University of Thessaly

Master Thesis

Precision landing for drones combining infrared and visible light sensors

Προσγείωση ακριβείας για μη επανδρωμένα αυτόνομα εναέρια οχήματα συνδυάζοντας υπέρυθρους και οπτικούς αισθητήρες

Author:
Giannis Badakis

Supervisors:
Spyros Lalis
Nikos Bellas
Christos Antonopoulos

A thesis submitted in fulfilment of the requirements for the degree of Master in the

Department of Electrical and Computer Engineering
University of Thessaly

Volos, October 23, 2020

ABSTRACT

One of the challenges in drone-based systems is to support automated landing without the intervention of a human operator. Most drones are already equipped with Global Position System (GPS) sensors, and therefore can navigate back to their home position. Nevertheless, this does not guarantee that they will land exactly at the desired position because standard commercial GPS receivers only have an accuracy of approximately $1 - 2$ meters.

Addressing this limitation, various efforts have been made to support precise landing on a given target, in the order of just few centimeters. More specifically, infrared-based mechanisms as well as RGB camera-based mechanisms have been developed for this purpose. In most cases, drones are equipped with only one of these mechanisms. There are also some cases where these mechanisms are used interchangeably, depending on weather conditions or the specific phase of the landing approach.

In this thesis, we develop an RGB camera-based sensor mechanism, which infers the position of the drone based on the detection of a special visual marker. Moreover, we combine the marker-based sensor with an infrared-based sensor concurrently using a fusion-based approach. This makes it possible to tolerate failures of any individual sensor subsystem that might occur due to occlusions or mere malfunctions. Furthermore, we strengthen the robustness of the precision landing operation by forcing the control logic to perform repeated landing attempts in case the landing target is lost during descent.

The implementation is done for the popular open-source ArduPilot software suite, through appropriate extensions in order to incorporate the marker-based sensor subsystem and allow its concurrent usage with the already supported infrared precision landing mechanism. We evaluate the developed mechanisms via simulation and by conducting a wide range of field experiments, using a flexible configuration which allows us to activate faults in the individual precision landing subsystem in a controlled way. Our results show that the proposed precision landing system increases robustness while maintaining satisfactory accuracy.

ΠΕΡΙΛΗΨΗ

Μία από τις προκλήσεις στα συστήματα που βασίζονται σε αυτόνομα εναέρια οχήματα (drones) είναι η υποστήριξη αυτόματης προσγείωσης χωρίς την παρέμβαση ενός ανθρώπου χειριστή. Τα περισσότερα drones είναι ήδη εξοπλισμένα με αισθητήρες Global Position System (GPS), κι επομένως έχουν τη δυνατότητα να μπορούν να επιστρέψουν στη θέση από την οποία απογειώθηκαν. Ωστόσο, αυτό δεν εγγυάται ότι θα προσγειωθούν ακριβώς στην επιθυμητή τους θέση, καθώς οι τυπικοί εμπορικοί δέκτες-αισθητήρες GPS έχουν ακρίβεια περίπου $1-2$ μέτρα.

Για να αντιμετωπιστεί αυτή η αδυναμία, έχουν καταβληθεί διάφορες προσπάθειες για την υποστήριξη προσγείωσης ακριβείας πάνω από έναν συγκεκριμένο στόχο, με ακρίβεια της τάξης μόλις λίγων εκατοστών. Πιο συγκεκριμένα, έχουν αναπτυχθεί μηχανισμοί που βασίζονται σε αισθητήρες υπέρυθρης ακτινοβολίας, καθώς και μηχανισμοί που βασίζονται σε RGB κάμερα. Στις περισσότερες περιπτώσεις, τα drones διαθέτουν μόνο έναν από αυτούς τους μηχανισμούς. Υπάρχουν, επίσης, ορισμένες περιπτώσεις όπου αυτοί οι μηχανισμοί χρησιμοποιούνται εναλλάξ, ανάλογα με τις καιρικές συνθήκες ή τη φάση της προσγείωσης του drone.

