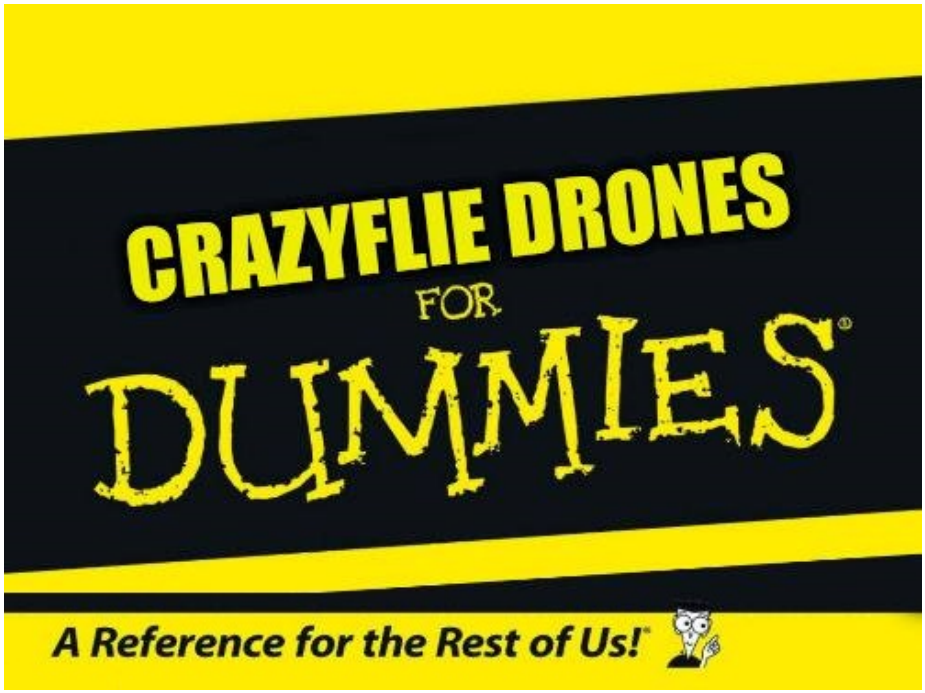
Crazyflie Guide
(AKA Drones for Dummies)



Typeset by Brian Raymond,
Contributed to by Tameez Latib, Joey Miller

Under the guidance of:
Prof. Tabuada's Cyber-Physical Systems Laboratory and
Prof. Fragouli's Algorithmic Research in Network Information Flow Laboratory

August 10, 2018

Contents

1 Native Installation **3**
- 1.1 Crazyflie Python Library . . . 3
- 1.2 Crazyflie Python Client . . . 3
- 1.3 Crazyflie Firmware . . . 3
- 1.4 Crazyflie LPS Node Firmware . . . 4

2 VM Installation and Software **4**
- 2.1 Installation . . . 4
- 2.2 Installed Software . . . 4
- 2.3 System tweaks . . . 5
- 2.4 Projects . . . 5

3 Running Crazyflie **5**
- 3.1 PC Client . . . 5
- 3.2 Updating Repos . . . 5
- 3.3 Updating Crazyradio Firmware . . . 6
- 3.4 Compiling Crazyflie Firmware . . . 6
- 3.5 Adding USB Drivers . . . 6
- 3.6 Bootloading the Crazyflie . . . 6

4 Custom Firmware **6**
- 4.1 Crazyflie . . . 6
- 4.2 Loco-Positioning System . . . 7

5 Debugging **7**
- 5.1 Crazyflie Firmware Debugging . . . 7
  - 5.1.1 STM32 . . . 8
  - 5.1.2 nRF51 . . . 10
- 5.2 Crazyflie Python Lib . . . 10
- 5.3 ROS . . . 11

6 Loco-Positioning System **11**
- 6.1 Configuring Drones . . . 11
- 6.2 Configuring Nodes . . . 11
- 6.3 Node Placement . . . 11

7 Setpoint Following **11**
- 7.1 Setup . . . 11
- 7.2 Code . . . 12

8 Logging **14**
- 8.1 Loggable Variables . . . 14
- 8.2 Skeleton Code . . . 14
- 8.3 AltLogger . . . 16

9 Custom Communication **16**
9.1 Crazyflie ⇌ Base Station 16
9.1.1 Crazyflie Code 16
9.1.2 Base Station Code 17
9.2 Crazyflie ⇌ Beacon 17
9.2.1 Crazyflie Code 17
9.2.2 Beacon Code 18
9.3 Crazyflie ⇌ Crazyflie 19

1 Native Installation

1.1 Crazyflie Python Library

In terminal, type the following to install cflib:

```
$ git clone https://github.com/bitcraze/crazyflie-lib-python.git
$ pip install -e path/to/cflib
```

