SPH ENGINEERING
SMART PLANES & HELICOPTERS

UgCS DDC

Version 3.0

INSTRUCTION

DRONE DANCE PATH GENERATION

Introduction

Dance trajectory of UAV may be described as a list of points with fixed constant time between each other. Trajectory is stored as binary file and is uploaded to UAV before flight. Current limitation of trajectory length is 1000 points. Commonly we use trajectory description with 4 points per second, i.e. 0.25 second between each point but you are free to set any FPS in general. So, UAV flies trajectory of 1000 points for about 4 minutes. You can increase flight time by decreasing FPS in cost of lesser smooth flight, or vice versa.

Each point is described with 12 bytes: X, Y, Z, R, G, B, Integers, 2 bytes each. X, Y, Z are position of UAV in centimeters from the center of DANCE scene. You can consider X, Y, Z as some local coordinates in some coordinate system, where X and Y are basis on earth plane and Z-axis is directed up. When you plan dance for multiple drones they all should be in the same coordinate system obviously.

R-G-B means Red-Green-Blue colors. Each R-G-B value should be integer, strictly between 0 and 255.

File with trajectory must have PATH extension, for example, file can be named "Apm-2.PATH".

There are no limits what way you decide to make trajectory. It can be software for 3D modelling, scripts or even game engines. All you need is to have file with correct format in the output. In addition, trajectory should be flyable with your hardware. For DJI F450 frame with RTK GPS we recommend not to exceed speed of 3 m/s and not to have UAVs closer than 2.5 meters from each other at all times.

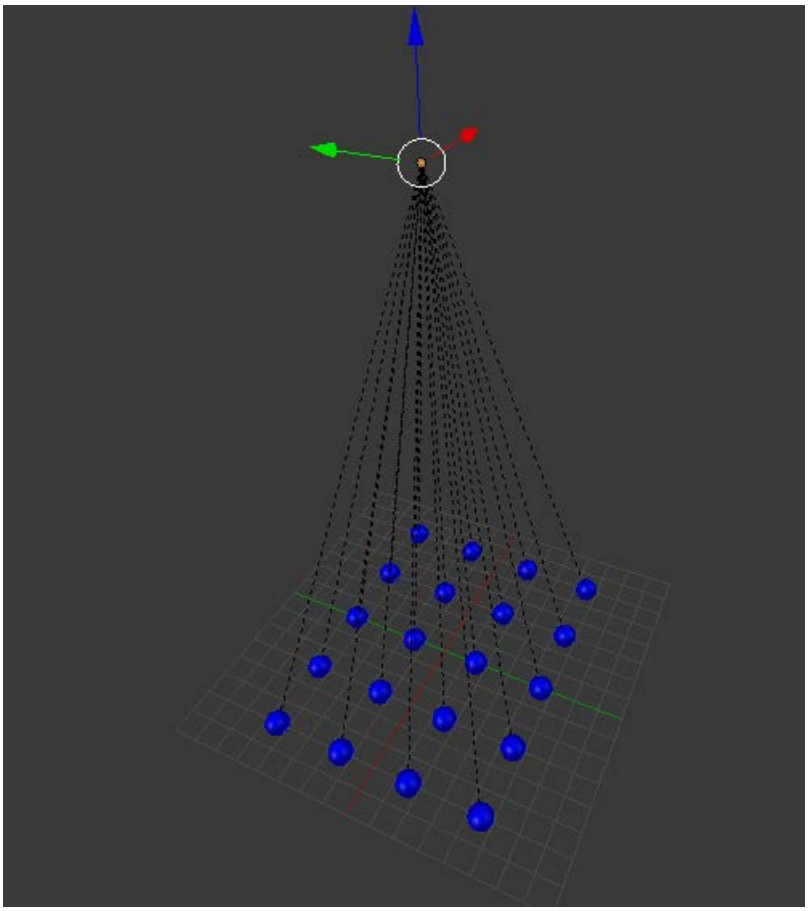
Creating dance path with Blender software

Blender is a professional, free and open-source 3D computer graphics software toolset used for creating animated scenes, visual effects, art, 3D models, etc… It has python console so it will be easy to get object position and write them into the file. Now we describe this approach in detail.

How to make animation with Blender software is out of scope of this manual, but we offer you a example of such animation.

You can see animation of 20 spheres. The size of animation is 2600 frames, so animation lasts 108 seconds in Blender. We have 20 active objects in scene with names from "Sphere_0" to "Sphere_19" and we want to make show with 20 drones from this animation.

When all your objects are in the same scene you obviously already have single coordinate system for all your scene objects and this is coordinate system of your scene itself. It has to be positive oriented (and Blender coordinate system is positive oriented) and has Z-axis direction looking up. In ardupilot system of coordinates, X-axis is looking north, but you can set up dance orientation later in **UgCS DDC** Client. On the Picture 1 you can see first frame of our animation. Red arrow means X-Axis.



