Planning Algorithm for a Quadrotor Drone

X. Liu (5456509), R. Martín Rodríguez (5507391), P. Féry (4660625), Y. Zhang (5390664)

***Abstract*— Micro-Aerial-Vehicles are expected to increase in automomy and utility in the near future. In addition to moving among buildings, they sometimes need to access indoor spaces. In the present document, the study of the motion planning for such transition is developed, for which a reference path will be computed and smoothed. Also, non-linear model predictive control is used for tracking and local obstacle avoidance.**

I. INTRODUCTION

In this report, a study of a quadrotor drone will be conducted. The aim of the project is to compute high performance trajectories to maneuver between preset start- and end-positions, while avoiding obstacles in its way. The project was conducted using Python, the model and the algorithms were custom made, with use of some external libraries [1], [2], [15], [19], [20].

The proposed solution relies on a global planner, specifically *RRT** [4], to generate the global path, and *minimum snap* for trajectory generation and smoothing. For the trajectory tracking, we employed both *non-linear geometric controller* [3], and a self-implemented *non-linear model predictive controller* [6], for an initial solution to local obstacle avoidance. Then, the system will be tested in two tasks, trajectory tracking and local obstacle avoidance, attending to the different obstacles present. Firstly, bounding boxes representing the walls of the environment and will be accounted for during global planning. Secondly, punctual, spherical obstacles will be added as unpredicted obstacles to be avoided by the local planner.

RRT* was chosen as the basis for global path definition instead of PRM or RRT. Its advantages are probabilistic completeness and asymptotical optimality. This will ensure that a path will be found and it will be as efficient - short - as possible. Those advantages make RRT* a very popular choice for quadrotor planning, used as global planner both in static environments [9], [10], and dynamic ones with a local planner, such as dynamic window [8], or MPC [11]. In order to allocate time to the path and smoothing it, minimum-snap-optimization [5] will be used, resulting in a high-quality reference trajectory to be tracked.

Finally, MPC is used as local planner. Conversely to other local planners, it allows to consider obstacles while optimizing some criterion (e.g. spatial or temporal difference with a reference). Its use is extensive in literature for the mentioned features, and has been extensively used, including similar combinations to the one presented [12], [13].

II. ROBOT MODEL

A. Dynamic Model and Equations of Motion

We first define the forces and moments exerted by the system. Considering that a rotor **i** turns at a certain speed ω_i, we can define its force and momentum exerted as $F_i = k_F\,\omega_i^2$ and $M_i = k_M\,\omega_i^2$ (magnitudes of $\mathbf{F_i}$ and $\mathbf{M_i}$ in (1)), where k_F and k_M are the encapsulated lift and moment coefficients of the propellers. Using these values, the total force and momentum exerted by the system are defined in the equations in (1).

$$\boldsymbol{F} = \sum_1^4 \boldsymbol{F_i} - \boldsymbol{F_G}\ ; \quad \boldsymbol{M} = \sum_1^4 (\boldsymbol{r_i} \times \boldsymbol{F_i} + \boldsymbol{M_i}) \tag{1}$$

Where $\mathbf{r_i}$ is the 3-dimensional vector connecting the centre of mass of the quadrotor and rotor **i**. With these values, we can establish the Newton-Euler equations, given that rotor forces F_i are applied in the Z-coordinate of the body frame while F_G (gravitational force), in that of the inertial frame. The linear equation of motion will be defined with respect to an inertial frame, while the angular equation will be defined in a body-fixed frame, in which the tensor of inertia remains constant. The resulting equations are as follows:

$$m\ddot{\boldsymbol{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} \tag{2}$$

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \boldsymbol{u_2} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} u_1 \\ \boldsymbol{u_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ -\gamma & \gamma & -\gamma & \gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{4}$$

Where I represents the tensor of inertia of the drone, $\ddot{\mathbf{r}}$ is the acceleration of the system, p, q and r the angular velocities in the body-fixed frame, and u_1 and $\boldsymbol{u_2}$ are the exerted thrust (scalar) and moments of the system (3-dimensional vector corresponding to the body-fixed axes), defined in (4). Where L is arm length of the robot and $\gamma = k_M/k_F$.

