Crazyflie Swarm Guide

Derek Watson, Aaron Spak, Preston Kemp, Rebecca Bevers, and Silvan Li
ECE 4012 Senior Design Spring 2019
Advisor: Dr. Mick West

1. Hardware

1.2.1 Crazyflie 2.0

The Crazyflie is an affordable miniature quadcopter manufactured by Bitcraze in Sweden. Right after we purchased our Crazyflies, they released the Crazyflie 2.1, but there should be no significant difference. These drones are ideal for this type of project because they are completely open source and programmable. If you want to add features to the drone's firmware, just compile it yourself and flash it using the GUI (described in section 2.1). You probably won't because the firmware provided by Bitcraze (which is hosted at https://github.com/bitcraze/crazyflie-release/releases) is very feature-packed, including support for a rudimentary PID controller, high-level Mellinger controller, the LPS deck, LED deck, and communication with the Crazyradio. You are encouraged to update the Crazyflie to the latest release because they are constantly adding features for smoother flight. As of this writing, all the Crazyflies are on 2019.02.1. What's even cooler is that you can flash the Crazyflie wirelessly. Just connect to it in the GUI and go to Connect->Bootloader. Instructions for all this are located at https://wiki.bitcraze.io/doc:crazyflie:client:pycfclient:index.

The drones will require assembly, so before you do anything regarding repairs, please don't make the same mistake I did and read this guide: https://www.bitcraze.io/getting-started-with-the-crazyflie-2-0/. There's a reason we bought 10 drones and only used 9 for our show. Although that did make things better for us as the 10th drone served as a show model for the Expo and having 9 drones makes a symmetric square in two dimensions. For the performances, the drones will also need the LPS deck and LED ring, which can be installed on the top and bottom (respectively) of the Crazyflie using the included header through-pins. Just make sure you have things lined up by looking for the front indicator on each deck. We also bought a ton of batteries, enough to have 3 for each drone. So while one is in use, always have another charging using the micro-USB breakout boards. We just ran everything off one powered USB hub that provided 3 amps. This still wasn't enough as each charger utilizes 500 mA and 10 can be charging at once. Theoretically a battery can charge faster than it's depleted on a drone, but that depends on how much the drones are flying.

Currently our scripts (both Bitcraze and ROS) are configured to control 9 drones. We were planning to make this number dynamic based on how many are present in the script, but that becomes difficult when drones have unique ID's and channels. More on this in section 1.2.3. When you connect using the GUI, all you have to do is specify the address of the drone you want to connect to. The GUI will scan all radios, channels, and datarates for that address

and provide you with the full URI in a dropdown. Connect to that, and now you've got access to a ton of logging variables and control functions.

We also discovered that the drones are severely affected by aerodynamic effects, including wind, downwash, and ground effects. The best tips we found about flying are:

- When taking off, start slow. This means don't immediately command the drone to take off to 1 meter above the ground. It will take off too fast and may not do so vertically. If the first flight position is more than 0.5 meter off the ground, insert an intermediate point that is 0.5 meter off the ground and have it wait there for a second. This should be implemented in the performance creation framework and GUI, but you've been warned.
- Don't have one drone hover over another for too long. If they are just passing over each other at a vertical distance of 1 meter, that should be fine, but any less than that or any longer period of time and the drone on the bottom will flip over and crash, guaranteed. The downwash force on these things is incredible.
- When landing, also take it slow, but not too slow. Go down to about 0.5 meter , wait a second, and command it to land. If it lands from too high, it will crash due to the gust of wind caused by ground effect.
- Keep things slow or short. If you have fast formation changes, make sure there's enough room for error, or program in intermediate points between the start and destination.

