Laboratory A: Introduction to LE Quad

Assistants: Giovanni Iacca and Maja Varga
(Software material prepared by Géraud L'Eplattenier)

16.09.2014 (Group 1)

1 INTRODUCTION

In this first laboratory we will introduce the tools that will be used during the rest of the course. The hardware we will use is "LE Quad" (LIS EPFL Quadrotor), a flying robotic platform (also called MAV, Micro Air Vehicle, UAS, Unmanned Aerial System, or UAV, Unmanned Aerial Vehicle) developed at the Laboratory of Intelligent Systems at EPFL for educational and research purposes. A view of the quadrotor can be seen in Fig. 1.

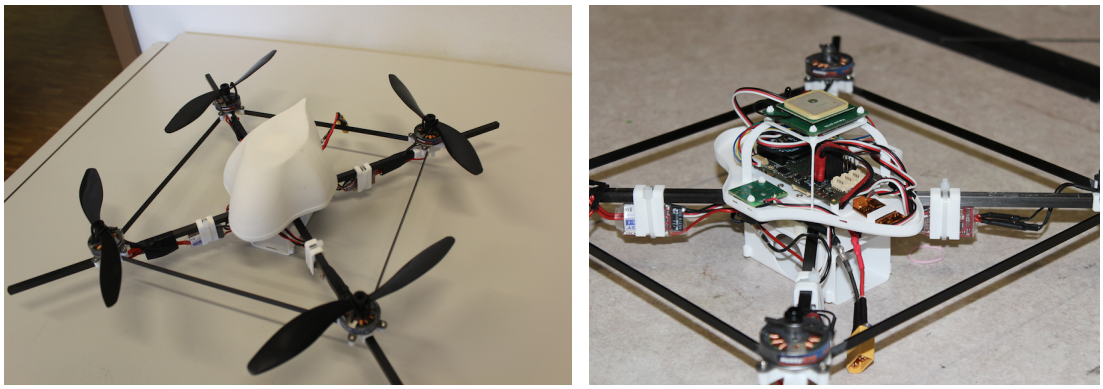

Figure 1: LE Quad, with and without the 3D-printed shell.

NOTE: LE Quad is an experimental platform of which very few prototypes exist, including a limited number dedicated to the Mobile Robots course. Please keep this in mind during the entire duration of the course, and handle hardware very carefully. Every damage due to lack of attention is your responsibility!

LE Quad is controlled by an approximately credit-card sized electronic board (in the following referred to “MAV’RIC board”) that runs software written in C (referred to “autopilot”, or “MAV’RIC application”). The board includes an Atmel AVR 32-bit processor with 512KB of flash memory and 64KB RAM, FPU and DSP instructions (Fig. 2). It also has a micro-SD slot for data logging, an Inertial Measurement Unit (IMU) composed of a three-axis gyroscope, three-axis accelerometer, and three-axis magnetic compass, and a piezo speaker for signaling the user. A barometric pressure sensor provides altitude above sea level. Motor controllers and RC servos can be connected to 8 servo outputs. Additionally, an external GPS can be plugged in, and four extension ports are available to connect digital sensors such as radars or acoustic arrays. A slot is also available for an 802.15.4 compatible XBee communication module used as a telemetry downlink via an open-source protocol called MAVLink. Finally, a small RC receiver can be plugged in to allow manual control and safety override (arming and disarming the motors, switching control modes).

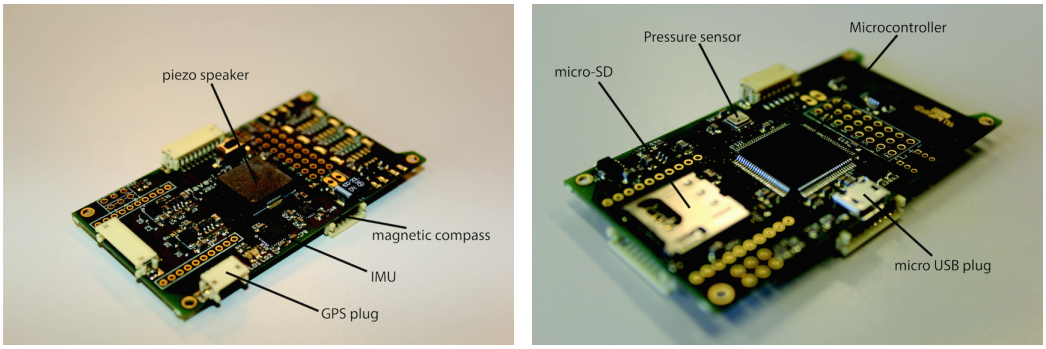

Figure 2: The MAV’RIC board: (left) top and (right) bottom view.