Finally, we will define the orientation of the quadrotor by means of the quaternions, defined from the Yaw-Pitch-Roll angles (ψ, θ and ϕ, respectively) as in (5). Resulting in the state space model in (7) with respect to the state vector **s**.

$$\boldsymbol{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} c(\phi/2)c(\theta/2)c(\psi/2) + s(\phi/2)s(\theta/2)s(\psi/2) \\ s(\phi/2)c(\theta/2)c(\psi/2) - c(\phi/2)s(\theta/2)s(\psi/2) \\ c(\phi/2)s(\theta/2)c(\psi/2) + s(\phi/2)c(\theta/2)s(\psi/2) \\ c(\phi/2)c(\theta/2)s(\psi/2) - s(\phi/2)s(\theta/2)c(\psi/2) \end{bmatrix} \tag{5}$$

$$\boldsymbol{s} = [x, y, z, v_x, v_y, v_z, q_0, q_1, q_2, q_3, p, q, r]^T \tag{6}$$

$$\begin{cases} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\ \begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R(\boldsymbol{q}) \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} \\ \begin{bmatrix} \dot{q_0} \\ \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_3 & -q_2 & q_1 \\ q_2 & q_3 & -q_0 \\ -q_1 & q_0 & q_3 \\ -q_0 & -q_1 & -q_2 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} - 2(\|\boldsymbol{q}\|^2 - 1)\boldsymbol{q} \\ \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \left(\boldsymbol{u_2} - \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \right) \end{cases} \tag{7}$$

Given that R($\mathbf{q}$) is the rotation matrix of the system as a function of the quaternion, see [7].

B. Workspace and Configuration Space

Since our quadrotor will move in the 3-dimensional space, its workspace will be $\mathbb{R}^3$. The coordinates of such workspace are $x-y-z$, describing the robot's longitude, latitude and altitude, respectively.

Given the quadrotor a rigid-body in 3D-space, its configuration space can be defined as $\mathbb{R}^3 \times SO(3)$ (that is, the set of all positions and orientations in 3D space). In addition to the cartesian coordinates, the quadrotor orientation also needs to be described with respect to the inertial frame. The chosen one will be quaternions, as mentioned in subsection II-A, for its low number of parameters and robustness against singularities.

Finally, thanks to the differential flatness of the system, the configuration space for planning can be reduced to $[x, y, z, \psi]$, resulting in $\mathbb{R}^3 \times \mathbb{S}^1$. Furthermore, given that the heading of the quadrotor is not relevant to our problem, its value will be fixed to zero. This allows us to carry out the global planning in $\mathbb{R}^3$, simplifying the task.

III. MOTION PLANNING

As it has already been stated, three different algorithms are used in combination in the present project.

A. RRT*

This sampling-based method first introduced in [4], derives from RRT, which is based on sampling positions in the free configuration space of the robot and checking for feasible connections between them until a path is found to the goal. A further step was introduced in RRT* for ensuring asymptotical optimality and probabilistic completeness. That is, further checking within a certain radius to the new sample whether a new connection exists for which the cost to reach the sample is smaller, and re-wire it accordingly.

The re-wiring radius will adapt to the number of dimensions d and nodes n to ensure the aforementioned properties of RRT*. The expression is taken from [14] and shown in (8), in which γ is a constant further explained in [14].

$$r = \gamma \left(\frac{\log n}{n} \right)^{1/(1+d)} \tag{8}$$

In this case, we have determined a value of nine for γ, following the indications in [14] and further fine-tuning, until a good balance between time consumption and path length was found.

The configuration space defined in II-B allows us to run the algorithm in $\mathbb{R}^3$. For this reason and the fact that the path will be smoothed afterwards, there is no need for a steering function, and so, samples will be connected by straight lines, using euclidean distance as the wiring cost. Finally, an additional function was designed, which makes the longest collision-free connections between points and removes unnecessary ones (points prunning), improving the performance of the optimization algorithms by reducing the total amount of points to optimize for.

B. Minimum snap optimization

Given the path generated by the RRT* algorithm, minimum snap will solve a constrained optimization problem in which a 7-th order polynomial is considered for every segment of the path. The standard cost function and constraints are defined in [5] as a QP problem. The cost function will minimize the snap of the resulting trajectory, expressed in quadratic form considering a vector of variables containing the coefficients of the polynomials of the trajectory. The constraints will impose matching positions of the polynomials at the waypoints, as well as position, velocity, acceleration and jerk continuity.

A further inclusion to the problem is time optimization. While the standard problem requires known time between every two consecutive points, such values will be optimized to improve final performance. To do so, the variable vector is expanded to include the time allocated for each segment of the path. Then, their L2-norm is weighed by a penalty $\mathbf{c}$ and added to the cost function, now minimized by means of the *SLSQP* solver from *scipy* [15].

In order to avoid surpassing the limitations of the quadrotor, actuator constraints were introduced as inequalities in the *SLSQP* solver to guarantee a feasible trajectory, using the approach in [18]. Finally, collision avoidance was introduced by checking collisions of the trajectory with the obstacles, adding additional points to the path and recomputing the trajectory.

