Lesson 4 Drone Integration

1. **Drone Integration Introduction**

Now that we've had a chance to control a drone in simulation, we will take a look at bringing those concepts to a real drone!

In this lesson, we will familiarize ourselves with some examples of commercially available drones. We will explore some of the features, functionality, and limitations of these aerial robots. We will focus on some example drones that are compatible with the Udacidrone interface, meaning they will be compatible with the code we've written for the simulator!

Whether or not you are following along with your very own drone, this lesson provides an introduction to some common parts of the drone ecosystem. For this lesson, we will be using an Intel Aero Ready to Fly drone as an example platform, but it is important to note that many of the features and functionality we discuss is quite typical of many drones.

We will go through some high-level topics (e.g. manual command and control), some of the core elements making up a drone system (not just the drone itself, but also pieces such as the remote control, ground stations, autonomous control, etc), and what it starts to look like as we move beyond having a drone as a toy to just fly around and start to run our own code on the drone, getting it to really do what we want and how we want it.

**Compatible Platforms**

All the python code written in the project is designed to communicate with the simulator and a real drone through the Udacidrone API. At the moment Udacidrone fully supports all PX4 drones and development for Crazyflie support is in progress. To ensure the best experience with integrating code onto a real drone, we recommend one of the following drones:

* Intel's [**Aero Ready to Fly**](https://click.intel.com/intel-aero-ready-to-fly-drone.html) drone
* Bitcraze's **[crazyflie 2.0](https://www.bitcraze.io/crazyflie-2/" \t "_blank)**

The crazyflie is the recommended platform for students who are new to flying drones. Udacity has partnered with Bitcraze to offer Flying Car Nanodegree students an exclusive discount for the crazyflie. In order to receive the Bitcraze discount offer via email, you will need to be subscribed to the Udacity email. You may check your email subscription settings in your Udacity account settings.

The following lessons are developed around the Intel Aero, however, crazyflie lessons are on their way!

**Purchasing**

**Intel Aero**

When buying the Intel Aero, there are a couple additional items to make sure you have in order to take full advantage of your Intel Aero and its onboard computer.

1. The battery needed for flight does not come with the Aero and needs to be purchased separately. Intel recommends either a 3S or a 4S LiPo battery with at least 4500mAh of capacity and an XT60 connector. Consult their [**detailed instruction manual**](https://github.com/intel-aero/meta-intel-aero/wiki/01-About-Intel-Aero#powering-the-intel-aero-ready-to-fly-drone-for-developers) for more details such as the maximum possible battery dimensions.
2. To connect the computer to a monitor, you will need a cable that has a micro-HDMI end for the Intel Aero. For example you may want a micro-HDMI to HDMI cable (e.g. a regular TV) or micro-HDMI to DVI or micro-HDMI to display port (e.g. a computer monitor).
3. To use the computer with a keyboard and mouse, you will need an OTG micro-usb cable and a USB hub with the number of ports you may need.
4. A wall power adapter to be able to use the computer without needing to run it off a battery. See Intel's [**guide for powering the intel aero**](https://github.com/intel-aero/meta-intel-aero/wiki/01-About-Intel-Aero#powering-the-intel-aero-ready-to-fly-drone-for-developers) to get an idea of the power requirements for the wall adatper. **Note that you will not be able to power the motors with these wall adapters, this is for bench work only!**
5. **Intel Aero Unboxing**

Let's start taking a look at what getting a drone like an Intel Aero looks like. In this series of lessons we will go through from unboxing our new drone through to running our backyard flyer script on it! Along the way, we will take a closer at:

* all the elements needed to fly a drone
* the specific control mechanics of the drone
* a commonly used open source ground station software
* a little bit on safety and regulations

Note that these lessons are built around the Intel Aero, however, many of the topics covered are valid for any PX4 based drone.

[**https://www.youtube.com/watch?v=\_4r9ghv6Mm8**](https://www.youtube.com/watch?v=_4r9ghv6Mm8)

**What's Inside?**

To summarize what we have found in the box:

* some instruction and getting started guides (which you should familiarize yourself with)
* the Intel Aero drone itself
* a remote control for manually piloting the drone
* a box of propellers

Note that the box does **not** include any batteries needed for flying the drone, so we will need to buy those separately. Intel recommends one of the following choices for a battery:

* 3 cell (3S, 11.1V) battery with at least 4000mAh of capacity
* 4 cell (4S, 14.8V) battery with at least 4000mAh of capacity

1. **Intel Aero First Boot**

Now that we have our drone out of the box, let's start getting it set up! We'll be walking through many of the pieces of [**Intel Aero's quick start guide**](https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/aero/intel-aero-rtf-drone-getting-started.pdf) to get familiar with all the components in the box, the elements of the Intel Aero, and how it connects to our backyard flyer. For those following along with your own Aero, make sure you also take some time to flip through that guide.

