System Implementation Document

DC System

Equipo Mii Dron

Índice

Tabla de Ilustraciones

Histórico del Documento

Título del documento	Versión	Fecha	Autor	Revisor
DC System: Mii ME3D	1.0	19/4/2019	Mikel Ruiz	Daniel Rodrigo
DC System: Mii ME3D	2.0	11/5/2019	Mikel Ruiz	Daniel Rodrigo

Prefacio

Este documento recoge la implementación final del sistema que da lugar al producto definido inicialmente en el OpsCon. La implementación de todas las partes que integran el sistema se va a mostrar en las siguientes etapas: Software, Hardware. En este documento no se va a volver a detallar las etapas de Design y Testing pues cada una de ellas cuenta con un documento específico en los cuáles se establecen todos los datos relativos a dichas etapas del ciclo de vida del sistema de reconstrucción 3D de la fachada de la ETSII empleando para ello un dron Parrot Bebop 2.

Documentos de referencia

Como documentos de referencia en primera instancia se van a utilizar aquellos que se han desarrollado a lo largo del ciclo de vida del producto, desde la definición del producto en el documento OpsCon , pasando por el establecimiento de los requisitos con los stakeholders hasta llegar a la fase de implementación tras conocer el diseño del producto. Por lo tanto, a continuación, se muestra una lista de los documentos de referencia empleados como base para la elaboración de este documento de implementación del sistema total.

- DC Design: Mii ME3D (v1.0)
- INCOSE, OpsCon and SysRS
- Hackday F Report
- DC Testing: Mii ME3D (v1.0)
- SysML Standard
- Course Slides

Introducción

Para la descripción de la implementación del sistema se va a mostrar en primer lugar, una breve descripción de los componentes tanto software como hardware empleados en el proyecto. Para posteriormente, describir la secuenciación del proyecto mostrando cada una de las fases seguidas en el proceso de implementación hasta llegar a la solución final. La implementación se va a basar fundamental en la interacción del código desarrollado, compilado y ejecutado en el host, para que interactúe con el target en nuestro caso el procesador del dron y permita el control del vuelo del mismo y la captura de las fotos en los puntos fijados por el equipo para conseguir una correcta reconstrucción de la fachada.

Descripción componentes

Componentes Hardware

La finalidad de este documento no es la descripción de los componentes utilizados en el proyecto, pero es necesario recordarlos para ponerse en situación.

Una descripción más amplia está presente en el documento DC Design del proyecto.

El dron empleado se corresponde con el dron Parrot Bebop 2. Cuya característica esencial es que es programable y cuenta con buenas prestaciones para la captura de imágenes, duración de batería y conectividad Wi-fi.

Las especificaciones técnicas más significativas se muestran en la propia página de la compañía, además de quedar completamente definidas en el modelo del sistema creado en la etapa de Diseño del sistema. (Parrot, 2019)

Ilustración 1.Dron Parrot Bebop 2

La duración de la batería, la calidad de la cámara y la estabilidad de vuelo son características de este modelo muy valorados para la elaboración del sistema objetivo de este Ingenia.

Componentes Software

Igualmente, como se ha indicado en la descripción de los componentes hardware, se va a mostrar ligeramente los paquetes software de trabajo a utilizar durante la implementación del sistema final. En documentación previa, se señaló los distintos tipos de software a utilizar:

- Software de navegación: destinado a la planificación de la ruta a seguir durante la captura de imágenes. Se utilizará un módulo ROS (Aerostack) con el que se indicarán los *waypoints* a seguir por el dron.

- Software de captura de imágenes: deberá coordinarse con el de navegación de manera que la captura de imágenes se lleve a cabo en los waypoints indicados, según haya sido planificado de antemano. Este software también deberá procurar la calidad de las imágenes (enfoque, obturación...), quedando esto limitado por el modelo de dron escogido y estas serán almacenadas en una base de datos, a la cuál se podrá acceder por medio de una sencilla interfaz creada en Python 2.7.

- Reconstrucción 3D: una vez obtenidas las imágenes pertinentes será empleado para reconstruir el modelo 3D de la ETSII, mediante el acceso a ellas por la interfaz creada y posterior operación del software especializado en la tarea de reconstrucción a partir de imágenes, testeado en el Hackday F, llamado Meshroom.

