Flying Drone Toolkit for Unity User Manual

Clockworks Games

Video tutorials are available on our YouTube channel:
https://www.youtube.com/channel/UCplB_hqldZkddN02tsih9UQ

Version History

Version 2.1 – July 2016

- Added dynamic waypoint following capabilty.

Version 2.0 – June 2016

- Enabled Flying Drone Toolkit for VR compatibility, in particular with the Oculus Rift CV1.

Version 1.2 – January 2016

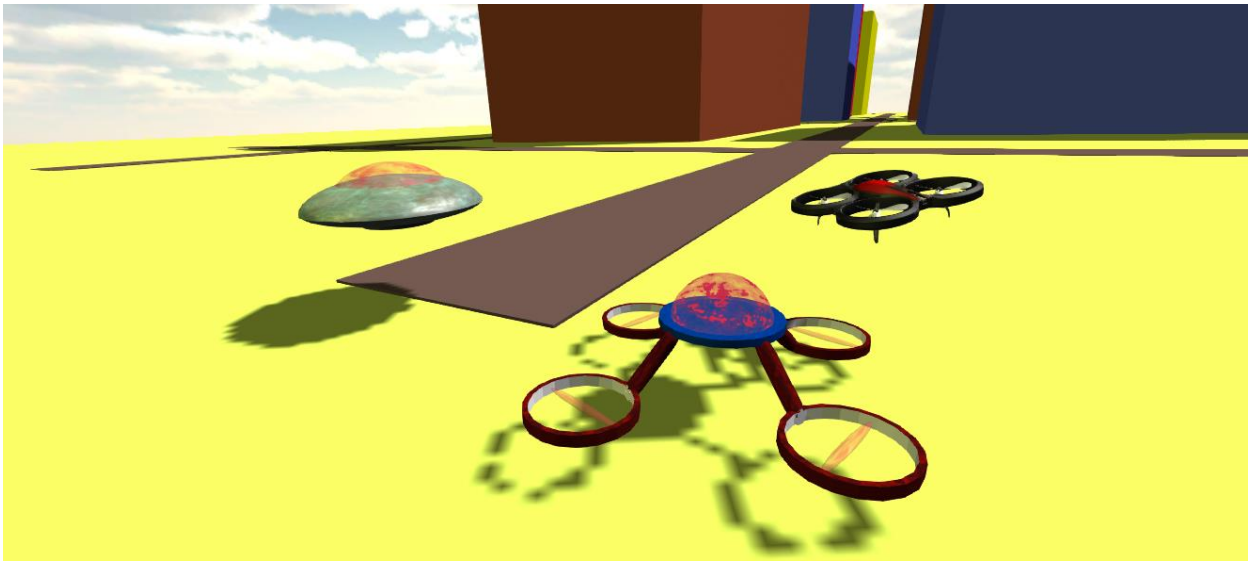
- Fixed bug that prevented drones from starting in manual mode.
- Added drone tilting capability.

Version 1.1 – March 2015

- Made compatible with Unity 5.0.

Version 1.0 – January 2015

- Initial version.



Introduction

Flying drones have been in the news quite a bit in recent years. At first, most focus had been on military drones. More recently, however, there have been reports of peaceful drones delivering pizza or online book orders. Small radio-controlled drones are available to hobbyists. Just look up "flying drone" on YouTube for many interesting examples.

The Flying Drone Toolkit makes it easy to add a flying drone to your own Unity games. The drone can be manually controlled, patrol among waypoints, or have simple following behavior. The software architecture for the drone is extensible to add more complex behaviors such as AI path planning. If there is interest in this Toolkit, I will publish new versions having examples of these more complex behaviors. The Toolkit also includes two sample flying drone 3D models. The Toolkit contains an example scene that should work right out of the box, sample 3D models, and drone scripts that you could add to your own games, or you can consider these assets as a starting point to which you can add your own 3D flying drone model or improve upon the provided scripts.

