Testing Abstractions for Cyber-Physical Control Systems - Technical Report on the Control Design of the Crazyflie Quadcopter

March 2022

Control Design of the Crazyflie 2.1 Quadcopter

In this technical report we provide an intuition of how the Extended Kalman Filter (**EKF**) and cascaded PID controllers work to control the Crazyflie quadcopter, together with the formalisation of the corresponding equations. This document should provide an intuition of what are the control algorithm specifications provided by the control engineer to the software engineer that actually implements the control software.

In this work we consider a Crazyflie equipped with an Inertial Measurement Unit (**IMU**) sensor, a camera for optical flow, and a vertical laser ranging sensor. The vertical laser provides a direct measure of the distance from the ground, while the combination of optical flow and IMU data allows the drone to estimate the horizontal speed.

The controller combines a state-estimator and a feedback controller. The state-estimator uses sensor readings to estimate the state $x(t)$ of the drone in real time, producing the estimated value $\hat{x}(t)$. The state vector $x(t) = [p(t), v(t), q(t), \omega(t)] \in \mathbb{R}^{13}$ includes the drone position $p(t) \in \mathbb{R}^3$, the drone velocity $v(t) \in \mathbb{R}^3$, the attitude $q(t) \in \mathbb{R}^4$ and the attitude rate $\omega(t) \in \mathbb{R}^3$. Figure 1 shows the axes definition for p. The attitude q is expressed using quaternions[1] and encodes the three angles: pitch (rotation θ around y axis), roll (rotation ϕ around the x axis), and yaw (rotation ψ around the z axis). The state estimate is then communicated to the feedback controller, that uses $\hat{x}(t)$ together with the reference values $r(t)$, to compute the voltage signals to be issued to the motors M_1, M_2, M_3 and M_4 illustrated in Figure 1.

When flying with optical flow data, the state estimator is implemented as an EKF [4, 5], while the feedback controller is a set of cascaded PID controllers [1]. The setup with state-estimator and feedback controller is standard in control theory and common to most control systems. The control design process provides us with equations used to model our quadcopter and equations to describe the estimator and the controller [2, 3].

The State Estimator

An EKF merges three types of information: (**i**) a model containing the a-priori information on the system dynamics, (**ii**) frequent but noisy inertial measurements, and (**iii**) more precise but sporadic optical flow and ranging measurements. Together with the state estimate, the EKF computes and updates the covariance matrix P of the estimate.

The a-priori model consists of two equations retrieved from physical modelling [3]: a state equation in the form

$$\dot{x}(t) = f(x(t), u(t)) + w(t),$$

[1] Quaternions are a four-dimensional extension of complex numbers, and a very convenient tool to represent rotations in the three dimensional space.

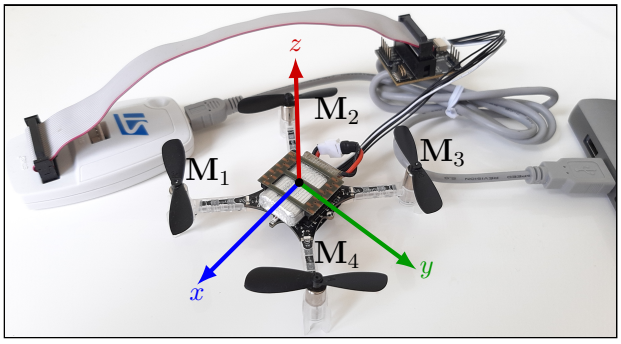

Figure 1: Crazyflie 2.1 with the STM debugger link.

where f describes the derivative $\dot{x}(t)$ of the state vector as a function of the current state $x(t)$, the input $u(t)$, and the process noise $w(t)$. The process noise is used to represent the uncertainty that we have on the a-priori model.

A second equation is used to describe the relation between the current state and the available measurement:

$$y_i(t) = h_i(x(t)) + v(t)\,.$$

Here, v is the noise affecting the measurement, and i is the temporal index of the specific measurement (the measurements being obtained at discrete time instants).

The process uncertainty w and measurement noise v are random variables with respectively covariance matrices Q and R. Based on physical modelling of the Crazyflie [2, 3] the function f can be written as

$$\dot{x}(t) = A(q,\omega)\,x(t) + B(q)\,u(t) + G,$$

where A and B are time-varying matrices that depend on the attitude q (expressed in quaternions) and its rate ω (whose time dependency has been dropped for readability) and $G = [0_{1\times5} - g\,0_{1\times7}]^T$ is the gravity acceleration. The matrices are defined as follows:

$$A(q,\omega) = \begin{bmatrix} 0_{3\times1} & I & 0 & 0_{3\times1} & 0_{3\times1} \\ 0_{3\times1} & -\frac{1}{m}\mathbf{R}_{BG}^T\{q\}D & 0 & 0_{3\times1} & 0_{3\times1} \\ 0 & I & 0 & 0 & -\frac{1}{2}q_v^T \\ 0_{3\times1} & 0_{3\times1} & 0 & 0_{3\times1} & \frac{1}{2}(q_w I + [q_v]_\times) \\ 0_{3\times1} & 0_{3\times1} & 0 & 0_{3\times1} & -I_B^{-1}[\omega]_\times I_B \end{bmatrix},$$

$$B(q) = \begin{bmatrix} 0_{3\times1} & 0_{3\times1} \\ \frac{1}{m}\mathbf{R}_{BG}^T\{q\} & 0_{3\times1} \\ 0 & 0_{1\times3} \\ 0_{3\times1} & 0_{3\times1} \\ 0_{3\times1} & I_B^{-1} \end{bmatrix}.$$