- Software empleado para el testeo de los requisitos, en este caso se ha empleado la funcionalidad y todas las posibilidades que otorga Google test implementado en Visual Studio. Esta etapa del ciclo de vida del sistema se encuentra perfectamente explicada en el documento de referencia (DC Testing: Mii ME3D (v1.0)).

La navegación y captura de las imágenes se ha implementado en un mismo código que realiza ambas funcionas, en el cual se lanza una rutina de captura de imágenes para ciertos waypoints fijados en función de la altura y anchura de la fachada, después de la cual se puede volver a lanzar tras un giro previo del dron permitiendo capturar las imágenes en tres direcciones que permitan realizar la reconstrucción 3D. Para ello, el equipo ha trabajado calibrando el ángulo de giro del dron para tener máxima precisión en la operación de la obtención de las imágenes.

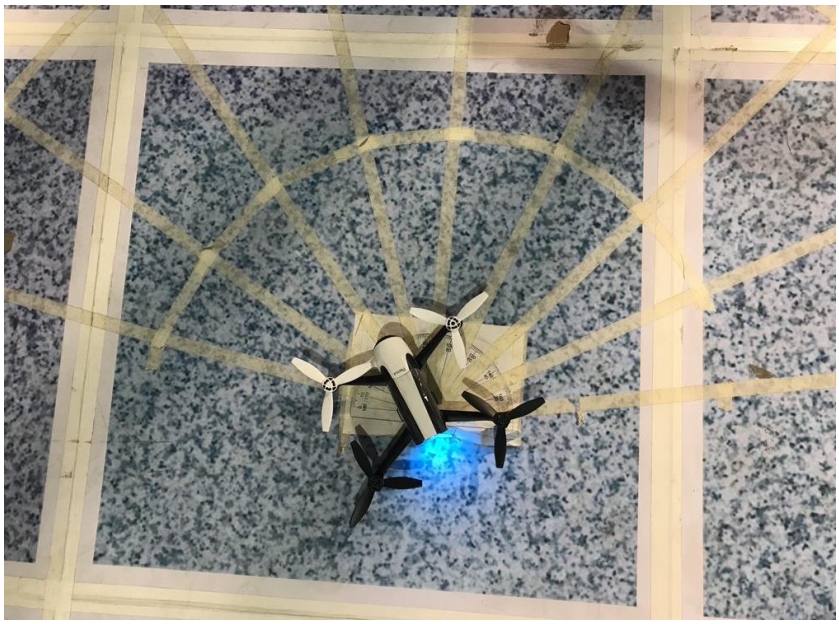

Ilustración 2. Calibración del ángulo de giro del dron en función del tiempo de activación de los motores.

Procedimientos

En este apartado, se va a mostrar como serían los procedimientos de implementación tanto de software como de hardware para conseguir cumplir con los requisitos fijados al comienzo del proyecto en el sistema final.

Para comenzar se nos ha proporcionado un ordenador particular para cada equipo con licencias para la instalación del software necesario. Inicialmente se debe instalar ROS y Aerostack, para posteriormente lanzar Aerostack, con el cuál se controla la trayectoria de vuelo del Dron para que se puedan capturar las imágenes justas y necesarias para la correcta reconstrucción de la fachada. El procedimiento de instalación de ROS en Ubuntu se muestra en la siguiente referencia. (Instalación de ROS en Ubuntu, 2019).

Una vez realizada la instalación de ROS nos disponemos a la instalación de Aerostack una extensión de ROS desarrollada en la escuela que permite a los desarrolladores controlar y elaborar la estructura del sistema aéreo, generalmente drones, de igual manera a continuación se muestra la referencia asociada a la instalación de Aerostack. (ETSII, 2019)

En el primer anexo de este código, se muestra el código destinado al control del vuelo del Dron. Este se ha implementado gracias a la librería rospy de Python que permite al usuario utilizar fácil y rápidamente ROS; la utilización de topics, servicios o el ajuste de determinados parámetros. Por lo tanto, la librería rospy es una API para el cliente, es decir, una interfaz de programación que permite al usuario acceder a las funcionalidades de ROS.

Posteriormente, el dron captura las imágenes en los puntos en los cuáles se le ha indicado en el propio código y estas van a parar a una base de datos que será accesible por medio de una interfaz creada que cuenta con un código en Python 2.7 que permite el envío y recibo de dichas imágenes.

