Towards a consequences-aware emergency landing system for unmanned aerial systems

Xiangyu Wu and Mark Mueller

***Abstract*— We present an algorithm with which an aerial robot is capable of planning a consequences-aware flight path. The robot is capable of reasoning about possible faults, and which emergency actions are available to it. In this paper, we focus on faults that would force the vehicle to land in a short amount of time, and create a system that allows the vehicle to reason about its ability to execute a safe emergency landing from its current state. Such a system may improve the safety and reduce the economic cost of aerial robots, by allowing them to operate more flexibly whilst still achieving suitable safety. Using an onboard camera, the flying robot is able to identify safe landing spots, and attempt to compute trajectories from a future state to the available landing spots. If no such emergency trajectories are found, the robot must return to the last state at which a safe emergency trajectory was available. Initial in-lab experiments are shown, validating the feasibility of the concept.**

I. Introduction

Aerial robots have been shown to be useful in many applications such as search and rescue, surveillance, infrastructure inspection and package delivery. However, faults can occur during the flight, which would make the aerial robots become out of control and could cause serious damages to property and injury to people. As a result, aerial vehicles are often prohibited from (or severely constrained when) operating in populated areas. The ability of aerial robots to find emergency landing sites and perform emergency landing are crucial for them to be widely used. The emergency landing problem can be divided into three parts:

1) Detection of an aerial robot's fault and assessment of the effect of that fault.
2) Detection and selection of emergency landing sites.
3) Trajectory planning and trajectory tracking during an aerial robot's emergency landing.

The focus of this paper will be on items 2 and 3. There is substantial work on fault detection and isolation for multicopters, and a survey of algorithmic approaches is given in [1]. Examples of works that examine partial failures of actuators for multicopters are given in [2]–[4], and complete failures are investigated in [5]–[9]. Flight after an actuator failure is discussed in [10] and [11] for an octocopter and hexacopter, respectively. Of course, in addition to algorithmic solutions, a designer may use additional mechanical solutions, such as adding a parachute to a vehicle [12] – though parachutes are attractive, they add a substantial amount of mass, cutting the vehicle's flight time and payload.

The authors are with the Department of Mechanical Engineering, at the University of California, Berkeley.
{wuxiangyu, mwm}@berkeley.edu

Previous work on the detection and selection of emergency landing sites of aerial robots are briefly introduced as follows: In [13], [14] the authors use edge information for the detection of safe landing places, while in [15] and [16], the authors used machine learning methods to detect safe landing regions. In [17] the authors propose initial designs for an autonomous decision system for UAVs to select emergency landing sites which consists of two main components: preplanning and realtime optimization. And in [18] the authors present a system to detect safe landing spot base on elevation map.

During flight, faults such as low battery, broken propellers, malfunction of motors, etc. could occur. These faults would limit the dynamics and reduce the available flight time of the vehicle. In this preliminary work, we focus on faults that would force the vehicle to land in a short amount of time, and create a system that allows the vehicle to reason about its ability to execute a safe emergency landing from its current state. Examples of such faults include battery faults, or sensor failures after which the vehicle's state estimate can only be relied upon for a short period.

For the detection of emergency landing site, our algorithm is based on the texture of the ground. The main reason of this choice is the computation efficiency: we want the image processing algorithm to run completely on board while being real time. For the trajectory planning part, we use the method in [19]. This algorithm is computationally efficient in the generation of motion primitives and verification of their feasibility. It is used to check the feasibility of sampled

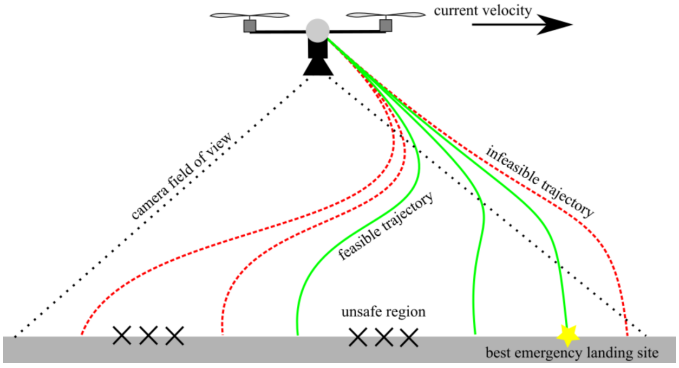

Fig. 1. During normal operations, the proposed system continuously uses an onboard, downward-facing camera to classify safe and unsafe candidate emergency landing positions (unsafe shown as checkered in the diagram), then tries to compute trajectories from its current state to the identified safe landing positions. Thus, the system is able to continuously guarantee a safe emergency procedure if an emergency situation were to occur mid-flight.

