Autonomy and app layer Workshop

Arnaud (Bitcraze)
BAM days
October 20th 2021

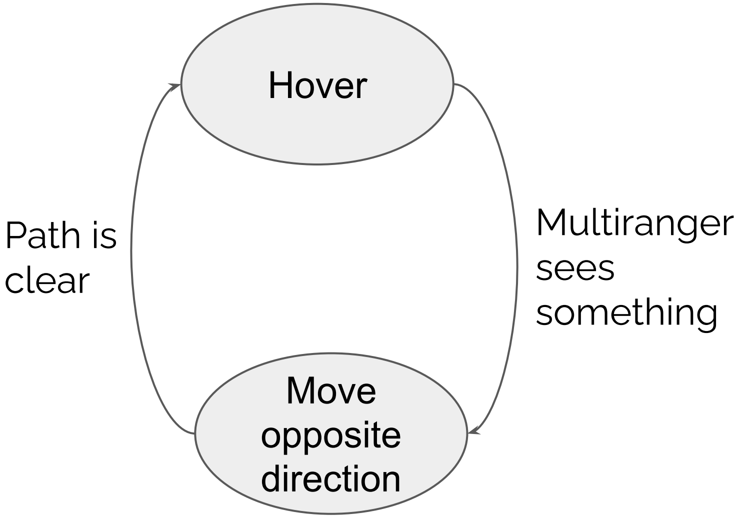
BAM days 19 -21 Oct.

