetar componentes de áreas conectadas, donde cada entrada se reemplaza
  por un número que representa a qué área conectada pertenece;
matriz_etiquetada ← etiquetar_componentes(matriz_binaria);
# Descartar áreas de tamaño menor a tamaño_minimo;
areas_filtradas ←
  [índice para índice, tamaño en enumerar(tamaños_componentes) si tamaño ≥
  tamaño_minimo];
# Obtener coordenadas;
coordenadas_horizontales_medias ← [];
foreach área en areas_filtradas do
    índices ← donde(matriz_etiquetada == área);
    fila_min ← mínimo(índices[0]);
    fila_max ← máximo(índices[0]);
    fila_media ← (fila_min + fila_max)//2;
    columna_media ← suma(índices[1])//longitud(índices[1]);
    elemento_medio ← (fila_media, columna_media);
    coordenadas_horizontales_medias.agregar(elemento_medio);
return coordenadas_horizontales_medias;
```

Una vez teniendo las coordenadas medias de todas las posibles áreas, se debe decidir si es posible tomar una de las rutas disponibles, y en caso de serlo, que acciones debe seguir el UAV para tomarlas. Para esto, se utiliza el módulo de selección de ruta.

2.2.5 Diseño del módulo de selección de ruta

El módulo de selección de ruta realiza dos funciones principales, hacer que el UAV avance y gire en una dirección especificada, o hacer que el UAV ascienda o descienda para tomar una ruta. El UAV debe ascender o descender en caso de que la ruta no se encuentre directamente en frente de la cámara, es decir, que el área de la ruta no sea atravesada por

la linea que corresponde a la mitad vertical de la imagen. En caso de que el UAV ascienda o descienda, no gira ni avanza en dirección de la ruta. Esto para que, una vez estando en frente de la posible ruta, pueda girar en la dirección correcta sin colisionar con un posible obstáculo en frente, o simplemente avance hacia adelante sin una ruta definida.

Desplazamiento vertical

En caso de que el UAV deba ascender o descender, se plantea la pregunta ¿Cuánto debe hacerlo?. Ya que al estar utilizando estimación de profundidad relativa no se puede determinar de manera directa la distancia vertical hasta una posible ruta, se debe estimar el cambio de la altura de vuelo del UAV mediante otro método.

Para esto, se realiza una estimación basada en el peor escenario posible. Supóngase que el UAV viaja a una velocidad v. La cada t segundos se realiza una captura de la imagen del dron. Esto quiere decir que la distancia máxima que el dron puede recorrer en este periodo (d) viene dada por 2.2:

$$d = v \cdot t \tag{2.2}$$

En este caso, d corresponde a la distancia que el UAV podría recorrer en el periodo t. El peor escenario posible sería que la posible ruta a elegir, o un posible obstáculo a esquivar, se encuentren exactamente a está distancia d. Si el obstáculo estuviera a una distancia menor que d, habría sido detectado en una captura anterior, suponiendo que la distancia inicial del dron con un posible obstáculo sea de al menos d.

Teniendo d, se puede estimar el peor escenario posible (es decir, la menor) distancia vertical que cubre la imagen. Para esto, se puede aplicar un cálculo trigonométrico, como se indica en la figura 4. Supóngase que la posible ruta se encuentra debajo del dron, por lo que h se define como la distancia en metros desde el límite inferior de la imagen hasta el centro de la misma. El ángulo α se puede definir como la mitad del campo de visión del dron en y. Este es un parámetro de hardware que se indica al inicio de la ejecución. Por lo tanto, teniendo la distancia estimada d, se puede calcular h como se muestra en 2.3:

$$h = d \cdot \tan(\alpha) \tag{2.3}$$

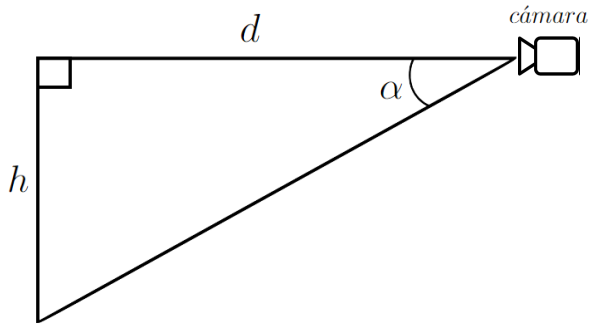

Figura 4: Representación del método para estimar altura absoluta

Una vez teniendo h, se puede realizar una equivalencia de pixeles a distancia física vertical. Ahora, se puede utilizar el algoritmo 2 para obtener la coordenada y del centro de la posible ruta que el dron puede tomar, la cual se denominará c_y. Sea res_y la resolución en y de la cámara del dron, la distancia que el dron debe bajar, denominada l , viene dada por la ecuación 2.4:

$$l = h \cdot \left(\frac{c_y - \frac{res_y}{2}}{\frac{res_y}{2}} \right) \tag{2.4}$$

El mismo razonamiento aplica en caso de que la ruta se encuentre por encima del centro de la imagen. Sin embargo, en este caso, el valor de c_y sería menor al del centro de la imagen, por lo que l correspondería a la distancia que el dron debería subir, y se definiría según 2.5:

$$l = h \cdot \left(\frac{\frac{res_y}{2} - c_y}{\frac{res_y}{2}} \right) \tag{2.5}$$

En caso de que la posible ruta se encuentre a una distancia mayor que d, la misma estimación de l se sostiene,como se muestra en la figura 5. Se definen s_y e i_y como las coordenadas de los límites superior e inferior de una posible área, respectivamente. Si la distancia real entre la posible ruta y la cámara es mayor a d, al moverse el dron la distancia l hacia abajo, el centro de la nueva imagen aún estará apuntando dentro del área de la posible ruta. El único cambio es que c_y no necesariamente corresponderá a la coordenada del centro de la imagen. Sin embargo, el área de la ruta posible en efecto se encontrará en frente de la cámara, por lo que en la siguiente captura se podrá determinar el ángulo en el que el dron deberá girar.