C. Nonlinear Model Predictive Control

Our local planner is iteratively solving the optimization problem:

$$\begin{aligned} \underset{x,u}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{N-1} ||x_i - x_{ref}||_Q^2 + ||u_i||_R^2 + ||x_N||_P^2 \\ \text{subject to} \quad & x_{i+1} = f(x_i,u) \quad i = 0,1,2,...N-1 \\ & u_i \leq u_{max} \quad i = 0,1,2,...N-1 \\ & ||p_i - p||_2 \geq r_{drone} + r_{obstacle} \quad i = 0,1,2,...N \\ & p_i \in p_{free} \end{aligned}$$

Where matrices Q, R, P represent state, control, and terminal cost, respectively, inspired by the work [16] and manually tuned. p_i represents the position elements in the state vector x at the time step i; and N is the time horizon of the MPC optimization. Despite the quadratic cost function, the non-linear dynamic model results in a non-linear optimization problem. Which will be solved by a Sequential Quadratic Programming (*SQP*) solver in Forces Pro [2].

We then have two settings for the different tasks: trajectory tracking and local planning. For the trajectory tracking task, our controller is considered to track the reference trajectory closely enough so that no collision with the environment will take place. No obstacle will be in the global path and the last two constraints are inactive. Collision avoidance is then guaranteed by RRT* and minimum snap. In the scenarios where an additional obstacle is on the way, the avoidance maneuver will cause a loss of such guarantee. To avoid this, the last two inequality constraints are added. Firstly, the distance between the current position and that of the obstacle is to be greater or equal than the sum of their radii, given the quadrotor and obstacle are modeled as spheres. Secondly, following the work in [17], we iteratively compute a cubic free space at each time step which will constrain the predicted position within the next n steps.

IV. RESULTS

As stated above, two tasks will be considered: trajectory tracking and local obstacle avoidance. In terms of real applications, the former represents scenarios with no local/dynamic obstacles on the way of the global trajectory; while for more, realistic scenarios -unstructured/dynamic environment-, MPC with local collision avoidance will be introduced. A summary of the results is shown in this video.

A. RRT*

It was observed that the algorithm, as expected, shows a time complexity of $O(n \log n)$. Also, once a number of iterations for which the path is consistently found, the path shortening with larger number of iterations is very small. The number of iterations required to consistently find a path, however, was found to range between 1500 and 2000 for all the scenarios tested. A final value of 2000 - 2500 was chosen, for which the computation time ranged between 100 - 200 seconds.

Regarding the path simplification introduced in section III, while the length of the simplified path is only about 1-5% smaller than that of the original path, it contains 2 to 3 times fewer points, speeding up the processing time of minimum snap with time optimization by up to 85%, as well as the length of the trajectory, as more points resulted in a more sinuous trajectory.

B. Minimum snap

It was observed that the collision avoidance constraint guaranteed collision-free trajectories in all cases. However, it was also observed to be an important source of variance in the computation time, since a single collision in the trajectory requires an entire new optimization. For comparison purposes, computation times will be provided for a single optimization. It is important to notice, however, that collisions in the first run of the optimization were approximately 80% less likely when time optimization was performed, or the total time provided ranged in values close to the optimal one.

The computation time of the basic implementation was observed to range between 3 to 4 seconds. However, it did not ensure that actuator constraints were complied with, forcing the choice of a more conservative completion time. When performing time optimization, the computation time increased significantly. The need for actuator constraints and the added time variables increased the time to 3-7 minutes per optimization for paths of 5-7 points of length. However, the resulting solution provided both with actuator constraints, optimal time allocation and collision avoidance.

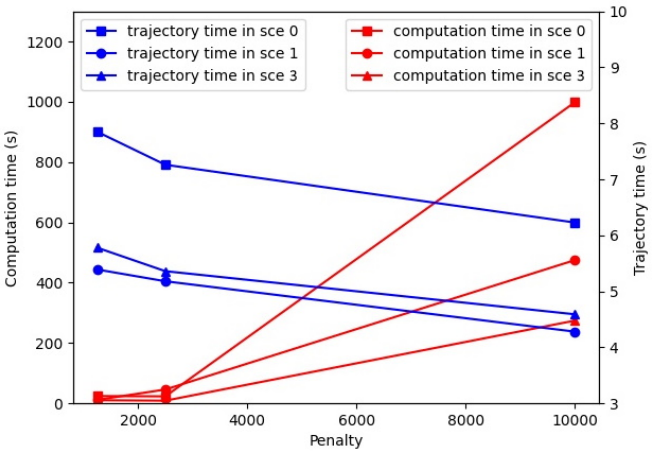

Fig. 1. Computation and trajectory time of minimum snap for different time penalties