During this laboratory, we will first see how MAV’RIC applications are structured and how to compile and flash the autopilot to the board. Then we will learn how to connect and interface with the board, get/set its parameters, and switch between flight modes. These are preliminary steps that are needed in the next TPs to develop and test your own autopilot.

The main software tools we will use during this laboratory and the rest of the course are:

- **Atmel® Studio**, an integrated development platform (IDP) for developing and debugging Atmel AVR® micro-controller (MCU) applications. Atmel Studio is an easy-to-use environment to write, build and debug applications written in C/C++ or assembly code.
- **X-CTU**, a software used for configuring and testing RF modems. Among other things, X-CTU allows to upgrade RF module firmware, save and retrieve modem configurations (profiles), and changing modem parameters.
- **QGroundControl**, a MAVLink-compliant Open Source Micro Air Vehicle Ground Control Station. This software allows to program the flight plan (by manipulating waypoints on a map), switch between flight modes (see below), visualize/change onboard param-

eters, as well as real-time plotting and logging of sensor and telemetry data from the MAV.

To begin the TP, download from Moodle the zip file `MobileRobots_Lab1.zip`, that includes all the files needed for this first laboratory. Unzip it on the Desktop, possibly into a folder containing your surnames (e.g. `SURNAME1_SURNAME2_SURNAME3`). In the following, this folder will be indicated as `<...>`. The folder contains:

- A pre-compiled fully operational autopilot (`<...>\MobRob_FullRelease`)
- The source-code of an example autopilot (`<...>\MobRob_Lab1`)
- Utility software (`<...>\Utilities`)

EXERCISE 1: COMPILING AND FLASHING THE AUTOPILOT ON THE MAV'RIC BOARD

MAV'RIC applications are written in C and built on top of a library containing various functions for handling the scheduling of user tasks, the hardware onboard, and for fusing sensor information (compass, barometer and GPS, if present) to obtain a unified 3D attitude, position and velocity estimate used for attitude, velocity and position control. The library is found in the folder `<...>\MobRob_Lab1\Library`, that is structured as follows:

- `communication`: handlers for onboard logging, state machine, and parameter setting;
- `control`: the control logics (attitude, stabilization, navigation, etc.);
- `hal`: the drivers of the hardware onboard (sensors, servos, piezo, file system, etc.);
- `runtime`: the task scheduler;
- `sensing`: sensor fusion functions (note that this folder is missing in this first TP);
- `util`: mathematical utility functions.

NOTE: Prior knowledge of C/C++ is required to develop MAV'RIC applications.

An example of MAV'RIC application is given in the folder `<...>\MobRob_Lab1`. A typical application is structured as a set of real-time (periodic) tasks for state estimation, vehicle stabilization, waypoint navigation, telemetry and control, which are scheduled based on their priority. The main control loop operates at 300 Hz, while higher-level tasks can be executed at lower frequencies depending on the application requirements.

```
 extern "C" {
  #include "led.h"
  #include "delay.h"
  #include "print_util.h"
  #include "central_data.h"
  #include "boardsupport.h"
  #include "tasks.h"
  #include "mavlink_telemetry.h"
  #include "piezo_speaker.h"
  #include "gpio.h"
  #include "spi.h"
  #include "sd_spi.h"
}

central_data_t *central_data;

void initialisation()
{
  central_data = central_data_get_pointer_to_struct();
  boardsupport_init(central_data);
  central_data_init();
  mavlink_telemetry_init();
  onboard_parameters_read_parameters_from_flashc(&central_data->mavlink_communication.
      onboard_parameters);
  central_data->state.mav_state = MAV_STATE_STANDBY;
  central_data->imu.calibration_level = OFF;
  piezo_speaker_quick_startup();
  // Switch off red LED
  LED_Off(LED2);
  print_util_dbg_print("Starting!\r\n");
}

int main (void)
{
  initialisation();
  tasks_create_tasks();

  while (1 == 1)
  {
    scheduler_update(&central_data->scheduler);
  }

  return 0;
}
```

Listing 1: The main file of a complete MAV'RIC application

This folder `<...>\MobRob_Lab1` contains:

- `asf`: Atmel Software Framework, a collection of system files for handling the AVR 32-bit processor;
- `config`: a collection of header files defining various internal parameters of the board;

- `boardsupport.c/.h`: a handler for hardware initialization;
- `central_data.c/.h`: a structure containing all the data shared by the tasks;
- `mavlink_telemetry.c/.h`: a series of definitions of the messages sent by the autopilot to the ground station (not present in this first TP);
- `tasks.c/.h`: the definition of the tasks (with associated scheduling rules). This is where the user can add new tasks that must be executed on the autopilot;
- `main.cpp`: the main application file, i.e. the entry point of the autopilot. Listing 1 shows an example of a complete MAV'RIC application main file (note that this code differs from the file that is actually present in `<...>\MobRob_Lab1`).

