CS4140 Lab Assignment 2021-2022

Koen Langendoen (course instructor)
Arjan J.C. van Gemund (founding father)

Embedded and Networked Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

April 2022

1 Introduction

The CS4140 Embedded Systems Laboratory [1] aims to provide MSc students ES, CS, CE, EE, and the like, with a basic understanding of what it takes to design embedded systems (ES). In contrast to other courses on ES, especially those originating from the EE domain (traditionally the domain where ES originate from), in this course, the problem of embedded systems design focuses on using *standard, programmable* hardware components, such as microcontrollers, FPGAs, and subsequently connecting them and programming them, instead of developing, e.g., new ASICs. The rationale is that for many applications, the argument in favor of ASICs (large volumes) simply doesn't outweigh the advantages of using programmable hardware. Consequently, the course focuses on *embedded software*. The goal of the 5 ECTS credits course is not to have the student *master* the multidisciplinary skills of embedded systems engineering, but rather to have the student *understand* the basic principles and problems, develop a systems view, and to become reasonably *comfortable* with the complementary disciplines.

The project chosen for this course is to design embedded software that controls and stabilizes an unmanned aerial vehicle (UAV), commonly known as a drone. In particular, the course centers around a quad-rotor drone, dubbed the Quadruple, developed (brewed!) in house by the Embedded Software group. This application has been chosen for a number of reasons:

- The application is typical for many embedded systems, i.e., it integrates aspects from many different disciplines (mechanics, control theory, sensor and actuator electronics, signal processing, and last but not least, computer architecture and software engineering).
- The application is contemporary. Today's low-cost drones can be flown with ease because of the embedded software that stabilizes the drone by *simultaneously* controlling roll, pitch, and yaw[1]. Without such control software, even experienced pilots find it difficult to fly a drone in (windy) outdoor conditions without crashing a few seconds after lift off.
- The application is typical for many air, land, and naval vehicles that require extensive embedded control software to achieve stability where humans are no longer able to perform this complicated, real-time task. Professional aerial vehicles such as helicopters, most airline and fighter jets totally rely on ES for stability. This also applies to submarines, surface ships, missiles, satellites, and space ships. Today's push towards self-driving cars is another domain that critically depends on much of the same technology.
- Last, but certainly not least, quad rotor drones are great fun, as shown by the rapidly increasing commercial interest in these toys (and this course).

Although designing embedded software that would enable a drone to autonomously hover at a given location or move to a specified location (a so-called *autopilot*) would by far exceed the scope of a 6 EECS credit course, the course project is indeed inspired by this very ambition! In fact, the project might be viewed as a *prototype study* aimed to ultimately design such a control system, where a pilot would remotely control the drone through a single joystick, up to the point where the UAV is entirely flown by software.

[1] roll, pitch, and yaw are the three angles that constitute a drone's attitude in 3D [2]. See Appendix A.

2 System Setup

The ultimate embedded system would ideally be implemented in terms of one chip (a SoC, system on chip) which would easily fit within a drone's limited payload budget. Instead our prototype ES will be implemented on a custom flight controller board (PCB) holding various chips (processor, sensors, etc.), which provides a versatile experimentation platform for embedded system development. The system setup is shown in Figure 1. The ES receives its commands via a so-called *PC link* from a so-called *ground station*, which comprises a PC (laptop) and a joystick. The PC link can be operated in (1) tethered, and (2) wireless mode. Although wireless mode is the ultimate project goal, *tethered flight* will be most extensively used. In this mode the drone is connected to the ground system through a USB cable providing low-latency and high-bandwidth communication. The motors are powered by a LiPo battery carried by the drone, whose charging state needs to be checked at all times as operating these batteries at a too low voltage ruins them instantly. Charging the batteries takes about an hour, and provides for 15 minutes or more of continuous flight. More information on the Quadrupel drone can be found at the CS4140 Resource webpage [2] (in particular, the Operations Manual).

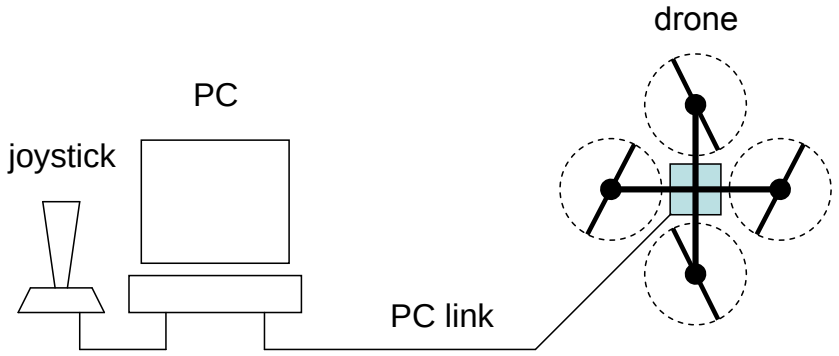

Figure 1: Hardware Setup

Rather than resorting to expensive sensors, electronics, FPGAs, etc., we explicitly choose a *low-cost approach*, which opens up a real possibility for students inspired by this course to continue working with drones and/or ES on a relatively low budget; the price tag for a Quadrupel drone is around €250. Hobby model drones are relatively low-cost due to their simplicity. As the rotors do not feature a swash plate there is no rotor pitch control. Consequently, rotor thrust must be controlled through varying rotor RPM (revolutions per minute). The four rotors control vertical lift, roll, pitch, and yaw, respectively, by varying their individual rotor RPM (see Appendix A). The accelerometer and gyro sensors required to derive drone attitude (which is the minimal information for a simple autopilot application) are also low cost. By using a standard microcontroller (with integrated Bluetooth support), the Quadrupel drone can be simply programmed in C with an open-source compiler (arm-gcc), avoiding the need for an expensive toolchain.