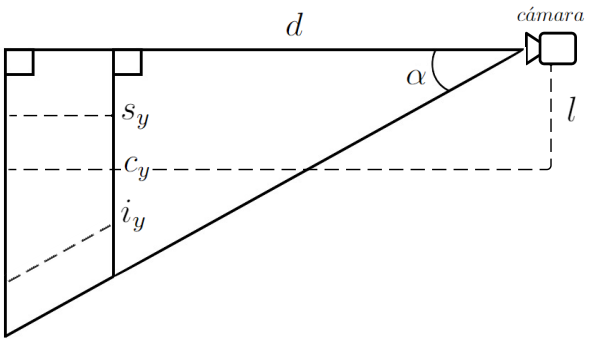

Figura 5: Representación del método para estimar altura absoluta en caso de distancia mayor a d

Desplazamiento horizontal

Para determinar la dirección en la que el dron debe girar horizontalmente, se considera el campo de visión horizontal de la cámara del mismo. El algoritmo 2 retorna la coordenada en x del centro de cada una de las posibles áreas que el dron puede tomar, por lo que para cada coordenada en x, el ángulo en el que el dron debe girar viene determinado por 2.6, donde $field\_of\_view\_x$ corresponde al campo de visión del dron en x, *angle* corresponde a la cantidad de grados que el dron debe girar, considerando que 0° se encuentra en el centro en x de la imagen, y res_x corresponde a la resolución en x de la cámara.

$$angle = \frac{c_x - \frac{res_x}{2}}{res_x} \cdot field\_of\_view\_x \tag{2.6}$$

Selección de ruta o instrucción

Cuando se realizan las estimaciones de desplazamiento vertical y horizontal, puede no identificarse una posible ruta, puede identificarse solo una, o pueden identificarse más de una. Como el objetivo de la flotilla de UAV´s es identificar explorar un terreno, se busca que se tomen la mayor cantidad de rutas posibles. Para esto, se asigna un identificador único a cada dron, y se escoge la ruta a tomar utilizando un algoritmo *Round Robin* basado en dicho identificador. En caso de que no se encuentre una posible ruta, el dron se desplazará hacia la izquierda o hacia la derecha, según su ID, hasta encontrar una posible ruta.

Como se mencionó anteriormente, para cada imagen el dron solo puede realizar, o bien un desplazamiento vertical, o uno horizontal, pero no ambos. Por lo que se debe determinar si el área que corresponde a la ruta que el dron debe tomar se encuentra directamente en frente de la cámara. Además, se debe considerar que el dron no se salga de los limites de altura definidos. En el caso de los drones Crazyflie, el límite de altura recomendado para el Flow Deck son 4 metros [6], mientras que la altura mínima puede ser definida según el uso.

En la figura 6 se muestra el diagrama de flujo para la selección de la instrucción que se enviará al dron.

2.2.6 Diseño de la interfaz con hardware

El sistema para la detección de rutas busca ser independiente del hardware, por lo que la interfaz con el hardware debe ser adaptada específicamente al UAV que se planea utilizar. En el caso de los UAV´s Crazyflie, estos se pueden controlar mediante radio utilizando el módulo de hardware CrazyRadio 2.0 [5], y el API de Python Crazyflie [4].