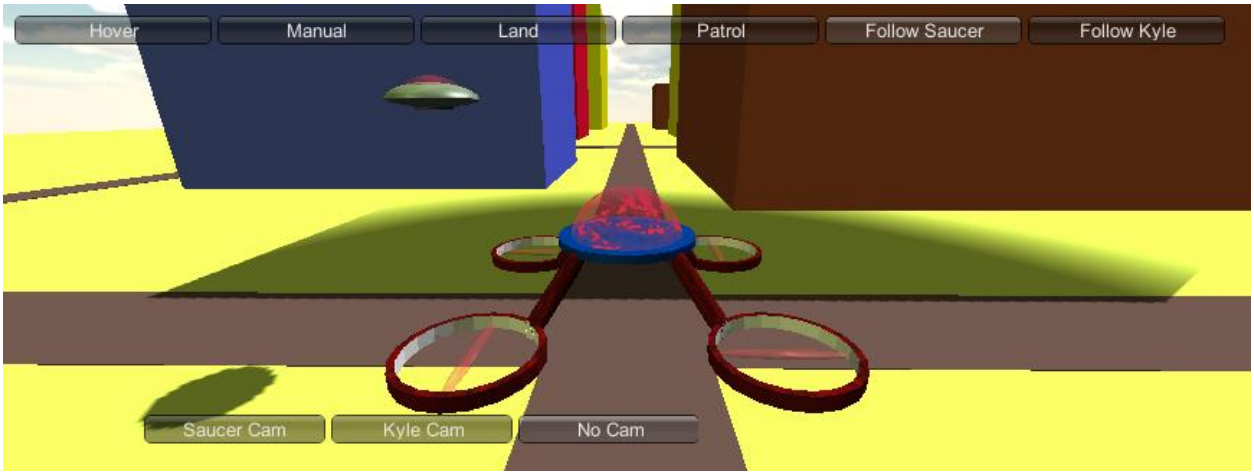
Included in the Toolkit

This Toolkit includes:

- 2 example flying drone 3D models, a Flying Saucer and the Dome Drone.
- A primary drone script implementing drone behaviors as state machines and related scripts for more detailed drone behaviors.
- A waypoint mechanism for automated patrolling behavior.
- A camera script for smoothly following the traveling drone.
- A script for allowing a drone to tilt in different ways (new in version 1.2).
- A sample scene showing flying drones in action.
- Support for Virtual Reality, in particular for the Oculus Rift CV1 (new in version 2.0).
- Scripts for dynamic waypoints to improve following behavior (new in version 2.1).
- This User Manual.
- A Programming Manual explaining how the drone behaviors could be extended.
- My commitment to make reasonable efforts to address problems you might encounter, and to improve the Toolkit.

Quick Start – Running the Drone City Sample Scene

To get a quick feel for how a Unity-based flying drone can work, try running the sample scene, Drone City. (You can preview this by this by viewing the introductory video on the Clockworks Games YouTube channel.) In case the procedure for doing this is not familiar to you, you can follow these steps.



- Create a new, empty project in Unity by selecting File menu -> New Project… Under the Create New Project, click on the Browse… button and navigate to the folder where you would like to set up your test project, then click the Create button.
- Import the Flying Drone Toolkit. This can be done from the Asset Store window. (Alternately, if you have the Unity Package file, you can import this using Assets -> Import Package -> Custom Package.)
- Select File -> Open Scene. Navigate into the FlyingDroneToolkit folder, then the DroneCityScene folder, then open the DroneCity scene file.[1]
- Click the Run button.

The sample scene consists of a simplified city in which city blocks are shown as colored blocks. A number of Flying Saucers and Dome Drones are patrolling the city. You are in control of one Dome Drone. You have command buttons across the top of screen that allow you to switch between these modes:

NOTE: For Manual mode to work (allowing you to fly the drone using the keyboard or a game controller), you need to set up the Drone Control Axes as described later in this manual. If you try using Manual mode without setting up the axes, there will be a short pause followed by an error message.

[1] If you are using Unity Pro, you may want add Antialiasing to the cameras to eliminate the "jaggy" effect from some of the straight lines. FXAA1PresetA worked well for us. This is optional, however. Don't worry if you don't have Unity Pro!

- Hover: The drone will adjust its altitude to a predetermined height and stay there.
- Manual: You can fly the drone directly using a game controller or these keys:
    - Forward / backward: w / s
    - Turn left / right: a / d
    - Move left / right: q / e
    - Up / down: r / f
- Land: The drone will lower itself to the ground and stop.
- Patrol: The drone will traverse among the waypoints. There is an invisible waypoint at most city intersections.
- Follow Saucer: The drone will follow one specific Flying Saucer.
- Follow Kyle: The drone will follow Robot Kyle.

You also have control of Robot Kyle, using the up (forward) arrow and left / right arrows.

Control buttons along the bottom of the screen control the picture-in-picture display:

- Saucer Cam: Display the Flying Saucer that can be followed in a small picture-in-picture.
- Kyle Cam: Display Robot Kyle in a small picture-in-picture.
- No Cam: No picture-in-picture.

Adding Flying Drones to Your Own Game

For this section, we are assuming that you have your own project for your own game containing at least one scene, perhaps created yourself from scratch or perhaps leveraging environment assets from the Unity Asset Store. There are many references on creating scenes in Unity, so these instructions will not describe the creation of your custom scene. We will assume that you have already done this. Then, to add the Flying Drone Toolkit:

- Import the Flying Drone Toolkit into your project as described above.
- There are several options for drone behaviors within your game. Based on what you want, different setup is required, as summarized in this table. These are not mutually exclusive; you can use any or all of these behaviors.