An important feature of the MAV'RIC board is that it includes a built-in hardware-in-the-loop simulator. This simulator runs on the autopilot itself and simulates the dynamics of the platform at the full update frequency of the inertial system and stabilization loop. Simulated gyroscope, accelerometer, compass, barometer and GPS readings are generated and passed to the attitude and position estimator. This is a powerful tool for debugging the autopilot code before deploying it into the real hardware.

> **NOTE:** In this laboratory, we will work with the built-in simulator only. Complete tests with full quadrotors will be done in the next TPs and mini-projects.

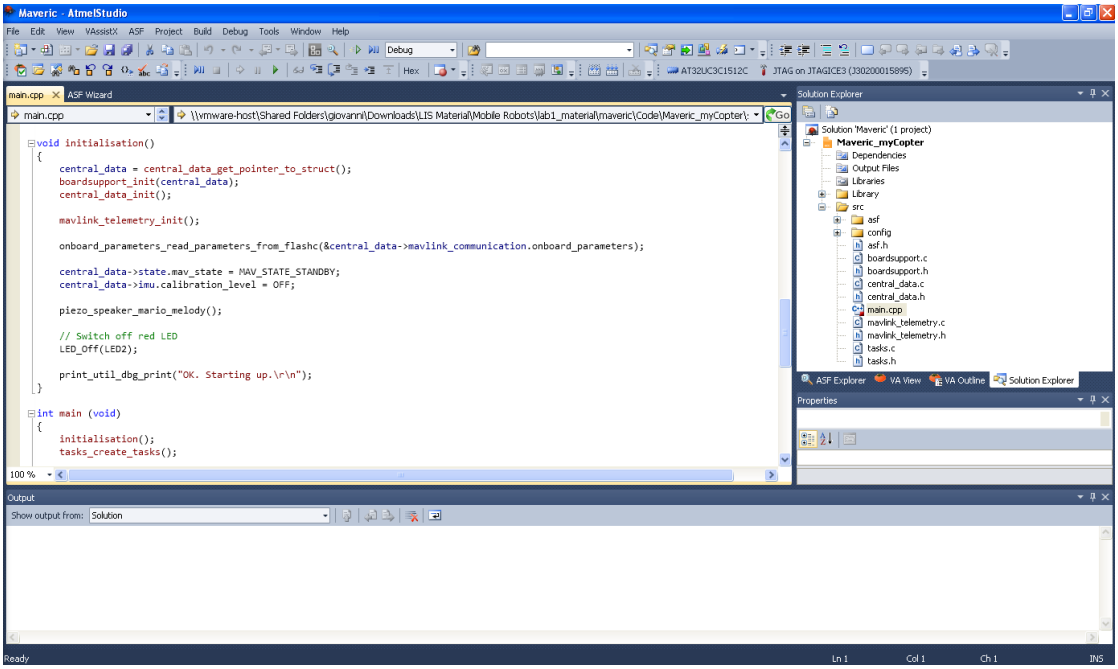

Figure 3: The main view of Atmel Studio

1. Open Atmel Studio. The interface is shown in Fig. 3.
2. Import the project from <...>\MobRob_Lab1\Mavric_MobRob.cproj.
3. Once the project is loaded (including the library files), select in the top toolbar the "Release" Solution Configuration.
4. In the "Build" menu, press "Rebuild Solution" (to clean and build a previously compiled solution) or simply "Build Solution".
5. Check the output in the console. You should not get any error, warning nor message. The final output should be something like:

```
  Finished building target: MegaFly2.elf
                "C:\Program Files\Atmel\Atmel Toolchain\AVR32 GCC\
                    Native\3.4.1057\avr32-gnu-toolchain\bin\avr32-
                    objcopy.exe" -O ihex -R .eeprom -R .fuse -R .lock
                     -R .signature "MegaFly2.elf" "MegaFly2.hex"
                "C:\Program Files\Atmel\Atmel Toolchain\AVR32 GCC\
                    Native\3.4.1057\avr32-gnu-toolchain\bin\avr32-
                    objcopy.exe" -j .eeprom  --set-section-flags=.
                    eeprom=alloc,load --change-section-lma .eeprom=0
                     --no-change-warnings -O ihex "MegaFly2.elf" "
                    MegaFly2.eep" || exit 0
                "C:\Program Files\Atmel\Atmel Toolchain\AVR32 GCC\
                    Native\3.4.1057\avr32-gnu-toolchain\bin\avr32-
                    objdump.exe" -h -S "MegaFly2.elf" > "MegaFly2.lss
                    "
                "C:\Program Files\Atmel\Atmel Toolchain\AVR32 GCC\
                    Native\3.4.1057\avr32-gnu-toolchain\bin\avr32-
                    objcopy.exe" -O srec -R .eeprom -R .fuse -R .lock
                     -R .signature -R .user_signatures "MegaFly2.elf"