The trajectory time is more or less aggressively optimized attending to the value of the time penalty, as observed in Figure 1. A relevant observation, however, was that for small values, up to approximately 25% (approximately 2700) of the initial value of the cost function, the algorithm solved the optimization without surpassing the actuator constraints of the system, even if they were not explicitly formulated

in the optimization (which explains the large in difference computation time in Figure 1. The resulting trajectories were similar to those of the constrained solution in a multitude of different paths. However, the lack of explicit actuator constraints requires the use of a MPC, which can track the reference even if it momentarily moves faster than the quadrotor.

C. Trajectory Tracking

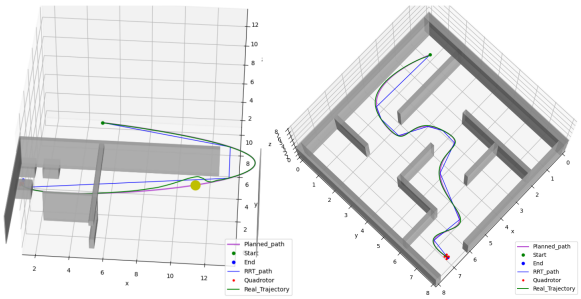

Fig. 2. Test scenarios used for simulation.
Left: the "warehouse", in a test-setup with a local obstacle (the yellow sphere). Right: the "maze", in a setup without local obstacle.

TABLE I
PERFORMANCE OVERVIEW FOR TRAJECTORY TRACKING (MEAN OVER 10 TESTS WITH SAME START AND GOAL).

scenario	controller	total time (s)	integrated tracking squared error (m)
maze	geometric	14.19	0.01
	MPC	17.79	0.97
warehouse	geometric	5.76	0.07
	MPC	5.36	7.15

From the results, we can see that both geometric controller and MPC controller can complete the tracking of the time-optimal trajectory computed by minimum snap reasonably well. It is worth noticing that geometric controller is more suitable for the pure tracking task, as it directly computes the desired thrust and rotation and feeds them into the control input, while MPC works in an optimization manner with being able to take more factors into account (quadrotor attitude, control input, etc). As we can see from Table I, especially in the warehouse scenario, the geometric controller gave much smaller deviation overall. As for the safety guarantee, both controller gave collision-free trajectories in the experiments. The simulation results are satisfying, especially considering the computed high-speed trajectory (24 meters in around 5 seconds).

D. Obstacle avoidance

For local obstacle avoidance, we only use the MPC controller, as geometric controller is not capable of performing obstacle avoidance. It is important to note that, when doing so, the quadrotor deviates relatively much from the reference trajectory and so, collision-free guarantee from minimum snap is not valid anymore. To avoid this, a the larges collision-free bounding cube is used as optimization constraint for the MPC, bounding the predicted quadrotor position within the cube for the next k time steps. Then, the policy can be made more conservative (safer but slower) by increasing k.

We tested such method in the warehouse scenario, the MPC controller was able to compute a collision-free local trajectory regarding both global and local obstacles. However, in order to bound the next k steps within the box, the total time of the trajectory needs to be increased with respect to the optimal (from approximately 5.36s to 8.57s). Otherwise, a conflict between the speed of the quadrotor and the strict bounding constraints results in infeasible solution of the optimization.

V. DISCUSSION

The system performs with proficiency at the tracking task. In static environments, the solution provides high speed trajectories that make full use of the actuation capabilities of the quadrotor and in a collision-free way. However, the computation time required to obtain peak performance was very high, in particular when introducing the actuator constraints. This results in an impossibility to apply the solution in real scenarios, unless trajectories can be computed offline.

For the collision avoidance task, the system proved capable of avoiding the punctual objects on the way. However, further tuning of the MPC would be needed to reduce the deviation from the reference. When accounting for static obstacles, the current solution required both a slower trajectory and a wider environment to maneuver. Also, when performing the avoidance maneuver in combination with cornering, or close to static obstacles, the system failed at avoiding the obstacles. This limits the usability of the solution and further work should be done in this regard, as will be discussed ahead.

A. Further works

As mentioned before, a faster solution for both actuator constraints in minimum snap and more robust MPC obstacle avoidance would be required in order to improve the practical usability of the solution. For the former, different solvers would need to be tested aiming for reduced computation time. For the latter, reformulation of the problem as a mixed-integer nonlinear program in which objects are avoided by means of six half-plane constraints per object, as in [21], would result in the optimal behavior, at the cost of higher computational load.

