SPH ENGINEERING
SMART PLANES & HELICOPTERS

UgCS DDC

DRONE DANCE TRAJECTORY GENERATION Version 3.0

User Guide

SPH ENGINEERING
DZIRNAVU IELA 62-9, RIGA
LV-1050, LATVIA

PH: +371 25453422
PH: +1 650 681 9755
E-MAIL.: UGCS@UGCS.COM

UGCS.COM/DDC
SPH-ENGINEERING.COM

Table of Contents

Introduction

An UAV dance trajectory may be described by a list of points separated by a certain constant time interval. The trajectory is stored as a binary file and written to an UAV before the flight. Currently the maximum number of points in a trajectory is limited to 1,000 points. As a rule, a trajectory is described by 4 points per second (i.e. 0.25 s between points), but you may set any number of points per second. So, UAV covers a 1,000 point trajectory in about 4 minutes. You may increase flight time by decreasing the number of points per second and thus making a flight less smooth, or vice versa.

Each point is described by 12 bytes: X, Y, Z, R, G, and B (two-byte integers) where X, Y, and Z specify the location of the aircraft (in centimeters) relative to the center point of the dance. You may consider X, Y, and Z as local coordinates in some coordinate system where X and Y are ground coordinates and Z axis is directed up. When you plan a dance for multiple drones, they all should obviously be in the same coordinate system.

R, G, and B stand for Red, Green, and Blue colors. Each R, G, and B value must be integer, strictly between 0 and 255.

The trajectory file must have PATH extension, for example, Apm-2.PATH.

You may create the trajectory using any method, such as 3D modelling software, scripts or even game engines. All you need to do is produce a file in the proper format. In addition the trajectory should be executable on your airframe in terms of physical capability. For example, for DJI F450 airframe with RTK GPS, we recommend not to exceed the speed of 3 m/s and to keep a minimum distance of 2.5 m between drones.

Creating a dance trajectory with Blender software

Blender is a professional, free, open-source 3D computer graphics software used to create animated scenes, visual effects, works of art, 3D models, etc. It has a Python console so it is easy to get an object position and write it to a file. This approach is described in more detail below.

Making an animation with Blender falls beyond the scope of this guide, but you are offered an example of such animation.

Below is an animation of 20 spheres. The animation size is 2,600 frames, so it lasts 108 seconds in Blender. In this case you have 20 active objects in the scene with the names ranging from Sphere_0 to Sphere_19 and you want to make a show with 20 drones using this animation.

When all your objects are in the same scene you obviously have a single coordinate system for all these objects, which is also a coordinate system for your entire scene. It must be positively oriented (NB: Blender coordinate system is positively oriented), with Z axis looking up. In ArduPilot coordinate system, X axis is looking north, but you may set up dance orientation in UgCS DDC Client later. Figure 1 shows the first frame of the animation. A red arrow means X axis.

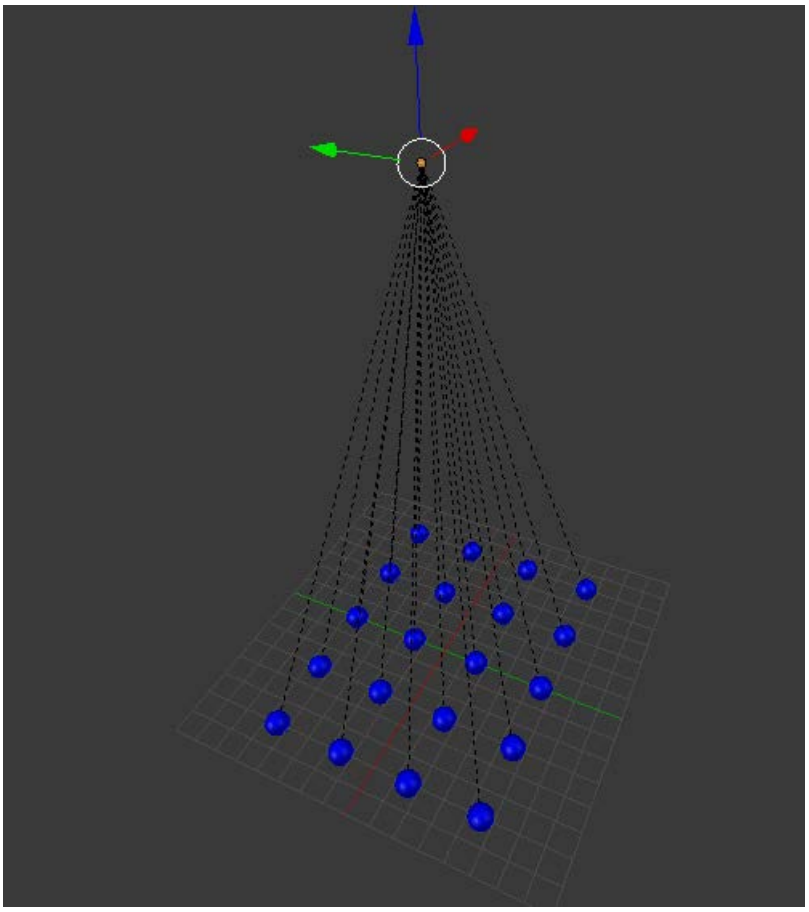

Figure 1. The first animation frame

Well, you have an animation and understand its coordinate system. Now, it is time to talk about colors. As you see, objects are changing colors during the animation. In this animation, a material diffuse color value is used to control actual drone color, therefore you must specify this value in the trajectory file.

