Attitude & navigation control

Lorenzo Stella, Pantelis Sopasakis and Panos Patrinos*

September 2016

Contents

1 Introduction

The goal of this part is to design a controller that allows the quadcopter to autonomously take-off, move through a sequence of checkpoints, and land.

1.1 Task

The students will be given a model of the dynamical system representing the quadcopter. After identifying the inputs/outputs and matching them against the available actuators/sensors of the quadcopter they will devise, based on that model, an LQR/LQG control loop that allows to:

- control the quadcopter manually using the remote controller (RC);
- navigate autonomously through a series of checkpoints;
- (optionally) perform autonomous take-off and landing.

*Contact us by email at: (P. Sopasakis) `p.sopasakis@gmail.com` and (P. Patrinos) `panos.patrinos@esat.kuleuven.be`

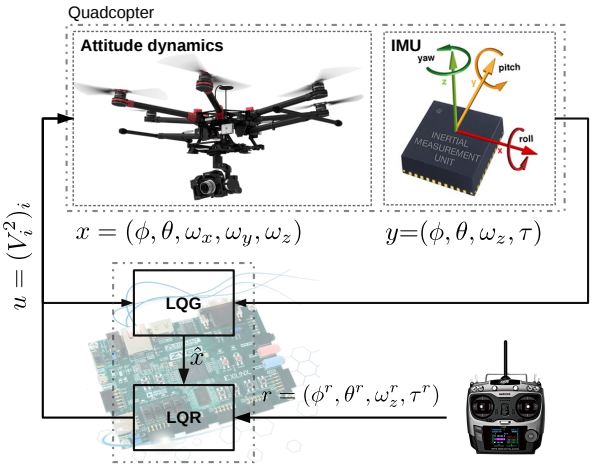

Figure 1: Structure of the attitude control loop involving an LQR controller and an LQG state estimator.

The controller can be structured in a two-level arrangement. An inner control loop (*attitude* controller) will operate at a high sampling frequency using feedback from the gyroscope and accelerometer — the so-called *inertial measuring unit* (IMU) — to maintain the quadcopter in a certain orientation and at a certain position in space. This will be used also, for example, to pilot the quadcopter manually using the radio controller (RC). The design (in a simulation environment) and implementation of such an attitude controller is the first main objective of this project.

For autonomous navigation, an outer control loop (*navigation* controller) will operate at a lower sampling frequency, using feedback from the sonar (for altitude) and camera (for position) and will provide set-points to the attitude controller.

1.2 Provided to the students

The students will be given the following material

- A nonlinear continuous-time mathematical model of the quadcopter,
- Reference material on LQR/LQG control.

1.3 Subtasks

This project will span two semesters. In the winter semester the main objective is to fly the quadcopter and make it respond to commands form the radio controller (RC).

First semester

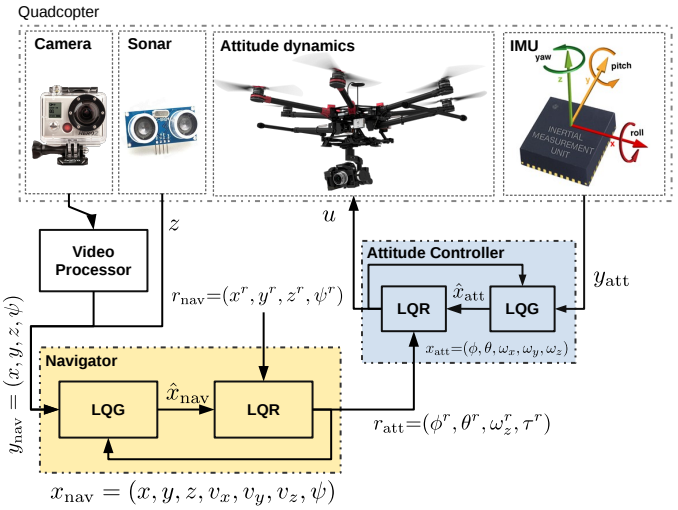

Figure 2: Structure of the navigation system: the RC has been replaced with a navigation system which receives position measurements from a video processor which is connected with an on-board camera and a sonar which measures the altitude.

T1.1.	Material reading + introduction to LQR/LQG.	2 weeks
T1.2.	Implement control loop in simulation environment.	3 weeks
T1.3.	Plug attitude controller onto Zybo[1].	2 weeks
T1.4.	Test attitude controller with RC inputs.	2 weeks

Second semester

T2.1.	Plug navigation controller onto Zybo.	3 weeks
T2.2.	Perform simple maneuvers.	3 weeks
T2.3.	Follow a prescribed path.	3 weeks

1.4 Milestones

M1. Full attitude control and navigation working in the simulation environment

- *Working simulations, with the proposed controller, of the system being able to correctly traverse a sequence of checkpoints (due for T2).*

[1]Zybo is a circuit development board by Xilinx — some basic information about this board can be found at `https://www.xilinx.com/products/boards-and-kits/1-4azfte.html`

M2. Attitude controller working on the quadcopter

- *Practical demonstration of the effectiveness of the attitude controller in manual mode (i.e., it correctly tracks the inputs from the RC) (due for T3).*

M3. Navigation controller working on the quadcopter

M3. *Practical demonstration of the effectiveness of the navigation controller in performing simple maneuvers (due for T4)*