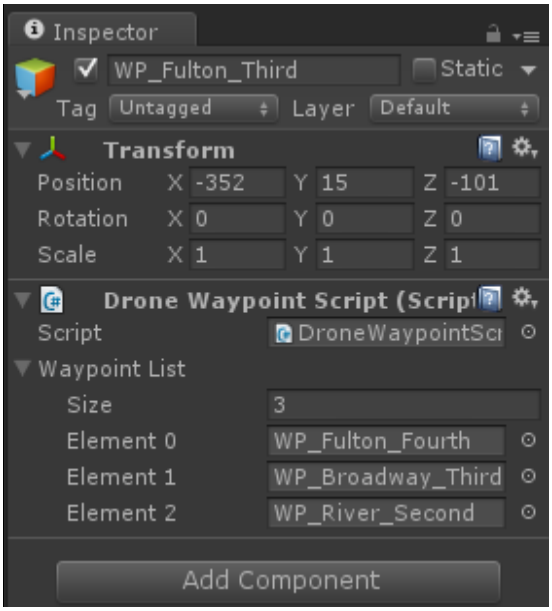
Drone Behavior	Waypoints	Drone Prefabs	Drone Control Axes	Drone Camera Script	Additional Cameras
Patrolling	X	X			
Constrained Following	X	X			
Direct Following		X			
Manual Control		X	X		
Main Camera follows drone		X		X	
Picture-in-Picture		X			X

Adding Waypoints

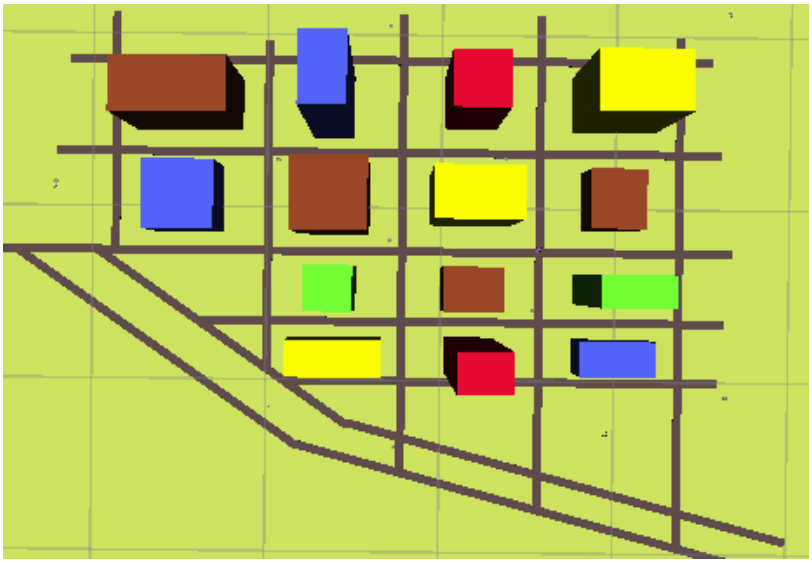
For your drone to patrol, you need to set up waypoints. These are also needed for "constrained following" in which a drone follows another drone or other game object, but is constrained to the paths connecting waypoints. Drones travel in a straight line between waypoints. Curved paths can be simulated using multiple waypoints.

To add a waypoint:

- Drag an fdWaypoint prefab onto your scene. Position it to be in the location that you want. This will be in a lateral position in X, Z. Set Y to be the altitude above the ground where you would like drones to fly. (Note that each individual drone also has a "Height Above Waypoint" parameter, so that your set of drones can be at different altitudes.)
- Once your waypoints are positioned, each of the waypoints needs to reference the waypoints that are its neighbors. To do this, each waypoint is edited in the Inspector. In the Drone Waypoint Script component, the Waypoint List size should be set to the number of neighboring waypoints. Each of those waypoints can then be dragged into Element 0, Element 1, … etc.



- It may seem confusing to keep a map of all your waypoints in your head if you have more than just a few. We suggest drawing a map of your environment, planning on paper where to position your waypoints, and assigning each a name. For example, this image shows an aerial view of Drone City. There is a waypoint at each intersection. Each street has a name.[2] Each waypoint is named based on its street intersection. From this diagram, it is straightforward to setup the waypoints correctly.



Adding Drone Prefabs

You can add any number of autonomous drones (that patrol or follow) to your game, as well as a drone that is controlled by the user.

[2] The names happen to be the names of streets in downtown Troy, NY, whose street map is very similar to that of Drone City.

To add a drone to your scene:

- Copy the fdDomeDrone prefab or the fdFlyingSaucer prefab to your Scene or Hierarchy. (There are variant versions of these prefabs in which the drones tilt. See the section on "Making Drones Tilt" for more information.)
- Position the drone prefab as you would for any game object.
- The Flying Drone Script component can be viewed in the Inspector, and has a number of parameters described in the Reference section at the end of this document. You can start by leaving most parameters set to their defaults, and determine whether any need to be adjusted based on trying the game. There are however a few that should be set as follows.
- Your drone should work for Hover, Maintain, and Land mode with no further changes.
- If you add a Next Waypoint, your drone also works for Patrol mode. If possible, select a waypoint that is closest to your drone's initial position in the game.
- If you add a Followee, your drone also works for Direct Follow mode (User Mode == Follow; User Follow Mode == Direct).
- If you add both a Next Waypoint and a Followee, your drone also works for Constrained Follow mode (User Mode == Follow; User Follow Mode == Constrained).
- If the Drone Control axes are set up according to the next section, then your drone also works in Manual mode.
- If you would like your drone to emanate a sound, you need to supply your own audio clip, which you would drag to the Audio Clip parameter of the Audio Source component.
- In the Inspector, set the Drone Mode according to the initial behavior you would like the drone to perform. The options are described in the Reference Section at the end of this document.