Para la comunicación con el hardware, se requieren 2 funcionalidades principales:

- Control de movimiento
- Lectura de imagen desde la cámara

El control del movimiento se debe definir como se indica en la figura 6. Los métodos *turn* y *set_flight_height* de la clase UAV se deben interfazar con el hardware para que giren la cantidad de grados indicados o cambien su altura, respectivamente.

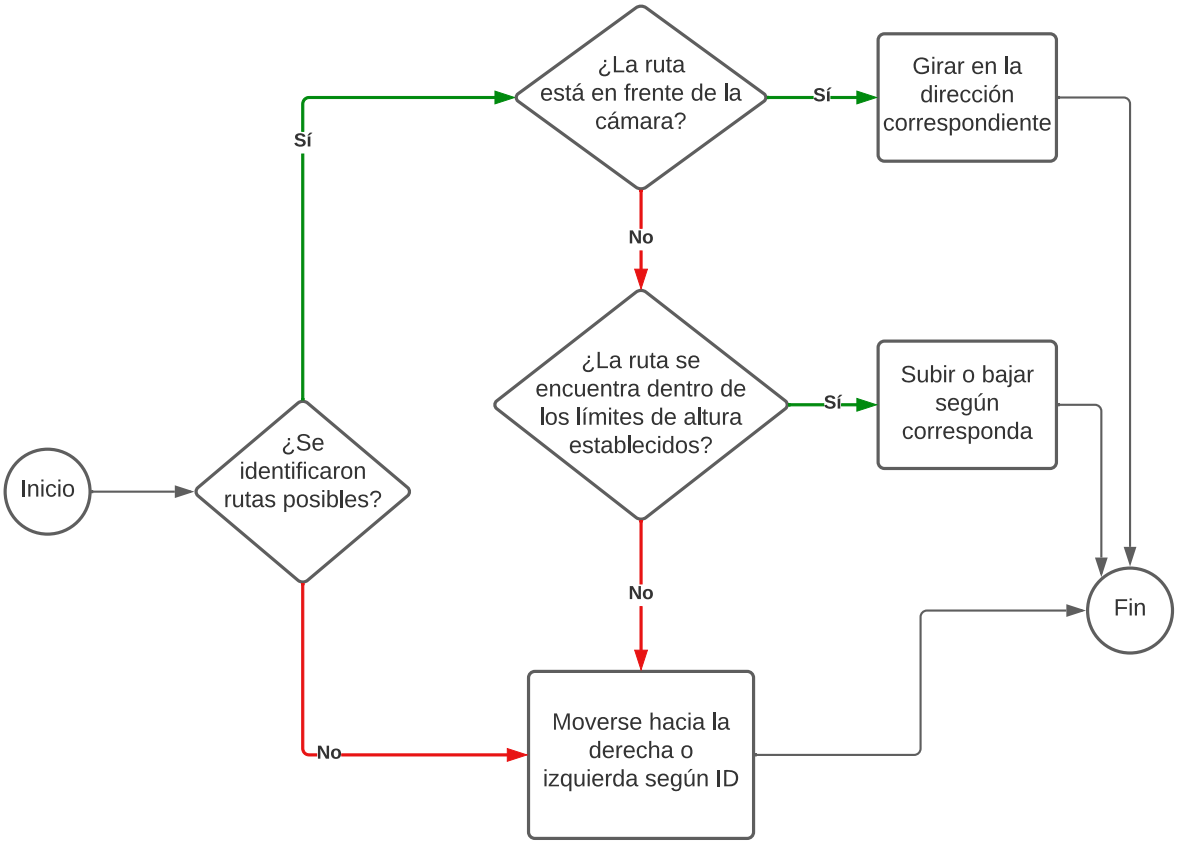

Figura 6: Diagrama de flujo para la selección de ruta

En el caso de los UAV´s Crazyflie, esto se logra mediante la clase *Motion Commander* [4]. Esta clase ofrece métodos para preparar el vuelo, despegar, moverse linealmente , incrementar o decrementar la altura, girar una cierta cantidad de grados, registrar datos de vuelo, entre otros. Por lo tanto, al seleccionar una ruta, los métodos de la clase UAV deben realizar un llamado con los respectivos parámetros a los métodos de la clase *Motion Commander*.

La otra funcionalidad necesaria para el sistema es la posibilidad de capturar imágenes desde la cámara. La lectura de video se hace desde el método *cv2.VideoCapture* de la biblioteca OpenCV [9].

En el caso de los UAV´s Crazyflie, el API de Python para controlar la cámara [3] ya se encuentra integrada con OpenCV [8], por lo que su implementación requiere indicar la fuente de video en el parámetro *video_source* en la clase UAV.