Finalmente, se realiza la reconstrucción 3D para la cual se ha empleado el software Meshroom que se ha probado que cuenta con una precisión bastante elevada, aunque su tiempo de operación se encuentra igualmente disparado. Esta decisión no está del todo tomada pues se está barajando la posibilidad de emplear un software de reconstrucción que parte de un video del objeto a reconstruir, llamado OpenDroneMap, aunque para ello es necesario contar con Windows 10 Pro. Por ello, seguramente no se pueda emplear esta solución y se opte por la utilización de Meshroom directamente.

Implementación

De manera previa a comenzar la implementación hay que ser consciente del modelo establecido en la etapa de diseño con cada una de las partes que integran el sistema y la trazabilidad de cada uno de los requisitos que no ha quedado del todo definida en la primera versión del documento de diseño. Por lo que es necesario, tener la visión global del conjunto en todo momento y realizar el testeo de cada uno de ellos en el sistema final implementado.

Aerostack

La implementación se ha iniciado con la puesta a punto de ROS y Aerostack, que permite el control del vuelo del dron y de la acción de captura de imágenes por medio del establecimiento de ciertos waypoints.

Aerostack es una extensión de ROS que permite a los desarrolladores de sistemas controlar y construir la arquitectura de sistemas aéreos, integrando soluciones computacionales heterogéneas.

Es una herramienta fundamental para crear sistemas de vuelo autónomos en entornos dinámicos, por lo que es una herramienta de investigación útil para probar nuevos algoritmos y arquitecturas. Información más detallada sobre la funcionalidad requerida en este proyecto se muestra en su página de GitHub. (Aerostack, 2019)

La versatilidad de este software lo hace idóneo para el propósito buscado. Se puede emplear con Hardware independiente como es el Dron Parrot Bebop 2, y permite controlar y crear misiones aéreas sin un operador asistente.

Sus características ligadas a ROS permiten crear un entorno de desarrollo perfecto para el proyecto, otorgando una librería con componentes para drones, tanto en C++ como en Python, los dos lenguajes de alto nivel más empleados. En nuestro caso el código ha sido desarrollado en Python y se recoge en el anexo A de este documento. El código está por el momento en fase de desarrollo por lo que la versión aquí incluida no cuenta con las sentencias relativas a la captura de imágenes.

Por útlimo, todas las configuraciones e intrucciones que se deben lanzar para la puesta en marcha del sistema y comenzar el pilotaje están

recogidas en el documento que se ha elaborado de manera simultánea a este llamado Manual de Usuario.

Meshroom

Tras barajar la idea de trabajar con distintos softwares de reconstrucción 3D se decidió trabajar finalmente con Meshroom por presentar un equilibrio entre precisión de la reconstrucción, necesidad de recursos y autonomía del software para realizar su trabajo.

Este software no cobra importancia hasta que no se ha realizado la captura de imágenes por parte del dron y no se han almacenado en una base de datos de accesibilidad por medio de una interfaz, programa en Python que permite la recepción y el envío de imágenes. Por lo tanto, a la entrada de esta etapa se encuentran un conjunto de imágenes que cuentan con unas características determinadas de posición y ángulo de giro determinado sobre la fachada de la Escuela a reconstruir, para permitir que Meshroom cuente con la información necesaria para la creación de dicho modelo 3D.

Como es normal en este tipo de software los requisitos de potencia de la tarjeta gráfica del ordenador son elevados y el tiempo computacional de creación del modelo se puede disparar considerablemente por lo que se presenta como una solución óptima en el caso que se cuente con los medios y el tiempo necesario para la creación del modelo, situación en la cuál se encuentra actualmente el equipo.

Hackday F

Durante la sesión del llamado Hackday F y los días posteriores se han conseguido importantes avances a notificar en la situación actual del sistema.

Los temas tratados en esta sesión fueron:

- Conocer e informarse a cerca del estado del arte del control del dron Parrot Bebop 2.
- Conseguir las primeras comunicaciones entre el Dron y el target (PC).
- Preparar el uso de Aerostack para el sistema.
- Programación del vuelo, sin GPS, empleando el driver de ROS, "Bebop_autonomy".
- Probar distintas operaciones de vuelo sin las hélices.
- Buscar todas las alternativas de vuelo posibles.
- Análisis preliminar del simulador de vuelo Sphinx.

Dichos temas tratados han sido ya realizados e incluso una serie pruebas de vuelo han sido realizadas para verificar el correcto funcionamiento e implementación del código del controlador ROS, "Bebop_autonomy", con un funcionamiento robusto y correcto.