landing sites and generate the reference trajectory to the target emergency landing site. Thanks to the computational efficiency of the image processing and reference trajectory generation programs. Our algorithm is demonstrated experimentally, and is able to run on a single-board computer on a quadcopter. We hope our proposed system could enable aerial robots to operate with improved safety and greater flexibility. An illustration of the system is shown in Fig. 1.

II. System overview

This section describes the major components making up the system. In the system, a downward looking camera is used for taking pictures. The image is then passed to the image processing part for the finding of safe sites. In image processing, we use texture information to detect obstacles on the ground. A site whose distance to obstacles is above a threshold is considered safe. Next, the safe sites are tested for whether they can be reached by the vehicle from its current state, if an emergency were to occur. During normal flight, the vehicle follows the nominal trajectory and the system runs constantly in the background to check if a safe landing site could be found and generates a trajectory to that place. If no safe landing site could be found during the flight, the vehicle must return to the last state at which a safe emergency trajectory was available. When the vehicle has a fault, it will switch from nominal trajectory tracking to emergency landing trajectory tracking and land on the safe sites selected by the system. Please refer to Fig. 2. for the block diagram of the system.

A. Vision-based safe landing spot detection

In real world applications, it is important to run all safety-related algorithms onboard the vehicle, since connection with external computers cannot be guaranteed. Given that many aerial robots are small in size, they often have limited energy storage and power supply capacity, and can only carry a limited amount of payload. As a result, for safe landing spot detection, deep learning methods such as Convolutional Neural Network is not a good choice because a GPU is required for the image processing to be real-time and a GPU adds a considerable amount of payload and power consumption to the system. In the proposed system, we detect safe landing places based on textures of the ground and the Canny edge detector [20] is used. Thanks to the computational efficiency of this method, we are able to do the process of image processing, feasibility test, landing site selection and reference trajectory generation in real time at 5Hz with an Odroid-XU4 on board.

In this system we are using a single downward-looking camera for taking pictures, and we assume both that the vehicle is oriented approximately horizontally and that the ground is flat. Given these assumptions, we may infer the distance between two sites on the ground directly from their pixel distance in the picture:

$$x_{\text{pixel}} = f\frac{x_{\text{camera}}}{z_{\text{camera}}} \tag{1}$$

$$y_{\text{pixel}} = f\frac{y_{\text{camera}}}{z_{\text{camera}}} \tag{2}$$

with the f here being the focal length of the camera, x_{pixel} and y_{pixel} represent a point's coordinates in pixels and x_{camera}, y_{camera} and z_{camera} represent a point's 3D coordinates (expressed in the camera frame). By the assumption of the vehicle being horizontal, the horizontal distance between an obstacle and a candidate landing point is proportional to the pixel distance in the camera's image coordinates, with proportionality constant equal to the focal length of the camera f divided by the vehicle's current height.

After getting a raw picture from the camera, the first step is to convert the picture to a grey-scale picture. The second step is Gaussian blur, which is used to reduce the noises and details of the picture. After that, the canny edge detector is used for edge detection and the result is shown in Fig. 4. It is assumed that if there are edges, there are obstacles at that place and vice versa – this method is clearly not perfect, but is likely to err on the side of caution while still being able to run on low-cost, constrained hardware. The fourth step is the calculation of each pixels distance to the nearest edge, which represents a sites distance to the nearest obstacle.

Then, a user-defined number of points (N_{sample}) are sampled uniformly at random in the image coordinates. For each point, the distance (in pixel-space) is computed to the nearest obstacle. To test the safeness of each site, a safety threshold (in meter) is defined based on an assumed accuracy with which the vehicle can follow a trajectory under emergency conditions, i.e., a safety margin is added to the threshold to guarantee that the emergency landing is safe considering final state error. The corresponding threshold (in pixel), which is inversely proportional to the height of the vehicle, is then compared with each site's distance to the nearest obstacle: if the distance (in pixel) is larger the the threshold (in pixel), then the site is marked as safe.