Checking flyability

Afterwards, you must check if your show is flyable, i.e. make sure that every UAV will fly its respective trajectory precisely enough and that UAVs will not collide during your show. You can do it with Blender's Python console and some scripting. To check for potential collisions, open Blender's Python console and run this script (copy/paste the script into the console):

```
import bpy
import math
sce = bpy.context.scene
# number of uavs in our scene
number_of_uavs = 20

# treshold distance in centimeters
d_treshold = 250
# length of animations in frames
frames_count = 2600
# scale
scale = 1.0

for i in range(0, number_of_uavs-1):
    ob = bpy.data.objects['Sphere_' + str(i)]
    for f in range(sce.frame_start, frames_count):
        if ((f-1) % 3 == 0):
            sce.frame_set(f)
            x = int(ob.matrix_world.to_translation().x*100 * scale)
            y = int(ob.matrix_world.to_translation().y*100 * scale)
            z = int(ob.matrix_world.to_translation().z*100 * scale)
            for k in range(i+1, number_of_uavs):
                if (k != i):
                    ob2 = bpy.data.objects['Sphere_' + str(k)]
                    x2 = int(ob2.matrix_world.to_translation().x*100 * scale)
                    y2 = int(ob2.matrix_world.to_translation().y*100 * scale)
                    z2 = int(ob2.matrix_world.to_translation().z*100 * scale)
                    dx = x2 - x
                    dy = y2 - y
                    dz = z2 - z
                    d = math.sqrt(dx*dx + dy*dy + dz*dz)
                    if (d < d_treshold):
                        print("Danger! Distance = " + str(d) + " between " + str(i) +
" and " + str(k) + " on " + str(f) + " frame")
```

Wait for several seconds, and the following text should be displayed in the console:

Danger! Distance = 236.6833327465202 cm between 0 and 19 in frame 2293
Danger! Distance = 224.30336600238527 cm between 0 and 19 in frame 2296
Danger! Distance = 244.95918027295895 cm between 0 and 1 in frame 2299

Danger! Distance = 214.6695134386809 cm between 0 and 19 in frame 2299
Danger! Distance = 238.16170976880395 cm between 0 and 1 in frame 2302
Danger! Distance = 208.77260356665576 cm between 0 and 19 in frame 2302
Danger! Distance = 232.85617878853893 cm between 0 and 1 in frame 2305
Danger! Distance = 205.7206844242941 cm between 0 and 19 in frame 2305
Danger! Distance = 228.50820554194547 cm between 0 and 1 in frame 2308
Danger! Distance = 205.9441671910132 cm between 0 and 19 in frame 2308
...
And the minimum value:

Danger! Distance = 187.736517491936 cm between 1 and 5 in frame 2503

It is not good, since you wanted a minimum distance to be about 2.5 meters throughout the whole show. That is why, you should change a scale by 1.3. You may do it either in Blender itself or in the script. So change the scale parameter to 1.3 and run the script again:

```
# scale
scale = 1.3
```

The following 3 warnings will be displayed:

Danger! Distance = 247.3115444131147 cm between 1 and 5 in frame 2500
Danger! Distance = 244.64872777106362 cm between 1 and 5 in frame 2503
Danger! Distance = 245.86785068406158 cm between 1 and 5 in frame 2506

You believe that 245 centimeters is OK. So let the scale be equal to 1.3 in future scripts.

Next, you need to check UAV speeds using a similar script. Assume that Blender animation is running at 24 FPS and your trajectory has 4 points per second.

```
import bpy
import math
sce = bpy.context.scene
number_of_uavs = 20 # we have 20 objects
scale = 1.3 # we want to scale scene by 1.3
scene_frames = 2600 # scene frame length
speed_treshold = 3 # speed treshold in meters per second
frame_rate = 4 # our target, autopilot, framerate. Usually it's 4 FPS.
blender_frame_rate = 24 # blender framerate

l_x = 0
l_y = 0
l_z = 0
```

```
for i in range(0, number_of_uavs):
    ob = bpy.data.objects['Sphere_' + str(i)]
    for f in range(sce.frame_start, scene_frames):
        if ((f-1) % (blender_frame_rate / frame_rate) == 0):
            sce.frame_set(f)
            if (f>(blender_frame_rate / frame_rate)):
                x = ob.matrix_world.to_translation().x * scale
                y = ob.matrix_world.to_translation().y * scale
                z = ob.matrix_world.to_translation().z * scale
                dx = l_x - x
                dy = l_y - y
                dz = l_z - z
                d = math.sqrt(dx*dx + dy*dy + dz*dz)
                s = d * frame_rate
                if (s > speed_treshold):
                    print("Danger! Speed = " + str(s) + " m\s for " + str(i) + " on "
+ str(f) + " frame")
                l_x = x
                l_y = y
                l_z = z
            else:
                l_x = ob.matrix_world.to_translation().x * scale
                l_y = ob.matrix_world.to_translation().y * scale
                l_z = ob.matrix_world.to_translation().z * scale
```