[**https://www.youtube.com/watch?v=jXKZwWdhLpA**](https://www.youtube.com/watch?v=jXKZwWdhLpA)

**Flight Modes**

[**https://www.youtube.com/watch?v=CemgWroChSE**](https://www.youtube.com/watch?v=CemgWroChSE)

**The Remote Control**

[**https://www.youtube.com/watch?v=\_Ei05vGbttg**](https://www.youtube.com/watch?v=_Ei05vGbttg)

1. **A Note on Safety**

[**https://www.youtube.com/watch?v=4-9CidAXkNI**](https://www.youtube.com/watch?v=4-9CidAXkNI)

**Drone Regulations**

Regardless of what drone you may be flying, make sure before you go out flying you have familiarized yourself with the drone regulations for your country.

**US Regulation Resources**

For those of you flying in the US, here are some links to important resources you should be aware of before flying a drone:

* [**Know Before You Fly**](http://knowbeforeyoufly.org/) - a great starting point for better understanding the different rules and regulations surrounding drones in the US.
* [**Registering your drone**](https://www.faa.gov/uas/getting_started/) - in the US, all drones above a certain weight need to be registered with the FAA and appropriately marked.
* [**Part 107**](https://www.faa.gov/uas/getting_started/part_107/) - for those of you looking to fly beyond the capacity of a hobbyist.

[**https://www.youtube.com/watch?v=gKulshbx8J0**](https://www.youtube.com/watch?v=gKulshbx8J0)

**Safety Tips**

When flying, here are some general safety tips that you should keep in mind. In general, it's always good to err on the side of caution. Most importantly, fly where and when you are comfortable!

* be aware of your surroundings, including people, obstacles, and other drones and aircraft
* yield to, and leave plenty of space with, manned aircraft
* don't fly over people
* make sure you are flying in a large enough space for your level of comfort and control
* if this is your first time flying a drone, practice flying the drone manually to get a feel for how it flies before going right to autonomous flights

There are a helpful set of [**community guidelines created by the Academy of Model Aeronautics**](https://www.modelaircraft.org/files/105.pdf) that are a great set of guidelines to follow for safety when flying.

It is also important to remember that most of these drones, including the Intel Aero, do **NOT** come with obstacle avoidance! The only obstacle avoidance is the pilot. Therefore it is very important that the pilot always keep an eye on the drone at all times, even during autonomous flight.

**Disclaimer**

PLEASE REMEMBER THAT YOUR RIGHT TO USE ANY CODE PROVIDED BY UDACITY IS SUBJECT TO OUR TERMS OF USE, THE FLYING CAR SPECIAL TERMS, HONOR CODE, AND COMMUNITY CODE OF CONDUCT. TO THE EXTENT UDACITY PROVIDES YOU CODE TO USE IN CONNECTION WITH DEMONSTRATING FLYING DRONES AND/OR OTHER AUTONOMOUS FLYING VEHICLES, SUCH CODE IS PROVIDED “AS IS” WITH NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. YOU ASSUME TOTAL RESPONSIBILITY AND THE ENTIRE RISK FOR YOUR USE OF ANY CODE AND/OR OPERATION OF ANY AUTONOMOUS FLYING/UNMANNED VEHICLES. TO THE FULLEST EXTENT ALLOWED UNDER THE LAW, UDACITY DISCLAIMS ANY AND ALL LIABILITY RELATING TO YOUR USE OF ANY CODE OR YOUR OPERATION OF ANY UNMANNED AIRCRAFT OR FLYING VEHICLE AS SET FORTH IN THE TERMS OF USE, INCLUDING, BUT NOT LIMITED TO PROPERTY DAMAGE, PERSONAL INJURY, OR DEATH. TO THE EXTENT THAT YOU USE UDACITY’S CODE AND/OR YOUR OWN CODE FOR THE PURPOSES OF THIS COURSE, INCLUDING, BUT NOT LIMITED TO, USE CONNECTED WITH A FLYING DRONE, OR AS OTHERWISE ALLOWED BY UDACITY, SUCH USE BY YOU SHALL COMPLY WITH ALL FEDERAL, STATE, PROVINCIAL, AND/OR LOCAL LAWS REGARDING THE LICENSURE, REGISTRATION, AND OPERATION OF UNMANNED AIRCRAFT OR OTHER VEHICLES. FOR EXAMPLE, BUT NOT IN ANY WAY AS A LIMITATION, THE U.S. FEDERAL AVIATION ADMINISTRATION HAS SPECIFIC RULES RELATING TO THE OPERATION OF UNMANNED AIRCRAFT. THERE ARE ALSO PRIVACY REGULATIONS THAT MAY BE APPLICABLE TO YOUR USE OF THE CODE AND/OR OPERATION OF UNMANNED AIRCRAFT AND/OR VEHICLES AND ARE REQUIRED TO COMPLY WITH THOSE LAWS.