To use USB radio without being root:

```
$ sudo groupadd plugdev
$ sudo usermod -a -G plugdev <username>
```

You will also either need to create or edit the following file:

```
# /etc/udev/rules.d/99-crazyradio.rules
SUBSYSTEM=="usb", ATTRS{idVendor}=="1915", ATTRS{idProduct}=="7777", MODE="
    0664", GROUP="plugdev"
# to connect Crazyflie 2.0 via usb
SUBSYSTEM=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="5740", MODE="
    0664", GROUP="plugdev"
```

1.2 Crazyflie Python Client

To install the client, clone the following repo and install:

```
$ git clone https://github.com/bitcraze/crazyflie-clients-python.git
$ sudo apt-get install python3 python3-pip python3-pyqt5 python3-pyqt5.qtsvg
$ pip3 install -e .
```

1.3 Crazyflie Firmware

To install the Crazyflie 2.0 firmware and be able to edit it:

```
$ git clone https://github.com/bitcraze/crazyflie-firmware.git
$ sudo add-apt-repository ppa:terry.guo/gcc-arm-embedded # For ubuntu 14.04
    64b
$ sudo add-apt-repository ppa:team-gcc-arm-embedded/ppa # For ubuntu 16.04 64
    b
$ sudo apt-get update
$ sudo apt-get install libnewlib-arm-none-eabi
```

Compiling is done with 'make'

1.4 Crazyflie LPS Node Firmware

To install the location positioning system (LPS) firmware and be able to edit it:

```
$ git clone https://github.com/bitcraze/lps-node-firmware.git
$ sudo apt-get install libncurses5:i386
$ sudo apt-get install gcc-arm-none-eabi
```

Compiling is done with 'make'

2 VM Installation and Software

2.1 Installation

In order to get the Crazyflie in the air quickly, Bitcraze has set up its own Virtual Machine that is preloaded with all the essential software and code necessary to begin flying the Crazyflie. The virtual machine is intended to be run in Oracle VirtualBox. If you wish to run the VM in WMPlayer you will need to install some additional tools yourself. The Virtual Machine itself has an online index. If you wish to get the latest version of the VM, it can be found on their VM release page on Github.

One error that may come up iswhen running the VM will say something to the effect that there is an issue with the USB controller. To fix this, be sure to select the USB 1.1 OHCI configuration. Once this is done, the virtual machine should hopefully install and boot smoothly. Once it finishes installing, boot up the VM and click on the desktop configuration file named Update all Projects to automatically update all Crazyflie Github Repos. The Following is all the information necessary about the VM. It can also be found in the README on the Desktop of the VM.

```
* The username is: bitcraze
* The password is: crazyflie
* The virtual machine has 30GB drive and 1 GB of RAM
* The virtual machine is using Xubuntu 14.04.4 LTS
```

2.2 Installed Software

The following software has been pre-installed (besides basic setup)

- Virtualbox guest additions
- gnu-arm-none-eabi + build tools
- Git
- gitg
- PyCharm
- Oracle Java JRE (for PyCharm)
- pyusb, pygame and pyqt
- PyQtGraph
- Qt4 and Qt Designer
- KDE Marble with Python-bindings
- KiCad
- Eclipse with compiling/debugging/flashing

- Leafpad
- EmbSys RegView for Eclipse
- dfu-util

2.3 System tweaks

The udev rules to access the Crazyradio and the NRF bootloader have been added to the udev configuration. They have also been added to the Virtual Box pre-set filters.

2.4 Projects

The following projects have been pre-cloned into the /home/bitcraze/projects directory:

- crazyflie-pc-client
- crazyflie-firmware
- crazyflie-bootloader
- crazyradio-firmware
- crazyradio-electronics
- crazyflie2-nrf-firmware
- crazyflie2-stm-firmware
- crazyflie2-stm-bootloader

3 Running Crazyflie

3.1 PC Client

The Crazyflie PC client can be run by using the following command:

```
# On virtual machine:
$ python3 /home/bitcraze/projects/crazyflie-pc-client/bin/cfclient
# On native install:
$ cfclient
```

Or by using the shortcut on the desktop.

Please note: If no other input device is passed to the VM the "VirtualBox USB Tablet" device will be used. Do not connect to the Crazyflie using this device, since the thrust will then be controlled by the mouse movements on the screen.

3.2 Updating Repos

Updating to the latest versions of all the repositories can be done by using the following:

```
# On virtual machine:
$ /home/bitcraze/bin/update_all_projects.sh
# On native install:
$ /home/desiredRepoToUpdate git pull
```

On the virtual machine, the shortcut on the desktop may also be used.