Σε αυτή τη διατριβή, αναπτύξαμε έναν μηχανισμό βασισμένο σε μια RGB κάμερα, ο οποίος εξάγει τη θέση του drone με βάση την ανίχνευση ενός ειδικού οπτικού σημαδιού. Επιπλέον, συνδυάσαμε τον συγκεκριμένο αισθητήρα με έναν επιπλέον υπέρυθρο αισθητήρα χρησιμοποιώντας μια προσέγγιση βασισμένη στη σύντηξη της πληροφορίας τους. Αυτό καθιστά δυνατή την ανοχή σφαλμάτων οποιουδήποτε μεμονωμένου αισθητήρα, που μπορεί να συμβεί λόγω αδυναμίας ανίχνευσης των επιμέρους στόχων προσγείωσης (οπτικό σημάδι ή πομπός υπέρυθρης ακτινοβολίας) ή απλών δυσλειτουργιών. Επιπλέον, ενισχύσαμε την αξιοπιστία της προσγείωσης ακριβείας, αναγκάζοντας τον αυτόματο πιλότο να εκτελεί επαναλαμβανόμενες προσπάθειες προσγείωσης σε περίπτωση απώλειας του στόχου προσγείωσης κατά την κατάβαση.

Η υλοποίηση έγινε για την πλατφόρμα λογισμικού ανοιχτού κώδικα ArduPilot, μέσω κατάλληλων επεκτάσεων προκειμένου να ενσωματωθεί το υποσύστημα που βασίζεται στην RGB κάμερα για την ανίχνευση οπτικών σημαδιών και να επιτραπεί η ταυτόχρονη χρήση του με τον ήδη υποστηριζόμενο μηχανισμό προσγείωσης ακρίβειας με χρήση υπέρυθρου αισθητήρα. Οι μηχανισμοί που αναπτύχθηκαν αξιολογήθηκαν μέσω προσομοίωσης καθώς και πραγματοποιώντας ένα ευρύ φάσμα δοκιμών στο πεδίο, έχοντας τη δυνατότητα να ενεργοποιούμε, μέσω κατάλληλου λογισμικού, σφάλματα με πλήρως ελεγχόμενο τρόπο σε οποιοδήποτε μεμονωμένο υποσύστημα προσγείωσης ακριβείας. Τα αποτελέσματά δείχνουν ότι το προτεινόμενο σύστημα προσγείωσης ακριβείας πετυχαίνει την επιθυμητή ανοχή σε μεμονωμένες βλάβες διατηρώντας ταυτόχρονα ικανοποιητική ακρίβεια.

ACKNOWLEDGEMENTS

First and foremost I would like to express my deep gratitude to Professor Spyros Lalis, my thesis supervisor, for his guidance, enthusiastic encouragement and useful critiques of this work. During my studies he inspired me to be interested in various fascinating fields of computer science and aerial robotics.

I also would like to thank my co-supervisors Professors Nikos Bellas and Christos Antonopoulos for their help and valuable advices throughout my undergraduate and postgraduate years.

I would like to express my very great appreciation to Manos Koutsoubelias who contributed to the completion of this work. If it were not for him, I would have not been able to perform the evaluation experiments presented in this thesis. Also, my special thanks are extended to Nasos Grigoropoulos for being an exceptional and very helpful colleague. His great advice and support were essential for my postgraduate studies.

Last but not least, my deepest gratitude goes to my family who have always supported me to accomplish my dreams.

Part of this work was co–financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE, project PV-Auto-Scout, grant agreement number T1EDK-02435.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Multi-rotor Unmanned Aerial Vehicles (UAVs), frequently called drones, are rapidly growing in popularity. Besides casual usage by individuals, drones play an increasingly important role in several application domains, such as cinematography and advertising [1], coverage of social and sport events [2], courier and delivery services [3, 4], agriculture [5], structural health monitoring [6], search and rescue operations [7, 8] and surveillance [9].