The PC acts as a development and upload platform for the drone. The PC is also used as user interface / ground station, in which capacity it reads joystick and keyboard commands, transmits commands to the ES, receives telemetry data from the ES, while visualizing and/or storing the data on file for off-line inspection. The PC and the drone communicate with each other by means of a RS232 interface (both in tethered and wireless mode).

Of course, the low-cost approach is not without consequences. Using low-cost sensors introduces larger measurement errors (drift, disturbance from frame vibrations) which degrades computed attitude accuracy and control performance. Using a low-cost microcontroller reduces the performance of the designs; the selected ARM Cortex M0 CPU, for example, runs at 14 MIPS and lacks floating-point arithmetic.

Nevertheless, experiments have shown that the low-cost setup is more than capable to perform drone control, and provides an excellent opportunity for students to get fully acquainted with embedded systems programming in a real and challenging application context, where dealing with limited-performance components is fact of life.

3 Assignment

The team assignment is to develop (1) a *working demonstrator* and (2) associated documentation (a *report*) of a control system, based on the quad-rotor drone available. In an uncontrolled situation, the drone, inherently unstable, will almost immediately engage in an accelerated, angular body rotation in an arbitrary direction, which typically results in a crash within seconds as an unskilled human will not be able to detect the onset of the rotation quickly enough to provide compensatory measures with the joystick. The embedded controller's purpose is to *control* 3D angular body attitude (φ, θ, ψ) and rotation (p, q, r), as defined in Appendices A and B. This implies that the body rotation will not just uncontrollably accelerate, but will be controlled by the embedded system according to the setpoints received from the joystick (specifying `roll`, `pitch`, and `yaw`). In effect, the embedded system acts as a *stabilizer* allowing a (slow) human to manually control drone attitude using the joystick.

The lab hardware available to the team are [2] (i) a twist handle joystick, (ii) 2 Linux PCs, (iii) 1 test board, including the microcontroller and sensors, and (iv) a drone. The drone shall be controllable through the joystick, while a number of parameters shall be controlled from the PC user interface. The status of the system shall be visualized at the PC screen, as well as on the drone (using the 4 LEDs on the flight controller board). Whereas the hardware is provided, in principle, *all* software must be *designed by the team.* Code on the Linux PC and drone shall be programmed in C. A description of the required system setup, together with the signal conventions appears in Appendix B.

Although the project intentionally has some degrees of design freedom, the following requirements *must* be fulfilled. The first set of requirements relate to safe Quadrupel operation such that this valuable piece of equipment is not damaged:

Safety The ES must have a *safe state* (for drone and operator) that shall be reached *at all times.* In this state, the four motors are *de*-energized (zero RPM). This shall be the initial state at application startup, as well as the final state (either at normal exit or abort). Consequently, joystick states other than zero throttle and neutral stick position are illegal at startup, upon which the system shall issue a warning and refrain from further action. Note, that the safe state is not to be implemented by merely zeroing all setpoints at the PC side as this does not guarantee that the ES (when crashed) will be able to reach this state.

Emergency stop It shall be possible to abort system operation at all times, upon which the system briefly enters a *panic state*, after which it goes to safe state. In the panic state rotor RPM is set at a sufficiently low rate to avoid propellor damage, yet at a sufficiently high rate to avoid the drone falling to ground like a brick causing frame damage. Panic mode should last for a few seconds until it can be assumed that touch-down has occurred, after which the drone must go into safe state. Aborting the system must be possible by pressing joystick button(s), and keyboard key(s) (see Appendix C).

Battery health The LiPo batteries powering the drone need to be handled with care as draining them will inflict fatal damage. In particular, the battery voltage should never drop below 10.5 Volt. As soon as the battery voltage (read out over the internal ADC) crosses the threshold, the drone should enter panic mode. Preferably, the pilot should be warned ahead of time so he can land safely without the risk of getting his flight aborted at an inconvenient moment.

Robust to noise As sensor data is typically noisy and may contain spurious outliers, feeding raw sensor data to the controllers may result in large variations of the four rotor control signals `ae1`, `ae2`, `ae3`, and `ae4`, possibly causing motor stalling. The on-board sensing module (InvenSense MPU-6050) therefore includes a Digital Motion Processor (DMP) that filters and fuses the raw accelerometer and gyro readings to arrive at a smoothed stream of attitude readings. Note however, that in the final stage of the project, the filtering must be done "by hand" on the microcontroller itself and care must be taken to remove obvious outliers and filter vibration noise such that the sensor data is sufficiently stable and reliable to be used in feedback control[2].

Reliable communication The PC link must be reliable, as any error in the actuator setpoints may have disastrous consequences, especially when the drone has lift-off. A wrong rotor RPM setpoint may immediately tilt the drone, causing a crash. Note that *dependability* is a key performance indicator in the project: *a major incident that is caused by improper software development (i.e., improper coding and testing) may lead to immediate disqualification.* Testing communication status at both peers may involve sending or receiving specific status messages to ascertain that both peers are operating and receiving proper data.

