Stable hovering architecture for nanoquadcopter applications in indoor environments

Sofia Huștiu, Marian Lupașcu, Ștefan Popescu, Adrian Burlacu, Marius Kloetzer
Dept. of Automatic Control and Applied Informatics
"Gheorghe Asachi" Technical University of Iași
Iași, Romania
{hustiu.sofia, lupascu.marian, popescu.stefan, aburlacu, kmarius}@ac.tuiasi.ro

***Abstract*— This research aims at designing a stable hovering architecture for small size quadcopters in indoor environments. The chosen system is a Crazyflie 2.0 nanoquadcopter. First, using the Newton-Euler equations, the dynamic nonlinear model is built. This model allows for simulations and feedback controller design. Second, a 3D indoor environment was created for real-time applications. A Kinect Sensor is considered for real time position measurements, at the same time with obtaining orientations from the gyroscope of the Crazyflie. For practical implementation, a combination between Python and Matlab facilities was considered. The first prototype of the proposed architecture was evaluated for different scenarios and the experimental results are detailed and commented.**

Keywords*— *nanoquadcopter; modelling; indoor application.

I. INTRODUCTION

Quadcopters are popular choices for robotic platforms, due to their agility, simplicity, and wide range of applications. Their first appearance was in the military field [1]. The scope of the quadcopter technology has changed over the years. Therefore, they start to be used in daily application as: delivery services, ground mapping, photography and movie making [2]. A quadcopter is a four rotor helicopter [3]. The particularity of the quadcopter is given by the fact that two opposite propellers spin clockwise, and the other two spin counter clockwise. The study of the quadcopter control problem is interesting because of its complexity: six degrees of freedom [4, 5]. Most commonly used quadcopters are big enough to carry cameras or packages to deliver [6], but they are expensive and require a large space to operate safely.

One of the current challenges is to integrate quadcopters in applications for indoor environments. The reduced space and lack of very important measurement such as GPS increases the complexity of having a stable hovering. For this research, we decided to use small size quadcopters and design a stable hovering architecture for indoor applications.

The chosen type of quadcopters is Crazyflie 2.0 [7-9]. Due to small size, it cannot carry any camera or other sensors besides the IMU and gyroscope. In order to recover the position a RGB-D sensor was included in the indoor environment. Thus, the bias between the internal IMU data and the external position measurements generate the input error needed to be minimized.

The paper begins with problem formulation, where an overview of the system is given. Next, the mathematical model for the nanoquadcopter dynamics is presented. The nonlinear behavior is modeled using the Newton-Euler approach. Also, internal structure of the nanoquadcopter and physical constraints are evaluated. For practical applications we present details about the protocol communication between PC and Crazyflie 2.0, by using Python and Matlab facilities. The constructed model and the developed communication routines will be used for completing a stabilizing architecture, and the current experimental results are discussed in section V. In the last section we address conclusions and future work.

II. PROBLEM FORMULATION

This research investigates the problem of automatically controlling the flight of a Crazyflie 2.0 quadcopter, such that it can follow a desired reference trajectory.

The Crazyflie 2.0 is a nanoquadcopter suitable for indoor environments [7]. It weighs less than 30 grams (including battery) and its diagonal size is less than 10 centimeters. The quadcopter has a 32-bit, 168 MHz ARM microcontroller, an internal gyroscope, and it communicates with a PC over the Crazyradio PA, which is a 2.4 GHz USB dongle that transmits up to two megabits per second in 32-byte packets. Fig. 1 presents the kit containing a Crazyflie 2.0 quadcopter and a Crazyradio PA.