bitcraze Awesome Meetup Days

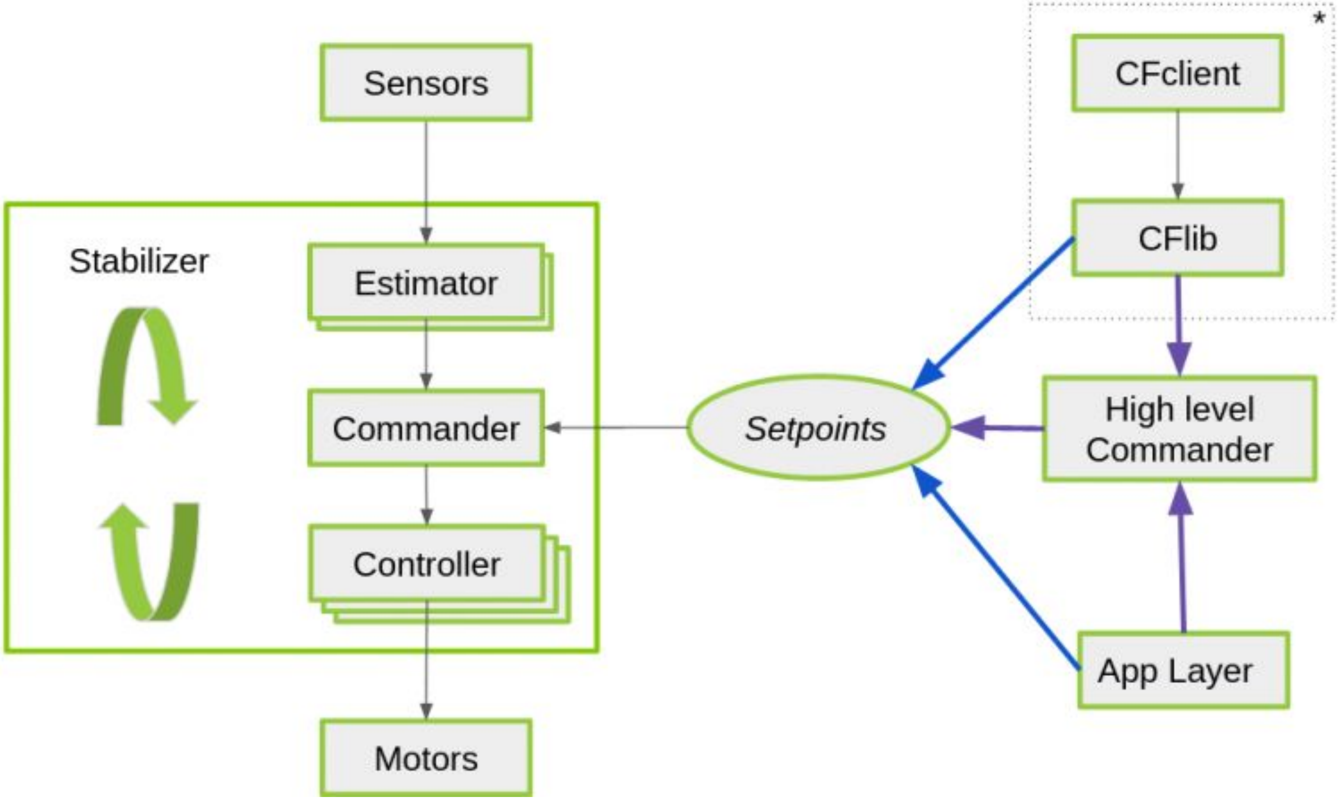
Example: The push demo

- Uses the multiranger deck and the flow deck
- Allows to push the Crazyflie around:
  - If an object is detected, move in the opposite direction
  - If an object is detected on the top, land
- Simple interactive demo to experiment with autonomous behaviors

State machine

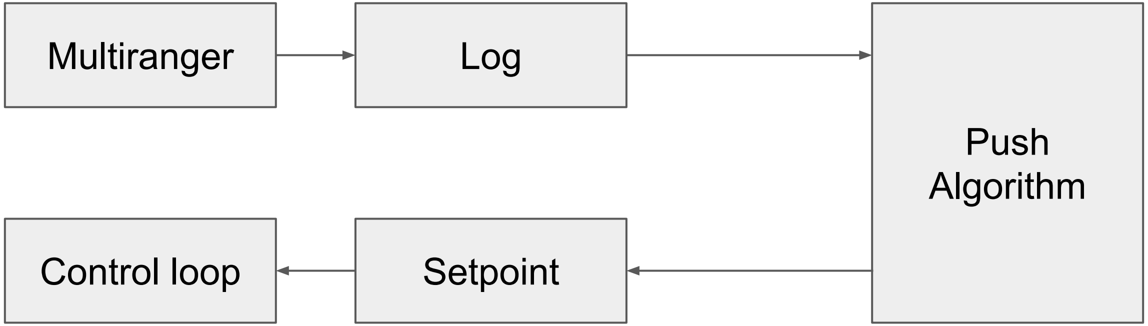


Controlling the Crazyflie



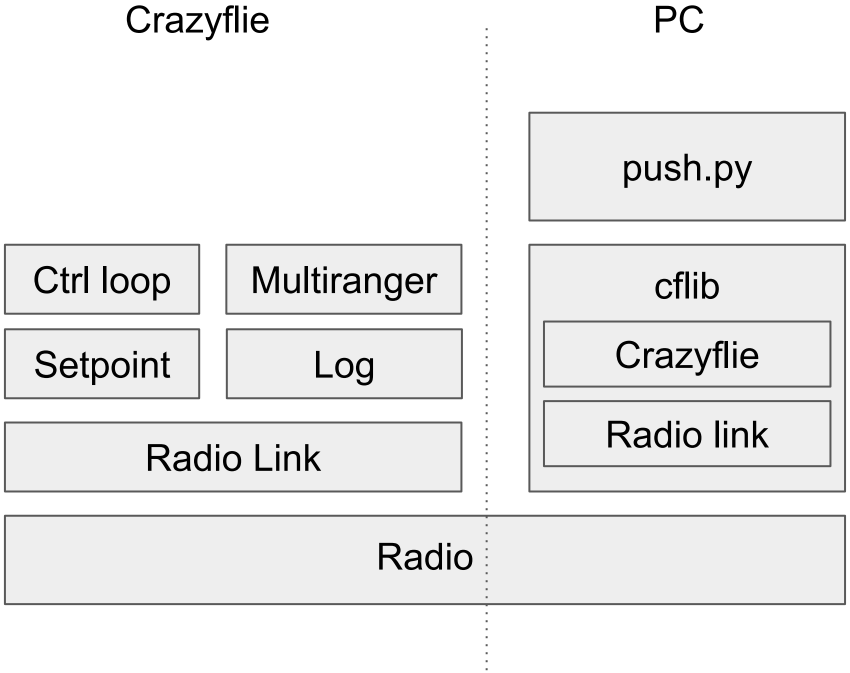
Basic architecture

- Multiranger, Control loop, Log and Setpoint live in the Crazyflie
- Where does the push algorithm go?
  - On PC as a python program?
  - In the Crazyflie? Where and how?