                     "MegaFly2.srec"
                "C:\Program Files\Atmel\Atmel Toolchain\AVR32 GCC\
                    Native\3.4.1057\avr32-gnu-toolchain\bin\avr32-
                    size.exe" "MegaFly2.elf"
                   text    data     bss     dec     hex filename
                 264232    7604   37908  309744   4b9f0 MegaFly2.elf
        Done executing task "RunCompilerTask".
        Using "RunOutputFileVerifyTask" task from assembly "C:\
            Program Files\Atmel\Atmel Studio 6.2\Extensions\
            Application\AvrGCC.dll".
        Task "RunOutputFileVerifyTask"
                                Program Memory Usage    :
                                    271836 bytes   51,8 % Full
        Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "Mavric_MobRob.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(
    PostBuildEvent)' != '') was evaluated as ('' != '').
```

```
Target "Build" in file "C:\Program Files\Atmel\Atmel Studio 6.2\Vs\
    Avr.common.targets" from project "C:\Documents and Settings\
    Giovanni\Desktop\MobileRobots\MobRob_Lab1\Mavric_MobRob.cproj" (
    entry point):
Done building target "Build" in project "Mavric_MobRob.cproj".
Done building project "Mavric_MobRob.cproj".

Build succeeded.
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Now it's possible to flash the compiled firmware (".hex" file) to the MAV'RIC board.

FLASHING A MAV'RIC APPLICATION

1. To flash the firmware, you need to enable the DFU (Device Firmware Update) mode on the MAV'RIC board. This can be done by pressing the push-button on the corner of the board while pressing the micro-controller's reset push-button (the one close to the micro-USB interface). After that, you should not see any LED blinking on the board.

2. Go to the `<...>\MobRob_Lab1` folder.

3. Double-click on `dfu_programming.bat` script. By default the compiled firmware file is found in `<...>\MobRob_Lab1\Release` and is named `MegaFly2.hex` (this setting can be changed in Atmel Studio). The script opens a terminal window, where you should see the following output:

```
    <...>\MobRob_Lab1>dfu-programmer at32uc3c1512 erase

    <...>\MobRob_Lab1>dfu-programmer at32uc3c1512 get
    dfu-programmer: no device present.

    <...>\MobRob_Lab1>dfu-programmer flash Release\MegaFly2.hex --
        suppress-bootloader-mem
    Validating...
    xxx bytes used (<yy%>) (Eg. 215220 bytes used (41.70%>))