You have the following output:

Danger! Speed = 3.962261911929965 m/s for 17 in frame 1741
Danger! Speed = 3.96226193855766 m/s for 17 in frame 1747
Danger! Speed = 3.9622613337829287 m/s for 17 in frame 1753
Danger! Speed = 3.9622602567461094 m/s for 17 in frame 1759
Danger! Speed = 3.962262463758294 m/s for 17 in frame 1765
Danger! Speed = 3.962261360219581 m/s for 17 in frame 1771
Danger! Speed = 3.96226023031527 m/s for 17 in frame 1777
Danger! Speed = 3.9622636203049737 m/s for 17 in frame 1783
...
Danger! Speed = 5.441835998950387 m/s for 19 in frame 733
Danger! Speed = 5.441835998950387 m/s for 19 in frame 739
Danger! Speed = 5.441828985714801 m/s for 19 in frame 745
Danger! Speed = 5.441835998950392 m/s for 19 in frame 751
Danger! Speed = 5.441827232408443 m/s for 19 in frame 757
Danger! Speed = 5.44183030069663 m/s for 19 in frame 763
Danger! Speed = 5.441830081532968 m/s for 19 in frame 769

Danger! Speed = 5.441830328092087 m/s for 19 in frame 775

It is too fast, therefore you need to reduce speeds twice. Run the Blender animation two times slower. You can do it either in Blender or in the script. All you need to do is change the blender_frame_rate parameter to 12 and run the script again.

```
blender_frame_rate = 12 # blender framerate
```

Now, you have maximum 3.38 m/s. It's OK.

Note that after you write trajectories to UAVs, you must check all speed values in **UgCS DDC** Client as well.

Generating PATH files

Now, all you need to do is generate 20 trajectory files, one file for each drone. You also need to slow down the show twice so that its duration will be about 4 minutes, and to scale it so that distances between drones at the start will be 4 meters instead of 3. Of course, you may scale the animation in Blender itself, but it can be done via a script as well.

Create the c:₩path folder.

Open Blender's Python console and run this script (copy/paste the script into the console)

```
# import blender packets
import bpy

# blender context scene
sce = bpy.context.scene
number_of_uavs = 20 # we have 20 objects
scale = 1.3 # we want to scale scene by 1.3
scene_frames = 2600 # scene frame length
frame_rate = 4 # our target, autopilot, framerate. Usually it's 4 FPS.
blender_frame_rate = 12 # blender framerate
# iterate through every drone
for i in range(0, number_of_uavs):
    # get object in scene (from Spehre_0 to Sphere_19)
    ob = bpy.data.objects["Sphere_" + str(i)]
    # create PATH file for every object
    file = open('c:/path/APM-' + str(i+1) + '.PATH', 'wb')
    # iterate through frames
    for f in range(sce.frame_start, scene_frames):
        # for every third frame get and save object position into file.
        # we should have 867 frames in output file
```

```
        if (((f-1) % (blender_frame_rate / frame_rate)) == 0):
            sce.frame_set(f)
            # get scaled position
            x = int(ob.matrix_world.to_translation().x * 100 * scale)
            y = int(ob.matrix_world.to_translation().y * 100 * scale)
            z = int(ob.matrix_world.to_translation().z * 100 * scale)
            # get color
            r = int(ob.active_material.diffuse_color.r * 255)
            g = int(ob.active_material.diffuse_color.g * 255)
            b = int(ob.active_material.diffuse_color.b * 255)
            file.write((x).to_bytes(2, byteorder='little', signed=True))
            file.write((y).to_bytes(2, byteorder='little', signed=True))
            file.write((z).to_bytes(2, byteorder='little', signed=True))
            file.write((r).to_bytes(2, byteorder='little', signed=True))
            file.write((g).to_bytes(2, byteorder='little', signed=True))
            file.write((b).to_bytes(2, byteorder='little', signed=True))
    file.close()
```

Wait half a minute and check that 20 PATH files are now in the c:₩path folder. These files should have 10,404 byte size, which corresponds to 867 frames (867*12=10,404). If you run the show at 4 points per second, it will last for 3 minutes and 37 seconds. You may now write these trajectories to UAVs using **UgCS DDC** software.