M4. Quadcopter able to follow traverse at least two checkpoints

- *Practical demonstration of the ability of the quadcopter to read the instruction at the take-off site, move to the next checkpoint, read the instruction and move to a third location (due for T5)*

M5. Final demonstration

- *Final demonstration of the fully working quadcopter (due for T6).*

2 Detailed description of each subtask

- *Task 1.1. Introduction to LQR/LQG.* An introductory lecture on LQR/LQG will be given on 10 October.
- *Task 1.2. Implement control loop in simulation environment.* Once a dynamical model is set, its parameters must be tuned according to the real system. Moreover, its inputs/outputs should be identified with sensors/actuators, and noise in measurements and dynamics should be appropriately modeled when devising the LQR/LQG control loop.
  - *Step 1: Identify the system inputs, states and outputs for the attitude and navigation control loops; determine the desired equilibrium point of hovering; linearize the system dynamics about this point; discretize the linear dynamics; compare the linearized system with the nonlinear one (in simulations[2]).*
  - *Step 2: Design an LQR/LQG attitude control system; determine and plot the poles of the controller and the observer; account for input constraints (by saturating the input in your simulations); perform closed-loop simulations using the nonlinear continuous-time system; fine-tune the controller; is it resilient to measurement noise and modeling errors? Plot closed-loop system trajectories and appraise your design choices.*

[2]Any programming language can be used to perform your simulations, e.g., MATLAB, Python, Julia, C/C++, etc. In your simulations, you have to test your controllers against the continuous-time nonlinear model of the quadcopter and account for modelling errors and measurement noise for your simulations to be realistic.

  - *Step 3: Design an LQR/LQG control system for navigation using the above attitude control system.*
  - *Step 4: Implement the attitude controller in ANSI C and test it thoroughly; be wary with units of measurement.*
  - *Step 5: Implement the navigation controller in Python and test it thoroughly.*

- *Task 1.3. Plug attitude controller onto Zybo.* The attitude controller must keep the quadcopter in a given orientation, and should therefore operate at a sufficiently high frequency. Because of this, it is a good idea to implement it in C and run it on the bare-metal core of the Zybo board so as to quickly and directly communicate with the hardware. The efficacy of the control loop can be tested on a test table, and a few iterations might be needed to tune the parameters of the model and the weights of the LQR/LQG controller more accurately.
  - *Details: Plug the attitude controller on Zybo and integrate it properly — for example you will have to provide the control actions via pulse width modulation (PWM); integrate all necessary parts (the controller needs to properly read the set-point signal from the RC and properly communicate the control actions to the motors); test the attitude controller using the RC.*

- *Task 1.3. Test attitude controller with RC inputs.* The attitude controller must be then integrated with the input signals from the RC, so to be able to further test the accuracy of the attitude controller in free flight.
  - *Details: Fine-tune the attitude controller by performing simple maneuvers with the RC; Move the quadcopter around in space and verify that its position coordinates are properly measured by the sonar and video processing system; translate the output of the video processing system into (x, y, ψ) coordinates.*

- *Task 2.1. Plug navigation controller onto Zybo.* The navigation part of the control loop can be implemented in Python on the Linux core, so to easily communicate with other modules (visual feedback, QR code[3] reading). It should moreover communicate with the attitude controller and provide set points to it, and replace the RC in driving the quadcopter throughout the maneuvers.
  - *Details: Integrate your Python implementation on the Zybo platform and test it by providing to it a fixed destination point; calibrate the LQR/LQG parameters; track a given trajectory.*

[3] A QR code is a type of *matrix barcode* — you may find more information at `https://en.wikipedia.org/wiki/QR_code`.

- *Task 2.2. Simple maneuvers.* Once the full control loop is correctly integrated with all the sensors and actuators, the students can proceed with refining the tuning of parameters and weights based on simple maneuvers, such as hovering inside of a square at a given altitude, or moving a few squares along a direction. This will allow, for example, to verify whether the controller is able to hold the position for a QR code to be read, or if its movement allow the camera to correctly sense the (x, y) position without errors.
- *Task 2.3. Follow the path.* The controller should now be fully integrated with other modules, including QR code reading, and ready to be tested on the task of following the sequence of instruction leading from one square to the next until the end. Further calibration of the controller may be required in this phase, and the students can freely decide how to reach the checkpoints: for example, one can perform only horizontal/vertical movements on the grid (easier), or go directly towards the destination, allowing for diagonal movements (harder).
- **Optional:** *Take-off and land autonomously.*

3 Documentation

All documentation related to this module can be found on Toledo (browse to `P&O Elektrotechniek EAGLE`, then `Course Information` in the left sidebar, then `Module Information`, and then `Attitude and navigation control`). This includes:

- A mathematical model of the quadcopter flight dynamics (`model.pdf`).
- A tutorial on the C programming language for people who already know Python, available at `CPY.pdf` on Toledo.
- A good reference for brushing up on LQR/LQG are the lecture notes of S. Boyd (`http://stanford.edu/class/ee363/lectures.html`, lectures 1–3 and 10). See `lqr-lqg.pdf` on Toledo.
- For those interested in a more in-depth understanding of LQR/LQG we propose the lecture notes of J. Hespanha. See `lqrnotes.pdf` on Toledo.