Capítulo 3

Diseño del código

La división modular que se explica en el capítulo anterior refleja la implementación a nivel funcional. A continuación, se explican los módulos de código presentes en `Trajectory Generation`. Se presenta una descripción de cada módulo, y se definen los métodos que contiene.

3.1 `capture_and_analyze_video`

Este módulo contiene los ciclos principales donde se ejecuta el programa. Para esto, se encarga de definir las variables de video necesarias, y ejecuta un ciclo hasta que se envíe una instrucción para detener la ejecución del programa, la cual está configurada por defecto como presionar la tecla "q". Cada ciclo toma un *frame* de la cámara o video cada `interval_seconds` segundos, muestra el video o toma una captura del *frame* en caso de estar configurado, y llama al módulo `complete_analysis` para realizar el análisis de la selección de ruta para el *frame* actual.

Funciones

`capture_and_analyze_video(drone,frames_list,num_drones,before_cicle_sleep_time,`

`interval_seconds,take_screenshots,dron_to_show,show_video,threshold_fraction,`

`transform,device,midas,accuracy,stop_event,print_analysis_time)`:

Inicializa el ciclo de ejecución para cada hilo de cada UAV.

Parámetros:

- `drone` (UAV): Instancia del dron a analizar
- `frames_list` (MatLike): Lista con los *frames* compartidos por todos los drones, para mostrar video en caso de estar configurado

- `num_drones` (int): Cantidad de drones instanciados
- `before_cicle_sleep_time` (float) tiempo que espera cada thread antes de empezar el ciclo, se utiliza para que no se ejecuten todos al mismo tiempo, utilizando calendarización Round-Robin
- `interval_seconds` (int): Define cada cuantos segundos se toma una captura de imagen para análisis
- `take_screenshots` (bool): Si se deben generar imágenes de cada captura o no
- `dron_to_show` (int): ID del dron que se quiere mostrar su video actual
- `show_video` (bool): Si se muestra video capturado de la cámara
- `threshold_fraction` (float): Fracción del valor máximo de profundidad de una imagen que se considera "profundo" para considerarse como posible ruta
- `transform` (torch.hub): Transformada de MiDaS
- `device` (torch.device): Dispositivo de MiDaS
- `midas` (torch.hub): MiDaS
- `accuracy` (int): En intervalos de cuantos grados se dividirá la imagen para que el dron considere una ruta
- `stop_event` (threading.event): Evento para detener la ejecución cuando se presione una tecla
- `print_analysis_time` (bool): Si se muestra en consola el tiempo de análisis para cada ejecución

Retorna:

- void

`show_current_frame_in_video(frames_list)`: Muestra el *frame* actual del video, en caso de que la opción esté activada

Parámetros:

- `frames_list` (MatLike): Lista con los *frames* compartidos por todos los drones, para mostrar video en caso de estar configurado

Retorna:

- void

3.2 choose_angle

Este módulo se encarga de seleccionar el ángulo según se estableció en la sección anterior.

Funciones

`choose_angle(depth_area,submatrices,drone)`:

Muestra el *frame* actual del video, en caso de que la opción esté activada

Parámetros:

- `depth_area` (list[list[int]]): Lista con la matriz binaria de áreas
- `submatrices` (int): Número de submatrices en los que se dividirá depth_area
- `drone` (UAV): Instancia del dron a analizar

Retorna:

- `angles` (list[float]): Lista de los ángulos de todas las posibles rutas a tomar por un dron
- `areas_in_front_of_camera` (list[tuple[string,int]]):Lista de tuplas que indica si un área se encuentra arriba, abajo o en frente del dron, y la diferencia entre el centro del área y el centro de la imagen en caso de que no esté en frente

`choose_angle(depth_area,submatrices,drone)`:

Muestra el *frame* actual del video, en caso de que la opción esté activada

Parámetros:

- `depth_area` (list[list[int]]): Lista con la matriz binaria de áreas
- `submatrices` (int): Número de submatrices en los que se dividirá depth_area
- `drone` (UAV): Instancia del dron a analizar

Retorna:

- `angles` (list[float]): Lista de los ángulos de todas las posibles rutas a tomar por un dron
- `areas_in_front_of_camera` (list[tuple[string,int]]):Lista de tuplas que indica si un área se encuentra arriba, abajo o en frente del dron, y la diferencia entre el centro del área y el centro de la imagen en caso de que no esté en frente

3.3 complete_analysis

Este módulo realiza la selección de rutas para cada imagen.