Here, $[\cdot]_\times$ denotes the skew-symmetric operator, $m \in \mathbb{R}$ is the mass of the drone, $\mathbf{R}_{BG}^T\{q\} \in \mathbb{R}^{3\times3}$ is the rotation matrix that rotates a vector from the inertial reference frame to the drone reference frame, $D \in \mathbb{R}^{3\times3}$ is the aerodynamic friction matrix, $q_v \in \mathbb{R}^3$ is a vector with the second, third and fourth components of the attitude quaternion q, $q_w \in \mathbb{R}$ is the first component of the attitude quaternion, I is the inertial matrix of dimension 3, and $I_B \in \mathbb{R}^{3\times3}$ is the inertial matrix.

The measurement equation for the ranging sensor is

$$y_{\text{r-}z} = \frac{x_{(3)}}{\mathbf{R}_{BG(2,2)}\{q\}},$$

where $y_{\text{r-}z}$ is the measured distance from the ground, $x_{(3)}$ is the third element of the state vector x (i.e. the distance of the drone from the ground), and $\mathbf{R}_{BG(2,2)\{q\}}$ denotes the element on the second row and second column of the rotation matrix, which is the cosine of the angle between the z-axis of the drone and the z-axis of the reference frame. The measurement equation for the optical flow in the x direction is (the equation in the y direction can be obtained by swapping y for x and vice-versa)

$$y_{\text{of-}x} = \frac{N_x \Delta t}{\theta_{py}}\left(\frac{\dot{x} R_{BG(2\times2)}\{q\}}{x_{(3)}} + x_{(12)}\right),$$

where $y_{\text{of-}i}$ is the measured pixel count (how many pixels has the image moved from last picture), N_i the number of pixels for the camera in each direction, Δt is the time interval during which the pixels have been counted, θ_{py} is the aperture angle of the camera in the y direction (substituted with θ_{px} for the x direction), and $x_{(12)}$ is the angular rate around the y axis (to be swapped with $x_{(11)}$ for the measurement in the y direction).

The EKF is implemented in three main steps: a prediction step, a covariance update step, and an correction step. The *prediction step* is implemented in the software by the function `kalmanCorePredict()`.[2] In the prediction step, the function f is discretised in time (thus instead of the continuous time $t \in \mathbb{R}$ we have the discrete time steps $k \in \mathbb{Z}$) and used to integrate the accelerometer and gyroscope data and update the state estimate $\hat{x}$. This merges the a-priori model of the dynamics with the IMU data. In the Crazyflie the IMU is sampled every $0.001s$, the prediction step is executed every $0.01s$ and it provides a good state estimate on a short time scale. The *covariance update step* is implemented by the function `kalmanCoreAddProcessNoise()`.[3].

[2] `https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/kalman_core/kalman_core.c`

[3] `https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/kalman_core/kalman_core.c`

It is executed every $0.001s$ (same as the sampling of the IMU) and it updates the covariance P_k of the state estimate. This is done according to the update equation

$$P_k = J_f(\hat{x}_{k-1})\, P_{k-1} J_f^T(\hat{x}_{k-1}) + Q,$$

where J_f is the Jacobian of the function f. This updates the covariance of the state estimate (i.e. how uncertain we are about it) according to the modelled system dynamics and the model uncertainty Q. After several iterations, the noise in the IMU data will make the estimate drift from the actual values. To recover from this drift, the estimate is corrected periodically in the *correction step* using the flow and ranging measurements. Those measurements are more precise than the IMU data since they are not differential quantities – thus no integration is needed to estimate the states. In the Crazyflie software, this is implemented respectively by the functions `kalmanCoreUpdateWithFlow()`[4] and `kalmanCoreUpdateWithTof()`.[5] In the correction step, the specific measurement function h_i from the model is used to compute the expected measurement form the sensors. The difference between the actual measurement y_i and the expected one $\hat{y}_i = h_i(\hat{x}_k)$, is called innovation, and is used to correct the state estimate. This is done according to the equation

$$x_k^c = x_k + K_k(\hat{y}_i - y_i),$$

where x^c is the corrected state and K_k is the so-called Kalman gain. The Kalman gain is computed at every execution on the base of the dynamics of the system, the uncertainty of the model (quantified as a variance), and the variance of the measurement noise, according to the formula

$$K_k = P_k\, J_{hi}(\hat{x}_k)\, [J_h(\hat{x}_k)\, P_k J_f^T(\hat{x}_k) + R]^{-1},$$

where J_{hi} is the Jacobian of h_i. Intuitively, a previous estimate that is more uncertain than the measurement noise (P_k greater than R) will result in a high Kalman gain (since the measurement is more reliable). Conversely a measurement with more variance than the uncertainty of the model (R greater than P_k) will lead to a low Kalman gain (since the process model is more reliable). The correction step is executed according to the availability of the measurements: the specification for the Crazyflie is that the laser ranging sensor should be sampled every $0.025\,s$ and the optical flow sensor every $0.1\,s$.