1. **Intel Aero Setup**

In this section, we will make sure that our Intel Aero is running the most up to date software to ensure Backyard Flyer will work as best as possible.

**Checking Intel Aero Version**

[**https://www.youtube.com/watch?v=TNoqm87Yd6c**](https://www.youtube.com/watch?v=TNoqm87Yd6c)

**Logging in to Aero**

The Intel Aero broadcasts a wifi hotspot that allows us to connect to the onboard computer and both get telemetry information from the drone and access the operating system of the onboard computer.

**Connect to Wifi**

The Intel Aero is broadcasting a wifi hotspot we can log into that should start with the prefix **Aero-**.

To connect, the password is 1234567890.

**SSH Into Aero Computer**

Once connected, we can then ssh into the Intel Aero with the following information:

* **ip address:** 192.168.8.1
* **username:** root
* **password:** none

For additional details on the network configurations of the Intel Aero, check out [**this guide on Intel Aero's Network Configuration**](https://github.com/intel-aero/meta-intel-aero/wiki/08-Aero-Network-and-System-Administration).

**Updating Intel Aero**

If the Intel Aero needs to be updated to the newest version of the operating system, we can follow along with these [**initial setup instructions**](https://github.com/intel-aero/meta-intel-aero/wiki/02-Initial-Setup). For easiest setup, you will need the following items:

* keyboard and mouse
* monitor
* micro HDMI cable to connect to a monitor
* USB OTG cable

Note that each version of the Intel Aero operating system comes bundled with a new version of the PX4 autopilot that is designed to work best with that version of the OS.

**Companion Computer OS**

During the setup, it was seen that the Intel Aero is running Yocto linux on board. For integrating the backyard flyer script, there will be no need to change the operating system on the drone. It is worth noting that the Intel Aero is also [**capable of running a full version of Ubuntu**](https://github.com/intel-aero/meta-intel-aero/wiki), so for any of you interested in exploring more applications on your own, keep in mind that that is possible.

1. **Getting Familiar with QGroundControl**

[**https://www.youtube.com/watch?time\_continue=2&v=z4\_m\_IIek-Q**](https://www.youtube.com/watch?time_continue=2&v=z4_m_IIek-Q)

1. **Configure PX4**

Now that we are familiar with QGroundControl and know how to connect to our drone, let's go ahead and check sensor calibrations and make some configuration changes to the PX4 autopilot.

[**https://www.youtube.com/watch?v=q2S4JOSX9wA**](https://www.youtube.com/watch?v=q2S4JOSX9wA)

**Configuring PX4**

To recap configuring the PX4 autopilot, we:

* checked to make sure all our sensors were calibrated and ran any calibrations required
* checked to make sure our radio was properly calibrated and determined which switches control which channels
* updated the battery settings to match the battery we are using
* updated the flight control mode options to ensure we have the following options available:
  + Manual - most basic level of control, the drone will not maintain altitude or position automatically, will just stay level when no input is provided
  + Altitude - maintains current altitude
  + Position Control - uses GPS information to hold the drone in position when the pilot gives no inputs
  + Offboard - this is the mode in which we give control of the drone to our backyard flyer script
* slowed down the flight speed when in Offboard mode by changing the following parameters:
  + MPC\_XY\_CRUISE - changed to 4 m/s
  + MPC\_XY\_VEL\_MAX - changed to 5 m/s

**Reminders for Fresh Install of PX4**

If we've just updated your Intel Aero, or have reflashed a new version of PX4 onto our autopilot, we will need to make sure to complete the initial setup of your autopilot:

* ensure sensors have all been calibrated
* ensure remote control has been calibrated
* ensure the values in the battery configuration match the battery you will be using

**For Generic PX4 Autopilot**

If we are working with a generic PX4 autopilot, PX4 is not by default set up to allow offboard control, so we will need to make sure to enable that. We can find the [**detailed instructions here**](https://dev.px4.io/en/ros/offboard_control.html), but in short, we will need to set the SYS\_COMPANIONparameter to the value 357600. This value tells the PX4 that offboard control will be coming over the normal telemetry link at a baud rate of 57600. For details on the options available for the parameter, we can [**look at PX4's documentation on the parameters**](https://dev.px4.io/en/advanced/parameter_reference.html#SYS_COMPANION).