Ilustración 3. Test de vuelo

En las tareas relativas a la creación del servidor de almacenamiento de imágenes, se está trabajando de manera conjunta con el equipo de NordTech, de tal manera que se cumplan los plazos para su implementación y por medio de una interfaz sencilla que se ha elaborado en Python 2.7 con la librería Tkinter. Por medio de esta interfaz de muy fácil manejo por el usuario se pueden enviar y recibir imágenes al servidor de almacenamiento. Con este conjunto de imágenes tomadas por el dron se accede al programa Meshroom que lleva a cabo la reconstrucción del objeto en cuestión, en nuestro proyecto la fachada trasera de la ETSII.

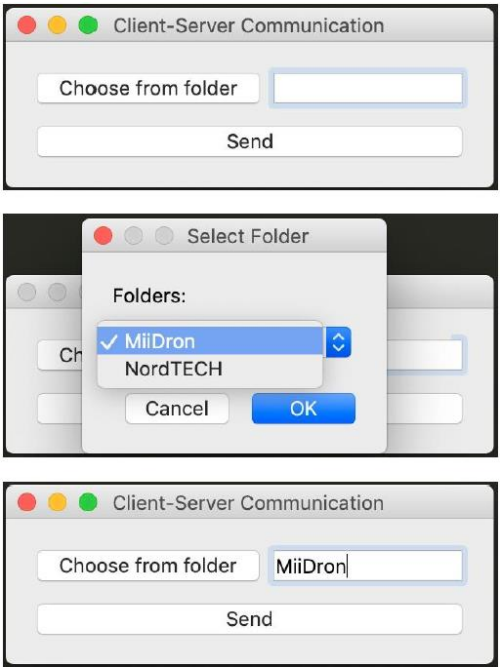

Ilustración 4. Interfaz de acceso a las imágenes de la base de datos.

A continuación, se muestra una prueba que se h realizado con el software Meshroom durante el Hackday F para justificar el correcto funcionamiento de la reconstrucción 3D.

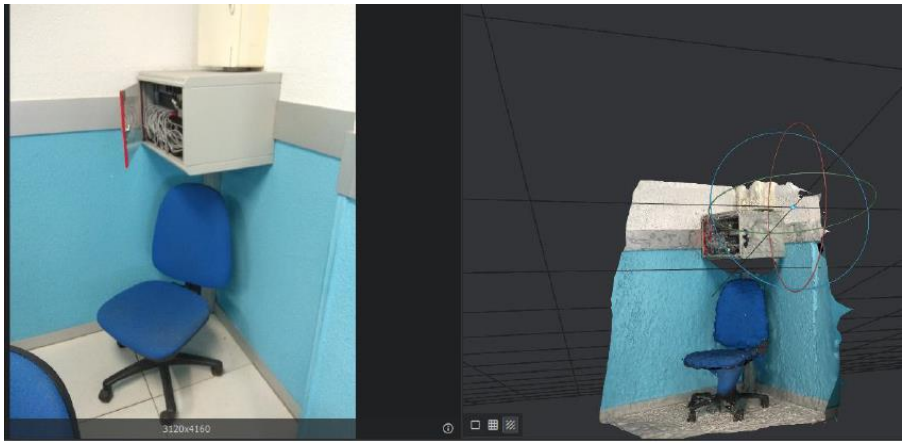

Ilustración 5. Reconstrucción 3D de una silla, con previa elección de las imágenes

Pruebas del sistema

Ilustración 6. Equipo trabajando en la nave de pruebas de vuelo.

Se ha conseguido completar la primera prueba de los elementos del sistema por separado. A diferencia de la ilustración 3, se ha conseguido realizar una reconstrucción 3D con el software Meshroom utilizando imágenes previamente obtenidas mediante el vuelo controlado por el código lanzado desde el ordenador proporcionado con sus correspondientes paradas en los waypoints para la obtención de imágenes.

El resultado se muestra a continuación, y nos permite concluir que una reconstrucción 3D con un grado de precisión y detalle respetable es posible.

Ilustración 7. Proceso de vuelo y captura de imágenes.