For obtaining a supervised indoor flight of this quadcopter we propose the architecture from Fig. 2, which is comprised of the following main parts:

(i) The quadcopter (plant to be controlled);

(ii) Software interface with the quadcopter, consisting in developing functions for sending commands and reading states;

(iii) Position feedback, i.e. indoor localization, solved by processing images with depth information gathered by a Kinect sensor [10];

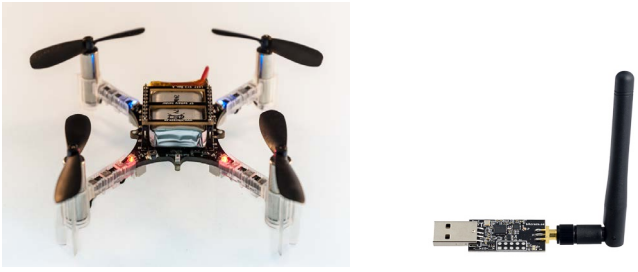

Fig. 1 Crazyflie 2.0 quadcopter (left) and Crazyradio PA communication dongle (right)

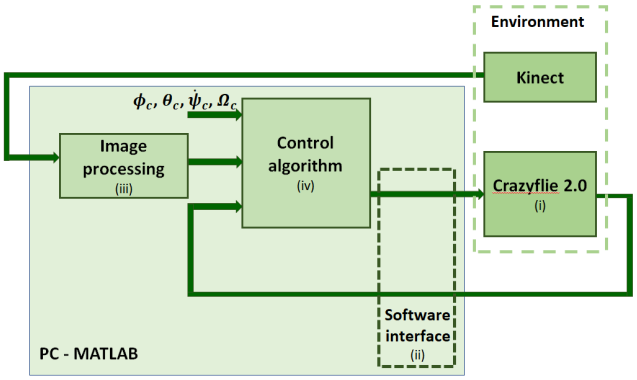

Fig. 2 Proposed architecture for supervised Crazyflie 2.0 experiments

(iv) Algorithm ensuring the stable flight while following a predefined reference trajectory.

This paper studies items (i), (ii) and (iii) from above. More specifically, for item (i) we discuss the quadcopter mathematical model and internal structure (Section III), which is further used for performing simulations and for finding suitable control laws in step (iv). For item (ii) we develop functions for establishing connection with Crazyflie 2.0, for sending commands and for reading important states – Section IV.A. Item (iii) includes functions for reading images and depth information from the Kinnect sensor and for detecting the quadcopter position in the workspace – Section IV.B. It worth mentioning at this point that the software routines for both steps (ii) and (iii) are created with the purpose of calling them from the Matlab environment. This is because many path planning and trajectory following algorithms (embedded in step (iv)) are developed under Matlab, and thus we intend to obtain user-friendly routines for Crazyflie 2.0. Step (iv) is left for future work, and thus this paper details the proposed architecture and workspace for Crazyflie 2.0 real-time experiments.

The workspace in which the quadcopter evolves is shown in Fig. 3, being constructed such that it allows the usage of the architecture from Fig. 2. Starting from a platform that was previously used for controlling planar mobile robots [11], we have added a Kinect sensor and a safety net, while 3D obstacles for further tests in step (iv) can be represented by polystyrene shapes. The working environment footprint is about 4 x 2.5 m, and the height is almost 2.8 m. From this height, the Kinect sensor can cover slightly more than the whole footprint, and an area of about 2.1 x 1.3 m at height of 1 m.

III. Crazyflie 2.0 Model and Structure

Subsection III.A details the mathematical model for the Crazyflie 2.0, while subsection III.B presents its internal structure yielding the available inputs and outputs. The information from this section summarizes important data from the model development and structure given in [12].

A. Dynamic Model for Crazyflie 2.0

To obtain the mathematical model, we consider the following hypothesis [13]:

- The structure is a rigid body;
- The structure is symmetrical;