**Disclaimer**

PLEASE REMEMBER THAT YOUR RIGHT TO USE ANY CODE PROVIDED BY UDACITY IS SUBJECT TO OUR TERMS OF USE, THE FLYING CAR SPECIAL TERMS, HONOR CODE, AND COMMUNITY CODE OF CONDUCT. TO THE EXTENT UDACITY PROVIDES YOU CODE TO USE IN CONNECTION WITH DEMONSTRATING FLYING DRONES AND/OR OTHER AUTONOMOUS FLYING VEHICLES, SUCH CODE IS PROVIDED “AS IS” WITH NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. YOU ASSUME TOTAL RESPONSIBILITY AND THE ENTIRE RISK FOR YOUR USE OF ANY CODE AND/OR OPERATION OF ANY AUTONOMOUS FLYING/UNMANNED VEHICLES. TO THE FULLEST EXTENT ALLOWED UNDER THE LAW, UDACITY DISCLAIMS ANY AND ALL LIABILITY RELATING TO YOUR USE OF ANY CODE OR YOUR OPERATION OF ANY UNMANNED AIRCRAFT OR FLYING VEHICLE AS SET FORTH IN THE TERMS OF USE, INCLUDING, BUT NOT LIMITED TO PROPERTY DAMAGE, PERSONAL INJURY, OR DEATH. TO THE EXTENT THAT YOU USE UDACITY’S CODE AND/OR YOUR OWN CODE FOR THE PURPOSES OF THIS COURSE, INCLUDING, BUT NOT LIMITED TO, USE CONNECTED WITH A FLYING DRONE, OR AS OTHERWISE ALLOWED BY UDACITY, SUCH USE BY YOU SHALL COMPLY WITH ALL FEDERAL, STATE, PROVINCIAL, AND/OR LOCAL LAWS REGARDING THE LICENSURE, REGISTRATION, AND OPERATION OF UNMANNED AIRCRAFT OR OTHER VEHICLES. FOR EXAMPLE, BUT NOT IN ANY WAY AS A LIMITATION, THE U.S. FEDERAL AVIATION ADMINISTRATION HAS SPECIFIC RULES RELATING TO THE OPERATION OF UNMANNED AIRCRAFT. THERE ARE ALSO PRIVACY REGULATIONS THAT MAY BE APPLICABLE TO YOUR USE OF THE CODE AND/OR OPERATION OF UNMANNED AIRCRAFT AND/OR VEHICLES AND ARE REQUIRED TO COMPLY WITH THOSE LAWS.

1. **Modifying Backyard Flyer**

**Updating Your Project Code**

In this section we will update the project code to be able to control a real drone.

While there is really only one line in the code that needs to change for this to work (the connection to PX4), we will be making several modifications to the project code for safety.

We will be making 3 different changes to our backyard flyer script in order to run properly and safely on our drone:

* **updating the connection type** - here we will change the connection being used to connect to a real drone instead of the simulator.
* **modifying the state machine** - for safety, we will slightly modify the state machine to give the pilot more control over the starting and stopping of the script.
* **shifting the coordinates of the box** - with a real drone we can't reset the location of (0,0,0) of our world, so we will shift our box coordinates to be centered on the local coordinates of the takeoff position.

**Updating Connection**

In our backyard flyer script, the connection being passed to Drone is MavlinkConnection('tcp:{0}:{1}'.format(args.host, args.port), threaded=False, PX4=False), which is a TCP connection to the simulator passing data as Mavlink messages.

When connecting to the Intel Aero, we will need to change the parameters passed into the connection class to set up a connection to the drone. For the Intel Aero, we will need a UDP connection ('udp:ip\_address:port) and tell the MavlinkConnectionclass we are connecting to a PX4 autopilot, by setting PX4=True. The modified connection class will be:

conn = MavlinkConnection('udp:192.168.1.2:14550', PX4=**True**, threaded=**False**)

For the Intel Aero, the IP address specified is the IP address of our computer when connected to the network created by the Intel Aero. To determine our IP address, we can:

* **Windows:** open the command prompt and type ipconfig. Look for the IP address for the wireless adapter.
* **OSX and Linux:** open the terminal and type ifconfig. Look for the IP address for the wireless adapter.

**For Generic PX4**

If we are setting up a generic drone running the PX4 autopilot, our connection to the drone is most likely over a dedicated telemetry link connected as a serial device on our computer. If that is the case, instead of 'udp:ip\_address:port', we can pass in 'serial\_port,baud\_rate'.

**Documentation for More Details**

For more details into the workings of the connection class and different parameters, check out the **[Udacidrone documentation on the connection class](https://udacity.github.io/udacidrone/docs/connection-api.html" \t "_blank)**.