B. Trajectory planning and feasibility test

For the trajectory generation, we used the method proposed in [19]. The reason for choosing this method is because of its computational efficiency: it is able to generate more than 1 million trajectories within a second. Here we present a very brief introduction to the trajectory generation method: this method calculates an optimal trajectory from an initial state to a final state in time T which minimizes the third derivative of the position (the jerk). In the context of this paper, the time T is a user-defined length of time, sufficiently short so that the vehicle could land in emergency conditions. The trajectory generator aims to minimize the cost function J_Σ, defined as below

$$J_\Sigma = \frac{1}{T}\int_0^T \|\boldsymbol{j}(t)\|^2\, dt. \tag{3}$$

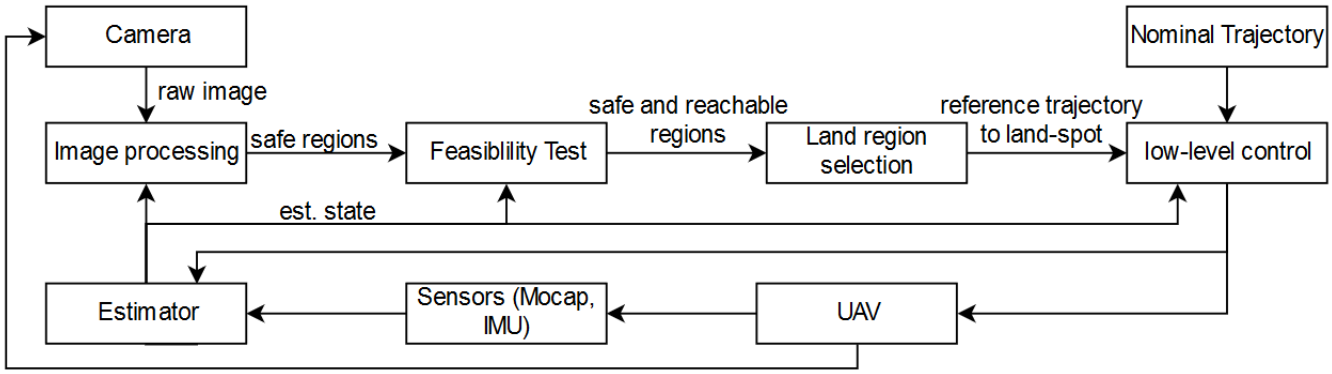

Fig. 2. A block diagram of the system, showing the relationship between various components.

Fig. 3. A representative (unprocessed) image, as may be taken from a UAV operating over a potentially sensitive site. The image is taken from `https://www.sensefly.com/`.

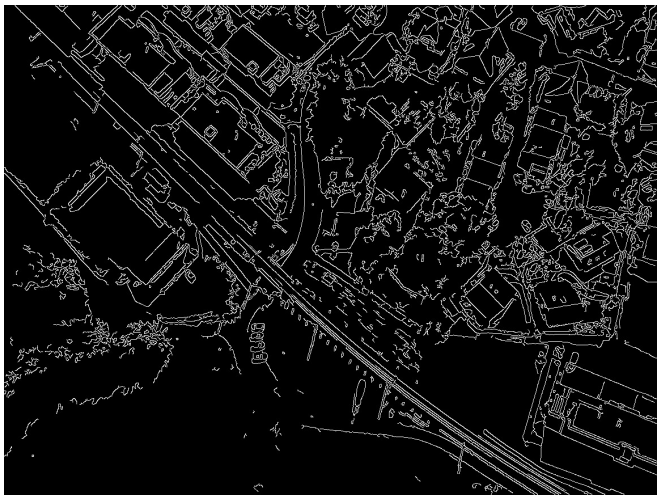

Fig. 4. The image of Figure 3 after edge detection.