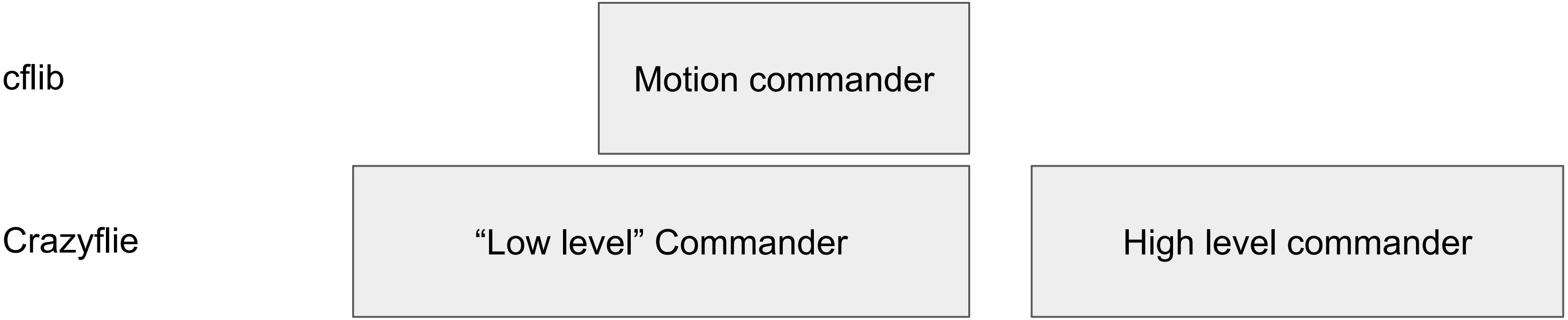
Writing algorithm on the PC

- From Python we use the package “cflib”.
- URI are used to tell the lib what Crazyflie to connect to and how. Eg. “radio://0/80/2M/E7E7E7E7E7”
  - Dongle, channel, datarate, address
- Cflib implement supports for Crazyflie subsystems, some deck even have specific driver
  - The multiranger is one of those

