Autonomous Navigation in Dynamic Environments using Monocular Vision

Liang Lu*, Javier Rodriguez-Vazquez, Adrian Carrio and Pascual Campoy
Universidad Politecnica de Madrid, Calle Jose Gutirrez Abascal 2, Madrid (Spain)

ABSTRACT

Autonomous navigation in dynamic unknown environments is a key research topic in robotics and has gained a lot of attention from the research community in the last years. In this paper, we propose a strategy for autonomous navigation in an environment with spheric obstacles. In this strategy, we use the YOLOv3 object detection model to detect the obstacles and an Iterative Perspective-n-Point (PnP) algorithm to estimate the center of the obstacle based on the result from the detector. Then using the obstacles' positions, a Receding Horizon Control (RHC) based planner is used to plan a trajectory using Covariant Hamiltonian Optimization (CHO) function, and a trajectory controller controller based on Model Predictive Control (MPC) is used to fly the planned trajectory. The proposed navigation strategy is evaluated with Rotors Gazebo simulator in a dynamic environment. Experimental results show that our autonomous navigation strategy is a valid approach for Unmanned Aerial Vehicle (UAV) navigation in dynamic environments.

1 INTRODUCTION

Navigation in dynamic environments is a key challenge in the area of autonomous navigation. It is still an unsolved problem because of the requirements of fast perception, planning and control algorithms when robots operating in a dynamic environment. During the last years, this problem has attracted a lot of researchers' attention and several methods have been proposed [1, 2, 3]. Among these methods, the following ones have gained a lot of interest within the research community, Artificial Potential Field (APF), geometry-based approach, Velocity Obstacle (VO), Partially Observable Markov Decision Process (POMDP), learning-based method and sampling-based strategy.

Our method can be thought of as a kind of geometry-based strategy because we describe the obstacles using geometric models (spheric obstacle in this case). In our method, first of all, we build a dataset of the obstacles we want to avoid and train the tiny version of the YOLOv3 model [4] to detect them. Next, the 3D positions of the centers of the obstaces will be computed using an Iterative PnP algorithm. Then, an online trajectory planner uses an RHC framework will be applied to plan a path. Finally, an MPC trajectory controller is employed to fly the planned trajectory.

The remainder of the paper is organized as follows. Section II presents problem formulation. In Section III, we introduce the proposed methodology. We show the experimental results and discussion in Section VI. And finally, Section IV concludes the paper and summarizes future research directions.

2 PROBLEM FORMULATION

2.1 Robot Model Assumption

A multirotor UAV has been used in this research. The multirotor UAV has 6 Degrees of Freedom (DoF), 3 DoF in translation and 3 DoF in rotation, and can fly freely in the 3-dimensional environment. The on-board sensors of the UAV are a front RGB camera and an Inertial Measurement Unit (IMU) sensor. The front RGB camera is used to detect obstacles within its the Field of View (FOV) and the IMU sensor is used for estimating the pose of the UAV. In this paper, the UAV will be modelled as a sphere that fully contains the UAV.

2.2 Environment Model Assumption

The operating environment in this paper is an environment with dynamic obstacles, modelled as spheres. The size of the obstacles is given but their positions are unknown. The initial point P_{init} and goal point P_g are given and the robot should move from P_{init} to P_g without collision. The robot will be thought as reaching P_g when the distance from the center of the robot to P_g is smaller than the threshold L_G. The environment model which is used in the paper is shown in Figure.1.

3 METHODOLOGY

The description of the proposed architecture for autonomous navigation is shown in Figure.2. There are 3 main parts in it, which are **robot localization** , **obstacles detecting and pose estimation** and **online RHC trajectory planner and controller**. The part of the robot localization is based on our previous work [5].

3.1 Obstacles Detection and Pose Estimation

In this paper, we train tiny YOLOv3 to detect the obstacles. It's an object detector that uses features learned by a deep convolutional neural network to detect an object. Given