3.3 Updating Crazyradio Firmware

Download the latest firmware and run the following commands:

```
$ cd /home/bitcraze/projects/crazyradio-firmware
$ python usbtools/launchBootloader.py
```

If you have not activated the USB filter for the NRF bootloader, pass the newly found "NRF BOOT" USB device to the virtual machine and then run the following command:

```
$ python usbtools/nrfbootload.py flash new_firmware_file.bin
```

3.4 Compiling Crazyflie Firmware

A version of the Crazyflie firmware that is upgradable using the Crazyradio bootloader can be built using the following commands:

```
$ cd /InstallLocation/crazyflie-firmware
$ make CLOAD=1 DEBUG=0
```

This firmware can then be downloaded to the Crazyflie using the cfclient or the "Flash using radio" make target in Eclipse.

3.5 Adding USB Drivers

Finally, be sure to right-click on the USB at the bottom of the Virtual Machine and select USB Settings. This will allow you to add USB drivers to the VM and is necessary to run a controller for flying the drone or allow it to recognize the CrazyRadio Dongle.

3.6 Bootloading the Crazyflie

Once the Drone has been assembled, it is necessary to flash the latest firmware onto the drone itself. Select the latest release zip file and use it for bootloading. To bootload the Crazyflie, hold the power button for 3 seconds and it should begin to flash blue LED's. If it makes the ordinary startup jingle and rotates its motors, turn it off and try again. Open up the CFClient and click on the connect tab on the top to open a drop down menu. Select Bootloader and then click on Initiate bootloader cold boot in the new window. It should connect to the Crazyflie. Once it is connected, click on browse and select the zip file you previously downloaded. Now click program and wait for it to finish flashing the firmware. once it has completed, be sure to click restart in firmware mode. Failure to do so will result in the Crazyflie not being updated to the new firmware.

4 Custom Firmware

4.1 Crazyflie

If the project you are currently working on requires you to modify the firmware, there are two repositories you will need to focus on. The first is the Crazyflie 2.0 Firmware and the second is the Crazyflie 2.0 NRF Firmware.

Clone these repositories and make your modifications as necessary. Once you are finished, run the make file and copy the binary created. The zip file downloaded in the last section contains 2 binaries and a json file. The json file specifies the names of the binaries so it is easiest to rename the binaries you generate from the make to the names of the binaries in the zip, extract the json file and then zip them up to create your new firmware package. The binaries are thankfully named appropriately such that it is easy to tell which one corresponds to NRF and the one for the ordinary firmware. Once you have your new zip file, follow the same steps in the section on Bootloading the drone to flash your modified firmware to the drone itself.

4.2 Loco-Positioning System

To modify the firmware for the LPS system, you will need to modify the lps node firmware repository and upload it the node using lps tools.

To install and run lps tools:

```
# To install, navigate to cloned folder and type:
$ pip3 install -e .[pyqt5]
# To run:
$ python3 -m lpstools
```

If trying to modify the firmware on 64bit Linux, make sure you have the following 32bit libraries:

```
$ sudo apt-get install libncurses5:i386
```

Then once files are modified, run the following in the root directory and upload the '.dfu' in 'lps-node-firmware/bin' with lpstools:

```
$ make clean; make
```

5 Debugging

5.1 Crazyflie Firmware Debugging

For debugging the Crazyflie, the debug adapter sold through bitcraze and a ST-Link/v2 will be needed. To install the proper ST-Link v2 drivers, do the following:

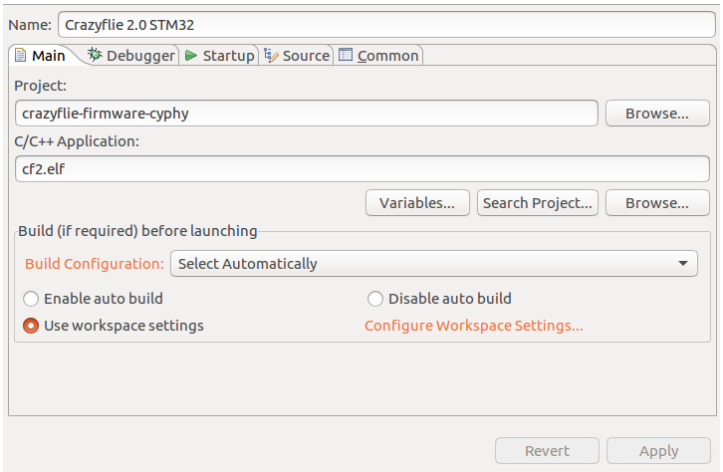
```
$ git clone https://github.com/texane/stlink stlink.git
$ cd stlink
$ make
# install binaries
$ sudo cp build/Debug/st-* /usr/local/bin
# install udev rules
$ sudo cp etc/udev/rules.d/49-stlinkv* /etc/udev/rules.d/
# restart udev rules
$ sudo restart udev
```

Also be sure to install Eclipse Photon and openOCD both on-system and within Eclipse as a debugging plugin (reference link for both).