1.2.2 Local Positioning System (LPS)

The LPS system is a combination of the "anchors," which are the devices placed around the room, and the "decks," which are add-on boards for the Crazyflie. The Crazyflie's firmware already has the ability to communicate with the decks and find its own position, at least if the system is setup and configured correctly. The Crazyflie GUI (section 2.1) is used to configure the system. You will have to enable the LPS tab in the View->Tools menu. Go to that, and you can configure the entire system through a single Crazyflie with a LPS deck. You will have to set the mode of the system to one of Two-way Ranging (TWR), Time-Difference of Arrival (TDoA) 2, or TDoA 3. TDoA 3 has some more future proof features is still in development, so we chose to use TDoA 2 as it offers enough features for this project. You will also have to manually set the positions of each anchor, which when used in combination will provide the Crazyflie with its position. Thankfully you have the option of saving these positions to a .yaml file. Once you have configured the positions, the Crazyflie will send out position updates to each anchor over its deck. Each anchor will save its position internally in ROM.

You may have noticed during configuration that each anchor also has a unique ID, 0-7. We placed the anchors so that the ID was its binary equivalent in (x, y, z). For example, anchor 0 is placed at (0, 0, 0), anchor 4 is placed at (3.05, 0, 0), and anchor 7 is at (3.05, 3.05, 2.36). These numbers come from the lengths of each side. However, actual positions depend on where you define your origin to be. The above example had the origin in a bottom corner. The performance creation tool we wrote assumes the origin is at the center of the floor. To make this easier, we just translated the positions in our software (subtract 1.5 from x and y). The tool for configuring the anchors and up to date firmware for the anchors can be found at

https://www.bitcraze.io/getting-started-with-the-loco-positioning-system/. But to do this, you need to first install the libusb driver, described in section 2.1.

All 8 anchors need to be powered, and thankfully there is a convenient way to do this. Each anchor has 4 methods of power input: barrel jack, wire terminal, microUSB, and FTDI debug solder joints. The barrel jack and wire terminals are wired together on the same side of a diode (https://wiki.bitcraze.io/_media/projects:lps:loco_node_reve.pdf), meaning the barrel jack can be an input and wire terminals an output. Then speaker wire can be used to connect the output wire terminals of one to the input terminals of another. By connecting each top corner to its respective bottom corner and running speaker wire down each vertical edge, the 8 anchors can be powered by 4 power adapters. The senior design lab has 1 amp adapters with the correct size barrel jack.

Here are some tips about the LPS system:

- Always start the drones facing the +x direction. Documentation says that the drones expect to be facing this way when they start because they cannot obtain yaw information from the LPS system.
- Start drones in these positions:

Example start

```
                    (+x)
                     |
9       5       2    |
                     |
7       4       1    |
                     |
6       3       0    |
                     |
(+y)---------------------------(tdoa 0)
```

- Use low-gauge speaker wire for the inter-anchor power connections as it provides a convenient power-ground pair.
- Orientation of the anchors does not matter. Just make sure the antenna is in the correct position.