Funciones

`complete_analysis(drone,image,transform,device,midas,threshold_fraction, submatrices, interval_seconds,print_analysis_time)`:

Realiza la selección de rutas para cada imagen

Parámetros:

- `drone` (UAV): Instancia del dron a analizar
- `image` (list[list[float]]: Matriz de la imagen a analizar
- `transform` (torch.hub): Transformada de MiDaS
- `device` (torch.device): Dispositivo de MiDaS
- `midas` (torch.hub): MiDaS
- `threshold_fraction` (float): Fracción del valor máximo de profundidad de una imagen que se considera "profundo" para considerarse como posible ruta
- `submatrices` (int): Número de submatrices en los que se dividirá depth_area
- `interval_seconds` (int): Define cada cuantos segundos se toma una captura de imagen para análisis
- `print_analysis_time` (bool): Si se muestra en consola el tiempo de análisis para cada ejecución

Retorna:

- `depth_area` (list[list[int]]): Lista con la matriz binaria de áreas
- `depth_estimation_matrix` (list[list[float]]): Matriz de la estimación de profundidad

`handle_no_routes(drone)`:

Se llama en caso de que el análisis no identifique ninguna ruta.

Parámetros:

- `drone` (UAV): Instancia del dron a analizar

Retorna:

- void

3.4 delete_files_in_folder

Módulo que se utiliza para limpiar una carpeta después de cada ejecución.

Funciones

`delete_files_in_folder(folder_path)`:

Se utiliza para limpiar una carpeta después de cada ejecución.

Parámetros:

- `folder_path` (string): Ruta de la carpeta a limpiar

Retorna:

- void

3.5 generate_images

Módulo que se utiliza para generar la imagen con la imagen original, la estimación de profundidad y las áreas identificadas

Funciones

`merge_images(image_paths, output_path)`:

Fusiona las imágenes en una carpeta

Parámetros:

- `image_paths` (list[string]): Ruta de la carpeta con las imágenes
- `output_path` (string): Ruta de la carpeta con de salida

Retorna:

- void

`save_plot(input_matrix, filename)`:

Fusiona las imágenes en una carpeta

Parámetros:

- `input_matrix` (list[list[float]]): Matriz de la imagen
- `filename` (string): Nombre del archivo a generar

Retorna:

- void

`generate_merged_images(image_matrix_lists,merged_image_name,tmp_folder)`:

Genera la imagen fusionada

Parámetros:

- `image_matrix_list` (list[list[list[float]]]): Lista de matrices de imagenes
- `merged_image_name` (string): Nombre del archivo de salida
- `tmp_folder` (string): Folder temporal donde se guardan las imagenes que se generaron desde una matriz

Retorna:

- void

3.6 hardware_interface

Estos son los métodos llamados por la clase UAV. La implementación del movimiento depende del hardware.

Funciones

`turn(id, direction)`:

Elige la dirección en la que el dron debe girar para tomar la ruta seleccionada. El rango en el que el dron puede girar corresponde al campo de visión de la cámara del dron, y está definido desde (-field_of_view_x/2) hasta (field_of_view_x/2), donde field_of_view corresponde al campo de visión de la cámara del UAV en el eje x, en grados. 0° es el centro de la imagen.

El dron debe seguir moviéndose hacia adelante después de girar.

Parámetros:

- `id` (int): El identificador del dron.

- `direction` (float): El ángulo al que el dron debe girar.

Retorna:

- void

`move_horizontally(id, direction)`:

Elige la dirección en la que el dron debe moverse si no hay una ruta disponible. Se mueve hacia la izquierda o hacia la derecha.

El dron debe seguir moviéndose hacia adelante después de moverse.

Parámetros:

- `id` (int): El identificador del dron.
- `direction` (string): La dirección en la que el dron debe moverse ("left" o "right").

Retorna:

- void

`set_flight_height(id, flight_height)`:

Este método permite definir la altura de vuelo del dron dentro de los límites establecidos.

Parámetros:

- `id` (int): El identificador del dron.
- `flight_height` (float): La nueva altura de vuelo del dron, en metros.

Retorna:

- void

`set_current_speed(id, current_speed)`:

Este método permite definir la velocidad actual del dron.

Parámetros:

- `id` (int): El identificador del dron.
- `current_speed` (float): La nueva velocidad del dron, en metros por segundo.

Retorna:

- void

3.7 load_model

Este módulo carga el modelo de MiDaS al principio de la ejecución.

Funciones

`load_migrated_model(model_type))`:

Este método carga el modelo y lo retorna.