[2]Note that the microcontroller only provides integer-arithmetic instructions. Consequently, the filters must be implemented in software using *fixed-point* arithmetic.

Dependability The same dependability standards apply to the embedded system itself. If, for some reason, PC link behavior is erratic (broken, disconnected) the code should immediately signal the error and go to panic mode.

Robustness Last but not least, the embedded control code must be as reliable as possible. An important aspect is the proper use of fixed point arithmetic for representing and handling of sensor values, filter values, controller values, various sorts of intermediate values, parameter values, and actuator values (the Cortex M0 processor does not provide floating-point arithmetic instructions). On the one hand, the values must not be too small, which would cause loss of precision (resolution). On the other hand, the values must not become too large, as this may cause integer overflow, with possibly disastrous consequences.

The next requirements relate to proper functionality of the ES. As mentioned earlier, the ES is intended to act as a controller that provides an order of stabilization to the drone attitude and rotation, such that the drone can be adequately controlled by a human. The requirements are as follows:

PC The PC shall act as the user's drone *control interface*. After verifying that the joystick settings are in neutral, it shall upload the ES program image to the drone (which at that point runs a simple boot loader that waits for a valid program to execute), read inputs from the keyboard and the joystick, send commands to the ES, read telemetry data from the ES, and visualize and store the ES data. At mission completion (or abort) the PC shall set the ES in the safe state, have the ES program terminate, which returns control to the boot loader that will wait for the next program to run.

drone The ES shall be operated in at least six control modes, enumerated 0 ("safe"), 1 ("panic"), 2 ("manual"), 3 ("calibrate"), 4 ("yaw control"), and 5 ("full control"). Mode 6 ("raw"), 7 ("height"), and 8 ("wireless") are optional. In modes 2 and up, the joystick and keyboard produce the control signals `lift`, `roll`, `pitch`, and `yaw` (see Appendix C for the joystick and keyboard mappings). The demonstrator shall commence in mode 0 at startup. A flight experimentation sequence always starts with selecting the proper mode, prior to ramping up motor RPM.

Safe mode In safe mode (mode 0), the ES should ignore any command from the PC link other than the command to either move to another mode or to exit altogether. Effectively, the ES should keep the motors of the drone shutdown at all times.

Panic mode In panic mode (mode 1) the ES commands the motors to moderate RPM for a few seconds such that the drone (assumed uncontrollable) will still make a somewhat controlled landing to avoid structural damage. In the panic state, the ES should ignore any command from the PC link, and after a few seconds should autonomously enter safe mode.

Manual mode In manual mode (mode 2) the ES simply passes on `lift`, `roll`, `pitch`, and `yaw` commands from the joystick and/or keyboard to the drone *without* taking into account sensor feedback.

Note that the keyboard must also serve as control input device (for instance, pressing 'a' increments `lift`, pressing 'z' decreases `lift`), such that the actual commands being issued to the ES are the *sum* of the keyboard and joystick settings. This enables the keyboard to be used as static *trimming* device, producing a static offset, while the joystick produces the dynamic relative control component. Also see Appendix C which prescribes the key map that must be used.

Calibration mode In calibration mode (mode 3) the ES interprets the sensor data as applying to level attitude and zero movement. As the sensor readings contain a non-zero DC offset, even when the drone is not moving, the calibration readings thus acquired are subsequently stored during subsequent mode transition to controlled mode, and used as reference during sensor data processing in controlled mode.

Yaw-controlled mode In yaw-controlled mode (mode 4) the ES controls drone yaw rotation in accordance with the yaw commands received from the twist handle. In this mode, the joystick control mapping is the same as in manual mode, except for the twist handle (`yaw`), which now presents a yaw *setpoint* to the *yaw controller*. The yaw setpoint is yaw rate (cf. r). Neutral twist (setpoint 0) must result in zero drone yaw rate ($r = 0$). Unlike in manual mode, drone yaw should now be *independent* of rotor speed variations as much as possible, i.e., when the drone starts yawing the ES should automatically adjust the appropriate rotor thrusts `ae1` to `ae4` to counter-act the disturbance. As the yaw sensor may drift with time, an additional trim function (through keyboard keys, see Appendix C) must be supplied in order to maintain neutral twist - zero yaw relation. In this mode, the relevant controller parameter (P) should also be controllable from the keyboard (Appendix C).

Full-control mode In full-control mode (mode 5) the ES also controls roll and pitch angle (φ, θ), next to yaw rate (r), now using all *three* controllers instead of just one. The same principles apply: neutral stick position should correspond with zero drone roll and pitch angle. As the drone has a natural tendency for rapidly accelerated roll and pitch rotation (it's an inherently unstable system), proper roll and pitch control is absolutely required in order to allow the drone to be safely controlled by a human operator. Similar to the yaw control case, *proper roll and pitch trimming prior to lift-off is crucial to safely perform roll and pitch control during lift-off* as the vehicle has a natural tendency to either pitch or roll due to rotor and weight imbalance. For the keyboard map of the roll and pitch trimming keys see Appendix C.