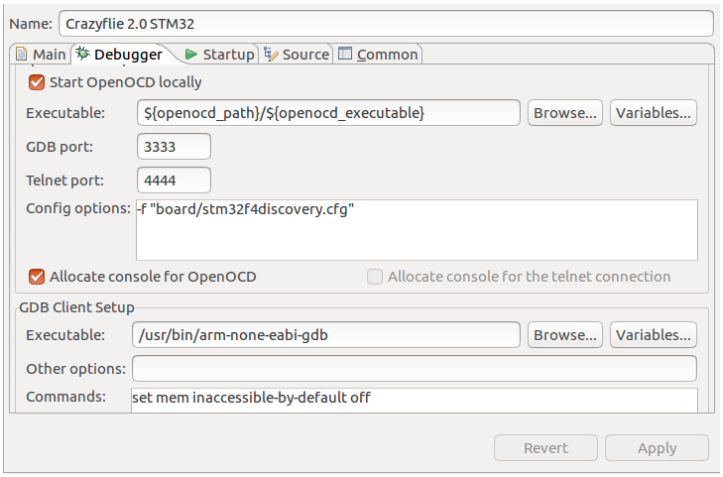
5.1.1 STM32

Within Eclipse:
Go to 'Run' → 'Debug Configuration'. Create a new 'GDB OpenOCD Debugging' configuration and call it 'Crazyflie 2.0 STM32.'

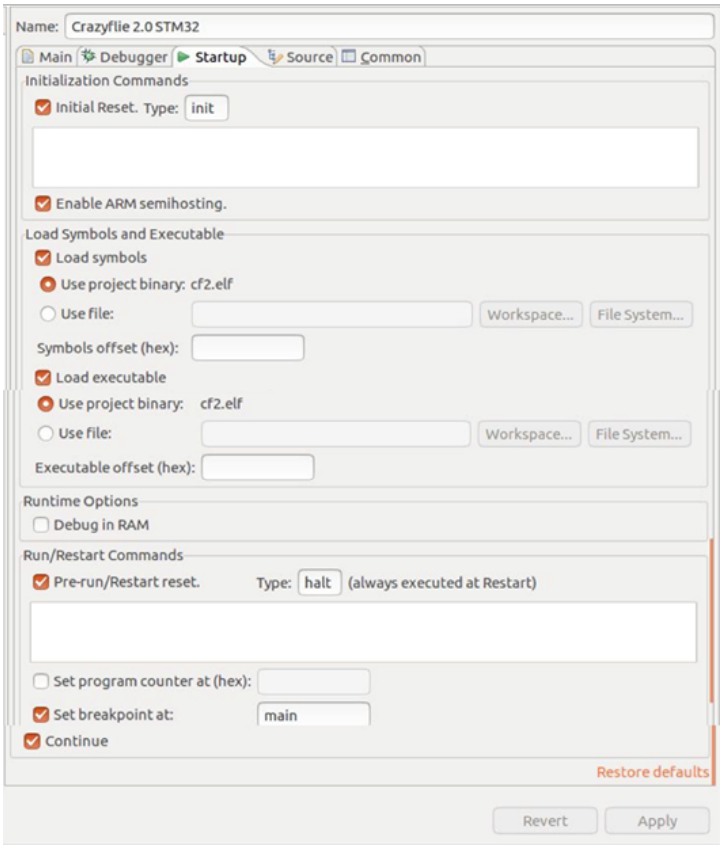
Under Main:
Make 'Project' your firmware folder and 'C/C++ Application' your cf2.elf file. Within 'Build (if required) before launching' section set 'Build Configuration' to 'Select Automatically' and click on 'Use workspace settings'.



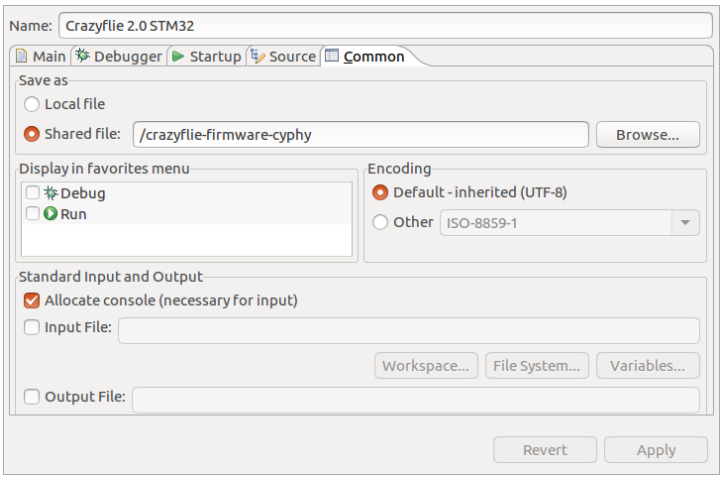
Under Debugger:
Within 'OpenOCD Client Setup' section ensure that executable is '${openocd_path}/${openocd_executable}', 'GDB port' is 3333, 'Telnet port' is 4444, and 'Config options' are ' -f "board/stm32discovery.cfg" '.
Within 'GDB Client Setup' ensure that 'Executable' is '/usr/bin/arm-none-eabi-gdb' and the 'Commands' are 'set mem inaccesible-by-default off'.