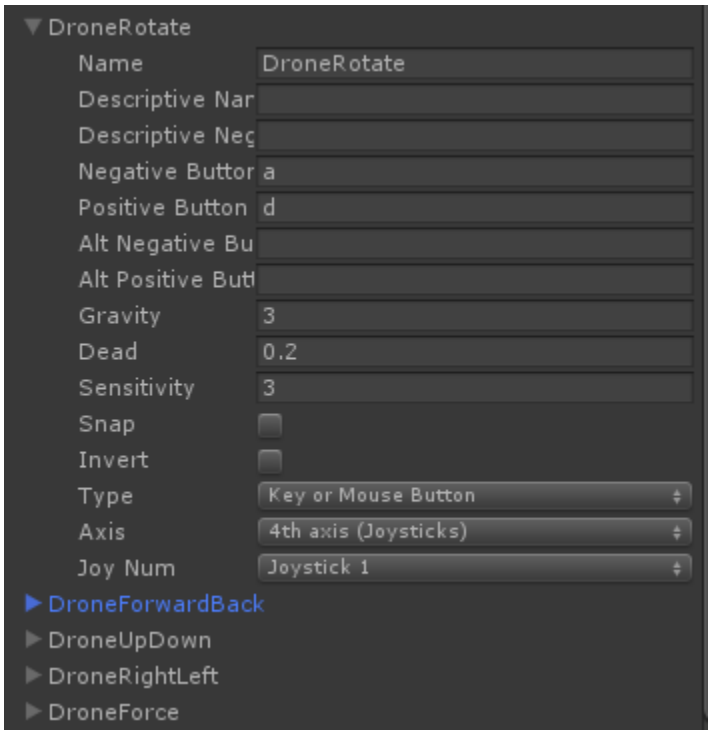
Setting up Drone Control Axes (joystick support new for version 2.0)

If you have a drone that you would like to control manually, you need to set up the control axes. Although a new project does come with Horizontal and Vertical control axes by default, for flying there can be a much larger number of axes. Luckily you can add more axes using the Unity Input Manager. The Flying Drones are controlled using five axes.

These settings will provide joystick control using an Xbox One controller. Other types of joysticks could likely be used, although the axis mapping may be different.

- Under Edit -> Project Settings -> Input, you need to define these five axes: DroneRotate, DroneForwardBack, DroneUpDown, DroneRightLeft, DroneForce. These axes can be added by increasing the Size parameter of the Axes by five. Assign these new axes to negative and positive keyboard buttons and controller axes. Suggested parameters are Gravity=3, Dead=0.2, Sensitivity=3, Snap=false, Joy Num=Joystick 1.

- An example for setting up DroneRotate is shown in the screenshot below. Rotation is assigned to the a / d keys, or to the 4th axis of a game controller. The others are set up similarly. Be sure to assign different keys and axes to each drone axis!



We recommend these key and game controller assignments:

- DroneRotate: a/d keys, X axis (as shown above)
- DroneForwardBack: s/w keys, Y axis
- DroneUpDown: f/r keys, 8th axis
- DrownRightLeft: q/e keys, 7th axis
- DroneForce: (no keys), 3rd axis

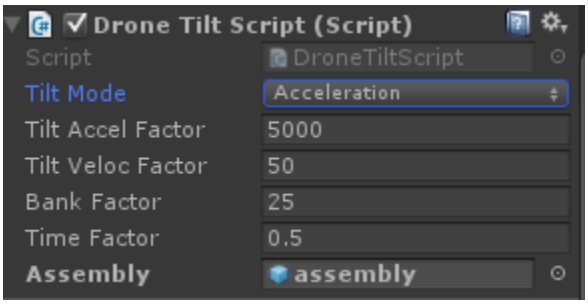
To switch between keyboard and joystick control:

- For each of the axes above, choose Type = “Key or Mouse Button” to use the keyboard, or “Joystick Axis” to use the game controller.
- When using the game controller, the suggested setting for Invert is true for DroneForwardBack and DroneUpDown. Otherwise, leave Invert unchecked.

Making Drones Tilt (new with version 1.2)

A script has been included called DroneTiltScript.cs, along with new prefabs: fdDomeDroneTilting and fdFlyingSaucerTilting. These are slightly modified versions of drones included with the Flying Drone Toolkit which use the tilting script.

The tilting script includes a few parameters that can be adjusted in the Inspector. These are described in the Reference section.