What drives this widespread use of drones is the fact that they can pilot themselves in a highly autonomous manner, thanks to an embedded autopilot subsystem, which continuously gathers data from various on-board sensors, processes them and takes all the steering decisions so that the drone performs the desired movement. This, in turn, makes it possible even for inexperienced persons to "fly" a drone, simply by issuing high-level commands, like *take-off*, *move upwards*, *goto waypoint*, *hover* and *return to home*. All that is needed is a device that sends these commands to the drone's autopilot over the air. This device can be practically anything. It can be a dedicated remote control console, or an ordinary smartphone/tablet running a suitable user interface that translates gestures, clicks and taps into navigation commands.

In the same spirit, a drone can also be flown by a *computer program*. Like the average drone user, the computer program does not need to know how to actually fly the drone; all it needs to do is to issue navigation commands in a meaningful sequence, based on the current attitude and position of the drone. This opens the way to a new class of drone-based applications, where entire missions are automated using suitably designed mission programs, with little or no human involvement. Indeed, several efforts are underway to provide suitable programming abstractions that simplify the development of computer-based missions [10–12]. While it is not possible to automate every imaginable mission, this is quite feasible in cases where the drone routinely needs to perform the same tasks over an area that is well-known, as this is typically the case in several agriculture and monitoring/surveillance scenarios.

1.1 THE NEED FOR PRECISION LANDING

Simplifying the development and execution of mission programs is necessary but not sufficient to achieve the vision of full automation.

Namely, in order to close the loop of automated operation, the drone must be able to take-off and land autonomously.

In most cases, take-off is relatively simple. The drone simply needs to perform a vertical upwards movement to a pre-specified altitude, from where it can start the mission. Landing, however, can be more tricky. Even though the drone knows its home position (typically the location from where it took-off) and can autonomously navigate back to it based on its on-board GPS receiver, this does not guarantee that it will land exactly at the desired position since the standard GPS accuracy is typically within a 1-2 meters [13].

Such inaccuracy is not acceptable if the objective is to support fully automated operation, without any human involvement. For instance, the drone may need to land on a plate through which it can re-charge its batteries [14] so that it can resume operation. This may need to be repeated a number of times in order to complete the mission. Furthermore, when the mission is completed, the drone may have to land into a hangar [15] that will protect it from rain or theft during periods of inactivity (of course, the charging plate can be part of the hangar infrastructure). In these cases, the drone must be able to land with very high accuracy, even in the order of a just few tens of centimeters.

A popular way to achieve high-precision landing is to use infra-red beacons [16]. More specifically, one or more infrared transmitters are placed in the landing zone at predefined locations. In turn, the vehicle is equipped with an infrared sensor that detects the emitted signals and reports the horizontal offset of the drone from the beacon. This information is combined with the ground distance, typically obtained via a barometer or range-finder, to guide the drone so that it smoothly lands on top of the beacon. This mechanism works quite robustly in a wide range of environmental conditions, in particular when there is poor or no ambient light (at night). The disadvantage is that the drone needs to carry extra sensor equipment just for this purpose. Also, the infrared transmitters in the landing zone have to be powered in a reliable way.

Another approach is to use an RGB camera in conjunction with image processing algorithms to detect a special landmark/object in the landing zone. Based on the offset of drone with respect to the landmark, it is possible to guide the drone so that it lands at the desired position in the landing zone. The advantage compared to the previous method is that most drones already carry such a camera as part of the application payload, and there is no need to equip the landing zone with active beacons. The most important disadvantage is that landmark detection is not robust in situations of low visibility.

1.2 AIM AND CONTRIBUTION OF THIS THESIS

The aim of the thesis is to increase the robustness of precision landing capability in order to be able to tolerate failures of individual sensor subsystems, which may occur due to the limitations or mere malfunctions of the respective sensors and corresponding subsystems. This work is done for the open-source ArduPilot software suite [17], which is very popular in the community and is widely used in real drones.