**Modifying the State Machine**

Currently our project code handles all the commands from taking control of the drone, to arming, through the flight and disarming. To give us, the pilot, more control over the environment and the running of the script, we will modify our project code to no longer handle the arming or disarming transition. Instead, that will be up to the pilot to do manually. This is done so the pilot can ensure everything is properly set up and that the flight environment is safe before letting our script control the drone.

Also note that at all times during the flight we have the ability to take back control of the autopilot by flipping the switch on the remote control back to manual control from offboard! This is very important and the pilot should be ready to do this at all points during the flight just in case something unexpected happens!

First, we will remove the handling of the arming transition, by modifying the checks being made when in the MANUAL state. We will change the code to passively wait for the drone's state to be updated by the pilot in order to advance our current flight state.

The current solution state\_callback is:

**def** **state\_callback**(self):

**if** self.in\_mission:

**if** self.flight\_state == States.MANUAL:

self.arming\_transition()

**elif** self.flight\_state == States.ARMING:

**if** self.armed:

self.takeoff\_transition()

**elif** self.flight\_state == States.DISARMING:

**if** **not** self.armed **and** **not** self.guided:

self.manual\_transition()

And once we make the necessarily changes, it becomes:

**def** **state\_callback**(self):

**if** self.in\_mission:

**if** self.flight\_state == States.MANUAL:

*# now just passively waiting for the pilot to change these attributes*

*# once the pilot changes, need to update our internal state*

**if** self.guided:

self.flight\_state = States.ARMING

**elif** self.flight\_state == States.ARMING:

**if** self.armed:

self.takeoff\_transition()

**elif** self.flight\_state == States.DISARMING:

**if** **not** self.armed **and** **not** self.guided:

self.manual\_transition()

Next we will need to remove the disarming transition and manual transitions made by the script. Once again, this is so that the pilot is the one who knowingly takes back control of the drone and disarms the motors.

For this change, we will modify both the velocity\_callback and the state\_callback. We will be completely removing all the code from the velocity\_callback as we are no longer interested in triggering the disarming transition. In your backyard flyer solution, we handled stopping the mission during the disarming transition, here, we will add some checks to stop the mission once the pilot has taken back control. In order to achieve this, we will add checks on the drone's state when in the LANDING flight state.

Our modified set of callbacks will now look like:

**def** **velocity\_callback**(self):

**pass**

**def** **state\_callback**(self):

**if** self.in\_mission:

**if** self.flight\_state == States.MANUAL:

*# now just passively waiting for the pilot to change these attributes*

*# once the pilot changes, need to update our internal state*

**if** self.guided:

self.flight\_state = States.ARMING

**elif** self.flight\_state == States.ARMING:

**if** self.armed:

self.takeoff\_transition()

**elif** self.flight\_state == States.LANDING:

*# check if the pilot has changed the armed and control modes*

*# if so (and the script no longer in control) stop the mission*

**if** **not** self.armed **and** **not** self.guided:

self.stop()

self.in\_mission = **False**

**elif** self.flight\_state == States.DISARMING:

*# no longer want the vehicle to handle the disarming and releasing control*

*# that will be done by the pilot*

**pass**

**Shifting the Coordinates**

The last change we will need to make to your Backyard Flyer script is handling the fact that we cannot set the local position (0,0,0) coordinate to wherever we would like. Therefore, when we create our set of coordinates to fly our box, we will shift the box to have the first corner be at the drone's current local position. We achieve this by modifying our calculate\_box function from:

**def** **calculate\_box**(self):

print("Setting Home")

local\_waypoints = [[10.0, 0.0, 3.0], [10.0, 10.0, 3.0], [0.0, 10.0, 3.0], [0.0, 0.0, 3.0]]

**return** local\_waypoints

to:

**def** **calculate\_box**(self):

print("Setting Home")

cp = np.array([self.local\_position[0], self.local\_position[1], -self.local\_position[2]]) *# get the current local position -> note we need to change the sign of the down coordinate to be altitude*

local\_waypoints = [cp + [10.0, 0.0, 3.0], cp + [10.0, 10.0, 3.0], cp + [0.0, 10.0, 3.0], cp + [0.0, 0.0, 3.0]]

**return** local\_waypoints

**Almost There!**

Now that we have our backyard flyer solution properly modified for running on the Aero, let's move on to test running the script and flying it for real!

Once again keep in mind that most of these drones, including the Intel Aero, do **NOT** come with obstacle avoidance! The only obstacle avoidance is the pilot, therefore it is very important that the pilot always keep an eye on the drone at all times, even during autonomous flight.