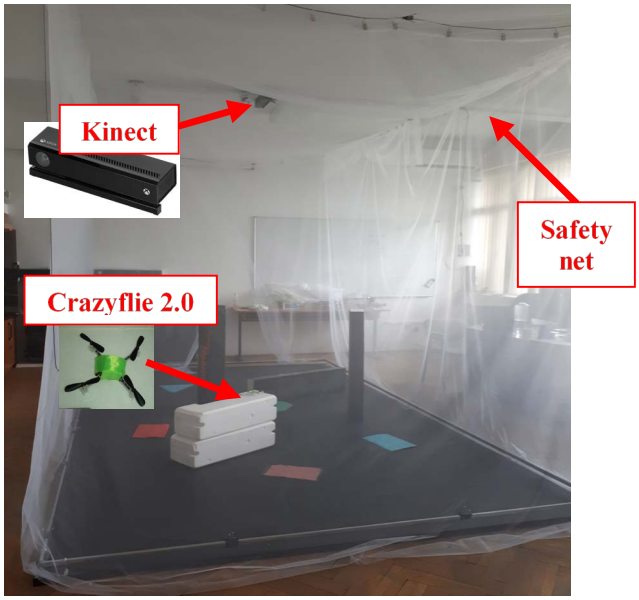

Fig. 3 Working environment

- The Center of Gravity (CoG) is in the middle of the quadcopter;
- The mass is constant.

It is also necessary to define two frames, as in Fig. 4: the body fixed frame in configuration "X", which is default for Crazyflie 2.0, and the inertial frame, denoted as Global inertial Frame (GiF), related to the Earth. Initially, the Crazyflie's body fixed frame is deployed in the origin of the GiF.

The following state variables of the model are chosen, all being expressed in the inertial frame: x, y, z denote the position of CoG, u, v, w are the linear velocities of CoG along the three axes, ϕ, θ, ψ are the roll, pitch, and yaw angles, respectively, and p, q, r are the angular velocities of the quadcopter.

Based on Newton-Euler equations and on the mentioned hypothesis, one can obtain the nonlinear model of the Crazyflie 2.0 quadcopter given by (1). For additional construction details see [12], but bear in mind that the model herein corrects some small errors. Also, we mention that it is possible to derive different quadcopter models as in [8,9], but this work will further focus on model (1).

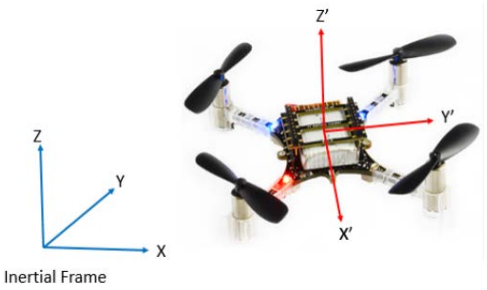

Fig. 4 Inertial frame (left) and body frame (right)

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{\psi}\\ \dot{\theta}\\ \dot{\phi}\\ \dot{u}\\ \dot{v}\\ \dot{w}\\ \dot{r}\\ \dot{q}\\ \dot{p}\end{bmatrix} = \begin{bmatrix} w(s_\psi s_\phi + c_\psi c_\phi s_\theta) - v(s_\psi c_\phi - c_\psi s_\phi s_\theta) + u \times c_\psi c_\theta \\ v(c_\psi c_\phi + s_\psi s_\phi s_\theta) - w(c_\psi s_\phi - s_\psi c_\phi s_\theta) + u \times s_\psi c_\theta \\ w \times c_\psi c_\phi - u \times s_\theta + v \times s_\phi c_\theta \\ p + r \times c_\phi t_\theta + q \times s_\phi t_\theta \\ q \times c_\phi - r \times s_\phi \\ r\frac{c_\phi}{c_\theta} + q\frac{s_\phi}{c_\theta} \\ rv - qw + g \times s_\theta \\ pw - ru - g \times s_\phi c_\theta \\ qu - pv + \frac{U_1}{m} - g \times c_\theta c_\phi \\ \frac{U_2 + pq(I_{xx} - I_{yy})}{I_{zz}} \\ \frac{U_3 + pr(I_{zz} - I_{xx})}{I_{yy}} \\ \frac{U_4 + pq(I_{zz} - I_{yy})}{I_{xx}} \end{bmatrix} \quad (1)$$

In order to simplify notations, model (1) uses notations s_x = sin(x), c_x= cos(x) and t_x = tg(x). Furthermore, m is the Crazyflie's mass, g denotes the gravitational acceleration, and I_{xx}, I_{yy}, I_{zz} are the moments of inertia around x, y, z axis, with numerical values given in Table I.