*Email address(es): liang.lu@upm.es

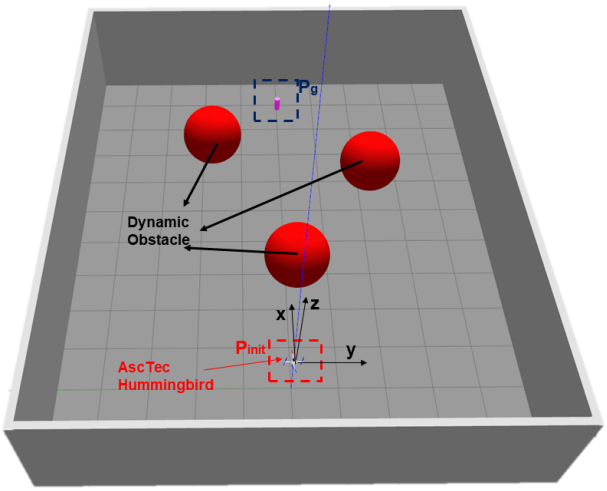

Figure 1: The environment model with dynamic obstacles.

the controlled experimentation conditions of this work, a very naive obstacle detection method (for example thresholding by color and circle detection) based on classical computer vision can be designed, but this could not work on future experimentation that includes real world flights. The reason of using YOLO instead of an easier method is to take un account the noise that would be introduced by the detector in the real world. The detector trained by tiny YOLOv3 has a very high frame rate and can be run real-time with a Graphics Processing Unit (GPU). The average processing time is 15ms using a GeForce GTX 1050Ti GPU. The output of the detector are the bounding boxes which enclose the detected obstacles. The dataset used to train the network has been derived from OpenImages V4.

In order to calculate the pose of the obstacles with sufficient accuracy, the bounding boxes should enclose the obstacle completely. And because the obstacles in our paper are spheric obstacles, the bounding boxes should have a square shape. As it can be seen from Algorithm 1, if the quotient between the bounding box's width w and height h is within a predefined range of $[\sigma_0, \sigma_1]$, the bounding box will be considered valid and used to compute the center of the obstacle.

With selected bounding boxes, we use an Iterative PnP algorithm [6] to compute the 3-dimensional position of the center of the obstacles. The average processing time of the iterative PnP is 14 ms when we run it a laptop with an Intel Core i7-8750H CPU..

3.2 *Online RHC trajectory planner and controller*

The RHC framework based trajectory planner is used in order to find the best trajectory from a trajectory library. In this paper, first, we use some ideas from Andreas Bircher *et al* [7] to generate online goal candidates and some ideas from Zheng Fang *et al* [8] to build the CHO objective function. Then the CHO objective function is used to generate a trajectory library. The initial point and goal point of the path are

Algorithm 1 Bounding Boxes Filter

Input: $Bounding\ Boxes\ From\ Detectors$
Output: $Bounding\ Boxes\ Selected$
1: $B_0 \leftarrow Bounding\ Boxes\ From\ Detectors$;
2: **for** $Bounding\ Boxes\ B\ in\ B_0$ **do**
3: **if** $\sigma_0 < B.width/B.height < \sigma_1$ **then**
4: $Bounding\ Boxes\ Selected \leftarrow B$;
5: **end if**
6: **end for**

the current robot position and the generated goal candidates, respectively. Finally, the trajectory with the lowest objective function value is selected as the best trajectory.

The Rapid Random Tree (RRT) framework [9] is used to generate the goal candidate nodes. As it can be seen from algorithm 1, first we set the number of maximum goal candidate nodes N_{max}, next a random node is generated in the predefined goal candidates searching area V_S, then we find a goal candidate by $Nearest$ and $Steer$ function from the RRT framework and store this goal candidate. The $Nearest$ function is responsible for searching for $P_{nearest}$ and $Steer$ function is used for generating P_{new}, detail information about these two functions can be found from [9]. At last, if the number of goal candidates is larger than N_{max}, the online goal candidates generation process is finished.

A modified objective function for CHO is built to create the trajectory library and search for the best trajectory. Our objective function measures four different aspects of the trajectory planning problem. First, in order to get a smooth path, we add a penalization based on dynamical criteria, like velocities and accelerations to the trajectory. Next, we penalize the trajectory by the distance from the trajectory waypoint to the objects to make the trajectory avoid obstacles. Then, the end of the trajectory is penalized by the distance from it to the final goal, which can help the trajectory planner plan a trajectory close to the final goal ξ_g. Finally, we penalize the trajectory by the distance from its waypoint to the ground to make the trajectory go far away from the ground. We describe these four items by f_s, f_o, f_g, f_a respectively, and define our objective function by summing their weights:

$$f(\xi) = w_1 f_s(\xi) + w_2 f_o(\xi) + w_3 f_g(\xi(1)) + w_4 f_a(\xi) \quad (1)$$

As described above, the trajectory is ξ and $\xi(s)$ is the function mapping the trajectory length s to the robot configurations, the initial and end configurations of the trajectory ξ is $\xi(0)$ and $\xi(1)$ respectively. The waypoints in the trajectory ξ are 3 DoF point $\{x, y, z\}$. w_1, w_2, w_3 and w_4 are the weights for each objective functions.