Following Dynamic Waypoints (new with version 2.1)

Drones can be made to follow other drones or any other GameObject (such as a robot on the ground), by setting User Mode to Follow and Followee to the drone or other GameObject to follow.

In versions of the Toolkit prior to version 2.1, you had a choice between two following behaviors:

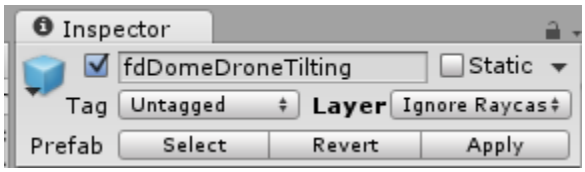
- User Follow Mode == Constrained. In this mode, the movement of the drone would be constrained to stay on the “tracks” between adjacent waypoints.
- User Follow Mode == Direct. In this mode, the drone would be free to move to any location. However, there was no obstacle avoidance to move around any obscuring object.

Starting with version 2.1 of the Toolkit, there is an additional option, which is for the GameObject being followed (e.g. the followee) to emit “dynamic waypoints” as it moves, creating a sort of memory of its path. The following drone is then capable of following around corners or through narrow passages, taking the dynamic waypoints into account.

To set this up, the DroneDynamicWaypointEmitter.cs script should be added to the GameObject being followed, and the DroneDynamicWaypointFollower.cs script should be added to the drone that is following. These components each have some settings that are documented in the Reference section at the end of this document. You may need to experiment to find the settings that work best for your situation.

Additionally, a few other configurations need to be made:

- The fdDynamicWaypoint script needs to be added to the appropriate slot in the DroneDynamicWaypointEmitter component in the Inspector.
- The GameObject being followed should have its layer set to “Ignore Raycast” so that the object being followed is not seem to be an obstructing object.



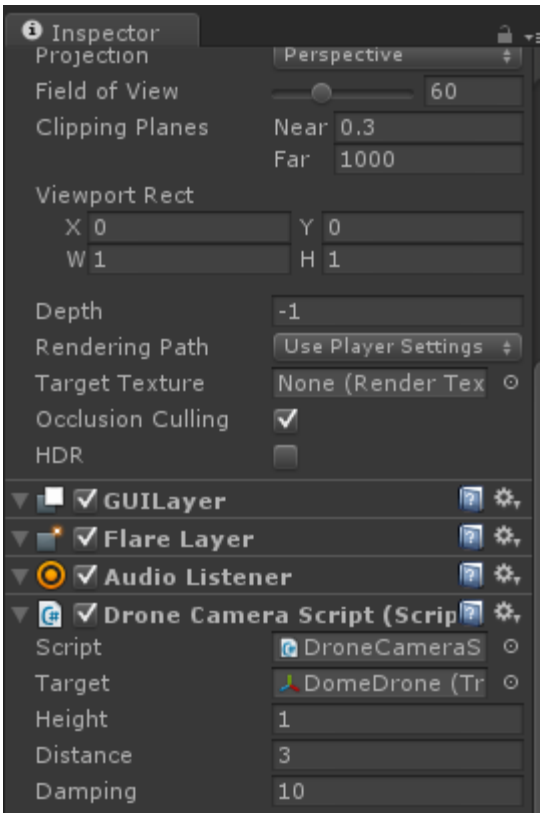
- You still have the choice between User Follow Mode of Direct or Constrained, with the same meanings as before.
- There is a new setting of User Maintain Height with options of Constant Height or Variable Height. Constant Height would be appropriate where the drone should maintain its altitude while, for example, following an object on the ground. Variable Height would be appropriate for following a flying object that is changing its altitude and maneuvering through narrow passages.

Adding the Drone Camera Script

In case you would like your main camera to follow one of your drones, a short script, DroneCameraScript.cs, is included for this purpose. Although the script is simple, it seems to work better for the flying drone than a number of “standard” third-person controller scripts that were tried. The script causes the camera to follow a drone smoothly in all types of 3D movements. Many existing 3rd person camera scripts seem to assume, for example, that the camera is following a player on the ground, and thus may not work smoothly for vertical rotations. (You may in fact find other uses for DroneCameraScript.cs beyond the Flying Drone Toolkit.)

To add DroneCameraScript.cs:

- Drag DroneCameraScript from the DroneScripts folder in the Project window onto your Main Camera in the Hierarchy.
- Click on your Main Camera and find the Drone Camera Script component in the Inspector.
- Drag the drone that you wish to follow onto the Target parameter.
- The Target and Height parameters can be changed to adjust the camera’s relative position to the drone.
- The Damping parameters adjusts the amount of lag in the camera’s movement relative to that of the flying drone. A higher number results in more rigid movement of the camera.

