At time of writing, the official ODrive developer guide may be found here This document is a condensed version of that page, serving as a quick-reference sheet for MRD lab development.

I. Dependencies
   - A. For windows; other systems will have slightly different requirements, see developer guide
     1. ARM GNU compiler
        - a) Create an environment variable ARM_GCC_ROOT whose value is the path ARM GNU compiler is installed in
        - b) Note: this is slightly different from adding to path. The path is itself an environment variable that you edit to add to, whereas here you need to add an entirely new environment variable.
     2. ARM GDB
     3. GNU MCU Eclipse’s Windows Build Tools (Windows only)
        - a) Note: this is the piece that contains “make”,
        - b) In theory, it should be enough to add the whole folder to the path, but in my case I had to explicitly add the subfolder that contains make.exe to the path as well before my build would work.
     4. OpenOCD
        - a) Developer guide suggests this is only necessary if using an ST-link/v2, but I found it necessary for general use as well. I recommend setting it up just in case.
     5. Tup
     6. Python
        - a) Requires version 3.7 or later.
     7. If you’re going to be using the ST-Link/v2 programmer, you also need some drivers for it; see the developer guide.
   - B. Important: if on windows, add all of these folders to path!
   - C. To check that all are installed and on path with correct versions, run from terminal:

```
arm-none-eabi-gcc --version
arm-none-eabi-gdb --version
openocd --version             # should be 0.10.0 or later
tup --version                 # should be 0.7.5 or later
python --version             # should be 3.7 or later
```

II. VSCode setup
   - A. Download VSCode here
   - B. Install extensions - within VSCode, hit ctrl-shift-X, then select:
     1. C/C++
     2. Cortex-Debug
     3. Include Autocomplete
     4. Path Autocomplete
     5. Auto Comment Blocks

C. Double-check that you set up the ARM_GCC_ROOT system variable as instructed in the dependencies

III. ODrive build configuration

- A. Clone the ODrive git repository (latest ODrive official) or the MRD Lab version with motor characterization
- B. Copy `Firmware/tup.config.default` to `Firmware/tup.config` and open it to edit parameters
  - 1. CONFIG_BOARD_VERSION
    - a) Our boards are v3.6-24V (as of 4/2/2021)
    - b) Note: you may need to uncomment this once you've confirmed it's set to the right board version
  - 2. CONFIG_USB_PROTOCOL
    - a) Leave as native unless you're having trouble running it on Mac; then you can consider native-stream
  - 3. CONFIG_UART_PROTOCOL
    - a) Default is ascii, but you can change it to native if you're running it off a UART connection to a PC and want to use the python tools
- C. See developer guide for protocol details (native, native-stream, etc.)

IV. Building and flashing ODrive firmware

- A. Requires an STLink-v2
- B. Restart VSCode and open file `ODrive_Workspace.code-workspace` in the ODrive root directory
- C. In VSCode
  - 1. Building the firmware - Terminal -> Run Build Task
  - 2. Flashing the firmware - Terminal -> Run Task -> flash
    - a) Or navigate to ODrive/Firmware within the builtin VS Code terminal and run `make flash`
- D. In the terminal
  - 1. Navigate to ODrive/Firmware and run `make`
  - 2. If you need to flash the firmware to the ODrive, use odrivetool dfu, after which you'll be able to connect with odrivetool

V. Possible errors

- A. When attempting to build, Intellisense claims that very basic C++ files (e.g. stdlib.h) are not on the path, and suggests installing them.
  - 1. That's not actually the issue; there's some indication that this might be a problem with the latest Intellisense extension. I tried downgrading to 26.2, which seemed to work, but then the issue returned next work session. On that occasion, re-upgrading Intellisense and rebooting fixed it.
- B. When attempting to build, "tup error: failed to inject dll: No such file or directory". Antivirus programs may delete tup32detect.exe when you build, in which case you'll need to download a new copy of that file, and whitelist it to prevent it from being deleted again in the future. See here for a discussion of the issue.