Parámetros:

- `model_type` (string): El nombre del modelo a utilizar

Retorna:

- `transform` (torch.hub): Transformada de MiDaS
- `device` (torch.device): Dispositivo de MiDaS
- `midas` (torch.hub): MiDaS

3.8 matrix_analysis

Este módulo se encarga de realizar las diversas operaciones matriciales necesarias para el funcionamiento del sistema. Sin embargo, muchas de estas funciones se aplican únicamente en métodos obsoletos, por lo que solo se definen las funciones que se utilizan en la solución final.

Funciones

`get_middle_horizontal_coordinates(binary_matrix, min_size)`:

Este método calcula el índice de fila del elemento horizontal medio promediando los índices de fila mínimo y máximo del área, como se explicó en la sección anterior.

Parámetros:

- `binary_matrix` (list[list[int]]): Matriz binaria que contiene los espacios de las posibles rutas a seguir.
- `min_size` (int): Tamaño mínimo del área para que sea considerada una ruta válida, en pixeles.

Retorna:

- `middle_horizontal_coordinates` (list[tuple[int]]): Las coordenadas del centro de cada una de las rutas identificadas en la imagen

3.9 MiDaS_depth_estimation

Este módulo se encarga de realizar la estimación de profundidad para los *frames* capturados.

Funciones

`estimate_depth(image,transform,device,midas)`:

Este método realiza la estimación de profundidad.

Parámetros:

- `image` (list[list[float]]): Matriz de la imagen a analizar
- `transform` (torch.hub): Transformada de MiDaS
- `device` (torch.device): Dispositivo de MiDaS
- `midas` (torch.hub): MiDaS

Retorna:

- `output` (list[list[float]]): Matriz de la estimación de profundidad

3.10 trajectory_generation

Es el punto de entrada para iniciar el programa. Se encarga de inicializar lar variables y los hilos para la ejecución.

Funciones

`trajectory_generation(drones,interval_seconds,take_screenshots,model_type,`

`show_video,threshold_fraction,accuracy,print_analysis_time)`:

Este método es el punto de entrada para el programa, una vez declaradas las variables necesarias.

Parámetros:

- `drones` (UAV): Lista de instancias de drones a analizar
- `interval_seconds` (int): Define cada cuantos segundos se toma una captura de imagen para análisis
- `take_screenshots` (bool): Si se deben generar imágenes de cada captura o no
- `model_type` (string): El nombre del modelo a utilizar
- `show_video` (bool): Si se muestra video capturado de la cámara
- `threshold_fraction` (float): Fracción del valor máximo de profundidad de una imagen que se considera "profundo" para considerarse como posible ruta
- `accuracy` (int): En intervalos de cuantos grados se dividirá la imagen para que el dron considere una ruta
- `print_analysis_time` (bool): Si se muestra en consola el tiempo de análisis para cada ejecución

Retorna:

- `void`

3.11 UAV

Esta clase representa la abstracción de los drones, y define los métodos que interactúan con el hardware cuando es necesario. Permite modelar las características y capacidades esenciales requeridas para la operación del sistema. El sistema está diseñado para ser utilizado con una flota de drones de números variables, lo que requiere la creación de una instancia para cada UAV real disponible.

Atributos

- `ID`: Representa el identificador del dron. Debe asignarse en orden ascendente, comenzando desde 0.
- `current_speed`: Denota la velocidad actual del UAV en metros por segundo. Esta información es crucial para la selección de rutas en las que el dron debe ajustar su altura. En caso de no poderse acceder, se puede elegir la velocidad media que se espera para el dron, o la velocidad máxima que puede alcanzar. En el caso de los drones Crazyflie, la velocidad máxima que permite el API es de 1 m/s [7].
- `field_of_view_x`: Representa el campo de visión del dron en grados a lo largo del eje x. Es necesario para determinar el rango de grados capturados por la cámara, ayudando en decisiones sobre cuánto debe girar el dron en una dirección específica.

- `field_of_view_y`: Representa el campo de visión de la cámara del dron en grados a lo largo del eje y. Es necesario para determinar los ajustes de altura que debe realizar el dron cuando sea necesario.

- `min_height`: Significa la altura mínima de operación del dron. Esta variable se define en función del uso previsto.

- `max_height`: Corresponde a la altura máxima de operación del dron. Se determina en función del uso previsto y las limitaciones de alcance del dron. Se recomienda que sea definida como un valor menor a la altura máxima que el dron puede identificar, esto para que el dron no pierda la capacidad para identificar su altura. Por ejemplo, en el caso de los drones Crazyflie la altura máxima que puede identificar el FlowDeck es 4 metros [6], por lo que se puede configurara la altura máxima de uso a 3.5 metros, por ejemplo.