The objective functions of f_s, f_o, f_g are the same from [8]:

$$f_s(\xi) = \int_0^1 c_o(\xi(s)) \| \frac{\mathrm{d}}{\mathrm{d}t} \xi(s) \| \mathrm{d}s \quad (2)$$

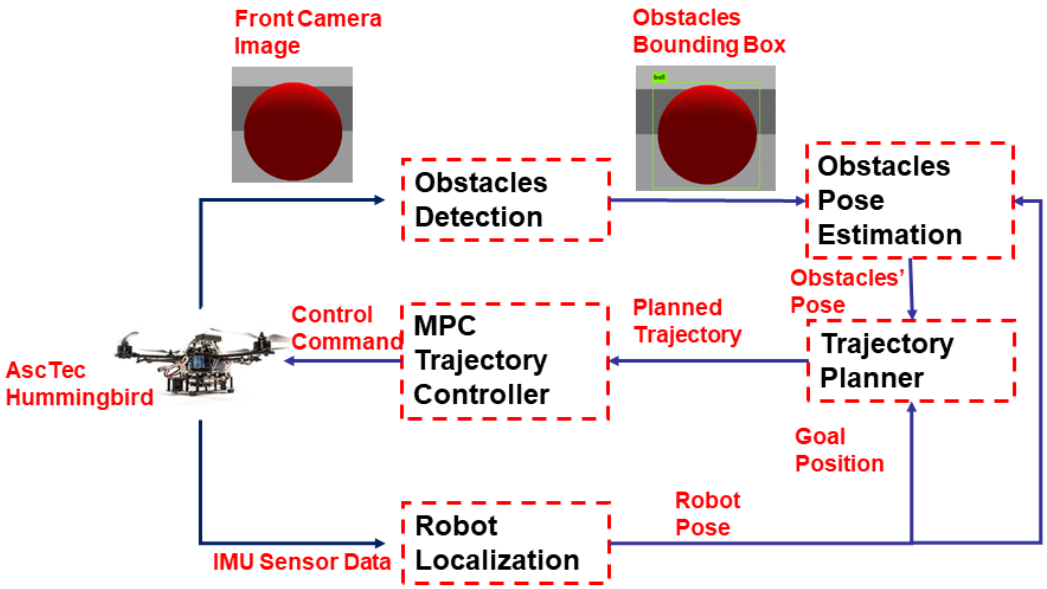

Figure 2: Architecture of the autonomous navigation method proposed.

$$f_o(\xi) = \frac{1}{2}\int_0^1 \|\frac{\mathrm{d}}{\mathrm{d}t}\xi(s)\|^2 \mathrm{d}s \tag{3}$$

$$f_g(\xi(1)) = \|\xi(1) - \xi_g\| \tag{4}$$

$c_o(\xi(s))$ in the objective function is the obstacle cost for the spheric obstacle obstacle:

$$c_o(\xi(s)) = \frac{w_0 R_o}{3}(1 - \frac{Dist(\xi(s))}{R_o})^3 \tag{5}$$

w_0 is the weight and R_o is the radius of the spheric obstacle obstacle. $Dist(\xi(s))$ is the distance from the waypoints in the trajectory to the center of the spheric obstacle.

The objective function of f_a is learned from [10]:

$$f_s(\xi) = \int_0^1 c_a(\xi(s))\|\frac{\mathrm{d}}{\mathrm{d}t}\xi(s)\|\mathrm{d}s \tag{6}$$

$c_a(\xi(s))$ in the objective function is the altitude cost:

$$c_a(\xi(s)) = \begin{cases} (Alt(\xi(s)) - \epsilon)^2, & Alt(\xi(s)) \leq \epsilon \\ 0, & Alt(\xi(s)) > \epsilon \end{cases} \tag{7}$$

$Alt(\xi(s))$ is the altitude of the waypoints of the trajectory ξ and ϵ is predefined the minimum value of the altitude.

The path library is generated by using the objective function $f(\xi)$ and the best trajectory is the trajectory with the lowest objective function value and optimized by minimizing the objective function $f(\xi)$.

After obtaining the trajectory, an MPC based trajectory controller which is similar to [11] is used to follow the robot to correctly follow the planned trajectory.