Raw mode In Raw mode (button 6) the ES must now use the raw (unfiltered) accelerometer and gyro data instead of the angles generated by the motion sensor. To obtain stable control the accelerometer and gyro readings must be smoothed and fused by a Kalman filter (for roll and pitch), and a Butterworth filter (for yaw control) The lack of floating-point instructions demands the use of fixed-point arithmetic. Note that switching to raw mode induces the need for recalibration (as the sensor module without DMP and filtering generates different signals). The easiest way to handle this is by maintaining a global state (DMP or RAW) and regard the press of button 6 as a toggle between the two.

Height-control With height-control (button 7) the ES must use the barometer to control the height of the drone, keeping it fixed at the height when button 7 was pressed. Button 7 effectively functions as an on/off button for the height control (semi auto pilot). When the pilot touches the throttle level on the joy stick, height control should be turned off immediately.

Wireless When switching to wireless control (button 8) the ES must be controlled through the Bluetooth link provided by the nRF51822 SoC. The main challenge is to down size the communication rate to the bandwidth provided by the Bluetooth module (whose driver operates at the highest interrupt level at regular intervals). Switching between tethered and wireless mode should only be allowed when the drone is on the ground (i.e., when the motors are off).

Note: In order to achieve optimum safety, a mode switch to modes 2, 3, 4, 5, and 6 should only be allowed to occur under zero motor RPM conditions (`ae1` to `ae4` = 0).

Profiling In order for the control algorithms to work properly, the control loop time, i.e., the total latency of computing controller output, writing to the drone, reading the resulting drone response returned by the sensor filter chain, filtering the sensor values to be processed by the controller *must be in the order of 2 ms or less.* (When using the DMP in modes 4 and 5, this processing time can be relaxed to 10 ms.) As the drone's response is nearly instantaneous, most of the control loop delay is caused by the ES code. Since the Cortex M0 core runs at only 14 MIPS some code timing measurements will have to be performed as to ascertain that the entire control loop fits within the control rate budget! For example, if the entire code takes 1.5 ms, in case of a 500 Hz control frequency (2 ms), there is only 0.5 ms left for other tasks, such as the communication through the 115,200 baud PC link (controller setpoints, PC telemetry data). Given the fact that the RS232 payload (8 bits) is embedded within a 10 bits protocol (1 start bit, 8 bits, 1 stop bit, no parity), this baud rate approximately corresponds to a 100 μs/byte communication bandwidth, i.e., only 10 bytes each 2 ms. Hence, *proper time budgeting and scheduling is a crucial step in the development of a stable and reliable drone control system!*

Logging In order to analyze system performance as well as to debug coding errors the ES must have a *data logging facility*, such that all relevant signals can be visualized and can be stored to PC file for off-line analysis (possibly buffered in the 128 KB on-board Flash memory first and flushed at the end of the mission). Signals to be logged include ES system time, system mode, incoming joystick/keyboard data, outgoing drone actuator data, incoming sensor data, and intermediate data from the sensor signal processing chain, and relevant controller data (e.g., control parameters). Note that the signals must be time-stamped, i.e., a line of signals in the log must start with system time (microsecond resolution). In this way, system performance (system reaction speed is crucial to proper control) can be observed as well as plotted against time. Some of the most relevant data should also be real-time visible on the PC screen (max 10 Hz update rate). Minimum measurements that must be conducted (and documented) include the internal control response time of the ES (typically, the control loop time in ms), the system response time as perceived by the user (i.e., the time between joystick movement or key-press, and drone response), and the associated drone yaw, roll, and pitch response (in terms of a time plot) in order to properly assess achieved stability performance. All measurements must adhere to the `gnuplot` / `matlab` data format.

The overall design may vary in quality and performance, which, of course, will directly translate into the course grade. Acceptable demonstrators include the following versions:

- version including full feedback control (will demonstrate mode 5 operation with fused angle information from the motion sensor module). This is the bare minimum to avoid failing the project.
- raw version including full feedback control (will demonstrate operation using raw accelerometer and gyro readings).
- height control (will demonstrate height stabilization).
- wireless version (will demonstrate stabilization, with wireless control). Obviously, this mode will earn the most credits, but is only allowed when tethered mode has been demonstrated to have sufficient quality/stability.

Additional brownie points can be obtained by implementing new features for example, a fancy GUI or smartphone control interface (replacing the joystick); use your imagination!

3.1 System Architecture

As mentioned earlier, the main challenge of the project is to write embedded software, preferably from scratch. However, as the project has to be completed in limited time (7 weeks), a sample skeleton program will be provided that implements the basic ES functionality of communicating commands and sensor readings over the PC link. This program effectively serves as a guide to the library functions operating the various hardware components of the Quadrupel drone. Note that the skeleton program consists of two parts, one for the PC side (e.g., for reading the joystick, and sending/receiving characters over the RS232 link), and one for the drone side (e.g., for reading the sensor values and writing the motor values). The sample program and additional information regarding hardware components and driver protocols are available on the project site [2].

Note that the sample program is purely provided for reference, and a team is totally free to adopt as much or as little as deemed necessary. That is, the team is completely free in developing their software, i.e., the system architecture, component definitions, interface definitions, etc. are specified, implemented, tested, and documented by the team.

3.2 System Development and Testing

An important issue in the project is the order of system development and testing. Typically, the system is developed from joystick to drone (the actuator path), and back (the sensor path), after which the yaw, roll, and pitch controllers are sequentially implemented. All but the last stages are performed in *tethered* mode. Each phase represents a project milestone, which directly translates into team credit. **Note:** each phase must start with an architectural design that presents an overview of the approach that is taken. The design *must first be approved* by the TA before the team is cleared to start the actual implementation.