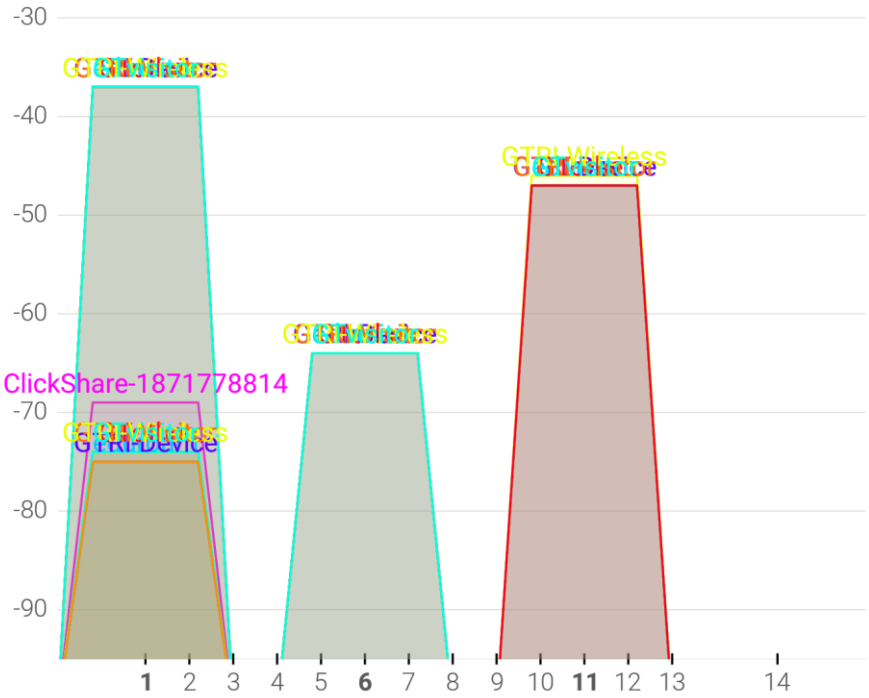
1.2.3 Crazyradio PA

The Crazyradio is an open source 2.4 GHz radio that comes with out of the box support for communicating with the Crazyflie. Bitcraze does release firmware updates for them, but I highly recommend not messing with it unless you have an SPI debugging tool. If the flash goes wrong, there is no way to rescue it. In contrast, the Crazyflie and LPS anchor have DFU modes and bootloaders to flash safely over USB.

When a radio is plugged in to a Linux machine, it can be referenced by the Crazyflie drivers (either Bitcraze or ROS) using an ID that corresponds to the order in which they were plugged in (starting with 0). The ID does not matter as the radios are not bound to a channel or datarate. In fact, the radios can communicate with multiple Crazyflies by using their unique IDs. But since there are 3 radios operating at the same time, they need to be on different channels. And since the radios operate in the same band as Wifi, we need to carefully choose the channels. The image below shows the 3 most common channels used by Wifi access points, as

well as the overlap. As you can see, the frequencies between channels 3/4 and 8/9 are completely free. According to https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/channels-frequencies-bands-bandwidth.php these channels correspond to the frequencies between 2422 – 2427 and 2447 – 2452. According to https://wiki.bitcraze.io/doc:crazyflie:client:pycfclient:index those frequencies would correspond to channels 22-27 and 47-52 on the Crazyradio. Considering channel 1 is a lot more crowded and louder, we set one radio to use channel 25, one radio to use channel 48, and one radio to use channel 50. We configured our drones to use the 2 Megabit datarate as this is more common for indoor usage and places where Wifi interference is strong. At higher datarates, range decreases, but packets are less likely to collide. Thus, the final set of URI's we decided on are:

- 0: radio://0/25/2M/E7E7E7E700
- 1: radio://0/25/2M/E7E7E7E701
- 2: radio://0/25/2M/E7E7E7E702
- 3: radio://1/48/2M/E7E7E7E703
- 4: radio://1/48/2M/E7E7E7E704
- 5: radio://1/48/2M/E7E7E7E705
- 6: radio://2/50/2M/E7E7E7E706
- 7: radio://2/50/2M/E7E7E7E707
- 8: radio://2/50/2M/E7E7E7E708



2. Standard Bitcraze Control API

This is the standard swarm control scheme provided by Bitcraze. It includes work done by Wolfgang Hoenig, a Ph.D. student at USC's Automatic Coordination of Teams (ACT) Lab,

but is mostly standalone. This means it only relies on Python and doesn't need any extra setup in ROS, Catkin, etc. It is just a standard Python 3 library and the control scripts are run from Python 3. This section will document how to get the standard control scheme installed and running. These libraries are great for testing quick algorithms and reflashing firmware.

2.1 Installation

There are a couple of different ways to gain access to Bitcraze's standard Crazyflie library. They have developed a GUI (cfclient) that includes and uses the library under the hood, and they have also provided the library (crazyflie-lib-python) for use on Windows, Linux, and MacOS. We do not provide instructions or tips for MacOS in this guide. The GUI stable releases can be found at https://github.com/bitcraze/crazyflie-clients-python/releases and a convenient installer for Windows that should work on a double click. If using Linux, it's probably best to use these stable releases, but if you don't need the GUI for its intended purpose (reflashing, setting LPS positions, testing hardware issues) then you can grab the Python library, cflib, master branch here: https://github.com/bitcraze/crazyflie-lib-python. Please be familiar with how to clone and pull from Git before doing this though.