While the robot is flying along the trajectory, we will check the status of several trajectory points in front of the robot. If the distance from one of the trajectory points to the center of the obstacles is smaller than the obstacles' radius, the trajectory planner will search for a new trajectory to fly.

Algorithm 2 Online Goal Candidates Generate

Input: *Current Robot Position*
Output: *Goal Candidate Nodes*

$P_0 \leftarrow Current\ Robot\ Position$;
$T \leftarrow P_0$;
$N_T \leftarrow 0$;
while $N_T < N_{max}$ **do**
  $P_{rand} \leftarrow SampleFree(V_S)$;
  $P_{nearest} \leftarrow Nearest(T, P_{rand})$;
  $(P_{new}, T_{new}) \leftarrow Steer(P_{nearest}, P_{rand})$;
  $Put\ P_{new}\ in\ Goal\ Candidate\ Nodes$
  $N_T \leftarrow N_T + 1$;
end while

4 EXPERIMENTAL RESULTS AND DISCUSSIONS

4.1 Experimental Setup

RotorS Gazebo simulation environment and Robot Operating System (ROS) have been used under Ubuntu 18.04. The Rviz/Gazebo environment can use real physical param-

eters of the robot and environment. All the experiments run on a laptop with Intel Core i78750H at 2.2GHz, a GeForce GTX 1050Ti GPU. The simulation of the proposed navigation method is integrated into our open source framework Aerostack[1]. The used environments are 3D indoor environments. The UAV in the simulation is the AscTec Hummingbird, which is equipped with a front RGB camera. The front camera takes charge of receiving information from the environment in which the robot operates. The UAV can fly freely in the 3-dimensional environment.

4.2 *Evaluation of Obstacle Pose Estimation*

We build the environment which is shown in Figure 3 to evaluate the performance of our obstacle pose estimation algorithm. The size of the environment is $10\,m \times 12\,m$. The multirotor UAV hovers at the point $(0, 0, 1.1)\,m$, the proposed approach for obstacle detection and pose estimation strategy are used to calculate the center of the obstacles by using the images captured from the UAV's front camera. We test our obstacle pose estimation strategy with two different spheric obstacles. The radius of these spheric obstacles is $1\,m$ and $1.5\,m$, respectively. The position of obstacle will generate randomly in the obstacle area. The obstacle area is a cube area, the center of the area is $(5, 0, 1.125)\,m$. The length (L), width (W) and height (H) of the obstacle area is $6\,m$, $6\,m$ and $1.5\,m$ respectively. We run our algorithm 1000 times for the

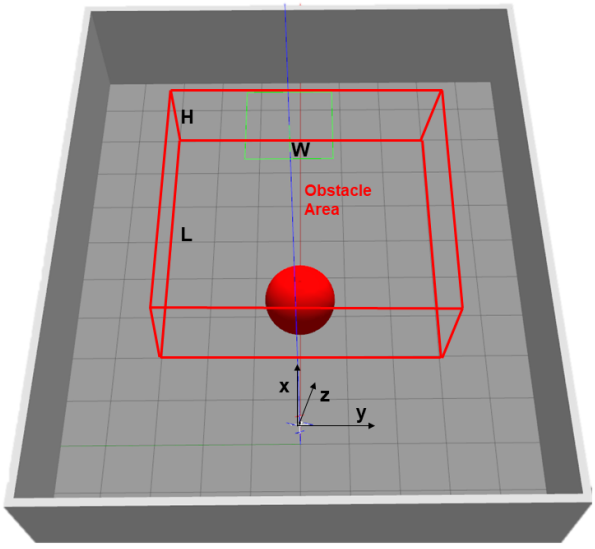

Figure 3: The environment used to evaluate the performance of obstacle pose estimation algorithm.

spheric obstacles mentioned above respectively, and compute the error which is the euclidean distance of the estimated obstacle center position and Gazebo ground truth. Then, Max Error (MaxE), Min Error (MinE), Mean Error (ME) and Root Mean Square Error (RMSE) is calculated and can be seen from Table 1

	obstacle radius = 1m	obstacle radius = 1.5m
MaxE (m)	1.9534	3.3951
MinE (m)	0.0105	0.0135
ME (m)	0.3061	0.2981
RMSE (m)	0.2315	0.3114

Table 1: Results of obstacle pose estimation.