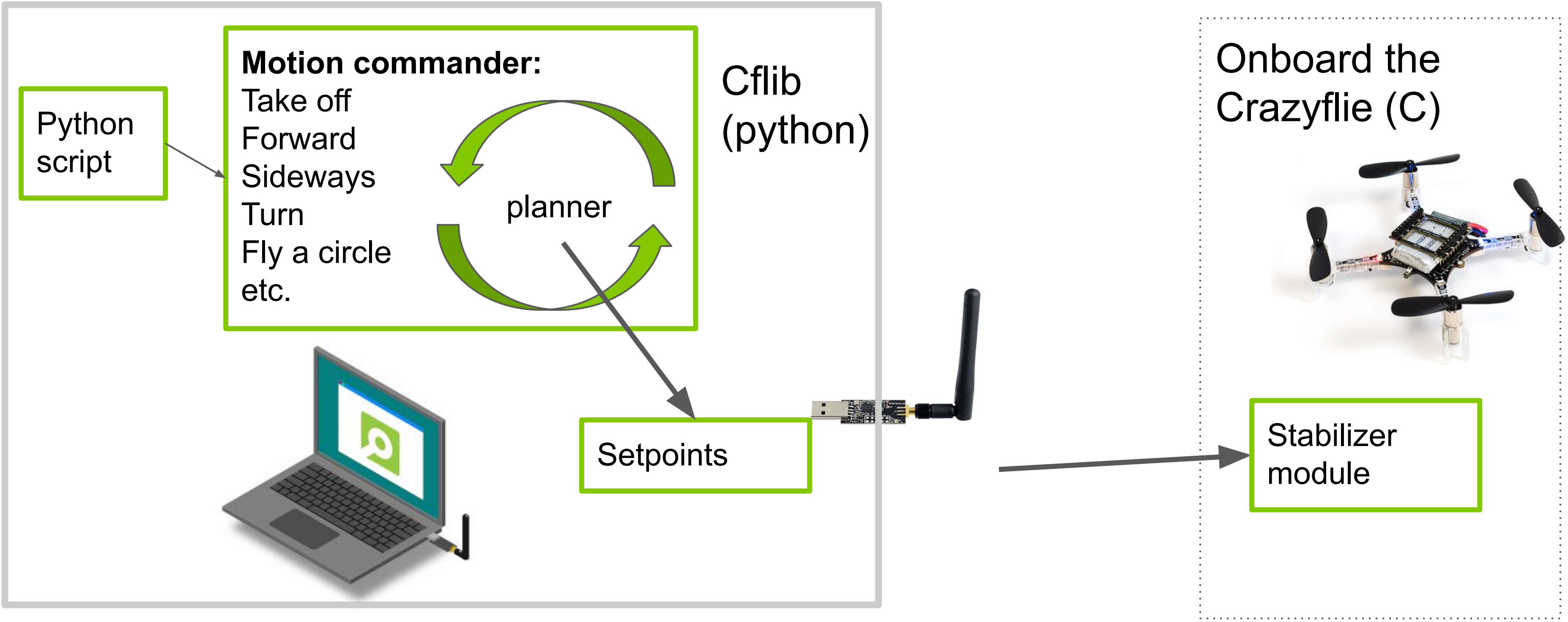


Sending setpoints

- "Low level" commander
  - Instantaneous setpoints, needs to be sent at regular intervals
  - 1 second watchdog
- High-level commander
  - Planner running in the Crazyflie
- Motion commander
  - Flow-deck-optimized planner running in cflib



Motion Commander



Python script

Motion commander:
Take off
Forward
Sideways
Turn
Fly a circle
etc.

planner

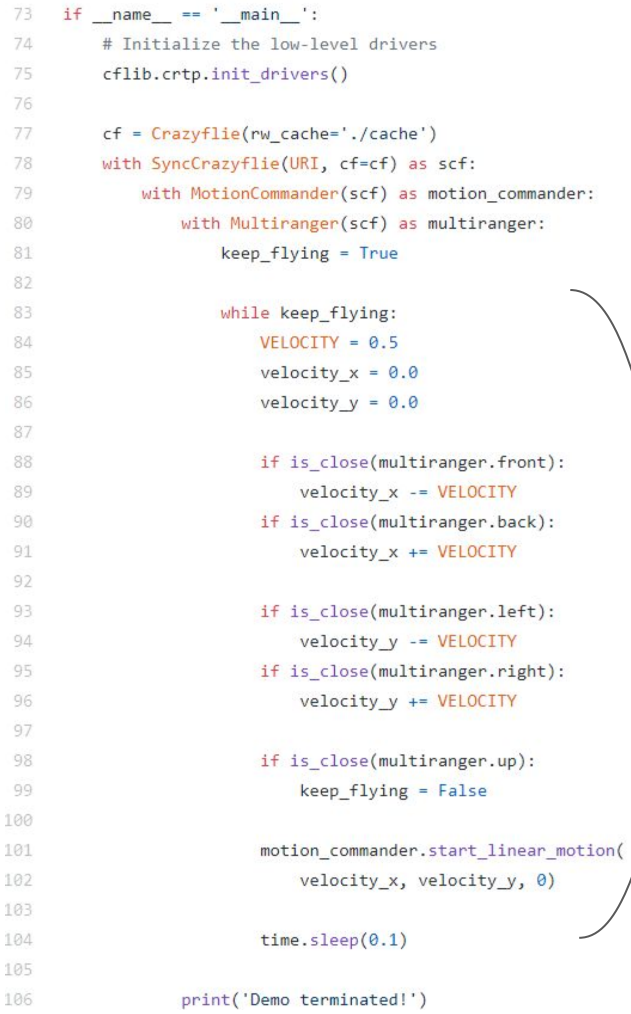
Cflib (python)

Setpoints

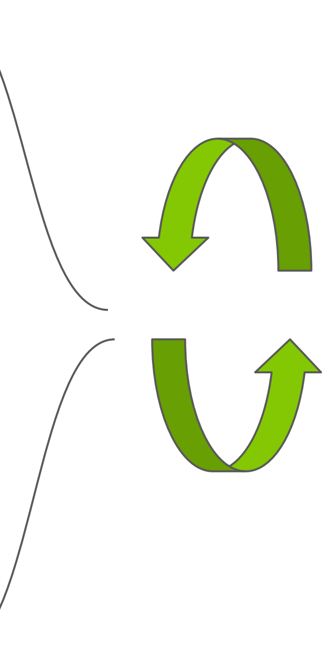
Onboard the Crazyflie (C)