However, Windows does not include the libusb drivers required for communicating with the Crazyradio, reflashing the Crazyflie over USB, or reflashing the LPS anchors. All of these use the same driver, but you will have to tell Windows to use that driver for each type of device. The instructions for this are located here: https://wiki.bitcraze.io/misc:usbwindows. Plug in each of the three devices one at a time and do this for each. These steps are not required for Linux, but permissions do need to be set for using the USB devices. Instructions for this are located at https://github.com/bitcraze/crazyflie-lib-python#platform-notes.

2.2 Setup

For all things Crazyflie, look at https://wiki.bitcraze.io/ for more info. Example scripts are located in the crazyflie-lib-python/examples folder. But if you install the library using pip, the library can be imported globally. To do this, run `pip install -e path/to/cflib`. Then you can import the library from anywhere, and also run the scripts located in the top level scripts folder. The nice part about these is that they're easy to understand, configure, and also offer full-featured logging tools.

2.3 Running scripts

The scripts were designed to be as simple as typing `./run_cfswarm.py performance.json` and pressing enter. Right now that's not the case if you change the number of drones in the performance. We also provided a calibrationtest.py script that individually command each drone to take off and land. The scripts work by spawning a thread for each drone and encapsulating the entire swarm in a Python class. Then in each thread, the

drone calls takeoff(height, duration). The duration doesn't actually make the drone take off slower, it just ensures the drone waits there until that time. The land(height, duration) command works similarly. Height is 0 if you want to land on the ground, higher if you're landing on a table. The send_setpoint() command takes in roll, pitch, yaw, thrust, and duration. In fact, if the drone is facing the +x axis, these values are a function of the destination point, as shown in the script. If it's not facing the +x direction, the drone will fly into a wall, guaranteed. This is why the high level commands should work better.

We began work on a script that utilizes the high-level Mellinger controller and associated go_to command, but enabling the Mellinger controller causes the drones to immediately fly into a wall. We do believe these methods will enable smoother and more reliable control of the drones, as the internal firmware will be doing the navigation as opposed to the controller PC. However, the ROS driver also implements these commands, so we recommend pursuing high level control alongside ROS integration.

3. Crazyflie ROS Control Scheme

3.1 Installation

Before installing the Crazyflie ROS driver, you must install ROS itself. The instructions for that and how to use ROS are located here: http://wiki.ros.org/ It's imperative that you have a good understanding of how ROS works, including the publish/subscribe model, services, and parameters before continuing. You must also understand rospy, the ROS interface for Python, located here: http://wiki.ros.org/rospy. ROS nodes are written in either C++ or Python, where C++ is often used for critical path code (hardware interface drivers) and Python is used for initialization, testing, or high-level control. We elected to write our choreography node in Python. The code base for the Crazyflie ROS driver is hosted at https://github.com/whoenig/crazyflie_ros. However, you don't need to install that as it is included in the Git directory as a submodule.

3.2 Setup

Our project segmented the ROS scheme from the Bitcraze scheme by placing the ROS files in the ros folder. Immediately after cloning the Git directory, you must enter the ros/src folder and run these commands:

```
git submodule init
git submodule update
```

This will pull down the latest commit for the crazyflie_ros driver and example scripts. This folder contains necessary drivers for the Crazyflie, but it conveniently omits the most important part: localization. That's what the lps-ros folder provides. The launch files contained in the launch/ folder were essential to understanding how this works, and it's what we based our choreographer launch file off of.

3.3 Running

The performances and Crazyflie resources are hosted at the top level of the Git folder. The ROS code should be compatible with the format of these files, which is common with the Bitcraze API. But to be honest, the ROS code got left behind once we switched to the Bitcraze API. The functions used in the choreographer script is very similar to the ones used in the Bitcraze script, like takeoff, go_to, land, and send_setpoint. But what those functions do under the hood is very different.