To this end, the main contributions are as follows:

- **Marker-based precision landing.** A new precision landing sensor is developed, using an on-board RGB camera in order to detect a special visual marker that is placed on the landing zone.
- **Fused precision landing.** The marker-based precision landing sensor is combined with the available infrared precision landing sensor (IRLock), using a fusion-based approach. The fused sensor can tolerate a single independent failure of any one of the two sensor subsystems hence works robustly as long as at least one of these sensors remains operational and delivers positioning information that can be used to guide a more accurate landing of the vehicle.
- **Refined precision landing approach.** To further increase the robustness of precision landing, the default precision landing operation of ArduPilot is extended so as to be able to tolerate (transient) target detection failures. More specifically, the control logic is modified so that the landing approach can be repeated a number of times in case the landing target is lost during descent.
- **Extensive evaluation.** The new precision landing capabilities are evaluated in an extensive way, using both simulations and tests with a real drone, for a range of indicative scenarios. To allow for flexible experimentation, we have added fault-injection support, which makes it possible to activate/set the type of faults to be applied to specific precision landing subsystems in a controlled way. Our results show that the marker-based sensor can achieve comparable accuracy to the infrared-based sensor. They also verify the increased robustness of the fused sensor and the more refined precision landing approach.

1.3 THESIS STRUCTURE

The rest of this thesis is structured as follows. Chapter 2 gives an overview of indicative related work. Chapter 3 briefly describes the structure of ArduPilot [17] software framework and the IRLock sensor subsystem that is currently used to support precision landing. Chapter 4 gives an overview of the technology behind the detection of special visual markers, and discusses the implementation of an independent sensor subsystem that can be used to support precision landing based on this technology. Chapter 5 describes the fusion-based approach, which combines the IRLock sensor subsystem and the marker sensor subsystem to provide a more robust sensor for precision landing, in a transparent way for the core autopilot framework. Chapter 6 presents the extension made to the ArduPilot control logic in order to support a more refined precision landing approach. Chapter 7 presents support that was added in order to enable the controlled injection of artificial faults into the individual sensor subsystems used for precision landing. Chapter 8 describes how the newly added functionality was tested and evaluated, using simulations as well as experiments in the real world. Finally, Chapter 9 concludes the thesis and points to directions for future work.

2

RELATED WORK

The problem of precision landing for vehicles with vertical landing capability has attracted a lot of attention especially in the last years with the rapid adoption of multi-rotor drones. In the following, we provide an indicate overview and identify the differences with our work.

A large body of work has investigated methods that detect pre-defined shapes using a downward-looking camera at the bottom of the UAV. In [18], the autonomous landing of an UAV on a ship is achieved using a visual-based line segment processing technique to detect a standard H mark on the landing deck. The same mark is used in [19] and [20] with an additional circle around it to enable the detection of the landing area from larger distances/heights. [21] proposes a shape consisting of two rings and two upside-down triangles that can be detected from different altitudes, to provide a full 3D pose estimation, roll, pitch and yaw. In [22], two disks of different color adjacent to the landing position are used in combination with a range finder to achieve pose estimation, while [23] explores beacon placement along two nested squares. A custom stripped pattern is tracked in [24] and [25] using an optical flow sensor together with an IMU and an altimeter, to support precision landing on stationary and moving landing pads, respectively.

Another popular approach is to use special visual markers. In [26], the on-board camera is used to determine the 3D position and orientation between the UAV and an apriltag, guiding the drone accordingly. A marker is also used in [27] to land a drone on a moving platform, using a control process that involves three modes for the initial search of the landing platform, the tracking of the platform and the final landing approach, respectively. [28] proposes a custom aruco marker, which consists of a small marker nested inside a bigger one, and validates the approach through real flight tests for a landing pad on a moving vehicle. [29] employs two apriltags placed at a known distance from each other to detect the landing target from different altitudes. A somewhat different design is presented in [30], featuring multiple aruco markers in different sizes surrounded by a circle with another nested marker at the center, leading to improved detection from both high altitudes and very low heights.

