

FLYING-THE-CRAZYFLIE PROJECT





This project uses the Crazyflie drone specimen, and programs it to fly autonomously. This section covers **behaviour planning** for a **single Crazyflie** using **onboard hardware**. The next project looks to integrate this functionality into the Crazyswarm framework with multiple drones. These approaches has been used with **an optic flow system** and then adapted to an **external localization technology**.

Flying the crazyflie alone is an important step as it serves to:

- Use as little functionality as possible for a particular scenario
- Create fast demos.

Various functionalities are designed and we will be able to explore them in greater depth in a series of tutorials. This page, instead, looks at what was achieved in the project. Here are the tutorials for more information.

- Crazyflie drone architecture
- Flight tips
- Trajectory generation
- Behaviour planning

 <p><u>Drone Guards and other flight tips</u></p> <p>🕒 less than 1 minute read</p> <p>Information about this drone and its framework.</p>	 <p><u>Trajectory Generation (/3)</u></p> <p>🕒 less than 1 minute read</p> <p>Developing trajectories for robots to follow.</p>	 <p><u>Flying the Crazyflie (/3)</u></p> <p>🕒 less than 1 minute read</p> <p>Information about this drone and its framework.</p>	 <p><u>Robot Behaviour Planning</u></p> <p>🕒 2 minute read</p> <p>Developing robot behaviours with an event-based planner.</p>
--	--	--	---

Each usecase has different requirements, whether simply to **reach a point**, or to do so in a way that takes into account the **environment and the constraints** we put on the drone. The approach here will therefore start with demo-able results in constrained environments, and then form a more **modular framework** that allow **more responsive, and more stable** drone usecases.

SETUP INSTRUCTIONS

The Crazyflie is assembled using [these instructions](#).

package contents

- 1x Crazyflie 2X control board with all components mounted
- 5 x CW propellers
- 5 x CCW propellers
- 6 x Motor mounts
- 1x LiPo battery (240mAh)
- 5 x Coreless DC motors
- 2x Short expansion connector pins (1x10, 2mm spacing, 8 mm long)
- 2x Long expansion connector pins (1x10, 2mm spacing, 14 mm long)
- 1x Battery holder expansion board
- 1x USB cable (only with the Crazyflie 2.1)

PYTHON TRAJECTORY PLANNER

First, we get a Crazyflie drone up in the air using a list of waypoints. Most applications require this alone. The Crazyflie framework is a **Python companion library** that can perform automated behaviours, given a set of points. This approach has been successfully coupled to an **optic flow system** which is readily attached to the drone. A demo of the procedure is available [here](#).

Note: Testing the optic flow system has revealed software errors such as erratic behaviour when navigating over crevices. This was an error detected in the Bitcraze changelog, further motivating the importance of consulting the changelog.

CUSTOM SHAPES

The python script can be further expanded. A [set of helper scripts](#) aid in generating known shapes.

Trajectory Generation (2)

- Sequence of (time, waypoint) pairs

```
genCircle.py
for t in linspace(0, T, 100):
    f.write("{}{}{}{}\n".format(
        t,
        r * cos(t + phase),
        r * sin(t + phase),
        height))
```

17

BEHAVIOUR PLANNING

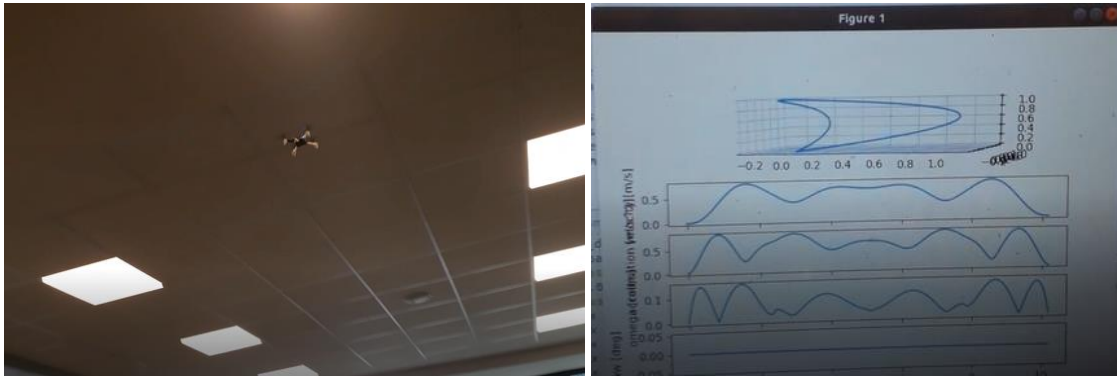
More intricate flight plans can be developed with better code organisation. Note: A quick starter on behaviour planning is covered in the behaviour planning tutorial. Testing these flight plans might require a drone, but the real drone is prone to accidents. A simulator is used for quick testing.

Download link: <https://github.com/udacity/FCND-Backyard-Flyer>



BEZIER TRAJECTORIES

Waypoint-following however, can only guarantee, well, reaching the waypoints. In the interest of stability, we might require a constant velocity, or velocity/acceleration thresholds. A Bézier curve can be used to specify the velocity over time of an object. We can then optimise a list of waypoints to a clearer route. Using a specific [github](#), we have helper scripts that serve to create more intricate trajectories. A demo of the procedure is available [here](https://youtu.be/DazGxY6YyCM) (<https://youtu.be/DazGxY6YyCM>).



Going further

Traditionally, robotics has **separated the generation of a trajectory from trajectory following**, and a neat research paper explains the variety of algorithms that can be used for this purpose. W. Hoenig, author of this paper, is incidentally the main contributor on the Crazyswarm github, a **framework for drone swarms** that offers the necessary infrastructure for **following simultaneous trajectories** (I write about it a little later).

The **use of spline trajectories** is commonplace in robotics, and a drone racing platform finds how to exploit **spline trajectories in 6 DOF**. The RPG drone racing framework explores this and other trajectory following algorithms on their github.

Going beyond with trajectory following, there has been a serious shortcoming in the last few years of drone development, and that is the poor localization offered by Bayesian filters. Research laboratories frequently use an external localization system to circumvent this. I used the Optitrack system for this purpose.

A recent paper trains a drone to follow a trajectory **with its VIO-IMU in zero-shot transfer learning**: this approach comes to show the progress autonomous drones have made over the last few years.