Under Startup:
Ensure it is the same as the following.



Under Common:
Within ‘Save as’ section check ‘Shared file’ and set the location to the folder of the crazyflie firmware.



Under the Build Targets view add the following targets by clicking 'New Build Target': "Make CLOAD DEBUG" with the parameters "all CLOAD=1 DEBUG=1" "Make CLOAD" with the parameters "all CLOAD=1 DEBUG=0" "Make DEBUG" with the parameters "all CLOAD=0 DEBUG=1" "Flash using debugger" with parameters "flash" "Flash using radio" with parameters "cload"

To run the debugger within Eclipse:
> Make DEBUG
> Flash using debugger
> Run → Debug Configurations → Crazyflie 2.0 STM32 → Debug

5.1.2 nRF51

5.2 Crazyflie Python Lib

The debugging of the python code is a lot simpler as you can just use the python debugger (pdb) library.

This can be done by either running the following terminal command:

```
$ python -m pdb myscript.py
```

Or adding 'import pdb; pdb.set_trace()' where you want to break the file you wish to debug and then running the file via terminal.

Within terminal, these are key commands:

```
$ h # help
$ l # list code around current break
$ s # step to next executed line
$ n # step to next line in current file
$ c # continue to next breakpoint
```

5.3 ROS

6 Loco-Positioning System

6.1 Configuring Drones

To install the Loco-Positioning deck onto the Crazyflie, be sure to place the flat end of the deck that has the '0xBC' logo face down over the battery. The drone must be powered off while installing and removing the deck. Push all the pins through the inserts on the deck and it should now be ready to use with the Loco-Positioning System (LPS).

Note: We found that for the Crazyflie to work, we needed to re-flash (see Section 3.4) the firmware as the drone is only configured for currently connected decks (e.g. flow deck and LPS deck).

6.2 Configuring Nodes

To flash node firmware to the Loco-Positioning Nodes, you will need to install the LPS configuration tool. Clone the following Git Repository and follow the instructions in the README. If you are running Windows on your device, you can alternatively install it by downloading the lps-tools-win32-install file. Once you have installed the configuration tool, you must download the firmware for the nodes. Select the .dfu file. To update the node, open up the LPS configuration tool and plug in the node via USB while holding down the button labeled DFU. If on Windows, you will need to follow instructions on installing the nodes USB driver with Zadig. In the client, click on the browse button and select the firmware file you recently downloaded. Now click update and finally press the reset button on the node once it completes the update. Once all the nodes are updated, connect them to the computer normally and configure the ID. If you wish to operate in TWR mode, make sure the Mode is set to Anchor (TWR). Click apply and assign every node a different ID. Mark them with electrical tape or some other method to easily keep track of the IDs of the nodes.

6.3 Node Placement

Place the anchors in the room at least 2 meters apart with a line of site to the flying volume. For TWR Mode, 4 anchors is the minimum with 6 being ideal and 8 if redundancy is needed.

7 Setpoint Following

7.1 Setup

To accomplish this, we used the Crazyflie2.0 with the 'Flow deck' and the 'Loco-Postioning deck' as the flow sensor aides in velocity/ height calculations while the loco-positoning deck aides in absolute position calculations. This is all done with Bitcraze's internal kalman filter and pid functions which are a part of the crazyflie firmware code.

7.2 Code

For setpoint following we heavily modified Luigi Pannochhi's code which utilizes the built-in commander python library provided by bitcraze. The python script takes in an input to specify which type of path following it should use: 'l' for loco-positioning based, 'p' for positioning based, and 'v' for velocity based. The names are a little deceiving, however, as both 'l' and 'p' use the beacons. The difference between the two is that 'l' creates setpoints internally using thrust, roll, pitch, and yaw while 'p' creates setpoints using x, y, z, and yaw.