**Disclaimer**

PLEASE REMEMBER THAT YOUR RIGHT TO USE ANY CODE PROVIDED BY UDACITY IS SUBJECT TO OUR TERMS OF USE, THE FLYING CAR SPECIAL TERMS, HONOR CODE, AND COMMUNITY CODE OF CONDUCT. TO THE EXTENT UDACITY PROVIDES YOU CODE TO USE IN CONNECTION WITH DEMONSTRATING FLYING DRONES AND/OR OTHER AUTONOMOUS FLYING VEHICLES, SUCH CODE IS PROVIDED “AS IS” WITH NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. YOU ASSUME TOTAL RESPONSIBILITY AND THE ENTIRE RISK FOR YOUR USE OF ANY CODE AND/OR OPERATION OF ANY AUTONOMOUS FLYING/UNMANNED VEHICLES. TO THE FULLEST EXTENT ALLOWED UNDER THE LAW, UDACITY DISCLAIMS ANY AND ALL LIABILITY RELATING TO YOUR USE OF ANY CODE OR YOUR OPERATION OF ANY UNMANNED AIRCRAFT OR FLYING VEHICLE AS SET FORTH IN THE TERMS OF USE, INCLUDING, BUT NOT LIMITED TO PROPERTY DAMAGE, PERSONAL INJURY, OR DEATH. TO THE EXTENT THAT YOU USE UDACITY’S CODE AND/OR YOUR OWN CODE FOR THE PURPOSES OF THIS COURSE, INCLUDING, BUT NOT LIMITED TO, USE CONNECTED WITH A FLYING DRONE, OR AS OTHERWISE ALLOWED BY UDACITY, SUCH USE BY YOU SHALL COMPLY WITH ALL FEDERAL, STATE, PROVINCIAL, AND/OR LOCAL LAWS REGARDING THE LICENSURE, REGISTRATION, AND OPERATION OF UNMANNED AIRCRAFT OR OTHER VEHICLES. FOR EXAMPLE, BUT NOT IN ANY WAY AS A LIMITATION, THE U.S. FEDERAL AVIATION ADMINISTRATION HAS SPECIFIC RULES RELATING TO THE OPERATION OF UNMANNED AIRCRAFT. THERE ARE ALSO PRIVACY REGULATIONS THAT MAY BE APPLICABLE TO YOUR USE OF THE CODE AND/OR OPERATION OF UNMANNED AIRCRAFT AND/OR VEHICLES AND ARE REQUIRED TO COMPLY WITH THOSE LAWS.

1. **Let’s go Fly!**

[**https://www.youtube.com/watch?time\_continue=3&v=XPH-WR0i-Xs**](https://www.youtube.com/watch?time_continue=3&v=XPH-WR0i-Xs)

1. **Crazyflie Introduction**

**Getting Started With the Crazyflie**

First things first, we will follow along with Bitcraze's [**detailed walkthrough**](https://www.bitcraze.io/getting-started-with-the-crazyflie-2-0/) to get our crazyflie all set up and ready to fly. Out of the box the crazyflie may not have the most up to date firmware, so we will make sure to follow Bitcraze's guide to making sure our crazyflie is completely up to date.

When following the guide, it's important to make sure we've installed the computer client and the necessary drivers for the antenna, as we will need to use both of these as we further configure the crazyflie and run our backyard flyer script on it.

**Additional Setup**

In addition to the setup steps outlined in the walkthrough, we want to configure our crazyflie radio connection to have a bit of a higher bandwidth. This will help to reduce some of the latency on getting information from the crazyflie when we are running the backyard flyer script.

To do so, we will open the client, navigate to Connect, select the Configure 2.0option and change the bandwidth from 250k to 2M.

**A Note on Perception**

The crazyflie is not equipped with GPS and therefore needs alternative means to be able to know its position information. The FlowDeck enables the crazyflie to determine position, however it does pose a few limitations that will need to be considered when controlling the crazyflie with the backyard flyer script.

With the FlowDeck, the Crazyflie uses optical flow to measure velocity to be used for a position estimate. This will drift over time, so the crazyflie may not be capable of doing very long straight flight segments when flying autonomously. This also means it doesn't really have an absolute frame of reference, which means it flies in a local XYZ frame, not an NED frame. In short, this means that when flying the backyard flyer, a 1m change North, will really just be a 1m change forward from the initial position and heading. This is the same for all subsequent commands (e.g. a 1m change E won't necessarily go E, but will go to the right 1m).

What was altitude is now height above the ground as the crazyflie uses a small laser range finder to determine its height above the ground. This results in 2 behaviors of importance. First, the sensor doesn't have unlimited range, therefore it is possible to fly "too high", being above the sensor's range and no longer being able to get good information. Second, it means that the crazyflie will follow the contours of the environment, meaning if we command a height of 1m but have the path go over a table, the crazyflie will increase it's height above the ground such that it is 1m above the table (its local perception of the ground).