- `current_height`: Indica la altura actual del dron, esencial para evitar que el dron supere los límites establecidos. En el caso de los drones Crazyflie, el sensor Flow-Deck puede proporcionar esta información. Esta altura debe estar dentro del rango de valores mínimo y máximo definidos al momento del inicio de la ejecución. El programa verifica que el dron se mantenga dentro de este rango, garantizando que la altura no se salga del rango en caso de que ubique una ruta posible arriba o debajo de la imagen. Además, se ajusta automáticamente en casos donde la altura respecto al suelo pueda variar, por ejemplo, en el caso que haya una pendiente debajo del dron.

- `resolution_x`: La resolución en la dirección x de la cámara del UAV. Se requiere para estimar posibles ubicaciones de rutas en la imagen.

- `resolution_y`: La resolución en la dirección y de la cámara del UAV. Se requiere para estimar posibles ubicaciones de rutas en la imagen.

- `video_source`: Corresponde a la fuente de video para el análisis. Utilizando OpenCV, se puede especificar el ID de la cámara asociada con la instancia del dron.

- `video_type`: Representa el tipo de video capturado con el parámetro de `video_source`. Dos tipos existentes son 'video' (para archivos de video, como para pruebas o configuraciones preexistentes) y 'camera' (para fuentes de video directas de la cámara UAV).

Métodos

- `turn(direction)`: Elige la dirección en la que el dron debe girar para seguir la ruta seleccionada. El rango permitido para la rotación del dron corresponde al campo de visión de la cámara y se define desde -field_of_view_x/2 a field_of_view_x/2.

- `move_horizontally(direction)`: Especifica la dirección horizontal (izquierda o derecha) en la que el dron debe moverse en caso de que no se detecte una ruta.

- `set_flight_height(flight_height)`: Este método permite definir la altura del dron dentro de los límites establecidos.

- `set_current_speed(new_speed)`: Este método permite definir la velocidad actual a la que se mueve el dron. Si no se puede determinar a través del hardware, puedes usar la velocidad promedio o la velocidad máxima del dron. En el caso de los drones Crazyflie, esto corresponde a 1 m/s [7]. Ajustar este parámetro ayuda a calcular el Escenario de Peor Caso para la distancia a una ruta u obstáculo.

Capítulo 4

Configuración y uso

En este capítulo se describe la puesta en marcha y utilización de UrbanPathGen.

4.1 Instalación

Para la instalación de UrbanPathGen, se deben seguir los siguientes pasos:

1. Instalar Python junto con el instalador pip desde [17]. Se debe corroborar que la versión instalada soporte PyTorch, esto se puede verificar desde [18]. Para el desarrollo de UrbanPathGen, se utilizó Python 3.10.

2. Clonar el repositorio de UrbanPathGen, el cual se puede encontrar en [10].

3. Seguir las instrucciones de instalación de CUDA, detalladas en [16].

4. Seguir las instrucciones de instalación de PyTorch, detalladas en [18].

5. Instalar los paquetes necesarios para el programa. Para esto, se puede ejecutar el comando `pip install -r requirements.txt` desde la carpeta raíz de UrbanPathGen.

6. Descargar el modelo de MiDaS a utilizar. Para esto, dirigirse a la carpeta `Trajectory Generation/torch_models` y ejecutar el *script* `force_refresh_model.py`. El modelo que viene configurado por defecto es `DPT_Hybrid`, sin embargo, se pueden seleccionar otros modelos disponibles, los cuales se pueden ver en el archivo `see_available_models.py`.

4.2 Estructura del repositorio

Al entrar en el repositorio, se encuentran dos carpetas:

4.2.1 Registro de pruebas

Esta carpeta contiene el registro de pruebas realizadas para el desarrollo. Además de dos *scripts*:

- `calc_average_time`: Permite ingresar la salida de la consola y calcular el tiempo medio de análisis para cada iteración en una ejecución
- `graph`: Se utiliza para gráficar datos. En este caso, una lista de tiempos promedio en función de la cantidad de drones.

4.2.2 Trajectory Generation

Es la carpeta que contiene todo el desarrollo de UrbanPathGen.Además de todos los archivos necesarios para su ejecución, los cuales se explicaron previamente, contiene las siguientes carpetas:

- image_analysis: En esta carpeta, se guarda la salida generada por el *script* `generate_one_image`.
- old_methods: Contiene métodos obsoletos para el cálculo de trajetorias. Se puede ignorar.
- test_images: Permite cargar imágenes de prueba para el programa.
- test_videos: Permite cargar videos de prueba para el programa.
- tmp_images: Guarda temporalmente los archivos generados por el *script* `generate_one_image`.