Table I. Crazyflie 2.0 parameters

Parameter	Value
m	0.0299 [Kg]
g	9.81 [m/s^2]
$I_{xx} = I_{yy}$	1.395 x 10^{-5} [Kg x m^2]
I_{zz}	2.173 x 10^{-5} [Kg x m^2]
C_T	3.1582 x 10^{-10} [N/rpm^2]
C_D	7.9379 x 10^{-12} [Nm/rpm^2]
d	0.03973 [m]

The generic inputs for the quadcopter are the total thrust, denoted by U_1 in (1), and roll, pitch, yaw moments, denoted by U_2, U_3, U_4, respectively. Equation (2) links these inputs to the speeds of the four motors [14]. In (2), ω_i, $i = 1, \dots, 4$ are the rotation speeds of the four DC motors, in rpm (revolutions per minute), C_D is the drag coefficient, C_T is the thrust coefficient , and the quadcopter's arm length is denoted by d, with numerical values from Table I.

$$\begin{aligned} U_1 &= C_T(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \\ U_2 &= dC_T\sqrt{2}(-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2) \\ U_3 &= dC_T\sqrt{2}(-\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2) \\ U_4 &= C_D(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2) \end{aligned} \quad (2)$$

B. *Internal Structure of Crazyflie 2.0*

The Crazyflie 2.0 quadcopter includes some internal controllers that cannot be overridden by the user. Thus, the inputs of the dynamic model (1), (2) cannot be directly send to Crazyflie 2.0, and instead of those, the user can impose the following signals:

- Command thrust Ω_c;
- Command (desired) roll and pitch angles: ϕ_c and θ_c, respectively, expressed in degrees;

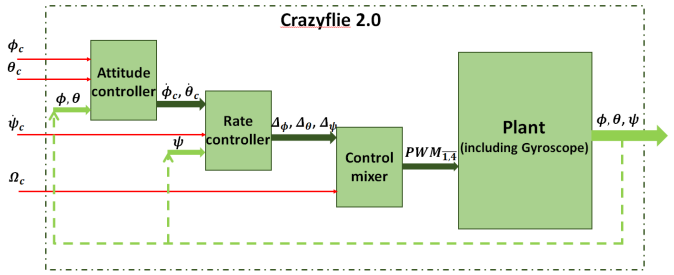

Fig. 5 Internal structure of the Crazyflie 2.0 quadcopter

- Command yaw rate $\dot{\psi}_c$, in degrees per second.

The internal structure of the Crazyflie 2.0 is given in Fig. 5, containing the internal controllers and the plant model. Command thrust Ω_c is given as a 16 bit number, ranging from 0 to 65535. Also, by exploring the Crazyflie's 2.0 firmware, we observe that the internal signals for driving the quadcopter (outputs of the "Control mixer" block) are PWM signals for each motor, rather than rpm values ω_i, $i = 1, \dots, 4$. The translation from PWM to RPM is given in (3).

$$RPM = 0.2685 \times PWM + 4070.3 \quad (3)$$

Thus, the block "Plant" from Fig. 5 basically corresponds to translation (3) and dynamic model (1), (2). Additionally, the quadcopter has a gyroscope sensor that returns the current values of ϕ, θ, ψ states and ensures internal feedback for the next described controllers that came with the stock firmware of the Crazyflie 2.0. The "Attitude controller" is a PI controller that outputs the desired roll and pitch angular velocities $\dot{\phi}_c$ and $\dot{\theta}_c$, based on send commands and on actual feedback ϕ and θ. The "Rate controller" further computes the necessary angular variations $\Delta\phi$, $\Delta\theta$, $\Delta\psi$ to create the angular momentum. The outputs from the rate controller are used as inputs in the last onboard controller. Along with the thrust Ω_c, the "Control mixer" block returns the PWM for each rotor, using relations (4).