Stabilizer module

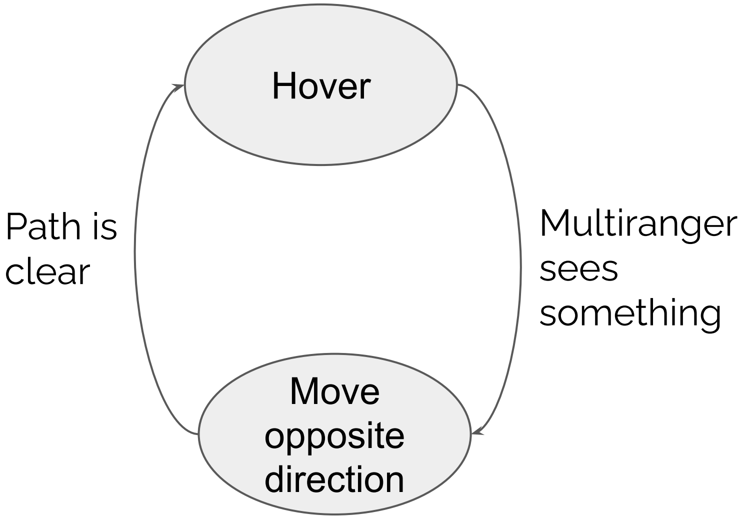
multiranger_push.py



```python
if __name__ == '__main__':
    # Initialize the low-level drivers
    cflib.crtp.init_drivers()

    cf = Crazyflie(rw_cache='./cache')
    with SyncCrazyflie(URI, cf=cf) as scf:
        with MotionCommander(scf) as motion_commander:
            with Multiranger(scf) as multiranger:
                keep_flying = True

                while keep_flying:
                    VELOCITY = 0.5
                    velocity_x = 0.0
                    velocity_y = 0.0

                    if is_close(multiranger.front):
                        velocity_x -= VELOCITY
                    if is_close(multiranger.back):
                        velocity_x += VELOCITY

                    if is_close(multiranger.left):
                        velocity_y -= VELOCITY
                    if is_close(multiranger.right):
                        velocity_y += VELOCITY

                    if is_close(multiranger.up):
                        keep_flying = False

                    motion_commander.start_linear_motion(
                        velocity_x, velocity_y, 0)

                    time.sleep(0.1)

            print('Demo terminated!')
```