    <...>\MobRob_Lab1>dfu-programmer reset
    pause
    Press any key to continue . . .
```

Now the firmware of the example autopilot is loaded on the MAV'RIC board and ready to use. Before starting the next exercise, repeat the same operations (1-3) with the complete autopilot that can be found in `<...>\MobRob_FullRelease`.

EXERCISE 2: INTERFACING WITH THE MAV'RIC BOARD

PAIRING XBEE MODULES

In this section you will learn how to pair two **XBee** modules, i.e. the RF modules used on the MAV'RIC board. This procedure is needed to create a Personal Area Network (PAN) connecting the ground station (monitored by QGroundControl, see next section) to one (or more) quadrotors. In this TP, each group will pair a single MAV'RIC board with a ground station.

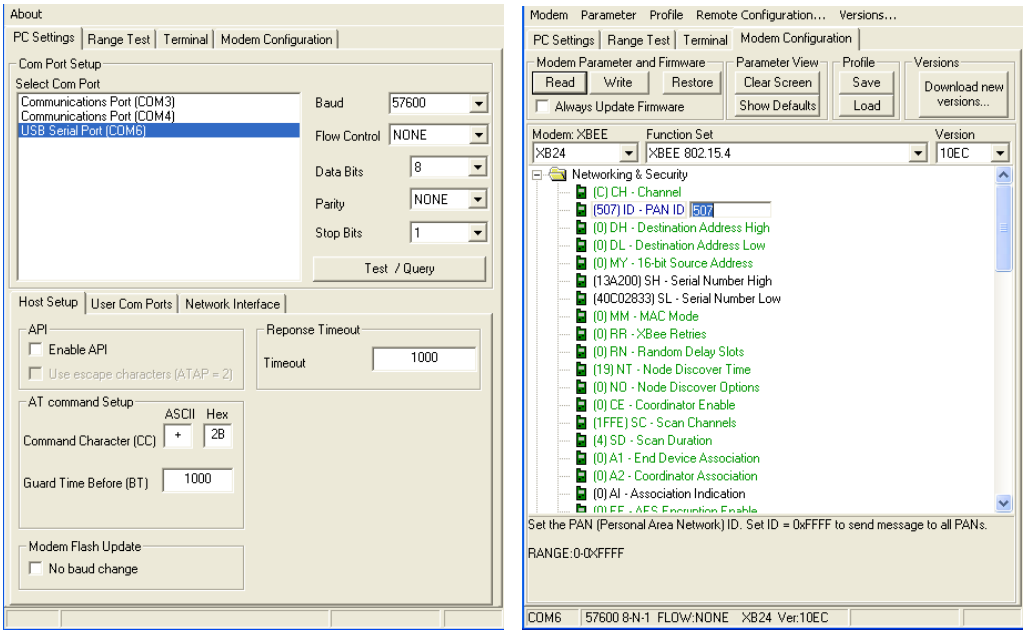

Figure 4: X-CTU: (left) main view for selecting ports; (right) configuration panel.

1. Plug your XBee ground station (attached to the the small red USB interface) in a USB port. The driver (pre-installed on the lab computers) should automatically load and detect the new device.

2. Open X-CTU (simply run the file `X-CTU.exe` present in `<...>\Utilities\XCTU`, you can ignore the error messages).

3. In the "PC Settings" tab, as seen in Fig. 4 (left), select the corresponding COM Port, e.g: "USB Serial Port (COM6)" (note that the port enumeration depends on the machine), and select the baudrate "9600" or "57600" (depending on the Xbee model). Test which baudrate works pushing the button "Test / Query" and checking the output message.

4. In the "Modem Configuration" tab, click "Read": if everything works properly, you should see a list of modem parameters, as shown in Fig. 4 (right).

5. In the "Modem Configuration" tab, click "Load" and load the file:
   `<...>\Utilities\XBee\XBee-Configuration.pro`

6. Under “Networking & Security → PAN ID” change the ID to a unique 4-digits ID.

   NOTE: Every XBee module used during the course is labeled with a unique ID. Please use this number as PAN ID, to avoid ID-conflicts with your neighbors.

7. Click “Save” to save the modem configuration for future use (don’t overlap the original “XBee-Configuration.pro” file).
8. Click “Write” to flash the configuration to the modem.
9. In the “PC Settings” tab, change the baud to “57600”.
10. In the “Modem Configuration” tab, click “Read” to check that the ID was correctly saved on the module.
11. Close X-CTU. This step is needed because X-CTU does not refresh USB port enumeration on the fly. So anything open after the software is running will not be listed in the “PC Settings” tab.