3.2.1 Actuator Path (mode 0, 1, and 2)

First the joystick software is developed (use non-blocked event mode) and tested by visually inspecting the effect of the various stick axes. The next steps are the development of the RS232 software on the PC, and a basic controller that maps the RS232 commands to the drone hardware interfaces (sensors and actuators). At this time, the safe mode functionality is also tested. If the design is fully tested, the system can be demonstrated using the actual drone.

3.2.2 Sensor Path (mode 3)

The next step is to implement the sensor path, for now, using the on-board motion processor that provides angular information. Calibration (mode 3) is an important step, and (zeroed) data should be recorded (logged) and presented at the PC under simulated flying conditions (i.e., applying RPM to the motors, and gently rotating the drone in 3D).

3.2.3 Yaw Control (mode 4)

The next step is to introduce yaw feedback control, where the controller is modified such that the `yaw` signal is interpreted as setpoint, which is compared to the yaw rate measurement r returned by the sensor path, causing the controller to adjust `ae1` to `ae4` if a difference between `yaw` and r is found (using a P algorithm). Testing includes verifying that the controller parameter (P) can be controlled from the keyboard, after which the drone is "connected" to the ES. By gently yawing the drone and inspecting `yaw`, r, and the controller-generated signals `ae1` to `ae4` via the PC screen, but without actually sending these values to the drone's motors, the correct controller operation is verified. Only when the test has completed successfully, the signals `ae1` to `ae4` are connected to the drone motors and the system is cleared for the yaw control test on the running drone.

3.2.4 Roll/Pitch Control (mode 5)

The next step is to introduce roll, and pitch feedback control, where the controller is extended in conformance with mode 5 using cascaded P controllers for both angles. Again, the first test is conducted with disconnected `ae1` to `ae4` in order to protect the system. When the system is cleared for the final demonstration, the test is re-conducted with connected `ae1` to `ae4`.

3.2.5 Raw sensor readings (mode 6)

This step involves doing the sensor fusion on the main controller, instead of relying on the sensing module's motion processor. Experience has taught that the Kalman filter presented in class [1] looks deceivingly simple, yet the details of implementing it with hand-written fixed-point arithmetic are far from trivial. Therefor the effectiveness of the digital filters must be extensively tested. Log files of raw and filtered sensor data need to be demonstrated to the TAs, preferably as time plots (matlab or gnuplot). Before flying may be attempted the TAs will thoroughly exercise the combined filter and control of your drone, to safeguard against major incidents resulting from subtle bugs.

3.2.6 Height Control (mode 7)

The implementation of this feature requires coding yet-another control loop to stabilize the drone's height automatically, freeing the pilot from this tedious task. It may only be attempted once full control (mode 5 or 6) has been successfully demonstrated to the TAs.

3.2.7 Wireless Control (mode 8)

The above condition (successful demonstration of mode 5 or 6) also applies to flying the drone without the USB cable. In the wireless mode the PC link is operated using the Bluetooth link. The particular challenge of this version is the low bandwidth of the link, compared to the tethered version, which only permits very limited setpoint communication to the ES. This implies that ES stabilization of the drone must be high as pilot intervention capabilities are very limited.

3.2.8 Methodology

The approach towards development and testing must be professional, rather than student-wise. Unlike a highly "student-like", iterative try-compile-test loop, each component must be developed and tested in isolation, *preferably outside of lab hours*, as to optimize the probability that during lab integration (where time is *extremely limited*) the system works. This especially applies to tests that involve the drone, as drone availability is limited to lab hours. The situation where there is only limited access to the embedding environment is akin to reality, where the vast majority of development activities has to be performed under simulation conditions, without access to the actual system (e.g., wafer scanner, vehicle) because it either is not yet available, or there are only a few, and/or testing time is simply too expensive.

3.3 Reporting

The team report, preferably typeset in Latex, should contain the following

- Title, team id, team members and student numbers, date
- Abstract (10 lines, specific approach and results)

- 1. Introduction (including problem statement)
- 2. Architecture (all software components + interfaces)
- 3. Implementation (how you did it, and *who did what*)
- 4. Experimental results (list capabilities of your demonstrator)
- 5. Conclusion (evaluate design, team results, individual performance, learning experience)
- Appendices (all component *interfaces*, component *code* for those components you wish to feature)

Specific items that *must* be included are a functional block diagram of the overall system architecture, the size of the C code, the control speed of the system (control frequency, the latencies of the various blocks within the control loop), and the individual contributions of each team member (no specification = no contribution). The report should be complete but should also be as minimal as possible. In any case, the report *must not exceed 10 pages* (A4, 11 point), *including* figures and appendices. Reports that exceed this limit are NOT taken into consideration (i.e., desk-rejected)!

3.4 Course Grading

Each team executes the same assignment and is therefore involved in a competition to achieve the best result. The course grade is directly dependent on the project result in terms of absolute demonstrator and report quality as well as relative team ranking. Next to team grading, *each team member is individually graded depending on the member's involvement and contribution to the team result.* Consequently, a team member grade can differ from the team grade.

A better result earns more credits. A result is positively influenced by a good design and good documentation (which presents the design and experimental results). A good design will achieve good stability, use good programming techniques, have bug-free software, demonstrates a modular approach where each module is tested before system-level integration, and therefore adhere to the "first-time-right" principle (with respect to connecting to the real drone).