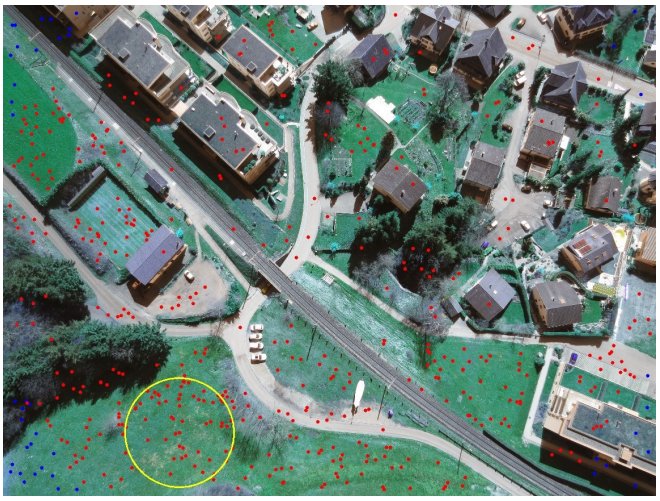

Fig. 5. The image of Figure 3 after all image processing. The points represent sampled candidate landing sites. If a site is marked red, that site is reachable for the vehicle under emergency. If a site is marked blue, that site is not reachable for the vehicle under emergency. The selected landing site is the center of the yellow circle, being that reachable site farthest away from an obstacle.

wherein $\boldsymbol{j}$ is the vehicle's instantaneous jerk along the trajectory and $\|\cdot\|$ denotes the Euclidean norm.

The cost function, J_Σ, is decoupled into a per-axis cost J_k by expanding the integrand in (3).

$$J_\Sigma = \sum_{k=1}^{3} J_k, \quad \text{where } J_k = \frac{1}{T}\int_0^T j_k(t)^2\,dt \tag{4}$$

For each axis k, the state $s_k = (p_k, v_k, a_k)$ is introduced, consisting of the scalars position, velocity, and acceleration. The optimal state trajectories can be solved with Pontryagins minimum principle (see [21]), and the solutions are polynomials in time.

This method also checks the feasibility of the trajectory: it checks if each candidate trajectory meets the dynamic constraints of the system and does not collide with obstacles. For the input feasibility, the algorithm constrains the thrust f produced by the vehicle lies in the range:

$$0 < f_{\min} \leq f \leq f_{\max} \tag{5}$$

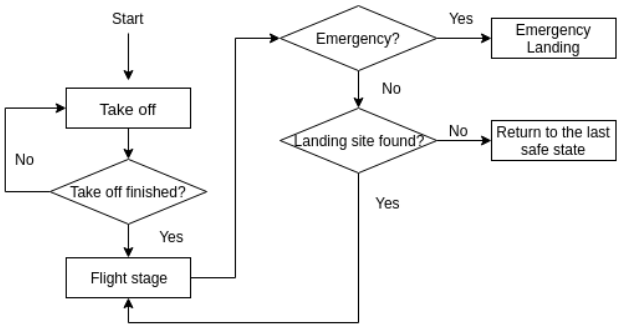

Fig. 6. A flowchart of the state machine of the system.

where $f_{\min}$ and $f_{\max}$ are the user-defined minimum and maximum normalized thrusts that the vehicle can achieve. The trajectory generator also allows for constraints on the angular velocity of the vehicle:

$$\|\omega\| \leq \omega_{\max} \tag{6}$$

where ω is the vehicle's angular velocity and $\omega_{\max}$ is the maximum allowed angular velocity. Furthermore, constraints need to make sure that the vehicle remains within a certain flight space. Planar constraints can be specified by specifying that inner product of the vehicle's position with the normal of the plane is grater than some constant value.

For the proposed system, this trajectory generation method is used to check the ability of the vehicle to reach candidate sites, and to generate a reference trajectory to the chosen target landing site. The safe sites from the previous image processing part are tested for reachability under emergency conditions. Finally, from the safe and reachable points, that site which is farthest from obstacles is chosen to be the current target landing site under emergency. See Fig. 5 for the feasible points selected from the example of Fig. 3. The pixel coordinates of the target landing site is transformed to world coordinates, using the intrinsic matrix of the camera, the position data of the vehicle and the assumption that the ground is flat. The program then generates a reference trajectory to the target landing site.

C. State Machine

The emergency system runs in the background, during normal flight operations, continuously seeking out safe and reachable landing points. When an emergency is detected, the system aborts the normal flight, and instead an emergency controller takes over. This emergency controller executes the currently valid emergency landing path, until the vehicle lands safely. If, in flight, no safe landing spots are detected, the vehicle must return to the previous position at which a safe position was detected – there, it may attempt to sample more points (until perhaps a safe emergency landing is detected and it may proceed), or it must await external instructions / execute an alternative. A flowchart is shown in Fig. 6.