12. Unplug the XBee ground station from the USB interface. Plug the other XBee module. Please pay attention when you do these operations.
13. Restart X-CTU.
14. Repeat the same steps (3-10) for the second module.

Note that the two modules are interchangeable, so there is no need to switch them again.

USING QGROUNDCONTROL

Now that you have the two XBee paired and the firmware loaded on the MAV’RIC board, you can monitor the connection between the ground station and the quadrotor.

- Plug your XBee ground station module and the MAV’RIC board in two USB ports.
- Open QGroundControl.
- In the “Instruments” view (Fig. 5), select the COM port corresponding to the ground station (e.g. COM6), the same as in the previous section, and select the correct baudrate (57600).
- Press the “Connect” button.
- If your MAV’RIC board is powered and the XBee modules paired correctly, you should start receiving telemetry data from the MAV, updated in real-time in the “Instruments” view. Also, your MAV should appear in the “Mission” view (Fig. 6), where you can plan and monitor your flight (note that you can choose different types of map selecting “Options → Map type”, but please select either “Bing Hybrid” or “OpenStreetMap”, as “Google Hybrid” doesn’t usually work).

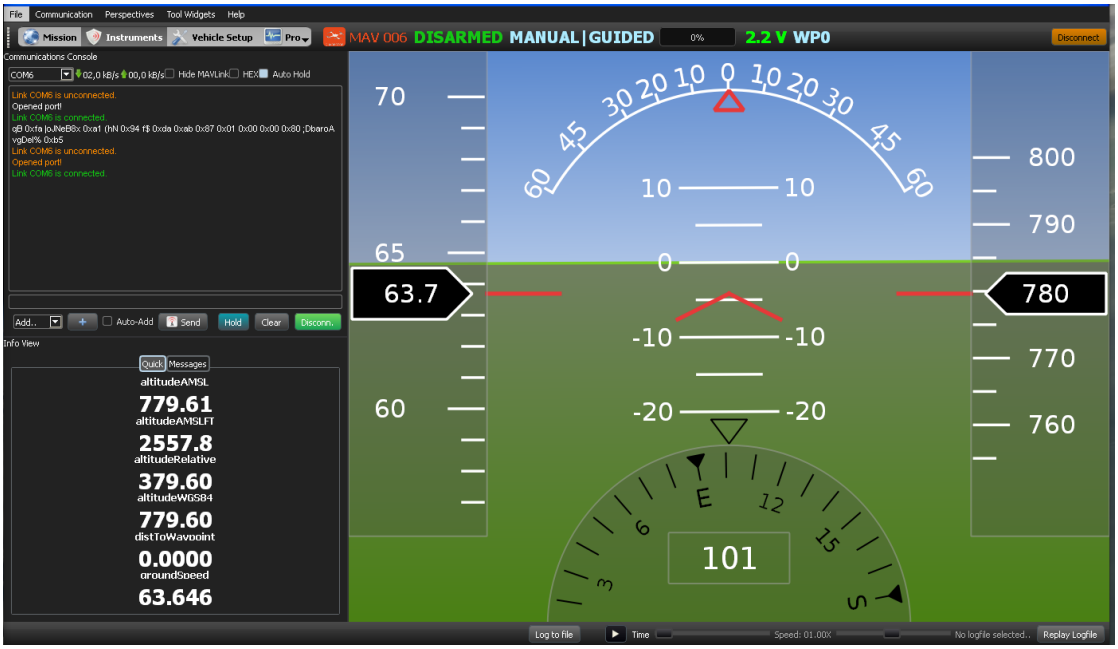

Figure 5: The Instruments view of QGroundControl

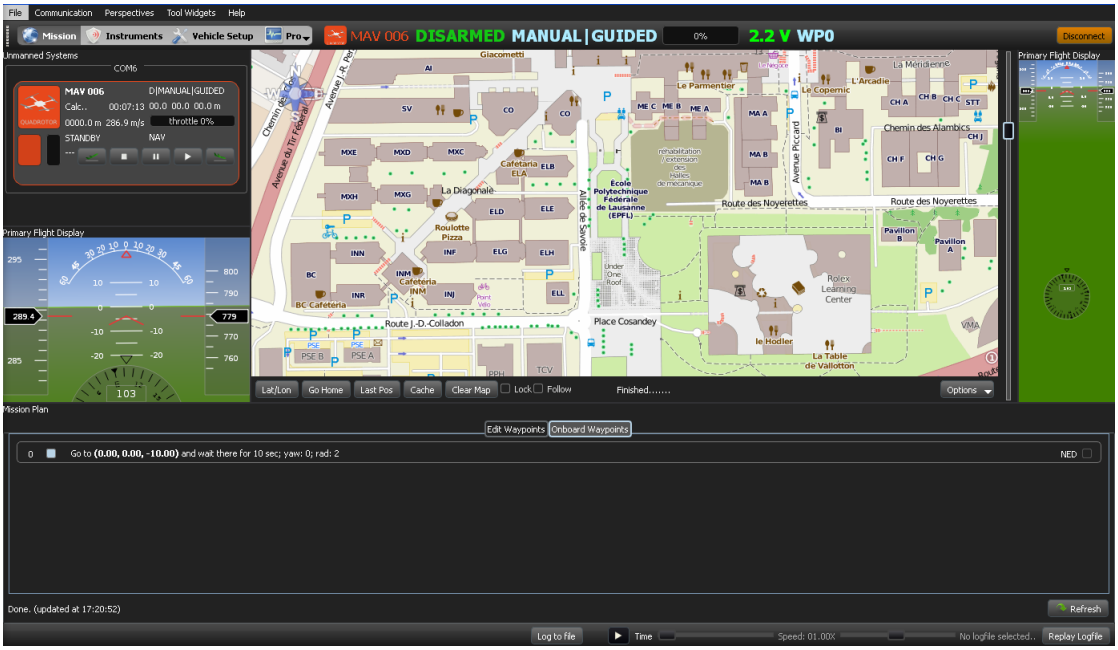

Figure 6: The Mission view of QGroundControl

Take a few minutes to explore the various tabs and tools available in QGroundControl. For example try to visualize and log on the PC some data in the “Plot” tab (“Pro” → “Plot”), as

seen in Fig. 7.

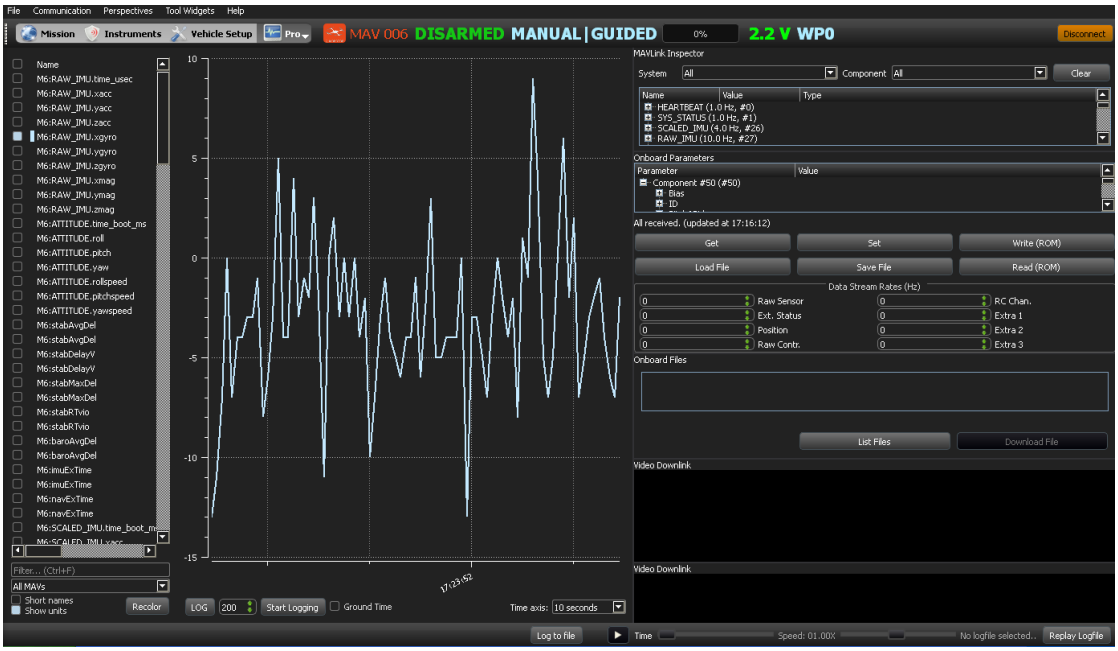

Figure 7: The Plot view of QGroundControl

EXERCISE 3: GETTING/SETTING PARAMETERS

It is possible to get and set parameters on the MAV'RIC board, in real-time, from QGroundControl. To do that, follow these instructions:

1. Open the Parameters view (select "Vehicle Setup" tab, and then check "Onboard Parameters" in the "Tool Widgets" menu). This will open the window shown in Fig. 8.
2. In the left panel, select "Advanced Config".
3. In the right panel, you will see a tree-like list with all the parameters received from the MAV'RIC board: you can expand each parameter and check the corresponding value. To refresh the list, click on the "Get" button. To send a new parameter configuration from the ground station to the board, click on "Set". Note that this is the configuration present in the volatile memory of the MAV'RIC board. To make this setting permanent, writing it to the EEPROM of the board, select "Write (ROM)". Similarly, to copy the parameters from the persistent to the volatile memory, select "Read (ROM)". It is also possible to write a parameter configuration to (or load it from) a file on the PC. The same operations are also possible in the central panel ("Advanced Config").