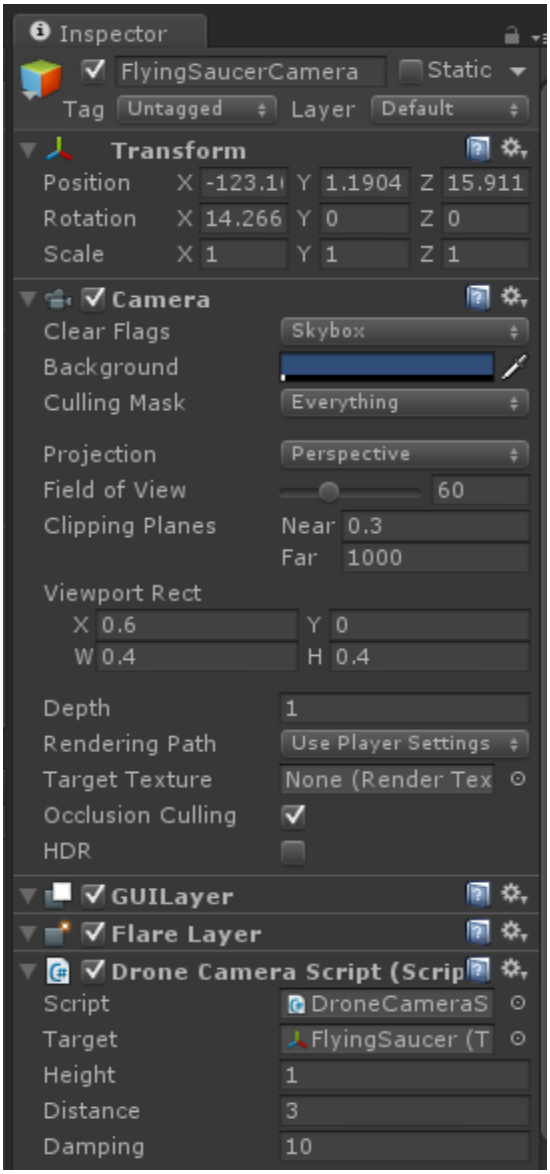


Adding Additional Cameras and Setting up Picture-in-Picture

Additional cameras can be placed in your scene, follow drones, and display as picture-in-picture. To do this:

- Add an additional camera. Remove its Flare Layer, GUILayer, and Audio Listener, so that it only has the Tranform and Camera components.
- Add the DroneCameraScript as in the previous section for the Main Camera. Add the drone to be followed as its Target.
- Within the Camera component of your new camera:
    - Adjust the Viewport Rect to be a subset of the entire screen.
    - Adjust the Depth to be greater than the Depth of the Main Camera.

For example, in this FlyingSaucerCamera, the Viewport Rect is set to display a picture-in-picture that is 0.4 times the width and height of the main display, in the bottom, right corner.



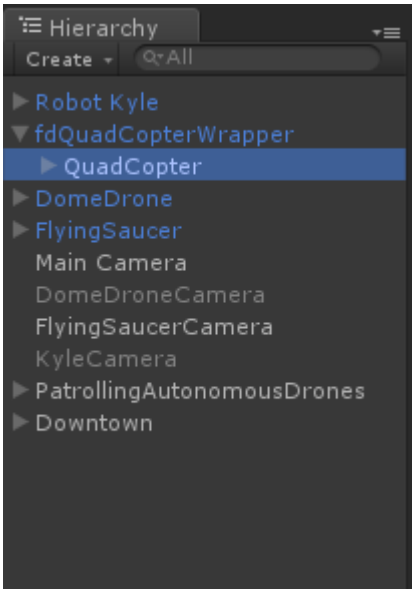
To integrate the QuadCopter Drone from sonaKiS

The Unity Asset Store has a very nice QuadCopter Drone for purchase. https://www.assetstore.unity3d.com/en/#!/content/21404.[3] In case you would like to use this asset, the Toolkit provides a prefab that makes it very easy. To use the sonaKIS QuadCopter in your scene, follow these steps:

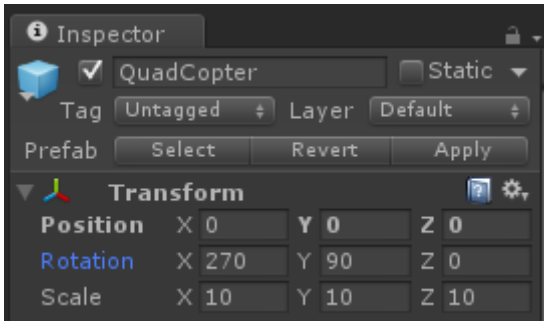
- Drag the fdQuadCopterWrapper prefab from the Prefabs folder to your Scene or Hierarchy. (Alternately, you can use fdQuadCopterWrapperTilting if you would like the quad copter to tilt.)
- Purchase the QuadCopter from the Unity Asset Store and import it into your project.

[3] This asset is a separate product from the Flying Drone Toolkit, from a separate author. It does however, work very nicely with the Flying Drone Toolkit.

- From the QuadCopter asset, drag the QuadCopter prefab as a child of the fdQuadCopterWrapper prefab in the Hierarchy. (If you are using fdQuadCopterWrapperTilting, drag the QuadCopter prefab as a child of the assembly subobject of the fdQuadCopterWrapperTilting prefab.)