Another technology that has been successfully used for precision landing are infrared (IR) LEDs with corresponding receptors / camera filters on the drone. The work in [31] tracks four IR beacons arranged in a T pattern to extract the 3D position and yaw of the UAV. A similar approach is followed in [32] and [33], using four beacons with a different placement. [14] and [16] employ the MarkOne Beacon System (an IR beacon at the landing position and the IR-LOCK sensor on the drone) together with a rangefinder in order to detect the vertical distance to the landing position.

There is also work that combines different technologies. For instance, [34] combines a visual H landing mark with 4 IR LEDs at its corners. The latter are used to detect the landing position from large distances using the front-view camera of the UAV, while the former is used in the final phase of the landing process once the image appears in the field-of-view of another camera at the bottom of the vehicle. Another combination of visual and IR-based detection is presented in [35], using a landing pad with an aruco marker with a translucent white acrylic surface backlit by an array of 264 IR LEDs, making the marker visible when using a standard RGB camera (without an IR filter) even at poor light conditions and at night.

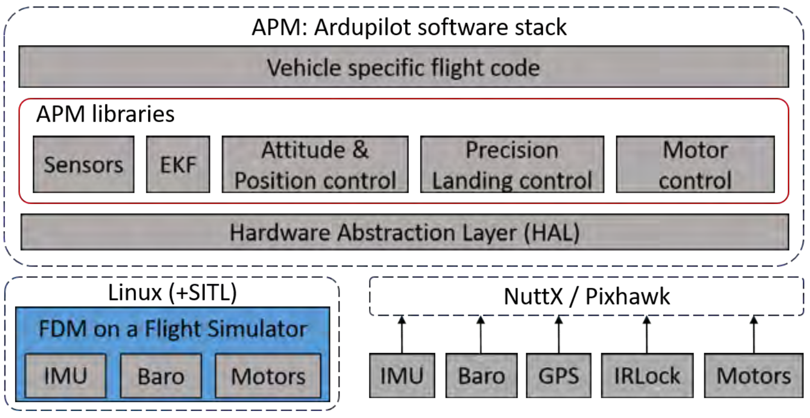
The vast majority of the approaches support precision landing of UAVs rely on a single type of sensor, typically a camera for detecting a visual shape or marker, or IR receptors / camera with an IR filter to detect IR-beacons. Also, work that combines different sensors typically uses them interchangeably, depending on weather conditions or the phase of the landing process. In our work, we use different sensor subsystems concurrently and combine their information using a fusion-based approach, in order to tolerate failures/inaccuracies of any single subsystem, which may occur due to the limitations or mere malfunctions of the respective sensors.

3

THE ARDUPILOT FRAMEWORK

This thesis makes use of the well-known open-source ArduPilot [17] framework (APM), an autopilot system that is used in multi-copters, helicopters, rovers and other autonomous vehicles. ArduPilot supports an autonomous precision landing flight mode based on the IRLock sensor subsystem, which feeds the autopilot's horizontal position controller with body frame vectors in direction of an IR beacon. In the following, we briefly present the structure of the ArduPilot software stack and the support for precision landing based on the IRLock sensor.

3.1 STRUCTURE



APM: Ardupilot software stack
Vehicle specific flight code
APM libraries
Sensors
EKF
Attitude & Position control
Precision Landing control
Motor control
Hardware Abstraction Layer (HAL)
Linux (+SITL)
FDM on a Flight Simulator
IMU
Baro
Motors
NuttX / Pixhawk
IMU
Baro
GPS
IRLock
Motors

Figure 1: ArduPilot autopilot structure.