- torch_models: Contiene *scripts* que permiten cargar un modelo, y verificar los modelos y transformaciones existentes.
- video_frames: En esta carpeta se guardan los *frames* de los flujos de video existente en caso de que la opción esté activada.

Además, en la carpeta Trajectory Generation se encuentran dos archivos, `example` y `generate_one_image`, los cuales corresponden a los casos de uso posibles, los cuales se explican a continuación.

`example`

Este archivo contiene los datos necesarios para la ejecución del programa. Para esta ejecución, se debe crear una instancia de dron por cada dron real disponible. Para esto, se define una lista llamada `drones` que contiene las instancias de cada uno de los drones. Las instancias se definen con los atributos indicados en la sección clase UAV. Algunos de

estos atributos corresponden a parámetros de hardware, los cuales deben ser indicados en función de los equipos disponibles. En este caso, se asume que todos los drones son del mismo modelo de hardware, por lo cual los parámetros dependientes del mismo son compartidos. En caso de no ser así, simplemente se inicializa la clase con atributos distintos. Entre las instancias de drones, se muestra una con el atributo `video_type = 'camera'`, y el atributo `video_source = 0` Esto quiere decir que esta instancia recibe video directamente desde la cámara con ID = 0, en este caso, la cámara de la computadora que ejecuta el programa. Para su utilización real, se debe indicar el ID de la cámara de cada dron que percibe OpenCV [9] [8]. En los demás casos, se toma como fuente un video. Esto se puede utilizar para escenarios de prueba.

Una vez creadas las instancias de los drones y agregadas a la lista, se puede iniciar el programa llamando a la función `trajectory_generation` utilizando la lista de instancias de drones y los parámetros descritos anteriormente.

Al iniciar la ejecución, el programa tardará algunos segundos antes de empezar a generar instrucciones. Esto debido a que se debe cargar el modelo, se deben crear los hilos para cada dron, inicializar las variables y empezar la captura de video.

Antes de la primera captura, el programa llamará al método `set_flight_height` de la clase UAV para realizar el despegue. Para esto, se configura la altura del dron como la altura mínima indicada al inicio de la ejecución.

`generate_one_image`

Este módulo genera una imagen compuesta por la imagen original, la estimación de profundidad de dicha imagen y las rutas identificadas en la imagen. Este módulo utiliza los mismos parámetros que `example`, sin embargo, en vez de llamar a `trajectory_generation`, realiza el análisis de una sola imagen utilizando `complete_analysis` y posteriormente toma la salida y genera la imagen utilizando el método `generate_merged_images`. La única diferencia para su uso con respecto a `example` es que se debe indicar en la variable `image_path` la ruta completa de la imagen. Una vez indicado este parámetro, la salida se guardará en la carpeta image_analysis.

Bibliografía

[1] F. Ahmed, J.C. Mohanta, A. Keshari, et al. Recent advances in unmanned aerial vehicles: A review. *Arab Journal of Science and Engineering*, 47:7963–7984, 2022. URL https://doi.org/10.1007/s13369-022-06738-0.

[2] S. S. Bashyal et al. Drone delivery systems: A literature review. pages 1–3, 7, 2019.

[3] Bitcraze. Ai-deck color camera module. https://store.bitcraze.io/products/copy-of-ai-deck-color-camera-module.

[4] Bitcraze. The crazyflie python api explanation. https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/user-guides/python_api/.

[5] Bitcraze. Crazyradio 2.0. https://www.bitcraze.io/products/crazyradio-2-0/.

[6] Bitcraze. Flow deck v2. https://www.bitcraze.io/products/flow-deck-v2/.

[7] Bitcraze. position controller pid. https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/controller/position_controller_pid.c#L593.

[8] Bitcraze. Video streamer. https://www.bitcraze.io/documentation/repository/aideck-gap8-examples/master/simple-examples/wifi-streamer/.

[9] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[10] Allan Calderón. Urbanpathgen. https://github.com/Calquito/UrbanPathGen.

[11] Samira Hayat, Evşen Yanmaz, and Raheeb Muzaffar. Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Communications Surveys & Tutorials*, 18(4):2624–2661, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] Intel. Midas. https://github.com/isl-org/MiDaS.

[14] A. Krizhevsky et al. Imagenet classification with deep convolutional neural networks. pages 1–2, 2012.

[15] MSI. Msi gf63. https://es.msi.com/Laptop/GF63-Thin-10SX-GTX/Specification.

[16] Nvidia. Cuda. https://developer.nvidia.com/cuda-toolkit.

[17] Python. Python. https://www.python.org/downloads/.

[18] PyTorch. Get started. https://pytorch.org/get-started/locally/.

[19] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3), 2022.