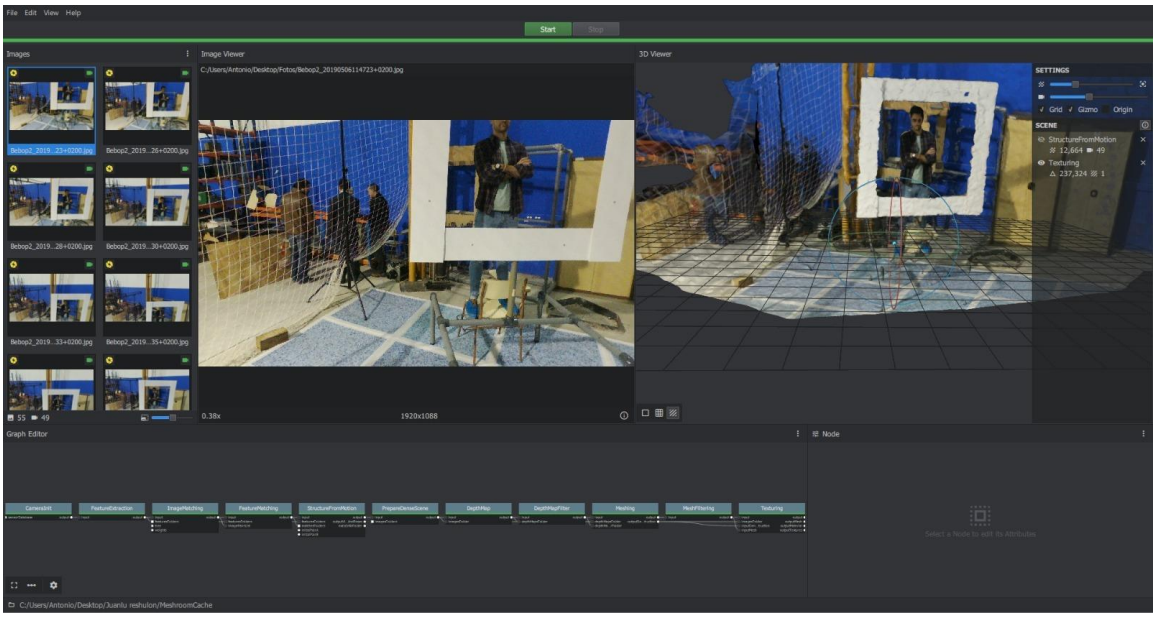

Ilustración 8. Reconstrucción 3D empleando las imágenes de vuelo del dron Parrot Bebop 2.

Logros obtenidos

- Lanzar rutinas de vuelo con captura de imágenes en distintos waypoints.
- Conocer el ángulo de giro del dron con respecto al tiempo de funcionamiento de los motores
- Primera reconstrucción realizada con imágenes capturas por el dron.
- Realización de determinadas pruebas asociadas a la fase de Testing del sistema.
- Obtención de material audiovisual para presentaciones futuras y para corroborar los logros.

Referencias

Aerostack. (Abril de 2019). Obtenido de Aerostack info: https://github.com/Vision4UAV/Aerostack/wiki/What-is-Aerostack

ETSII, V. . (Mayo de 2019). Obtenido de https://github.com/Vision4UAV/Aerostack/tree/master/installation

Instalación de ROS en Ubuntu. (Mayo de 2019). Obtenido de http://wiki.ros.org/es/hydro/Installation/Ubuntu

Parrot. (Abril de 2019). *Dron Parrot Bebop 2*. Obtenido de https://www.parrot.com/es/drones/parrot-bebop-2#t%C3%A9cnicos