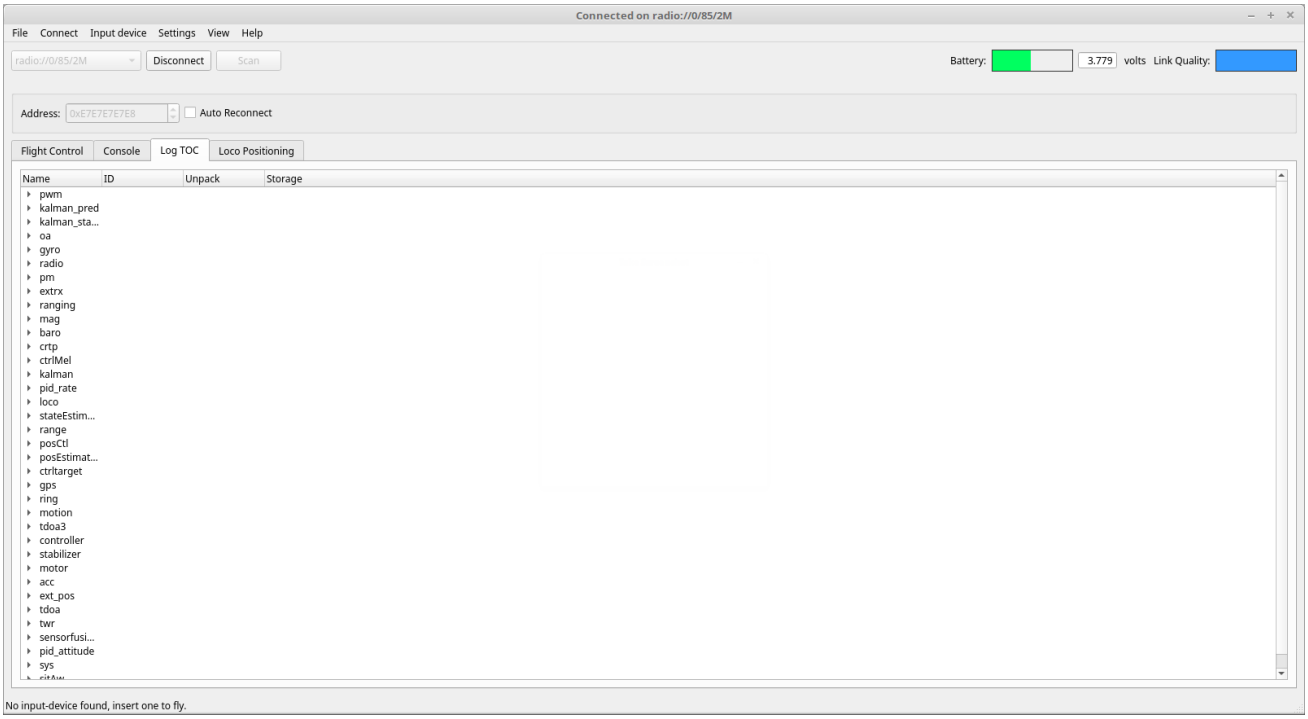
The following is a look at each of these setpoint following functions. Note that there are globals DT, T, VMAX, and START_HEIGHT. DT is the default dT for go_straight_d, T is the default time of flight, VMAX is the maximum velocity limit, and START_HEIGHT is the initial hover height.

```
def go_straight_d(cf, d_x, d_y, z, t, dt=DT):
        if (t == 0):
                return
        steps = int(t/dt)
        v = [d_x/t, d_y/t]
        for r in range(steps):
                cf.commander.send_hover_setpoint(v[0], v[1], 0, z)
                time.sleep(dt)

def loco_follow_paths(scf):
        cf = scf.cf
        cf.param.set_value('flightmode.posSet', '1')
        for position in sequence:
                for i in range(200):
                        cf.commander.send_setpoint(position[1], position[0],
                                                   position[3], int(position[2] * 1000))
                time.sleep(0.01)
        time.sleep(0.1)

def pos_follow_paths(scf):
        cf = scf.cf
        cf.param.set_value('flightmode.posSet', '1')
        cf.commander.send_hover_setpoint(0,0,0,START_HEIGHT)
        time.sleep(1)
        cf.commander.send_position_setpoint(0,0,START_HEIGHT,0)
        time.sleep(1)
        for position in sequence:
                cf.commander.send_position_setpoint(position[0], position[1],
                position[2], 0)
                time.sleep(1)

def vel_follow_paths(scf):
        cf = scf.cf
        cf.param.set_value('flightmode.posSet', '1')
        cf.commander.send_hover_setpoint(0,0,0,START_HEIGHT)
        time.sleep(1)
        movement = sequence[0]
        for position in sequence:
                movement = (position[0]-position_internal[0],
                                   position[1]-position_internal[1],
                                   position[2], position[3])
                t = T
                if (abs(movement[0]/T) > VMAX or abs(movement[1]/T) > VMAX):
                        t = abs(movement[0]/VMAX) if abs(movement[0]/VMAX) >
                                abs(movement[1]/VMAX) else abs(movement[1]/VMAX)
                go_straight_d(cf, movement[0], movement[1], movement[2], t)
                time.sleep(1)
                for i in range(4):
                        position_internal[i] = position[i]
                time.sleep(0.1)
```