On the debit side, to avoid lack of progress a design may sometimes require excessive assistance from the Teaching Assistants (consultancy minutes), which will cost credits.

3.5 Lab Order

In the absence of the course instructor, the TAs have authority over all lab proceedings. This implies that each team member is held to obey their instructions. Students that fail to do so will be expelled from the course (i.e., receive *no* grade).

Safety (systems dependability) is an important aspect of the code development process, not only to protect the drone equipment, but also to protect the students against unanticipated reactions of the drone. The RPM of the rotors can lead to injury, e.g., as a result of broken rotor parts flying around as a result of a malfunction or crash. Therefore, students who are in the immediate vicinity of the drone *must* stay out of range and wear goggles.

As mentioned at the course web page, lab attendance by all team members is *mandatory* for reasons of the importance of the lab in the course context, and also because of team solidarity. Hence, the TAs will at least perform a presence scan at the start and at the end of each lab session. Any team member whose aggregate AWOL (absent without leave) is more than 30 minutes will be automatically *expelled* from the course.

Although it is logical that team members will specialize to some extent as a result of the team task partitioning, it is crucial that each member fully understands all general concepts of the entire mission. Consequently, *each team member is expected to be able to explain to the TAs what each of his/her team members is doing, how it is done, and why it is done.* The TAs will regularly monitor each team member's performance. Team members that display an obvious lack of interest or understanding of team operations risk having their individual grades being lowered or even risk being expelled from the course. As a special check on member performance, and even fraud, each member must *author* each C *function* he/she has written. Note that C functions *must be single-authored*. Functions that have multiple authors simply implies that those functions are too big, which is unacceptable. The entire source tree (PC and ES) must be made available to the TAs at all times.

Fraud (as well as aiding to fraud) is a serious offense and will always lead to (1) being expelled from the course and (2) being reported to the EEMCS Examination Board. As part of an active anti-fraud policy, all code must be submitted as part of the demonstrator, and will be subject to extensive cross-referencing in order

to hunt down fraud cases. It is NOT allowed to reuse ANY computer code or report text from anyone else, nor to make code or text available to any other student attending CS4140 (aiding to fraud). An obvious exception is the code that has been made available on the CS4140 Resource web site. NOTE: an excuse that one "didn't fully understand the rules and regulations on student fraud and plagiary at the Delft Faculty of EEMCS" will be interpreted as an insult to the intelligence of the course instructor and is NOT acceptable. In case of doubt one is advised to first send a query to the course instructor *before acting*.

Finally, note that the lab sessions only provide a forum where teams can test their designs using the actual drone hardware. However, these sessions are *not* enough to successfully perform the assignment. An important part of the team work has to be performed *outside lab hours*, involving tasks such as having team meetings, designing the software, preparing the lab experiments, and preparing (sections of) the report. In order to enable (limited) experimentation outside lab hours, one flight controller board per team can be made available outside lab hours, once a €50 deposit has been made. The deposit is reimbursed once the board has been returned and it is established that the board is *fully functioning*.

Acknowledgments

The founding father of this course is Arjan van Gemund, who is now enjoying the fine life of a retired professor. He conceived the lab back in 2006, and guided numerous people in implementing essential building blocks (HW and SW) for several generations of quadcopters. Marc de Hoop, Sijmen Woutersen, Mark Dufour, Tom Janssen, Michel Wilson, Widita Budhysusanto, Tiemen Schreuder, Matias Escudero Martinez, Shekhar Gupta, and Imran Ashraf all contributed in one way or another. Despite their fine work, it was decided to overhaul the FPGA-based quad-rotor platform (from Aerovinci) completely, as it was costly to maintain and technology had moved on quite a bit. Cheap, powerful microcontrollers readily available, and accurate sensing out of the box allowed for a radical new design as undertaken by Ioannis Protonotarios (HW + low-level software) and Sujay Narayana (software beta tester), leading to conception of the Quadrupel drone in 2016.

References

[1] CS4140 Course Web Site. `http://www.st.ewi.tudelft.nl/~koen/cs4140/`.

[2] CS4140 Resources `http://www.st.ewi.tudelft.nl/~koen/cs4140/Resources/`.

[3] B. Etkin, L.D. Reid, *Dynamics of Flight, Stability and Control*, 3rd Ed., 1996, Wiley.

[4] QR Simulator. A.J.C. van Gemund. `http://www.st.ewi.tudelft.nl/~koen/cs4140/Resources/`.

A UAV Theory of Operation

In Figure 2 we describe the variables that are standard terminology in describing aerial vehicles location and attitude (NASA airplane standard). We use two coordinate frames, the earth frame and the (drone) body frame. When the drone (body frame) is fully 3D aligned with the earth frame, both x axes are pointing in the direction of the drone's flight direction. The y axes are pointing to the right, while the z axes are pointing *downward*. The variables are defined as follows:

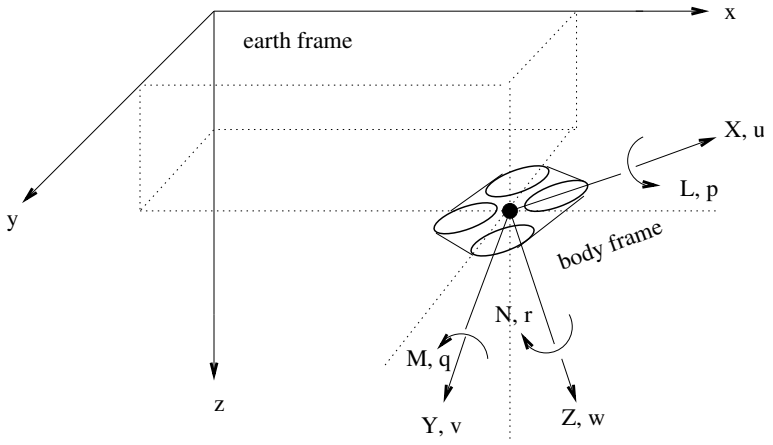

Figure 2: Standard drone variables

- x, y, z: coordinates of the drone CG (center of gravity) relative to the *earth frame* [m]
- φ, θ, ψ: drone attitude (body frame attitude) relative to the *earth frame* (Euler angles) [rad]
- X, Y, Z: forces on the drone relative to the *body frame* [N]
- L, M, N: moments on the drone relative to the *body frame* [N m]
- u, v, w: drone velocities relative to the *body frame* [m s^{-1}]
- p, q, r: drone angular velocities relative to the *body frame* [rad s^{-1}]

The forces X, Y, Z result in (3D) acceleration of the vehicle, while the moments L, M, N result in rotation (change in roll, pitch, yaw). In hover, Z determines altitude (to be controlled by `lift`), while L, M, N need to be controlled by `roll`, `pitch`, and `yaw`, as explained later on.

The Euler angles φ, θ, and ψ are not included in Fig. 2 as their graphical definition is less trivial. They describe the attitude of the body frame relative to the earth frame, and are defined in terms of three successive rotations of the *body frame* by the angles ψ, θ, φ, respectively[3]. Before applying the rotations the body frame is fully aligned with the earth frame. First, the body frame is rotated around the z axis (body frame, earth frame) by an angle ψ. Next, the body frame is rotated around the y axis (body frame) by an angle θ. Finally, the body frame is rotated around the x axis (body frame) by an angle φ. Note, that the order of rotations (ψ, θ, φ) is significant as different rotation orders will produce a different final body frame attitude. The Euler angle definition is shown in Figure 3, where the index (i) denotes the various body frames in the course of rotation (0 initial body frame, 3 final body frame).

The forces X, Y, Z and moments L, M, N are applied externally through gravity, and the four rotors. The above forces and moments are given by

$$\begin{aligned} X &= 0 \\ Y &= 0 \\ Z &= -b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \end{aligned}$$

[3]In drone and helicopter terminology, the three rotations are known as *yaw*, *pitch*, and *roll*, respectively, and are referred to as *heading*, *attitude*, and *bank* in airplane terminology.

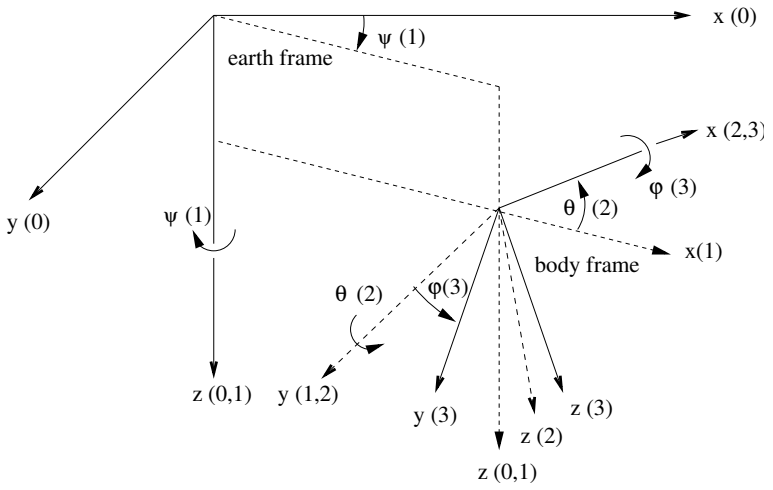

Figure 3: Body frame attitude using Euler angles

$$
\begin{aligned}
L &= b(\omega_4^2 - \omega_2^2) \\
M &= b(\omega_1^2 - \omega_3^2) \\
N &= d(\omega_2^2 + \omega_4^2 - \omega_1^2 - \omega_3^2)
\end{aligned}
$$

where $\omega_1, \ldots \omega_4$ denote rotor RPM, and b and d are drone-specific constants. The above equations directly follow from the rotational direction of the rotors as shown in Fig. 4. The rotor RPM ω_i is approximately proportional