State machine



Hover

Multiranger sees something

Move opposite direction

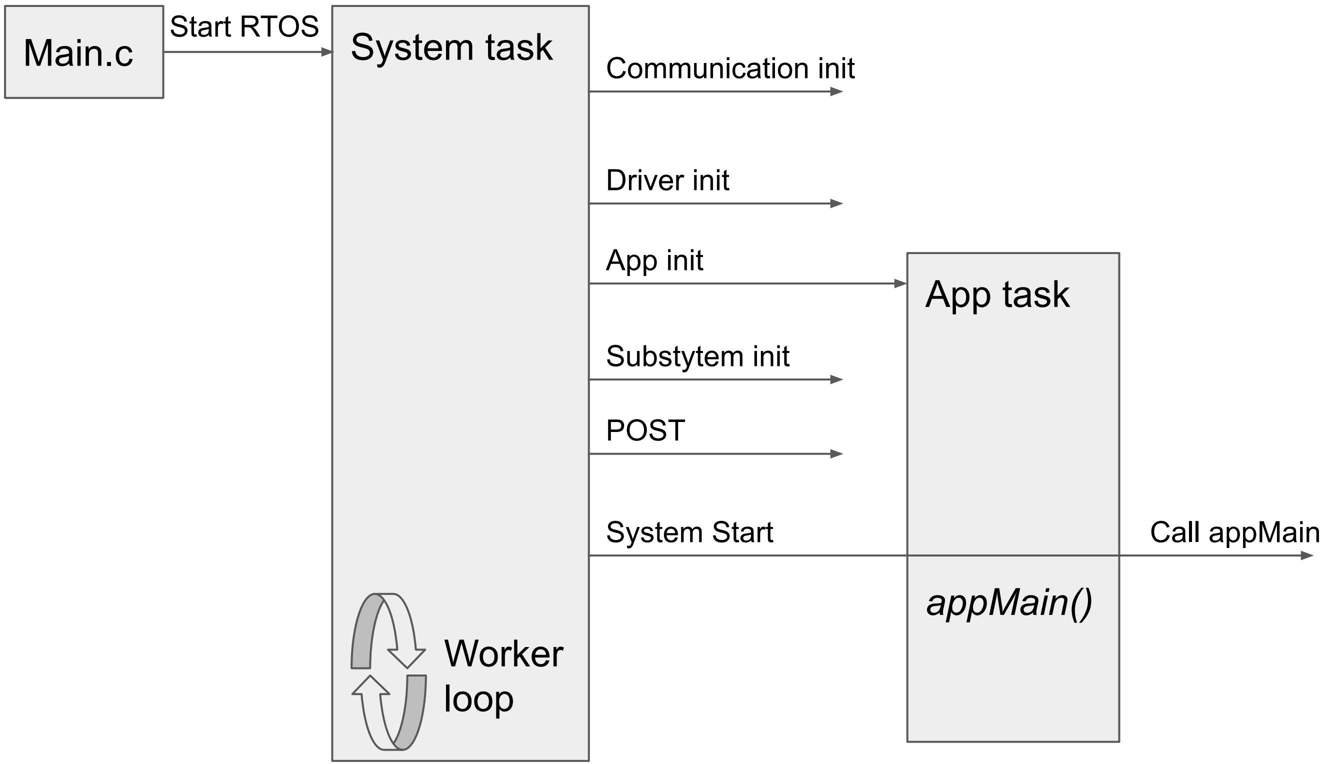
Path is clear

Running onboard: App layer

- Traditional way
    - Get the Crazyflie firmware source code
    - Find a place to put your code
    - Write code
    - Now you have a fork, hard to maintain over time

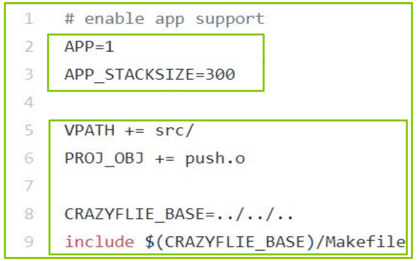
- Out Of Tree build
    - No need to fork!
    - Where does the code go?
- App layer
    - Will call “appInit()” during startup
    - “appInit()” default implementation calls “appMain()” in a thread after POST
- OOT + App layer
    - No more fork, Crazyflie firmware can be a git submodule, much happiness!

Crazyflie boot sequence (illustration)



demos/app_push_demo/

Makefile

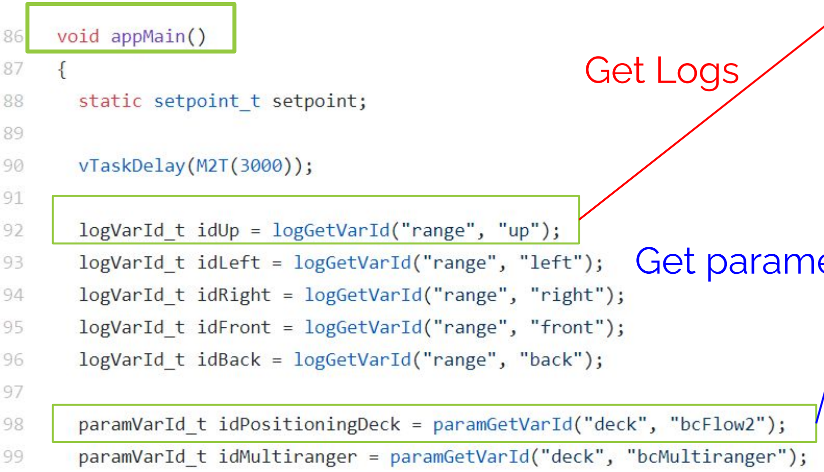


```
# enable app support
APP=1
APP_STACKSIZE=300

VPATH += src/
PROJ_OBJ += push.o

CRAZYFLIE_BASE=../../..
include $(CRAZYFLIE_BASE)/Makefile
```