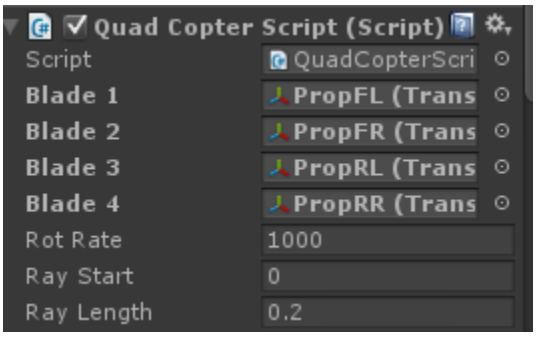


- With the QuadCopter selected in the Hierarchy, adjust the Rotation values in the Inspector as shown below. Position values should all be zero. Rotation X should be 270; rotation Y should be 90. You may also want to adjust the Scale value to change the size of the QuadCopter. Any value is fine, but all of X, Y, and Z should have the same Scale value.



- Remove the Animator component from the QuadCopter prefab. (Optional, but it is not used.)
- Expand the QuadCopter in the Hierarchy to reveal the 4 propellers: PropFL, Prop, FR, Prop RL, and Prop RR. Select the fdQuadCopterWrapper in the Hierarchy. In the Inspector, scroll to find the Quad Copter Script component. Drag and drop each of the 4 propellers into the 4 Blade slots in the Quad Copter Script, as shown below.

You can now set up the configuration of your QuadCopter as described in the “Adding Drone Prefabs” section above.



Acknowledgements

The excellent metal textures used on the Dome Drone and Flying Saucer models are from the Metal Textures Pack (https://www.assetstore.unity3d.com/en/#!/content/12949) and are used with permission of the author, http://nobiax.deviantart.com/.

The QuadCopter asset from sonaKiS (https://www.assetstore.unity3d.com/en/#!/content/21404) is very complementary to the Flying Drone Toolkit and integrates easily as described above. If you like this style of flying drone, we recommend this asset. It needs to be purchased separately.

Robot Kyle, Kyle's animation clips, and the aircraft audio clips are free assets from Unity Technologies and are included here only as part of the Drone City sample scene.

Finally, thanks to the users of the Flying Drone Toolkits who have requested new features and commented on current functionality. We are listening and working to constantly improve the Toolkit!

Reference: Flying Drone Script Parameters

The parameters described here are all accessible in the Unity Inspector.

The drone modes described below should be configured before starting a game. They can also be changed dynamically within the Inspector while the game is running. When this is done, some changes take effect immediately; others take effect the next time the drone reaches its destination.

Drone Modes

- User Mode – These control the overall behavior of the drone.
    - Maintain – The drone simply stays where it is, hovering in the sky or parked on the ground.
    - Hover – The drone is flying, and stays in place although subject to some random drifting movement.
    - Manual – The drone can be controlled by the user with control over four axes: forward/backward, up/down, left/right, rotate.
    - Land – The drone loses altitude until landing on the ground.
    - Patrol – The drone traverses among waypoints.
    - Follow – The drone follows another GameObject.
- User Travel Mode – These control how the drone travels to its destination.
  NOTE: Direct and Incremental are similar to each other. You may want to experiment to determine which works best for your situation.
    - Direct – The drone incrementally turns to face its destination and moves toward it.
    - Incremental – The drone incrementally turns to face its destination and moves toward it. This may work best for following.
    - Turn-then-Move – The drone first turns in place to face its destination, then moves laterally in a straight line to reach it. This may work best for patrolling.
- User Follow Mode
  These are specifically for Follow Mode, above.
    - Constrained – Drone is constrained to the paths between waypoints. This precludes following off the path, but also keeps the drone from crashing into obstacles, assuming the waypoint paths are clear.
    - Direct – The drone will move directly toward its followee, but will hit obstacles if any.
- User Motion Mode – These control the random drifting motion of the drone.
    - None – No random drifting motion.

    - Sine – The random motion is sinusoidal, analogous to a swinging pendulum. This mode is more compute intensive and seems to result in some issues after a running for a while.
    - Simple – The random motion is simply linear, alternating up and down. This is recommended, due to performance.
- User Maintain Height Mode – These whether the drone can vary its altitude while following other objects.
    - Constant Height – The height of the drone does not change. This is appropriate, for example, if the drone is following an object on the ground.
    - Variable Height – x Variable Height would be appropriate for following a flying object that is changing its altitude and maneuvering through narrow passages.

Other Parameters

You can change the behavior of a flying drone by experimenting with and adjusting these parameters. Generally, the default versions should be reasonable, at least as a starting point. You can experiment with the effect of these by changing the values in the Inspector while running a game.