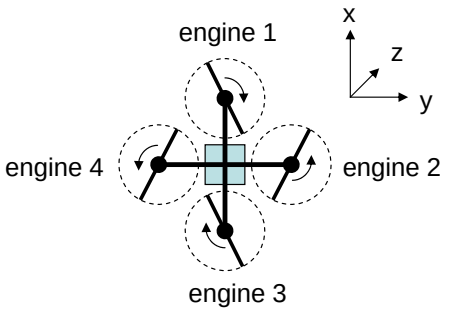

Figure 4: drone rotor layout (top view)

to the voltage applied to the engines, which, in turn, is proportional to the duty cycle c_i of the PWM signals generated by the drone interface electronics, which are controlled by `aei`. (Note that `aei` refers to incorrect terminology, as `e` stands for engine, while the drone's rotors are driven by motors. Switching terminology would be a major undertaking as all developed reference code and documentation refers to these labels, so the `e` should be interpreted as electrical motor.) As a result, the above equations can also be directly expressed in terms of the actuator signals `ae1` to `ae4` according to

$$
\begin{aligned}
X &= 0 \\
Y &= 0 \\
Z &= -b'(ae_1^2 + ae_2^2 + ae_3^2 + ae_4^2) \\
L &= b'(ae_4^2 - ae_2^2) \\
M &= b'(ae_1^2 - ae_3^2) \\
N &= d'(ae_2^2 + ae_4^2 - ae_1^2 - ae_3^2)
\end{aligned}
$$

where b' and d' are drone-specific constants. Note, that in order to compute the `aei` to produce a desired lift

(through Z), roll (through L), pitch (through M), and/or yaw (through N), the above system of equations must, of course, be inverted. An example solution is given in the drone simulator code [4].

The other variables u, v, w and p, q, r are governed by the standard dynamical and kinematic equations that govern movement and rotation of any body in space [3], which are also implemented in the drone simulator [4].

B Signal Definitions

In the interest of clarity and standardization, in following we define some of the most important signal names as to be used in the project. We refer to the generic system circuit shown in Figure 5. The four most important

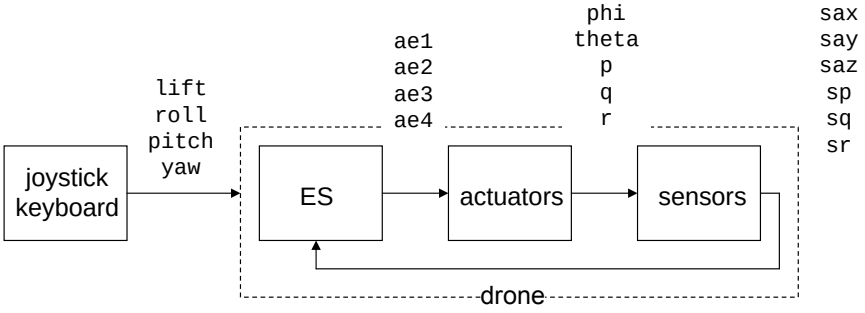

Figure 5: Standard system variables

signals originating from joystick or keyboard (see Appendix C) are called

- `lift` (engine RPM, keyboard: `a/z`, joystick: throttle)
- `roll` (roll, keyboard: `left/right arrows`, joystick: handle left/right (x))
- `pitch` (pitch, keyboard: `up/down` arrows, joystick: handle forward/backward (y))
- `yaw` (yaw, keyboard: `q/w`, joystick: twist handle)

The actuator (servo) signals to be sent to the drone electronics are called

- `ae1` (rotor 1 RPM)
- `ae2` (rotor 2 RPM)
- `ae3` (rotor 3 RPM)
- `ae4` (rotor 4 RPM)

The sensor signals received from the drone electronics are called

- `sp` (p angular rate gyro sensor output)
- `sq` (q angular rate gyro sensor output)
- `sr` (r angular rate gyro sensor output)
- `sax` (a_x x-axis accelerometer sensor output)
- `say` (a_y y-axis accelerometer sensor output)
- `saz` (a_z z-axis accelerometer sensor output)

Note that the sensor signals `sax, sp` etc. do *not* equal the actual drone angle and angular velocity θ, p etc. as defined earlier. Just like other drone state variables such as x, y, z, u, v, w, etc., they can*not* be directly measured (unfortunately), which is why the sensors are there in the first place. Rather, the sensor signals are an (approximate!) indication of the real drone state, and proper determination (recovery) of this state are an important part of the embedded system. For simulation purposes [4], where the true drone state variables are obviously available (i.e., simulated), the drone state variables are called

- `x, y, z` (x, y, z)
- `phi, theta, psi` (φ, θ, ψ)
- `X, Y, Z` (X, Y, Z)
- `L, M, N` (L, M, N)
- `u, v, w` (u, v, w)
- `p, q, r` (p, q, r)

which implies that these names must be used when referring to these signals in the C code.

C Interface Requirements

The following prescribes the minimum interface requirements. Apart from these requirements, appropriate on-screen feedback on drone and embedded systems performance earns credits.

C.1 Joystick Map

- joystick throttle up/dn: `lift` up/down
- joystick left/right: `roll` up/down
- joystick forward/backward: `pitch` down/up
- joystick twist clockwise/counter-clockwise: `yaw` up/down
- joystick fire button: abort/exit

The other buttons can be used at one's own discretion.

C.2 Keyboard Map

- `ESC`: abort / exit
- `0` mode 0, `1` mode 1, etc. (random access)
- `a/z`: `lift` up/down
- left/right arrow: `roll` up/down
- up/down arrow: `pitch` down/up (cf. stick)
- `q/w`: `yaw` down/up
- `u/j`: yaw control `P` up/down
- `i/k`: roll/pitch control `P1` up/down
- `o/l`: roll/pitch control `P2` up/down

The mode 5 P1 and P2 control parameters simultaneously apply to both the pitch and roll cascaded P controllers, which are identical. The other keys can be used at own discretion.

C.3 Drone Map

- `Blue LED`: blink at a 1 s rate to indicate the program is running properly

The remaining LEDs can be used at one's own discretion (typically used for status/debugging).