8 Logging

8.1 Loggable Variables

To see all the variables which can be logged, open the cfclient, connect to the crazyflie, and enable the Log TOC by selecting 'View → Tabs' in the top menu bar and checking the box next to 'Log TOC.' This should then enable the following tab, which may be perused to see all variables logged by each of the key words.



Logging is done by creating a LogConfig with the name of the measurement to be recorded, and then adding variables with the names formatted according to the Log TOC.

8.2 Skeleton Code

```
import cflib
import logging
from cflib.crazyflie.log import LogConfig

dataLog = errorLog = pos_writer = log_timestamp = log_conf = Logger = None

def begin_logging(cf):
        global Logger
        Logger = SkeletonLogger(cf)
        Logger.start_logging()
```

```
class SkeletonLogger:
        def __init__(self, cf):
                self.cf = cf
                logging.basicConfig(level=logging.ERROR)

        def start_logging(self):
                global pos_writer
                pos_log_file = open(self.directory + self.log_timestamp
                        + '_pos.csv', 'wb')
                pos_writer = csv.writer(pos_log_file)
                pos_writer.writerow(['time', 'x_pos', 'y_pos', 'z_pos'])
                log_pos = LogConfig(name='Position', period_in_ms=10)

                try:
                        log_pos.add_variable('kalman.stateX', 'float')
                        log_pos.add_variable('kalman.stateY', 'float')
                        log_pos.add_variable('kalman.stateZ', 'float')
                        self.cf.log.add_config(log_pos)
                        # This callback will receive the data
                        log_pos.data_received_cb.add_callback(print_pos)
                        # This callback will be called on errors
                        log_pos.error_cb.add_callback(log_error)
                        # Start the logging
                        log_pos.start()
                except KeyError as e:
                        print('Could not start log configuration,'
                                '{} not found in TOC'.format(str(e)))
                except AttributeError:
                        print('Could not add Position log config, bad configuration.')

        def print_pos(timestamp, data, self):
                print([timestamp, data['kalman.stateX'], data['kalman.stateY'],
                        data['kalman.stateZ']])

        def log_error(self, logconf, msg):
                """Callback from the log API when an error occurs"""
                print('Error when logging %s: %s' % (logconf.name, msg))
                if self.errorLog == None:
                        if not os.path.exists(DIRECTORY):
                                os.makedirs(DIRECTORY)
                        self.errorLog = open(DIRECTORY + datetime.datetime.now().
                            strftime("Error Log %Y-%m-%d_%H:%M:%S"), 'a')
                else:
                        self.errorLog.write('Error when logging %s: %s\n'
                                % (logconf.name, msg))
```

This code may be copy/pasted into a python script, and the begin_logging function be called by passing a Crazyflie handle. Note that if using a SyncCrazyflie handle called 'scf,' one should instead pass 'scf.cf' to begin_logging.

8.3 AltLogger

AltLogger.py is the robust, custom logger class that we created. To use, simply put 'include AltLogger as AltLogger' in your python script header, and then once having created the Crazyflie handle, call AltLogger.begin_logging(). Called with just the Crazyflie handle, this function saves data about stability, position, acceleration, and gyroscope to .csv files in a folder called '/LoggedData/' in the same directory as file being ran. The following is the header for the function:

```
allowedItems = ['stab', 'pos', 'acc', 'gyro']
defaultPath = './LoggedData/'

def begin_logging(handle, arg1=None, arg2=None)
# By default, arg1 = allowedItems and arg2 = defaultPath
# To pass something other than allowedItems, pass a list subset of allowedItems
# To pass something other than defaultPath, pass a string that is a valid file path
# Note: None, either, or both of these may be passed, order independent
```