The architecture of the ArduPilot software suite is shown in Figure 1. The core libraries include attitude and position estimation provided by an Extended Kalman Filter (EKF), precision landing control, motor control and more. In addition, there are libraries that implement a variety of drivers, for the Inertial Measurement Unit (IMU), the barometer, the GPS, the IRLock subsystem and the motors of the vehicle. For example, the IMU sensor library reads gyro and accelerometer data, performs calibration and provides data to the main-vehicle code and other libraries. These libraries are used to support the autonomous operation of different vehicle types, such as Copter (multi-copter aerial vehicles), Plane (winged aerial vehicles) and Rover (ground vehicles).

Thanks to its hardware abstraction layer (HAL), ArduPilot is portable to a large number of different platforms. More specifically, the HAL library defines the generic interface for the rest of the framework, which is implemented for concrete platforms and board types using a corresponding sub-library. For example, the HAL-Linux library is used to support Linux based boards as well as software-in-the-loop (SITL) operation, while the Pixhawk library is used to support the NuttX real-time operating system on Pixhawk boards.

3.2 SENSOR ACCESS

The ArduPilot framework supports access to a wide range of sensors. Each type of sensor is integrated into the rest of the software framework using a generic pattern, which is illustrated in Figure 2. Each sensor is accessed through a so-called front-end driver, which can be associated to one or more so-called back-end drivers. The number of back-ends depends on the number of individual physical sensors that are available on the vehicle. These are determined at initialization time, through automated discovery/detection mechanisms (e.g., polling a bus or serial connection using a suitable protocol) or based on pre-specified configuration information.

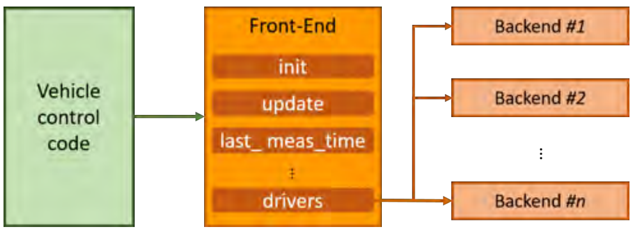


Figure 2: ArduPilot sensor driver architecture.

The main vehicle control code acquires sensor data by invoking the front-end, through an interface that consists of several function calls. In turn, the front-end is responsible for invoking the back-end(s) in order to retrieve the data that was produced by the respective physical sensor(s). Note that each sensor type may provide measurements at its own rate, which may also be (very) different from the frequency with which the front-end is being invoked from the main control code. For this reason, the front-end includes a method for checking the availability/freshness of sensor data. This method is typically invoked before invoking the method that returns the (most recent) sensor data.

3.3 PRECISION LANDING

The main control flow of ArduPilot when precision landing is enabled is illustrated in Figure 3, showing how the different sensors are used at different stages of the processing. More specifically, the IMU, GPS and barometer sensors are used to calculate the vehicle's current position/-pose (using an Extended Kalman Filter). The IRLock, barometer and range-finder sensors are used to calculate the vehicle's position relative to the landing target, which, in turn, is used to drive the precision landing of the vehicle. This control loop runs periodically, typically at a frequency of 400 Hz.

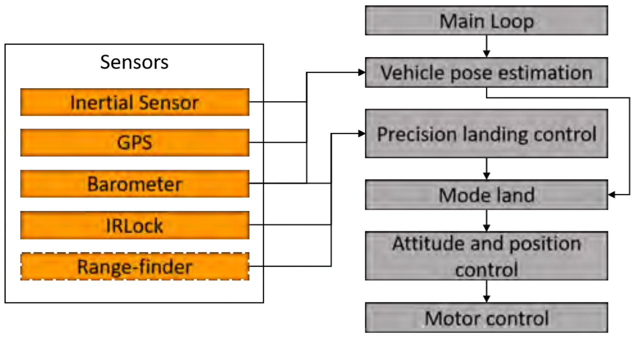


Figure 3: APM control flow with support for precision landing.