App layer

OOT build

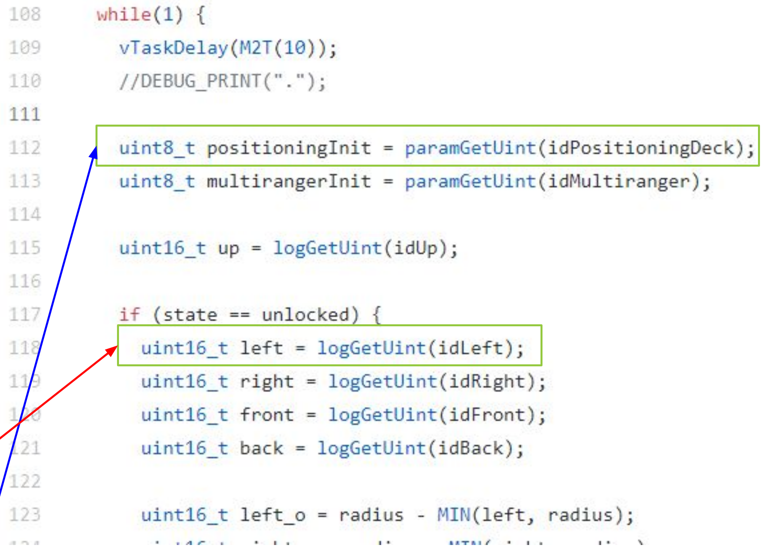
push.c



```
void appMain()
{
  static setpoint_t setpoint;

  vTaskDelay(M2T(3000));

  logVarId_t idUp = logGetVarId("range", "up");
  logVarId_t idLeft = logGetVarId("range", "left");
  logVarId_t idRight = logGetVarId("range", "right");
  logVarId_t idFront = logGetVarId("range", "front");
  logVarId_t idBack = logGetVarId("range", "back");

  paramVarId_t idPositioningDeck = paramGetVarId("deck", "bcFlow2");
  paramVarId_t idMultiranger = paramGetVarId("deck", "bcMultiranger");
```

Get Logs

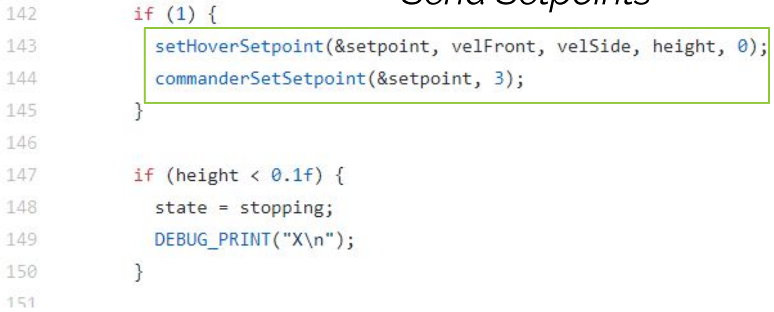
Get parameters



```
  while(1) {
    vTaskDelay(M2T(10));
    //DEBUG_PRINT(".");

    uint8_t positioningInit = paramGetUint(idPositioningDeck);
    uint8_t multirangerInit = paramGetUint(idMultiranger);

    uint16_t up = logGetUint(idUp);

    if (state == unlocked) {
      uint16_t left = logGetUint(idLeft);
      uint16_t right = logGetUint(idRight);
      uint16_t front = logGetUint(idFront);
      uint16_t back = logGetUint(idBack);

      uint16_t left_o = radius - MIN(left, radius);
```

⋮

Send Setpoints



```
      if (1) {
        setHoverSetpoint(&setpoint, velFront, velSide, height, 0);
        commanderSetSetpoint(&setpoint, 3);
      }

      if (height < 0.1f) {
        state = stopping;
        DEBUG_PRINT("X\n");
      }
```

More hooks, better API and future

- Hooks exists to implement an OOT estimator
- More hooks can/should be added to enable more OOT experiments
  - What do you need?
- Experiments to improve the build system using KBuild
  - One plan is to host most useful but niche functionality in our repos not compiled by default
- The in-firmware API should be improved and defined
  - Ideally, the same functionality would be available roughly the same way in Python and in the Firmware, to ease algorithm port
- The Crazyflie could run uPython, is it interesting?
  - Could allow to directly port code from the PC to the firmware with few to no modifications

Questions?