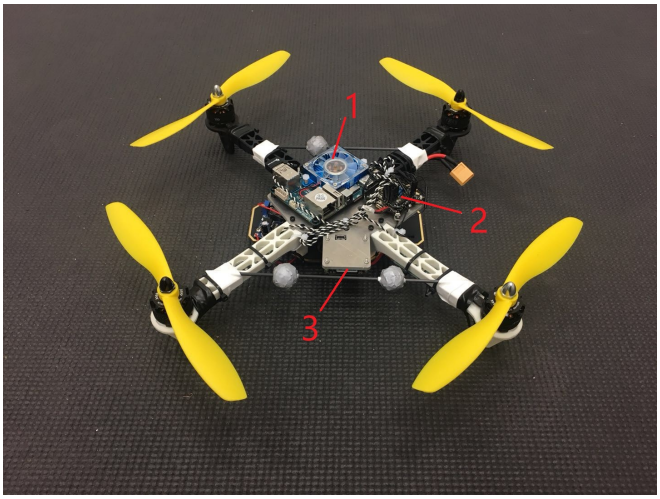

Fig. 7. The Quadcopter used in the experiment. (1) The onboard computer, (2) the embedded flight computer with IMU sensors, and (3) the downward looking camera

III. Experimental validation

To test the system, we used a custom built Quadcopter as shown in Fig. 7. The hub-to-hub distance of the vehicle is 330mm, the propeller diameter is 203mm. A downward looking mvBlueFOX-MLC200w camera is used for taking pictures with a resolution of 752×480 pixels. A Crazyflie 2.0 [22] running a modified version of the PX4 software stack [23] is used as flight control board which runs the low-level controller of the UAV, ultimately sending control signals to the vehicle's motor controllers. The image processing, landing region selection, feasibility test and estimation parts are run on a Odroid-XU4 computer board. The overall weight of the vehicle is 858g. The Robot Operating System (ROS) [24] is used as middleware.

The experiments were done in an indoor lab space of dimension $7 \times 6 \times 5$m. For the localization of our vehicle during the experiment, we used a commercial motion capture system, which is equipped with eight motion capture cameras for high-rate, high-fidelity state estimation. The data from the motion capture system is transmitted to a laptop for parsing and then published as a ROS message. An Extended Kalman Filter runs on the onboard computer, fusing motion capture information and control commands. Note, though, that the proposed algorithm does not, in principle, rely on having access to a motion capture system and would work if alternative methods of estimation / control were used, such as using GPS signals. The estimated state is then used for feasibility test, reference trajectory generation, and coordinate transformation between image and the world frame. During the experiment, we use a joystick to simulate an emergency event, which then triggers an emergency landing of the vehicle.

During the experiment, we put several boxes on the ground to emulate obstacles. The vehicle's trajectory in experiment is a horizontal sine wave, of amplitude 1m, at a constant height of 3m, flying over a variety of obstacles (boxes, in

the experiment). During the flight, the onboard system continuously processes images, identifying safe and reachable landing positions at an overall rate of 5Hz.

In the experiment, we simulate faults such as battery faults or sensor failures after which the vehicle's state estimate can be only relied on for a short period. The time allowed for the vehicle to land is set to $T = 2.5$s. The maximum normalized thrust of the vehicle is set to $f_{\max} = 15\text{m/s}^2$, and the minimum thrust is set to $f_{\min} = 5\text{m/s}^2$. At each time instant, the system sample $N_{\text{sample}} = 1000$ candidate points.

The experiment shows that our initial emergency landing system is able to detect and select safe landing spot, generate corresponding reference trajectory to that site and let the UAV follow the reference trajectory to land there. The emergency landing site finding and reference trajectory algorithm can run at 5Hz on Odroid XU4 and is proved to be computationally efficient.

IV. Conclusion and future work

In this paper we proposed a system which enables the unmanned aerial vehicles to plan a consequences-aware flight path. In this initial version of the system, we focus on faults that force the vehicle to land in a short amount of time. The system allows the vehicle to select its target emergency landing site based on its current state and the environment around it and land at the selected site. Finally, an experiment is done to verify the function of the system.

For future work, we will include additional failure modes to the system, for example, degraded dynamics and sensor failures. In addition, the image processing part will be improved to achieve higher-fidelity recognition of safe spots. In addition to the indoor experiment, we plan to validate the system through out-door flights.