$$\begin{cases} PWM_1 = \Omega_c - \frac{\Delta_\phi}{2} - \frac{\Delta_\theta}{2} - \Delta_\psi \\ PWM_2 = \Omega_c + \frac{\Delta_\phi}{2} - \frac{\Delta_\theta}{2} + \Delta_\psi \\ PWM_3 = \Omega_c + \frac{\Delta_\phi}{2} + \frac{\Delta_\theta}{2} - \Delta_\psi \\ PWM_4 = \Omega_c - \frac{\Delta_\phi}{2} + \frac{\Delta_\theta}{2} + \Delta_\psi \end{cases} \quad (4)$$

The gyroscope outputs ϕ, θ, ψ will also be read by our Software interface from Fig. 2, explained in the next section.

IV. Software Interface and Position Detection

A. *Matlab to Crazyflie 2.0 Connection*

The data connection with the nanoquadcopter is done through a 2.4 GHz Crazyradio PA dongle Fig. 2, containing a nRF24LU1 microcontroller and a 20 dBm amplifier that allows communication on a distance of up to 1 Km [15]. Software resources are represented by the libraries [16] developed by the

Crazyflie manufacturers which are written in the Python and Matlab programming language.

In order to communicate with the nanoquadcopter, a Python module was implemented. This module scans all devices at the time of execution, resulting in a list of possible candidates to connect with.

The send/receive procedure is done using a Matlab module by embedding four values: roll ϕ_c (the angle of rotation around the x axis, which takes values between -180 and 180 degrees), pitch θ_c (the rotation angle of the y axis, with the values between -90 and 90 degrees), yaw ψ_c (the rotation angle around the z axis, which can take the values between -180 and 180 degrees) and thrust Ω_c (with range discussed in Section III.B). Moreover, this module is linked through a UDP socket with the communication Python module.

B. Position Detection

In this subsection we discuss about the third step in the supervision architecture depicted in Fig.2. The main goal is to detect the position of the nanoquadcopter. This task is completed by using the data acquired with a Kinect V2 RGB-D sensor [6].

The environment is analysed using the colour image and the depth image. The obstacles distribution is obtained from the colour image, while the 3D mapping is directly linked to the depth image and the Kinect internal parameters. The colour image is calibrated with the depth image, which allows direct access to both colour information and depth data.

The following procedure was used for detecting the nanoquadcopter position: (1) an analysis is carried in the YCbCr space, considering a specific colour marker; (2) the resulting 2D position is mapped into the depth data, which combined with the intrinsic parameters leads to the 3D position. Both the nanoquacopters position and the positions of the obstacles are referred to GiF using equation (5):

$$P_{GiF} = R_{180}^{y} \times P_{Kinect} + T_{2.8}^{Z}, \tag{5}$$

where R is the rotation matrix $(3x3)$ by axis y with 180 degrees, $T_{2.8}^{Z}$ is the translation vector $(3x1)$ by axis z with 2.8 meters). GiF is placed at the end of the z axis of the Kinect.

The delay times introduced by routines mentioned in Section IV will be analysed in the next section, together with model validation and preliminary steps towards real-time experiments.

V. Simulations and Experimental Results

A. Open-loop experiments

For validating the model from Section III, we have simulated the structure from Fig. 5 in Matlab and we compared the results with the behaviour of the Crazyflie 2.0 quadcopter. These are open-loop experiments, since they do not use the feedbacks and the control block from Fig. 2.

We have applied the step input from (6) to both the structure from Fig. 5 (simulations performed in Matlab – Simulink) and to the Crazyflie 2.0 evolving in the environment from Fig. 3.