1. **Crazyflie Backyard Flyer**

**Backyard Flyer Modifications**

Just like with the Intel Aero and the simulator, the crazyflie is also able to be controlled (to an extent) through the Udacidrone API.

We will need to make the following modifications to our backyard flyer script to be able to control our crazyflie:

1. update the connection
2. modify arming and disarming flow
3. update waypoints
4. update takeoff target altitude
5. update waypoint acceptance thresholds

**0. Upgrade udacidrone**

You'll want to make sure you have the latest version of udacidrone. You can install the latest version by activating your virtual environment and then doing a pip install -U git+https://github.com/udacity/udacidrone.git

**1. Update Connection**

The crazyflie's default firmware uses it's own communication protocol, Crazy RealTime Protocol (CRTP), instead of Mavlink, therefore we will need to change to using the CrazyflieConnection that can be found in Udacidrone starting with version 0.3.0 (**NOTE: you may have to update your version of Udacidrone**).

1. add an import for CrazyflieConnection

**from** udacidrone.connection **import** CrazyflieConnection

1. At the bottom of the backyard\_flyer.py script, we will replace our connection object

*# replace*

conn = MavlinkConnection('tcp:{0}:{1}'.format(args.host, args.port))

*# with*

conn = CrazyflieConnection('radio://0/80/2M')

This will create a connection to the crazyflie. The input string is the URI of the crazyflie, which is defined as a string formated as 'radio://interface id/interface channel/interface speed. For our setup, we have kept the default interface id and interface channel, but have increased the speed from the default value of 250K to 2M. Each of these values can be adjusted through the crazyflie desktop client. In an area with multiple crazyflie's flying around, it is best to change the interface id and channel to make sure each crazyflie is unique.

**2. Modify Arming and Disarming**

The idea of being armed or in guided mode does not exist with the crazyflie, therefore we will need to modify those parts of the flow of the script. Even more generally, the concept of state for the crazyflie is very different than what is used with the simulator or with PX4. As a result, the state\_callback() callback will never be called! Since the state callback was responsible for takeoff transition, we will need to modify another one of the callbacks to handle that purpose.

**Arming / Takeoff**

We will choose to augment the local position callback with code needed to be able to issue the takeoff command as needed, by adding the following code to the top of the callback:

def local\_position\_callback(self)

if self.flight\_state == States.MANUAL:

self.takeoff\_transition()

...

Once the first local position message is received the takeoff command will be issued, since the drone starts in the MANUAL state. Note that we've made the takeoff altitude much smaller than what was used in the simulator since we will be flying in an indoor environment.

**Disarming / Ending Mission**

The same problem exists at the other end of the flight; with no armed and guidedinformation, we don't know when to consider the flight complete and the mission over. Instead we will use the landing condition as the end of the flight. To do this, we will **REPLACE** velocity\_callback() with the following:

**def** **velocity\_callback**(self):

**if** self.flight\_state == States.LANDING:

**if** abs(self.local\_position[2] < 0.01):

self.manual\_transition()

We will still use the manual transition function as it contains all the code to consider the flight as completed and stop the connection and the script.

**3. Update Waypoints**

In the simulator, we had the ability to reset our "zero" position for the world to our current position, allowing us to command a 10 meter box with fairly straightforward commands. This is not possible with the crazyflie, so we will modify the coordinates of the box to take into account the position of the drone at takeoff (when the box coordinates are created). To do this, we will modify calculate\_box() to be:

**def** **calculate\_box**(self):

cp = self.local\_position

cp[2] = 0

local\_waypoints = [cp + [1.0, 0.0, 0.5], cp + [1.0, 1.0, 0.5], cp + [0.0, 1.0, 0.5], cp + [0.0, 0.0, 0.5]]

**return** local\_waypoints

Notice we've set each of the coordinates to simply be shifted by the current position of the crazyflie when the calculate box function is run. Also notice, since we are flying out crazyflie inside, we have changed the coordinates to be a much smaller box, here a 1m box at an altitude of 0.5m.

**4. Update Takeoff Target Altitude**

For that same safety reason, we've also changed the target\_altitude in the takeoff transition to match the 0.5m flight altitude for the box. **If you have your own crazyflie and are planning to fly it indoors, we highly recommend changing the size of your box to make sure it can fly comfortably in the space you are using!**

**5. Update Waypoint Acceptance Thresholds**

When flying the backyard flyer in the simulator, the box was 10 meters on each side, so an acceptance threshold for a waypoint of 1 meter was reasonable. Now that we've reduced the box size to something more appropriate for inside (e.g. 1 meter on a side), we need to update the acceptance thresholds for waypoints to ensure the drone doesn't preemptively transition from one waypoint to another.