Picture 1. First frame of animation.

We have the animation and understand the coordinate system. Now couple of words about colors. As you can see, objects are changing colors during animation. We use material

diffuse color value for controlling drone color in this animation and we need to save this value in trajectory as well.

Check if show is flyable

Afterwards we need to check if our show is flyable. We must be sure that every UAV will fly this trajectory precisely enough and that UAVs will not collide during show. We can do it with Blender python console and some scripting. To check potential collisions, open Blender python console and run this script (copy₩paste script into 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")
```

After several seconds following text should be displayed in the console:

Danger! Distance = 236.6833327465202 between 0 and 19 on 2293 frame
Danger! Distance = 224.30336600238527 between 0 and 19 on 2296 frame
Danger! Distance = 244.95918027295895 between 0 and 1 on 2299 frame
Danger! Distance = 214.6695134386809 between 0 and 19 on 2299 frame
Danger! Distance = 238.16170976880395 between 0 and 1 on 2302 frame
Danger! Distance = 208.77260356665576 between 0 and 19 on 2302 frame
Danger! Distance = 232.85617878853893 between 0 and 1 on 2305 frame
Danger! Distance = 205.7206844242941 between 0 and 19 on 2305 frame
Danger! Distance = 228.50820554194547 between 0 and 1 on 2308 frame
Danger! Distance = 205.9441671910132 between 0 and 19 on 2308 frame
…
And the minimum value is

Danger! Distance = 187.736517491936 between 1 and 5 on 2503 frame

It's not good, we want minimum distance to be near 2.5 meters throughout the whole show. So, let's change scale by 1.3. We can do that in Blender itself, or inside the script. We need to change scale parameter to 1.3 and run script again:

```
# scale
scale = 1.3
```

You will get 3 lines of warnings:

Danger! Distance = 247.3115444131147 between 1 and 5 on 2500 frame
Danger! Distance = 244.64872777106362 between 1 and 5 on 2503 frame
Danger! Distance = 245.86785068406158 between 1 and 5 on 2506 frame
We think that 245 centimeters is ok. So let our scale be in future scripts 1.3.

Next, let check the speeds with similar script. Assume that blender animation is running with 24 fps and our trajectory will have 4 fps.

```
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
```

Ok. We have following output:

Danger! Speed = 3.962261911929965 m₩s for 17 on 1741 frame
Danger! Speed = 3.96226193855766 m₩s for 17 on 1747 frame
Danger! Speed = 3.9622613337829287 m₩s for 17 on 1753 frame
Danger! Speed = 3.9622602567461094 m₩s for 17 on 1759 frame
Danger! Speed = 3.962262463758294 m₩s for 17 on 1765 frame
Danger! Speed = 3.962261360219581 m₩s for 17 on 1771 frame
Danger! Speed = 3.96226023031527 m₩s for 17 on 1777 frame
Danger! Speed = 3.9622636203049737 m₩s for 17 on 1783 frame

…
Danger! Speed = 5.441835998950387 m₩s for 19 on 733 frame
Danger! Speed = 5.441835998950387 m₩s for 19 on 739 frame
Danger! Speed = 5.441828985714801 m₩s for 19 on 745 frame
Danger! Speed = 5.441835998950392 m₩s for 19 on 751 frame
Danger! Speed = 5.441827232408443 m₩s for 19 on 757 frame
Danger! Speed = 5.44183030069663 m₩s for 19 on 763 frame
Danger! Speed = 5.441830081532968 m₩s for 19 on 769 frame
Danger! Speed = 5.441830328092087 m₩s for 19 on 775 frame

Too fast, we want to lower speed by 2. So, lets run blender animation 2 times slower. We can do it in Blender or in script. Just change blender_frame_rate parameter to 12 and run script again

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

We have maximum 3.38 m₩s after that. It's ok for us.

Note that after you upload trajectories to UAVs you have to check all speed values in **UgCS DDC** Client as well.

Generate PATH files

Now all we need to do is to generate 20 trajectory files, one file for each drone. In addition, we want to slow down show two times, so that the duration is close to 4 minutes and to scale it, so that distance between drones on startup would be 4 meters instead 3. Of course, we could scale animation in Blender itself, but it can be done also via script.

Create folder c:₩path

Open Blender python console and run this script (copy₩paste script into 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 in the c:₩path folder. These files should have the size of 10404 bytes which is equal to 867 frames (867*12=10404). If we run the show with 4 FPS it will last for 3 minutes and 37 seconds. These paths are ready to be uploaded to UAVs via **UgCS DDC** software.