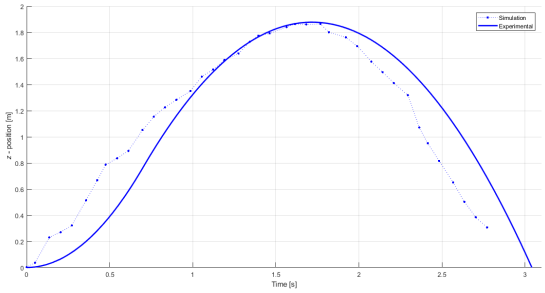

Fig. 6 Evolution of z position: numerical simulation (black curve) and experimental results (blue points)

The angle inputs were kept to 0. The hovering thrust (for which $\dot{z} = 0$) is 41950, so the thrust from (6) should first force the lifting of the quadcopter (until 0.7 s), then the z position should increase some more from inertia, and finally it should decrease.

$$\Omega_c = \begin{cases} 55000, & t \in [0, 0.7]s \\ 35000, & t > 0.7\ s \end{cases} \tag{6}$$

The simulated and the real-time behaviour of z state are given in Fig. 6. As expected, the simulated behaviour has an inflexion point at t = 0.7 s, while the experimental results are slightly with respect to obtained values versus time. Moreover, in numerical simulation the x and y states remained 0 (initial values), but in real time experiments the Crazyflie 2.0 has a fairly strong drift and the obtained x exceeded 1 m in some tests.

This fact shows the necessity for a closed-loop stabilizing control, as in the proposed architecture from Fig. 2. However, this is not captured in the current paper, because currently the delay introduced by image processing from block (iii) (Fig. 2) is around 0.067 s, which proves to be quite high for this fast system. However, using only depth data from the Kinect sensor could reduce the delay of the image processing block to approximately 0.03 s.

On the other hand, the delays introduced by sending commands and reading angles to / from Crazyflie are very

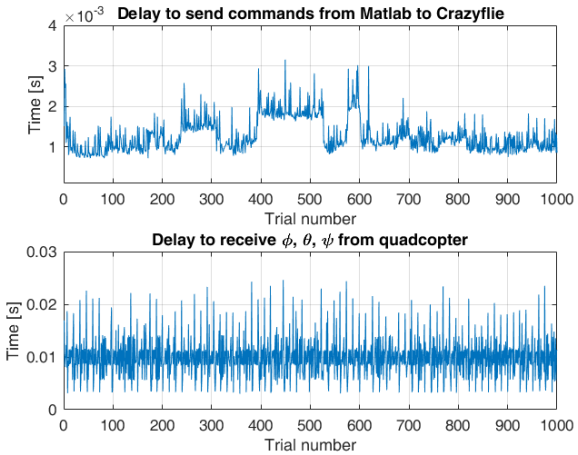

Fig. 7 Delays introduced by block (ii) from Fig. 2

small, about 10^{-3} s and 10^{-2} s, respectively. If one intends to also read the battery level and the current PWMs of each motor, the delay is larger (around 0.1 s), but such readings are not to be done often. Fig. 7 shows the introduced delays for communicating with the quadcopter, on 1000 send and receive function calls (trials).

B. *Closed-loop simulation*

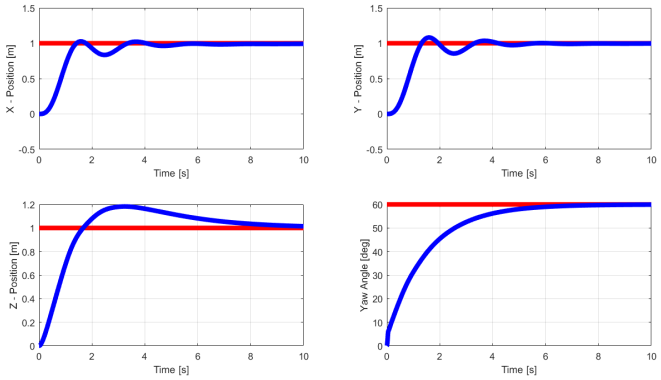

Fig. 8 Simulated results for reaching a stable hovering at $z = 1$ m

In Matlab we have performed some preliminary tests regarding the drone stable hovering. For this, we have replaced block (iv) form Fig. 2 with a PID controller, with parameters tuned around the values returned by Ziegler-Nichols method. Fig. 8 shows the results obtained when imposing constant reference values x, y, $z = 1$ m, ϕ, $\theta = 0$, $\psi = 60$ deg. The results from Fig. 8 show slightly better performances when compared to similar tests reported in [12].