V. Acknowledgement

This research was supported by a 2017 Seed Fund Award from CITRIS and the Banatao Institute at the University of California.

References

[1] Y. Zhang, A. Chamseddine, C. Rabbath, B. Gordon, C.-Y. Su, S. Rakheja, C. Fulford, J. Apkarian, and P. Gosselin, "Development of advanced FDD and FTC, techniques with application to an unmanned quadrotor helicopter testbed," *Journal of the Franklin Institute*, vol. 350, no. 9, pp. 2396–2422, 2013.

[2] M. Ranjbaran and K. Khorasani, "Fault recovery of an under-actuated quadrotor aerial vehicle," in *IEEE Annual Conference on Decision and Control (CDC)*, IEEE, 2010, pp. 4385–4392.

[3] H. A. Izadi, Y. Zhang, and B. W. Gordon, "Fault tolerant model predictive control of quad-rotor helicopters with actuator fault estimation," in *Proceedings of the 18th IFAC World Congress*, 2011, pp. 6343–6348.

[4] A. Chamseddine, Y. Zhang, C. A. Rabbath, C. Join, and D. Theilliol, "Flatness-based trajectory planning/replanning for a quadrotor unmanned aerial vehicle," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 4, pp. 2832–2848, 2012.

[5] A. Freddi, A. Lanzon, and S. Longhi, "A feedback linearization approach to fault tolerance in quadrotor vehicles," in *Proceedings of the 18th IFAC World Congress*, 2011, pp. 5413–5418.

[6] A. Akhtar, S. Waslander, and C. Nielsen, "Fault tolerant path following for a quadrotor," in *IEEE Annual Conference on Decision and Control (CDC)*, Dec. 2013, pp. 847–852.

[7] A. Lanzon, A. Freddi, and S. Longhi, "Flight control of a quadrotor vehicle subsequent to a rotor failure," *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 2, pp. 580–591, 2014.

[8] M. W. Mueller and R. D'Andrea, "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[9] M. W. Mueller and R. D'Andrea, "Relaxed hover solutions for multicopters: Application to algorithmic redundancy and novel vehicles," *The International Journal of Robotics Research*, vol. 35, no. 8, pp. 873–889, 2016.

[10] A. Marks, J. F. Whidborne, and I. Yamamoto, "Control allocation for fault tolerant control of a VTOL octorotor," in *UKACC International Conference on Control*, IEEE, 2012, pp. 357–362.

[11] M. C. Achtelik, K.-M. Doth, D. Gurdan, and J. Stumpf, "Design of a multi rotor MAV with regard to efficiency, dynamics and redundancy," in *AIAA Guidance, Navigation, and Control Conference*, 2012, pp. 1–17.

[12] Fruity chutes, "Drone parachute launcher, light weight, simple, and world class parachutes!" *(Accessed 26 Feb. 2018)* `https://fruitychutes.com/uav_rpv_drone_recovery_parachutes/drone_multicopter_quadcopter_recovery_parachutes.htm`, 2018.

[13] D. Fitzgerald, R. Walker, and D. Campbell, "A vision based forced landing site selection system for an autonomous uav," in *2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, Dec. 2005, pp. 397–402.

[14] Y. F. Shen, Z. U. Rahman, D. Krusienski, and J. Li, "A vision-based automatic safe landing-site detection system," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 294–311, Jan. 2013.

[15] X. Guo, S. Denman, C. Fookes, and S. Sridharan, "A robust uav landing site detection system using mid-level discriminative patches," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, Dec. 2016, pp. 1659–1664.

[16] X. Guo, S. Denman, C. Fookes, L. Mejias, and S. Sridharan, "Automatic uav forced landing site detection using machine learning," in *2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Nov. 2014, pp. 1–7.

[17] J. Ding, C. J. Tomlin, L. R. Hook, and J. Fuller, "Initial designs for an automatic forced landing system for safer inclusion of small unmanned air vehicles into the national airspace," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, Sep. 2016, pp. 1–12.

[18] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 111–118.

[19] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, Dec. 2015.

[20] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[21] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. I*. Athena Scientific, 2005.

[22] Bitcraze. (2018). Crazyflie 2.0, [Online]. Available: www.bitcraze.io/crazyflie-2 (visited on 02/25/2018).

[23] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, 2012.

[24] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.