- Force Multiplier – Controls the force on the flying drone in manual mode. A higher value will cause the drone to accelerate and travel faster. This parameters pertains directly to forward / backward force, and is further adjusted with the three following parameters.
- Up Down Factor – Adjusts the Force Multiplier above so that the up / down force can be different from the forward / backward force.
- Left Right Factor – Adjusts the Force Multiplier above so that the left / right force can be different from the forward / backward force.
- Rotate Factor – Adjusts the Force Multiplier above so that the rotational force can be different from the forward / backward force.
- Force Factor – The degree to which the acceleration function increases the force, when using a joystick.
- Hover Height – The height of the drone in hover mode. This would usually be similar in height to the waypoints.
- Dist Tol – Tolerance for how close the flying drone needs to be to its target destination.
- Wp Dist Tol – Tolerance for how close the flying drone needs to be when its target is a waypoint.
- Dot Tolerance – Tolerance for the flying drone to be facing in the correct direction. This should be set very close to but not more than 1.0.
- Height Above Waypoint – When patrolling, the distance should target this height above each waypoint. This allows patrolling drones to each be at a different altitude to avoid collisions.
- Motion Mag Max – The maximum magnitude of the seemingly random, drone drifting motion.

- Motion Time Incr – The timing or speed of the seemingly random, drone drifting motion.
- Motion Mag Height – The height at which the seemingly random motion has its full range. Below this, the motion is dampened.
- Motion Mag Delta - When the drone is this distance from the ground, it is treated as zero from ground. Should be a very small number greater than zero. You may want to adjust this, for example, if your propellers keep spinning when on the ground, but you had intended them to stop.
- Patrol New Next – If true, patrolling drone will not return directly to its previous waypoint unless it is the only choice.
- Followee – The GameObject that is followed in Follow mode. This can be set in the inspector or under script control.
- Next Waypoint – When patrolling or following in constrained mode, the Next Waypoint is maintained by the Toolkit scripts, but an initial Next Waypoint must be assigned using the Inspector.
- Prev Waypoint – The most recent waypoint reached. This is maintained by the Toolkit scripts.

These parameters are used in Direct and TurnThenMove travel mode:

- Journey Delta Vert – The amount of vertical movement in each FixedUpdate call when patrolling or following. Should be a very small number. A larger number results in faster movement.
- Journey Delta Horiz – The amount of horizontal movement in each FixedUpdate call when patrolling or following. Should be a very small number. A larger number results in faster movement.
- Journey Delta Rotate – The amount of rotational movement (in degrees) in each FixedUpdate call when patrolling or following. Should be a very small number. A larger number results in faster movement.

These parameters are used in Incremental travel mode:

- Smooth Time – The approximate time to reach the target in incremental travel mode. A smaller value will cause the drone to travel faster.
- Smooth Time Rotate – The approximate time to rotate to the correct direction in incremental travel mode. A small value will cause the drone to rotate faster.
- Smooth Max Speed – The maximum travel speed in incremental travel mode.
- Smooth Max Speed Rotate – The maximum rotational speed in incremental travel mode.

Tilting Parameters

If using the Drone Tilt Script, the following parameters are available:

- Tilt Mode provides a choice of Acceleration, Velocity, and Bank.

    - Acceleration: The tilting of drone is in the direction of acceleration. When the drone decelerations, there is negative tilt. There is no tilting when the drone rotates. This is most appropriate for quadcopter-type drones.
    - Velocity: The tilting is in the direction of the drone's movement velocity. This looks pretty good, although may be a less accurate portrayal of accurate flight physics than Acceleration tilting. There is no tilting when the drone rotates.
    - Bank: There is tilting only when the drone rotates, leaning into the direction of turn. This is most appropriate for airplane-type drones.
- Tilt Accel Factor, Tilt Veloc Factor, Bank Factor: These correspond to the three modes described above and can be adjusted for more / less tilting.
- Time Factor: Controls how quickly the turning will occur.
- Assembly: This must be connected to a game object within the drone from which the overall drone can pivot. This should already be set in the prefabs.

Dynamic Waypoint Parameters

If using the Dynamic Waypoint scripts, these parameters can be adjusted. There are separate parameters for the Dynamic Waypoint Emitter and the Dynamic Waypoint Follower.

Dynamic Waypoint Emitter

- Frequency – How often to emit a waypoint (in seconds).
- Distance Threshold – Only emit a waypoint if the object has moved at least this much since the last emitted waypoint.
- Wp Visible – True to display the waypoint (as a translucent red sphere). This is useful for debugging, but should be turned off for the finished product.
- Auto Delete – If true, the emitted waypoints will automatically be deleted after a time delay.
- Wp Lifetime – Time that each emitted waypoint should exist before being automatically deleted, if Auto Delete is true (in seconds).
- Wp Prefab – The fdDynamicWaypoint prefab must be dragged to this slot.
- Wp Offset (x, y, z) – Offset of each emitted waypoint from the position of the emitter, for example if you would like the waypoints to be always be a bit higher than an object at ground level.

Dynamic Waypoint Follower

- Cast Radius – The radius of a sphere that is "cast" from the drone to the object being followed or its emitted waypoints to determine whether there is an unobstructed path. This should be approximately the size of the drone.