The Controller

The feedback controller has the objective of computing the voltages to be fed to the four motors to bring the drone in the desired state. To do so, it uses the estimated state $\hat{x}_k$ and reference signal r_k. An equal thrust for the four motors is used to generate vertical thrust and control the vertical z direction. A difference in thrust between the two front motors (M_1 and M_4) and rear motors (M_2 and M_3) is used to make the drone lean forward or backward, i.e. to control the pitch. Analogously, a difference in thrust between the motors on the right (M_1 and M_2) and the two motors on the left (M_3 and M_4) is used to make the drone lean to the side, i.e. to control the roll. When the drone leans in any direction, the motors thrust is not purely vertical any more (in the absolute reference frame) and has some horizontal component as well. This latter component can be used to control the horizontal position: more specifically, a pitch different from zero makes the drone move forward or backward, equivalently, a non-zero roll makes the drone move to the side. Finally, as the pairs of motors on the two diagonals rotate in opposite directions (M_4 and M_2 rotate clockwise, while M_1 and M_3 rotate counter-clockwise), it is possible to leverage the conservation of angular momentum and use a difference between those pairs to make the drone rotate around the vertical axis, i.e. to control the yaw.

The software implements different controllers. In the optical flow setup, in particular these are different PID controllers.[6] Each of the three PID controllers takes care of a specific direction and the code is implemented in a modular way. The vertical position z is controlled by a single PID algorithm. The idea of the PID is that the control action u_k is computed on the base of the error $e_k = r_k - \hat{x}_k$ as the sum of three components:[7]

(**i**) a proportional component $P_k = K_p\, e_k$,

(**ii**) an integral component $I_k = I_{k-1} + K_i\, dt\, e_k$, and

(**iii**) a derivative component $D_k = K_d\, (e_k - e_{k-1})/dt$,

[4] `https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/kalman_core/mm_flow.c`

[5] `https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/kalman_core/mm_tof.c`

[6] `https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/controller_pid.c` `https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/pid.c`

[7] The formulae present in the actual implementation are slightly more complicated due to implementation details that are inessential to provide an intuition of how the controller works.

where dt is the duration of the discrete time-step between subsequent control activations. The control action is defined as

$$u_k = P_k + I_k + D_k,$$

and the values of K_p, K_i, and K_d are the weights of the three components.

To control the horizontal position, a series of four cascaded PID controllers are used for each the x and y direction. Cascaded controllers are used when more measurements are available in order to control a single quantity, and those measurements affect each other sequentially. In our case, for example, these measurements are the pitch rate, the pitch, the longitudinal speed and the longitudinal position. As discussed above, the pitch can cause movement in the longitudinal direction, thus we can use the following dynamical chain: difference in thrust between the pairs $\{M_1, M_4\}$, and $\{M_2, M_3\}$ affects the pitch rate $\dot{\theta}$. The pitch rate in turns affects the pitch θ, which affects the longitudinal speed v_1, which affects the longitudinal position p_1.

This chain is achieved by having one PID controller for the longitudinal position which receives the desired position and computes as control action u_k the reference speed for another PID controller that controls the longitudinal speed. This latter provides the reference for the pitch controller, which provides the reference for the pitch rate controller. The pitch rate controller finally computes the actual voltage value to be actuated to the motors. The very same structure is used for the y direction that is controller through the roll ϕ.

Finally, the yaw is controlled by two cascaded PID controllers: one controlling the ψ that provides the reference for the yaw rate $\dot{\psi}$ controller that computes the difference between the pairs $\{M_1, M_3\}$ and $\{M_2, M_4\}$.

Each of the mentioned PID controller has its own parameters K_p, K_i, and K_d. The attitude and attitude rate PIDs are executed every $0.002\,s$ while the position and speed PIDs are executed every $0.01\,s$.

References

[1] Karl Johan Åström and Tore Hägglund. *Advanced PID Control*. ISA - The Instrumentation, Systems and Automation Society, 2006.

[2] Julian Förster. System identification of the crazyflie 2.0 nano quadrocopter, 2015-08.

[3] Marcus Greiff. Modelling and control of the crazyflie quadrotor for aggressive and autonomous flight by optical flow driven state estimation, 2017. Student Paper.

[4] Mark W Mueller, Michael Hamer, and Raffaello D'Andrea. Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1736, May 2015.

[5] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. Covariance correction step for kalman filtering with an attitude. *Journal of Guidance, Control, and Dynamics*, pages 1–7, 2016.