Now that we know how to change parameters on-the-fly, we will use this feature to enable a remote controller (or a joystick) and manually switch the quadrotor to different flight modes.

Figure 8: The Vehicle Setup view of QGroundControl

EXERCISE 4: SWITCHING FLIGHT MODES

LE Quad can be operated under different modes, ranging from little to full autonomy. The first mode is an attitude control mode where the pilot controls the attitude of the quadrotor via a remote controller. When GPS is available, the quadrotor can be switched to a velocity mode, where the pilot controls the 3D velocity of the quadrotor (translational rate command mode). The pilot can then activate a position hold mode, in which the autopilot will try to maintain its current 3D position. The last mode is a fully autonomous mode, where the autopilot performs GPS (waypoint) navigation.

In the pre-compiled application, all the flight modes are available. In this exercise you will use a remote controller (or, alternatively, a joystick) to switch to the various modes and test them.

> **NOTE:** Before starting, switch the MAV'RIC board to simulation mode, setting the parameter `Sim_mode` to 1 as seen in the previous section.

> **NOTE:** The flight mode shown in the upper toolbar of QGroundControl has different names for the various flight modes, namely "MANUAL" corresponds to attitude control, "MANUAL | STABILIZE" to velocity control, "MANUAL | GUIDED" to position hold and "AUTO" to GPS navigation.

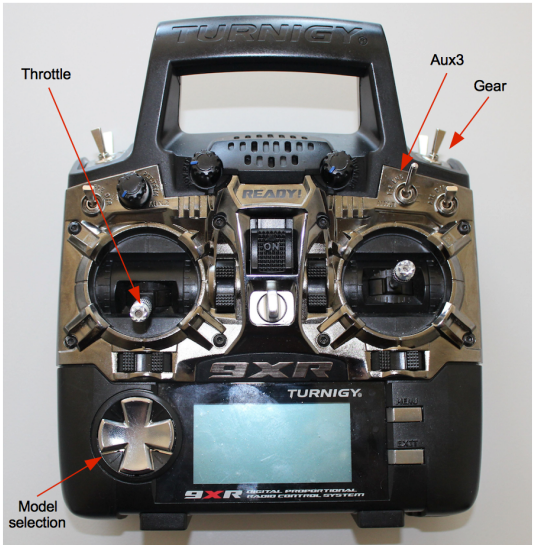

Figure 9: The remote controller

CASE 1: REMOTE CONTROLLER

1. To use the remote controller, plug the satellite receiver (attached to the remote) into the RC1 port on the MAV'RIC board. Please note that every receiver is pre-paired with a different remote, so use only the receiver coupled with your remote.

2. Activate the remote controller. To do that, set the following parameters on the MAV'RIC, as explained in the previous section (there's no need to save them on the EEPROM):

   a) set `remote_active` to 1

   b) set `remote_use_mode` to 1

3. Switch on the remote (central button "On").

4. Make sure that all the switches on the remote are pushed in the upper position, and the throttle (left stick) is all down.