In ROS, takeoff, land, and go_to are services provided by the crazyflie driver. When you call those from another script, the crazyflie_driver receives the service call and then runs the service. ROS parameters are also very difficult to pass in from the command line. In our framework, the roslaunch_cf.py script calls the actual roslaunch command, which takes in a node and a launch file (contained in the node's launch folder). You can pass in args by using <arg_name>:=<arg_value> with roslaunch. This will pass these arguments into the launch file, which needs to have a corresponding line with: `<arg name="<arg_name>" default=""/>` to accept it at the top level and then `<param name="<arg_name>" value="$(arg <arg_name>)" />` inside the node initialization in order to pass it to that. And then the node Python script needs to have a line with `<arg_name> = rospy.get_param('~<arg_name>')`. Yes it's convoluted and messy, but that's ROS and how it's designed.

You can change the launch file by changing the input to roslaunch. As you can see in the cf_swam.launch file, in order to add multiple drones, you just copy the group of node initialization commands. The important ones are the crazyflie_driver and dd_choreography nodes. On retrospect, you probably shouldn't need the lps nodes as the crazyflie driver should be updated with LPS code. And if you use high level commands (like go_to) then the drone firmware will take care of the localization. That's up to you to figure out.

4. Simulation in V-REP

V-REP is a simulation program created by Coppelia Robotics. It is cross-platform and features support for a variety of frameworks. More details can be found on their website: http://www.coppeliarobotics.com/index.html. This installation guide will detail how to install and run V-REP with the simulation script on Ubuntu 18.04.

4.1 Installation

The simulation script requires ROS and rospy in order to function properly. If these have not been installed yet, follow the instructions in section 3.1. The script was written with Python 2.7, but later versions should work.

To install V-REP, navigate to the Coppelia Robotics download page, located at http://www.coppeliarobotics.com/downloads.html. V-REP Player is the free tier of V-REP and does not include sample models. It may also lack some functionalities the simulation script and associated plugins require. V-REP Pro Edu features the same functionality as V-REP Pro, and

is the recommended version if being used for educational purposes. After downloading the package for V-REP, extract it to the desired location. In order to start the simulation, open a terminal in the directory V-REP has been extracted to, and run the following command:

```
./vrep
```

Finally, install the V-REP ROS interface, which can be found here http://www.coppeliarobotics.com/helpFiles/en/rosInterf.htm. The source code for the plugin can be found here https://github.com/CoppeliaRobotics/v_repExtRosInterface. Create a directory in the directory where V-REP is installed, and follow the instructions listed on the Github page.

4.2 Setup

Before starting V-REP, make sure to start roscore by typing the following command into any terminal:

```
roscore
```

The V-REP ROS interface requires roscore running in order to initialize, otherwise it will be disabled until restarting V-REP. Start up V-REP and navigate to File > Load Scene. From there, navigate to the ninedrone.ttm file that was included in the swarm resources file. Open the scene and there should be nine drones in a square corresponding to their initial positions. To add more drones, either copy and paste or navigate to File > Load Model and select Drone_0.ttm.

If no modifications are being made to the scene or the script, the rest of the setup section is not necessary. However, it is recommended to read through it to understand how each simulated quadcopter functions and communicates.

Before running the simulation script, make sure all quadcopter scripts are initialized properly. Find the left toolbar in V-REP and click on the Scripts button, which should look like this in version 3.6.1.

Find all scripts labeled “Non-threaded child script (Quadcopter_target)” or “Non-threaded child script (Quadcopter_target#<number>)”. Double click on each one and find line 53 in the Lua script. It should look similar to the following line:

```
pos_update=simROS.subscribe('/target_update0','geometry_msgs/Point','update_callback')
```

If necessary, change the first argument of the subscribe function (‘/target_update0’) to the name of the topic that will be publishing commands. The topic name must be the same as the name of the topic in the Python script that is publishing.

4.3 Running

To run the simulation, first click the triangular “Play” button on the top toolbar of V-REP. Open a terminal in the directory that the simulation script is saved, and run the following command:

```
./run_sim.py <performance_file>
```

Replace <performance_file> with the json file that stores the desired performance. Since the scale of the simulator is not exactly the same as the physical Crazyflies, scaling variables in the Python script can be adjusted as required.

The simulated drones are not perfect, and will crash if directed to move too far away too rapidly. In order to prevent this make sure the simulated quadcopters are near their initial locations. The python script compensates for this by generating intermediate points with a max timestep for a smooth transition between points.