9 Custom Communication

9.1 Crazyflie ⇌ Base Station

9.1.1 Crazyflie Code

When consoleCommInit is called, it creates the header for our CRTP packet and this allows us to direct it at a particular port. It also creates a new task/process. This process will block until it receives a packet with the right header. Then, we can interpret this custom data from the PC. In the sample code, we also call consoleCommPuts, which puts a string into the buffer ready to be sent and consoleCommPflush, which puts and flushes.

```
// consoleComm.c

void consoleCommInit() {
        if (isInit) return;
        messageToPrint.size = 0;
        messageToPrint.header = CRTP_HEADER(CRTP_PORT_CONSOLE, 0);
        vSemaphoreCreateBinary(consoleLock);
        xTaskCreate(consoleCommTask, CONSOLE_COMM_TASK_NAME,
        CONSOLE_COMM_TASK_STACKSIZE, NULL, CONSOLE_COMM_TASK_PRI);
        isInit = true;
        consoleCommPflush("Console Comm init!");
}
```

```
// consoleComm.c

void consoleCommTask(void * prm) {
        crtpInitTaskQueue(CRTP_PORT_CONSOLE);

        while(1) {
                crtpReceivePacketBlock(CRTP_PORT_CONSOLE, &messageReceived);

                // process data in messageReceived.data
        }
}
```

9.1.2 Base Station Code

For the Base Station to communicate it needs to send a packet to the drone using the CRTProtocol. We included this in the commander class but it could be done anywhere with the correct includes.

```
def send_message(self, str):
        for i in range(len(str)/29 + 1):
                pk = CRTPPacket()
                pk.port = CRTPPort.DRONE
                pk.data = struct.pack('<30s', str[29*i:(30*(i+1))-(i+1)])
                self._cf.send_packet(pk)
```

9.2 Crazyflie ⇌ Beacon

9.2.1 Crazyflie Code

When the drone receives a packet from the beacon we check the type of packet it is. If it is of type LPS_TWR_RELAY_B2D (B2D = Beacon to Drone), then we can process it within beaconComm's analyzePayload() function. Currently, we have this function make a call to consoleCommPflush().

```
// lpstwrtag.c
case LPS_TWR_RELAY_B2D:
{
        beaconAnalyzePayload((char*)rxPacket.payload);
        ranging_complete = true;
        messageToSend = 0;
        messageExpected = 0;
        return 0;
        break;
}
```

Note that in sendMessageToBeacon(), we set the messageToSend flag which tells lpstwrtag.c to communicate our custom messages instead of its normal poll → ans → final → report routing. Within the CRTPPAcket we set the header to LPS_TWR_RELAY_D2B instead of LPS_TWR_POLL which is just a #define.

```
// lpstwrtag.c
void initiateRanging(dwDevice_t *dev) {
//      ...
        if (messageToSend) {
                messageExpected = 1;
                memcpy(txPacket.payload, message, LPS_MAX_DATA_SIZE);
                txPacket.payload[LPS_TWR_TYPE] = LPS_TWR_RELAY_D2B;
        }
        else {
                txPacket.payload[LPS_TWR_TYPE] = LPS_TWR_POLL;
        }
        txPacket.payload[LPS_TWR_SEQ] = ++curr_seq;
        txPacket.sourceAddress = options->tagAddress;
        txPacket.destAddress = options->anchorAddress[current_anchor];
        dwNewTransmit(dev);
        dwSetDefaults(dev);
        dwSetData(dev, (uint8_t*)&txPacket, MAC802154_HEADER_LENGTH+2+28);
//      ...
}
```

The file/header pair beaconComm has functions similar to those of consoleComm, but on beaconCommFlush (and the subsequent call to beaconCommSendMessage) it puts the data into a buffer in lpstwrtag, which will be sent on the next communication cycle between the beacon and drone.

```
// beaconComm.c
static bool beaconCommSendMessage(void) {
        sendMessaeToBeacon(message);
        consoleCommPflush(message);
        messageLength = 2;
        // First byte is the header, and the second is the sequence number
        return true;
}
//      ...
void sendMessageToBeacon(char * msg) {
        messageToSend = 1;
        consoleCommPflush(msg);
        memcpy(message, msg, LPS_MAX_DATA_SIZE);
}
```

9.2.2 Beacon Code

On the beacon's side of the firmware, we just modified uwb_twr_anchor.c to listen for the header which specifies the packet as being a part of our custom communication. Note that RELAY_D2B is defined the same as LPS_TWR_RELAY_D2B in the Crazyflie side of the firmware. RELAY_B2D signifies the packet as being from the Beacon to the Drone (Crazyflie).

```
// uwb_twr_anchor.c
case RELAY_D2B:
{
        txPacket.payload[TYPE] = RELAY_B2D;
        txPacket.payload[SEQ] = rxPacket.payload[SEQ];
        memcpy(receivedMsg, rxPacket.payload + 2, PAYLOAD_LENGTH - 2);

        // do something to process the data

        dwNewTransmit(dev);
        dwSetDefaults(dev);
        dwSetData(dev, (uint8_t*)&txPacket, MAC802154_HEADER_LENGTH+2+
           PAYLOAD_LENGTH);

        dwWaitForResponse(dev, true);
        dwStartTransmit(dev);

        break;
}
```

9.3 Crazyflie ⇌ Crazyflie

Under progress