4.3 *Experiments of Autonomous Navigation*

We use the environment shown in Figure 1 to test the performance of our autonomous navigation system. In the environment, there are 3 dynamic obstacles which have a sinusoidal trajectory. The initial point of the robot is $(0, 0, 0)\,m$. The coordinate x, y and z are randomly generated between $[8, 8.5]\,m$, $[-3, 3]\,m$ and $[0.75, 1.5]\,m$, respectively. We run our autonomous navigation algorithm 50 times in the environment. Table 2 shows the results after running the algorithm 50 times. In the table, the Successful Rate (SR), Path Length (PL), Time to the Goal (TG), Maximum Velocity (MaxV) and Mean Velocity (MeanV) express the performance of successful flight from the initial point to the goal point, the mean length of the path, the average time to reach the goal, the average maximum velocity, and the average velocity for the 50 runs.

SR (%)	82
PL (m)	9.8848
TG (s)	28
MaxV (m/s)	0.7012
MeanV (m/s)	0.3223

Table 2: Results after 50 runs of our navigation algorithm in the dynamic environment.

Figure 4 shows 3 flying trajectories from the 50 runs. The figure of top left, bottom left and bottom right correspond respectively to the top view, left view and normal view of the 3 flying trajectories in the test environment. In the figure, the red point is the initial point which is $(0, 0, 1.1)\,m$, the yellow, green and blue points are the goal points for different trajectories and the purple, brown and blue line lines correspond to the 3 different trajectories. The red spheric obstacles are the dynamic obstacles which move with a sinusoidal trajectory.

A video description of these flights for the 3 trajectories can be seen from https://vimeo.com/347564788.

4.4 *Discussions*

As it can be seen from Table 1, there are some errors in the obstacle pose estimation. However, the SR in Table 2

[1] www.aerostack.org

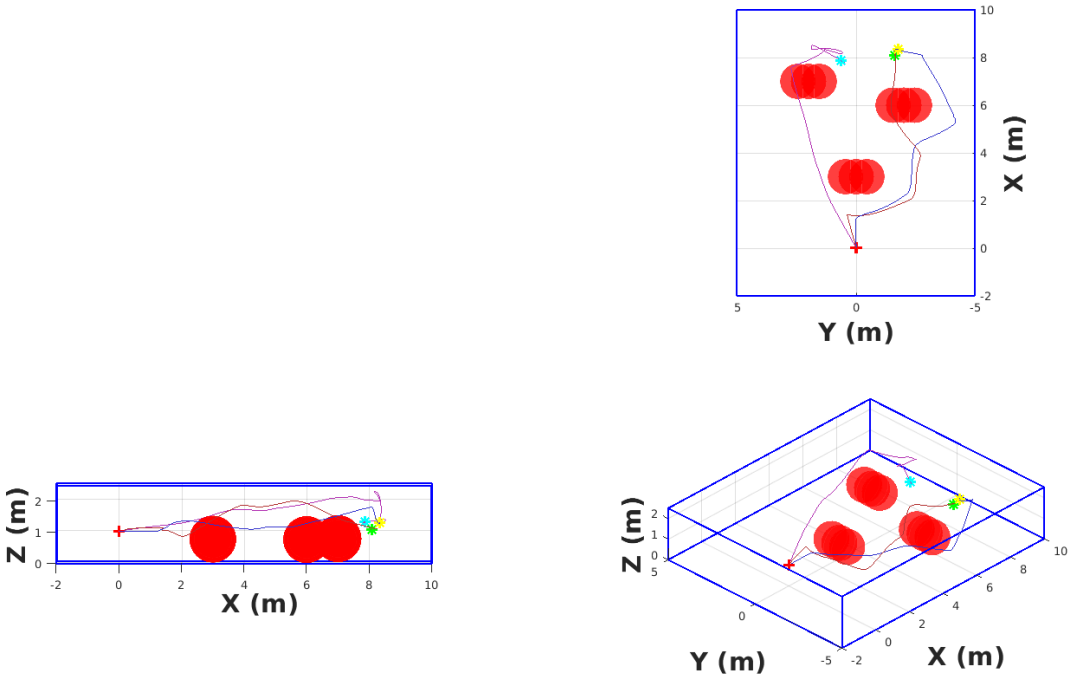

Figure 4: Three views of three flight trajectories.

shows that it is still valid for our proposed trajectory planner. The SR in Table 2 also shows that our navigation strategy can be used for navigation in an environment with dynamic spheric obstacles with a maximum velocity of $0.7012\,m/s$ and an average velocity of $0.3223\,m/s$.