ANEXO A. Código Relevante

En este anexo se va a mostrar el código relativo al control de vuelo, que como ya se ha indicado en este documento se encuentra en continua actualización por lo que el código aquí expuesto seguramente no sea la última versión de este.

```
#!/usr/bin/env python

import rospy
import time

from std_msgs.msg import String
from std_msgs.msg import Empty
from geometry_msgs.msg import Twist

def takeoff():
    pub_takeoff = rospy.Publisher("bebop/takeoff", Empty, queue_size=10 )
    rospy.init_node('bebop_MD', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    time_toff = 3
    t0_toff = rospy.Time.now().to_sec()
    t1_toff = 0
    print("--- Take Off ---")
    while (t1_toff < time_toff):
      t1_toff = rospy.Time.now().to_sec() - t0_toff
      pub_takeoff.publish(Empty())
      rate.sleep()

def land():
    pub_land = rospy.Publisher("bebop/land", Empty, queue_size=10 )
    rospy.init_node('bebop_MD', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    time_tl = 3
    t0_tl = rospy.Time.now().to_sec()
    t1_tl = 0
    print("--- Land ---")
    while (t1_tl < time_tl):
```

```
        t1_tl = rospy.Time.now().to_sec() - t0_tl
        pub_land.publish(Empty())
        rate.sleep()

def fwd_bck(d,s):
    pub_fw = rospy.Publisher("bebop/cmd_vel", Twist, queue_size=10 )
    rospy.init_node('bebop_MD', anonymous=True)
    rate = rospy.Rate(10) # 10hz

    vel_msg = Twist()

    ##Init
    #linear
    vel_msg.linear.x=s
    vel_msg.linear.y=0
    vel_msg.linear.z=0
    #angular
    vel_msg.angular.x=0
    vel_msg.angular.y=0
    vel_msg.angular.z=0

    speed = abs(vel_msg.linear.x)
    distance=d #in metters

    distance=float((distance*5.0)/15.0)
    t0 = rospy.Time.now().to_sec()
    current_distance = 0

    print("--- fwd_bck ---")
    while (current_distance < distance):
        #publish the velocity
        pub_fw.publish(vel_msg)
        #actual time
        t1 = rospy.Time.now().to_sec()
        #actual distance
        current_distance = speed*(t1-t0)

    print("--- Stop ---")
    vel_msg.linear.x=0
```

```
    pub_fw.publish(vel_msg)
    rate.sleep()
    distance=0


def up_dw(d,s):
    pub_up = rospy.Publisher("bebop/cmd_vel", Twist, queue_size=10 )
    rospy.init_node('bebop_MD', anonymous=True)
    rate = rospy.Rate(10) # 10hz

    vel_msg = Twist()

    ##Init
    #linear
    vel_msg.linear.x=0
    vel_msg.linear.y=0
    vel_msg.linear.z=s
    #angular
    vel_msg.angular.x=0
    vel_msg.angular.y=0
    vel_msg.angular.z=0

    speed = abs(vel_msg.linear.z)
    distance=d #in metters
    distance=float((distance*5.0)/15.0)

    t0 = rospy.Time.now().to_sec()
    current_distance = 0

    print("--- up_dw ---")
    while (current_distance < distance):
       #publish the velocity
       pub_up.publish(vel_msg)
       #actual time
       t1 = rospy.Time.now().to_sec()
       #actual distance
       current_distance = speed*(t1-t0)
    print("--- Stop --")
    vel_msg.linear.z=0
```

```
    pub_up.publish(vel_msg)
    rate.sleep()
    distance=0
def lf_rt(d,s):
    pub_lf = rospy.Publisher("bebop/cmd_vel", Twist, queue_size=10 )
    rospy.init_node('bebop_MD', anonymous=True)
    rate = rospy.Rate(10) # 10hz

    vel_msg = Twist()

    ##Init
    #linear
    vel_msg.linear.x=0
    vel_msg.linear.y=s
    vel_msg.linear.z=0
    #angular
    vel_msg.angular.x=0
    vel_msg.angular.y=0
    vel_msg.angular.z=0

    speed = abs(vel_msg.linear.y)
    distance=d #in metters
    distance=float((distance*5.0)/15.0)

    t0 = rospy.Time.now().to_sec()
    current_distance = 0

    print("--- rt_lf ---")
    while (current_distance < distance):
       #publish the velocity
       pub_lf.publish(vel_msg)
       #actual time
       t1 = rospy.Time.now().to_sec()
       #actual distance
       current_distance = speed*(t1-t0)

    print("--- Stop ---")
    vel_msg.linear.y=0
    pub_lf.publish(vel_msg)
```

```
    rate.sleep()
    distance=0


if__name__== '_main_':
    try:
        print("--- Start ---")
        #Take Off
        takeoff()
        time.sleep(1)

        #Initial Position
        lf_rt(1,0.2)
        time.sleep(0.5)
        fwd_bck(1,0.2)
        time.sleep(0.5)

        #Navigation Matrix
        for i in range(1,3): #
            lf_rt(0.5,-0.2)
            time.sleep(0.5)
            for j in range(1,5):
                up_dw(1,0.2)
                time.sleep(0.5)
                #Snapshot
            lf_rt(0.5,-0.2)
            time.sleep(0.5)
            for k in range(1,5):
                up_dw(1,-0.2)
                time.sleep(0.5)
                #Snapshot

        #Land
        land()
    except rospy.ROSInterruptException:
        pass
```