5. Make sure the "MobRob" model is selected: to do that, click on the right direction on the cross-shaped button on the left of the remote. Then navigate to select the "MobRob" model (left/right to change page, up/down to select model), and click right again.

6. Arm the motors moving simultaneously both sticks in the bottom-right position (to disarm them, move the two switches in the bottom-left position).

7. You can now switch to the different flight modes:

   - "Gear" switch upper position: attitude control
   - "Gear" switch lower position **and**:

– “Aux 3” switch upper position: velocity control
– “Aux 3” switch middle position: position hold control
– “Aux 3” switch lower position: GPS (waypoint) navigation control

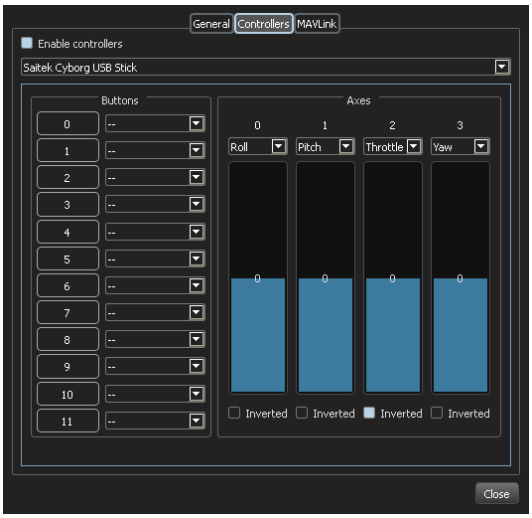

Figure 10: The Joystick Configuration panel of QGroundControl

Figure 11: The joystick

CASE 2: JOYSTICK

First, configure the joystick as follows:

1. Make sure that QGroundControl is closed
2. Connect the joystick to an USB port
3. Open QGroundControl
4. In QGroundControl, go to "File → Settings → Controllers"
5. In the Joystick Configuration panel (10), click on "Enable Joysticks"
6. Choose your joystick in the list
7. Link, in the upper tabs of the joystick window, the joystick axes with the axes of control:
   - Axis 0: Roll
   - Axis 1: Pitch
   - Axis 2: Throttle
   - Axis 3: Yaw
8. Check the axis direction (select the corresponding "Inverted" checkbox if needed):
   - Roll: negative when titling left, positive when titling right
   - Pitch: negative when pitching down (pushing forward the joystick), positive when pulling
   - Throttle: negative when no gas, positive when full gas
   - Yaw: negative when turning left, positive when turning right
9. Make sure that on the left side of the panel the buttons are not assigned to any specific instruction. These assignments are done in the MAV'RIC library, and are set as follows:
   - Button 1 (trigger): arm/disarm
   - Button 3: attitude control
   - Button 5: velocity control
   - Button 2: position hold control
   - Button 6: GPS (waypoint) navigation control
10. Close the joystick configuration window

NOTE: The joystick inputs are sent by QGroundControl **ONLY** in simulation mode.

You can now deactivate the remote controller to enable joystick control. To do that, set the following parameters on the MAV'RIC, as explained in the previous section (there's no need to save them on the EEPROM):

1. set `remote_active` to 0
2. set `remote_use_mode` to 0

You can spend some time playing with the remote controller (and/or the joystick) and testing the various flight modes. Have fun!