In this paper, we build a Gazebo simulation environment and use Rotors Gazebo simulator to evaluate our algorithm, Rotors Gazebo model incorporates a good dynamic model for the aerial robot which can provide realistic flight behaviors and it is widely used in the research community [12, 13].

5 CONCLUSIONS AND FUTURE WORKS

In this paper, a method for autonomous UAV navigation in an environment with dynamic obstacles has been presented. An obstacle detector based on YOLOv3 and an iterative PnP algorithm are used to estimate the relative position of the obstacles. Then, an online RHC trajectory planner is used to plan a path and finally, the robot is controlled using an MPC controller in order to guide it to the goal waypoint. The experiment results show that this is a valid approach for UAV navigation in dynamic environments.

In the future, we will improve the performance of our detection and pose estimation algorithm to reduce the errors in obstacle pose estimation. An Extended Kalman filter or Unscented Kalman Filter will be also used to predict the future position of the obstacle and used in our trajectory planner to improve its performance. A real flight will also be implemented to evaluate the performance of the proposed algorithm.

ACKNOWLEDGEMENTS

The work reported in this paper is sponsored by the Chinese Scholarship Council (CSC).

REFERENCES

[1] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. de la Puente and P. Campoy, "Laser-Based Reactive Navigation for Multirotor Aerial Robots using Deep Reinforcement Learning," Intelligent Robots and Systems (IROS), *2018 IEEE/RSJ International Conference in.* IEEE, 2018, pp.1024-1031.

[2] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, A. Carrio, R. A. Suarez-Fernandez, J. L. Sanchez-Lopez and P. Campoy, "A fully-autonomous aerial robotic solution for the 2016 International Micro Air Vehicle competition," Unmanned Aircraft Systems (ICUAS), *2017 IEEE International Conference in.* IEEE, 2017, pp.989-998.

[3] A. Carrio, C. Sampedro, A. Rodriguez-Ramos and P. Campoy, "A review of deep learning methods and applications for unmanned aerial vehicles," Journal of Sensors, 2017: 1-13.

[4] J. Redmon, A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv preprint arXiv: 1804.02767, 2018.

[5] H. Bavle, J. L. Sanchez-Lopez, P. de la Puente, A. Rodriguez-Ramos, C. Sampedro and P. Campoy "Fast and Robust Flight Altitude Estimation of Multirotor UAVs in Dynamic Unstructured Environments Using 3D Point Cloud Sensors," aerospace, 2018, 5(3): 1-21.

[6] A. Rodriguez-Ramos, C. Sampedro, A. Carrio, H. Bavle, R. A. Suarez-Fernandez, Z. Milosevic and P. Pascual, "A Monocular Pose Estimation Strategy for UAV Autonomous Navigation in GNSS-denied Environments," Proceedings of the International Micro Air Vehicle Conference and Flight Competition, 2016, pp.17-22.

[7] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova and R. Siegwart, "Receding Horizon NextBestView Planner for 3D Exploration," on Robotics and Automation (ICRA), *2016 IEEE International Conference.* IEEE, 2016, pp.1462-1468.

[8] Z. Fang, S. Yang, S. Jian, G. Dubey, S. Roth, S. Maeta, S. Nuske, Y. Zhang and S. Scherr, "Robust Autonomous Flight in Constrained and Visually Degraded Shipboard Environments," Journal of Field Robotics, 2017, 34(1): 25-52.

[9] S. Karaman and E. Frazzli, "Sampling based algorithms for optimal motion planning," *International Journal of Robotics Research*, 30(7): 846-894, 2011.

[10] M. Zucker, N. Ratliff, A.D. Dragan, M. Pivtoraiko,M. Klingensmith, C.M. Dellin, J.A. Bagnell, and S.S.Srinivasa, " CHOMP: Covariant hamiltonian optimizationfor motion planning," *International Journal of Robotics Research*, 2012.

[11] M. Kamel, M. Burri and R. Siegwart, "Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles," *arXiv preprint arXiv: 1611.09240*, 2016.

[12] T. Cieslewski, E. Kaufmann and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," Intelligent Robots and Systems (IROS), *2017 IEEE/RSJ International Conference in.*, IEEE, 2017, pp. 2135-2142.

[13] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," Intelligent Robots and Systems (IROS), *2016 IEEE/RSJ International Conference in.*, IEEE, 2016, pp. 5332-5339.