VI. Conclusions

This paper includes our initial steps towards obtaining a real-time control architecture for supervised flight of an indoor nanoquadcopter. After presenting the intended structure, we have presented the results concerning three of the main blocks. First, the model and the internal structure of the Crazyflie 2.0 quadcopter are detailed. Then, the communication routines for sending and receiving information to/from the quadcopter and from the Kinect sensor are mentioned. All the developed functions can be called from Matlab environment, thus ensuring a good support for further including various path generation and trajectory following techniques. Numerical simulations were compared with open-loop results from experimental platform, and closed-loop simulations are mentioned. Time delays introduced by communication and quadcopter detection are also investigated. Further work is directed to reducing the position detection delays and to implementing control algorithms for obtaining a stable hovering and then the flight along a predefined trajectory.

Acknowledgment

This work was partially supported by a grant of Romanian Ministry of Research and Innovation, CNCS-UEFISCDI project PN-III-P1-1.1-TE-2016-0737.

References

[1] M. Bilal Kadri, N. Aziz Jumani, and Z. Pirwani, “Dynamical modelling and control of quadrotor”, Transactions on Machine Design, 4(2), 2016.

[2] J. Preiss, Wolfgang Honig, G. Sukhatme, and N. Ayanian, “Crazyswarm: a large nano-quadcopter swarm”, IEEE International Conference on Robotics and Automation (ICRA), 2017.

[3] T. Luukkonen, “Modelling and control of quadcopter”, Aalto University, Independent research project in applied mathematics, Espoo, 2011.

[4] F. Sabatino, “Quadrotor control: modelling, nonlinear control design, and simulation”, MSc. Thesis XR-EE-RT 2015:XXX, KTH, 2015.

[5] D. Gheorghita, I. Vintu, L. Mirea, and C. Braescu, “Quadcopter control system modelling and implementation”, 19th International Conference on System Theory, Control and Computing (ICSTCC), 2015.

[6] Z. Tahir, M. Jamil, S. A. Liaqat, L. Mubarak, W. Tahir and S. O. Gilani, ”State space system modelling of a quad copter UAV”, Indian Journal of Science and Technology, 9(27), pp. 1-5 , 2016.

[7] Crazyflie 2.0 documentation, available at: https://wiki.bitcraze.io/projects:crazyflie2:index, 2018.

[8] B. Landry, “Planning and control for quadrotor flight through cluttered environments”, MSc. Thesis, MIT, 2014.

[9] M. Greiff, “Modelling and control of the crazyflie quadrotor for aggressive and autonomous flight by optical flow driven state estimation”, MSc. Thesis, Lund University, 2017.

[10] Kinect for Windows SDK, available at: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn799271(v=ieb.10), 2018.

[11] .M. Kloetzer, S. Magdici, and A. Burlacu, “Experimental platform and Matlab toolbox for planning mobile robots”, 16th International Conference on System Theory, Control and Computing (ICSTCC), 2012.

[12] C. Luis and J. Le Ny. “Design of a trajectory tracking controller for a nanoquadcopter”, Technical report, Mobile Robotics and Autonomous Systems Laboratory, Polytechnique Montreal, 2016.

[13] A.A.J. Lefeber. “Controlling of a single drone. Hovering the drone during flight modes.” Technical Report TU/E Eindhoven, Department of Mechanical Engineering, 2015.

[14] C. Balas, “Modelling and linear control of a quadrotor”, MSc. Thesis Cranfield University, 2007.

[15] Crazyradio PA, available at: https://www.bitcraze.io/crazyradio-pa, 2018.

[16] Library imported for application development, available at: https://github.com/bitcraze/crazyflie-lib-python, 2018.