In the local\_position() callback, you can change the waypoint acceptance threshold by changing the following line:

**def** **local\_position\_callback**(self):

...

*# change this line:*

**if** np.linalg.norm(self.target\_position[0:2] - self.local\_position[0:2]) < 1.0:

...

*# to this line:*

**if** np.linalg.norm(self.target\_position[0:2] - self.local\_position[0:2]) < 0.2:

...

You may find you also need to change it for the acceptance on velocity to trigger than landing transition:

**def** **local\_position\_callback**(self):

...

*# change this line:*

**if** np.linalg.norm(self.local\_velocity[0:2]) < 1.0:

...

*# to this line:*

**if** np.linalg.norm(self.local\_velocity[0:2]) < 0.5:

...

Note here we've adjusted the threshold to 0.2 meters for waypoints and 0.5 m/s for landing. You may find you need to play around with that threshold based on the size of the box you are flying.

**Flying!**

And now we are ready to fly our crazyflie! To run the script we will:

1. Plug in the crazyflie and set it on the ground, with the front pointed in the desired direction. This starting position is the bottom left hand corner of our box, so we will make sure to point it towards our open space designated for flying.
2. Plug in the Crazyradio into the computer
3. Make sure you're in the fcnd workspace, and then run the script, python backyard\_flyer.py
4. We will see the connection complete, a short pause (this is for the sensors to calibrate) and then the crazyflie should takeoff and complete the mission!
5. **Crazyflie Keyboard Control**

Having played around with the crazyflie and flying it with the phone control, we know it is definitely not the easiest of drones to fly. Using the crazyflie desktop client, the radio and a game controller is one way to get a better level of control on the crazyflie. For another way, we have written a script to allow for keyboard control of the crazyflie, using the same controls from the simulator.

Bitcraze has a [**python API for the crazyflie**](https://github.com/bitcraze/crazyflie-lib-python)that we've leveraged to be able to provide a method of much better control over the crazyflie than the mobile app does.

Below is the code for the script:

"""

This script contains a keyboard controller using the MotionCommander.

Info on API element used:

https://github.com/bitcraze/crazyflie-lib-python/blob/master/cflib/positioning/motion\_commander.py

"""

**import** logging

**from** pynput **import** keyboard

**import** cflib.crtp

**from** cflib.crazyflie.syncCrazyflie **import** SyncCrazyflie

**from** cflib.positioning.motion\_commander **import** MotionCommander

URI = 'radio://0/80/2M' *# ENSURE THIS MATCHES YOUR CRAZYFLIE CONFIGURATION*

*# Only output errors from the logging framework*

logging.basicConfig(level=logging.ERROR)

**class** **KeyboardDrone**:

**def** **\_\_init\_\_**(self, mc):

self.mc = mc

self.velocity = 0.75

self.ang\_velocity = 120

self.sleeptime = 0.5

*# self.max\_hight = 0.8*

*# self.hight = 0.0*

print('Press u for taking off!')

**def** **on\_press**(self, key):

**if** key.char == 'w':

self.mc.start\_forward(self.velocity)

**if** key.char == 'u':

self.mc.take\_off(0.3)

**if** key.char == 's':

self.mc.start\_back(self.velocity)

**if** key.char == 'a':

self.mc.start\_left(self.velocity)

**if** key.char == 'd':

self.mc.start\_right(self.velocity)

**if** key.char == 'c':

self.mc.start\_down(self.velocity)

**if** key == keyboard.Key.space:

self.mc.start\_up(self.velocity)

**if** key.char == 'l':

print('Kill engines')

**return** **False**

**if** key.char == 'q':

self.mc.start\_turn\_left(self.ang\_velocity)

**if** key.char == 'e':

self.mc.start\_turn\_right(self.ang\_velocity)

**def** **on\_release**(self, key):

self.mc.stop()

**if** \_\_name\_\_ == '\_\_main\_\_':

cflib.crtp.init\_drivers(enable\_debug\_driver=**False**)

**with** SyncCrazyflie(URI) **as** scf:

*# We take off when the commander is created*

mc = MotionCommander(scf)

drone = KeyboardDrone(mc)

**with** keyboard.Listener(on\_press=drone.on\_press, on\_release=drone.on\_release) **as** listener:

listener.join()

We've saved the script as keyboard\_control.py. In addition to requiring that the proper drivers have been installed, it requires the following python dependencies:

* cflib - the crazyflie python API

**Flying**

To use the script to control the drone over the keyboard:

1. power on the crazyflie and set it on a level surface
2. plug in the crazyradio PA to the computer
3. ensure the URI string in the script appropriately matches the interface id, channel and speed parameters for the crazyflie
4. run the script, python keyboard\_control.py
5. enjoy flying with ease!