Also, it is important to notice that both the environment and obstacles were known and the yaw angle has not been considered. In real scenarios, this should be accounted for, since the orientation of mounted sensors is important for detecting obstacles. This would also allow for the implementation of incremental RRT*, more suitable to unknown environments.

REFERENCES

[1] J. D. Hunter, *"Matplotlib: A 2D Graphics Environment"*, Computing in Science & Engineering, vol. 9, no. 3, 2007, pp. 90-95

[2] Alexander Domahidi and Juan Jerez, *"FORCES Professional"*, Embotech AG, url=https://embotech.com/FORCES-Pro, 2014-2019

[3] T. Lee, M. Leok, N. H. McClamroch *"Geometric tracking control of a quadrotor UAV on SE(3)"*, 49th IEEE conference on decision and control (CDC). IEEE. 2010, pp.5420-5425

[4] S. Karaman, and E. Frazzoli, *Sampling-based algorithms for optimal motion planning*, Int J Rob Res, vol. 30, pp. 846-894, 2011

[5] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors*, in 2011 IEEE international conference on robotics and automation. IEEE, 2011, pp. 2520–2525

[6] David Q. Mayne et al., *Constrained model predictive control: Stability and optimality*, Automatica, vol. 36, no. 6, pp. 789-814, 2000

[7] Basile Graf, *Quaternions and Dynamics*, 2007, p. 3

[8] T. Jia, S. Han, P. Wang, W. Zhang and Y. Chang, *Dynamic obstacle avoidance path planning for UAV*, 2020 3rd International Conference on Unmanned Systems (ICUS), 2020, pp. 814-818, doi: 10.1109/ICUS50048.2020.9274865.

[9] B. Tong, Q. Liu, C. Dai and Z. Jia, *A RRT*-based kinodynamic trajectory planning algorithm for Multirotor Micro Air Vehicle*, 2020 IEEE International Conference on Information Technology,Big Data and Artificial Intelligence (ICIBA), 2020, pp. 422-437, doi: 10.1109/ICIBA50161.2020.9277168.

[10] G. Kulathunga, D. Devitt, R. Fedorenko, S. Savin and A. Klimchik, *Path Planning Followed by Kinodynamic Smoothing for Multirotor Aerial Vehicles (MAVs)*, 2020 International Conference Nonlinearity, Information and Robotics (NIR), 2020, pp. 1-7, doi: 10.1109/NIR50484.2020.9290162.

[11] P. Lin, S. Chen and C. Liu, "Model predictive control-based trajectory planning for quadrotors with state and input constraints," 2016 16th International Conference on Control, Automation and Systems (ICCAS), 2016, pp. 1618-1623, doi: 10.1109/ICCAS.2016.7832517.

[12] A. Iskander, O. Elkassed and A. El-Badawy. *Minimum Snap Trajectory Tracking for a Quadrotor UAV using Nonlinear Model Predictive Control*. 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), 2020, doi: 10.1109/NILES50944.2020.9257897.

[13] P. Niermeyer, V. S. Akkinapalli, M. Pak, F. Holzapfel and B. Lohmann, "Geometric path following control for multirotor vehicles using nonlinear model predictive control and 3D spline paths," 2016 International Conference on Unmanned Aircraft Systems (ICUAS), 2016, pp. 126-134, doi: 10.1109/ICUAS.2016.7502541.

[14] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli and M. Pavone, "Revisiting the Asymptotic Optimality of RRT," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 2189-2195, doi: 10.1109/ICRA40945.2020.9196553.

[15] Pauli Virtanen et al. (2020) *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*. Nature Methods, 17(3), 261-272.

[16] Song, Yunlong and Scaramuzza, Davide, "Policy Search for Model Predictive Control with Application for Agile Drone Flight", IEEE Transaction on Robotics, 2021

[17] B. Brito, B. Floor, L. Ferranti and J. Alonso-Mora, "Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments", IEEE Robotics and Automation Letters, 2019

[18] T. Talha, and M. Nahon, "Constrained Control Allocation Approaches in Trajectory Control of a Quadrotor Under Actuator Saturation.", 2020 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2020

[19] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, pp. 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (https://www.nature.com/articles/s41586-020-2649-2).

[20] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings and Moritz Diehl. *CasADi - A software framework for nonlinear optimization and optimal control*. Mathematical Programming Computation (2018).

[21] Kushleyev, Mellinger, Kumar, "Towards A Swarm of Agile Micro Quadrotors", DOI: 10.15607/RSS.2012.VIII.028, 2012.