ArduPilot supports a variety of landing modes. Besides some special modes that can be used only in SITL operation, the landing modes that can be actually used during real operation are:

- **None.** As the name suggests, no sensor is used to guide the vehicle in order to perform a more precise landing. In this case, the ground target position has the same horizontal coordinates as the vehicle at the time when the autopilot was instructed to commence the landing procedure.
- **IRLock.** This mode employs the IRLock sensor subsystem (discussed in more detail below). IRLock provides to the main control logic the horizontal offset of the drone with respect to an IR beacon. The ground distance of the drone has to be provided by an additional sensor, which can be the barometer or a more accurate ranger-finder (such as a sonar or a LiDAR).
- **Companion.** The vehicle's displacement with respect to the landing position is provided by an external so-called companion system. This can be an on-board SoC device or a remote station. In this case, the positioning information typically includes both horizontal and vertical displacement information and is communicated to the autopilot system through the Micro Air Vehicle link (MAVLink) protocol using a special message. These messages are decoded and made accessible to the main control code of the autopilot through a corresponding front-end, in the same spirit as this is done for other on-board sensors.

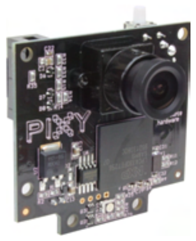
Note that IRLock only provides horizontal displacement information with respect to the landing target. The same may also hold for an external system that is used in the Companion mode. In these cases, the ground distance of the vehicle has to be provided using an additional sensor. This can be the vehicle's barometer or a more accurate range-finder sensor (such as a sonar or LiDAR).

In each iteration of the control loop, the autopilot retrieves the vehicle's displacement vector with respect to the landing target (the latter is obtained depending on the land mode, as discussed above). Based on this vector, the autopilot control logic issues commands to the motors/actuators of the vehicle in order to adjust the vehicle's horizontal and vertical position as needed to (gradually) approach the desired landing position.

3.4 THE IRLOCK MECHANISM

Infra-red technology is widely used in order to support precision landing in UAVs. The standard approach is to place infra-red transmitters on the landing zone, and a receiver on the vehicle. The transmitter emits infra-red beacons while the sensor detects these signals and guides the vehicle accordingly.

One of the most popular solutions is the IRLock target tracking mechanism [36], which is already supported in the ArduPilot autopilot framework (as discussed above). IRLock is based on the Pixy vision sensor [37] with a custom IRLock filter and a 3.6 mm lens (Figure 4a). It runs the IRLock specialized firmware, which reports the position of the LEDs at 50 Hz.


(a) Pixy vision sensor with IRLock filter and specialized firmware.

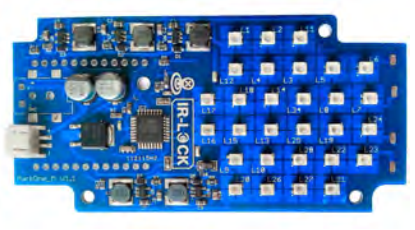

(b) The MarkOne beacon system.

Figure 4: IRLock target tracking mechanism [38].

The IRLock filter used by the Pixy firmware blocks the visible light while allowing specific wavelengths of infra-red light to get through. The tracking range depends upon the infra-red receiver and transmitter. More specifically, to increase the detecting range, one may cluster multiple LEDs to increase the size of the infra-red point, employ a lens with a narrower field of view (FoV) in order to increase the tracking range, and/or use high-power LEDs.

Given that sunlight also emits infra-red light, the Pixy sensor may falsely detect shiny objects or may be confused by surfaces (such as water) that reflect sunlight. To avoid such false detections, the IRLock target tracking mechanism is typically used in combination with the MarkOne Beacon system (Figure 4b), which provides robust and reliable target detection in practically all lighting conditions and operation environments. Notably, when using the MarkOne beacon, the detection range can be over 15 meters.

4

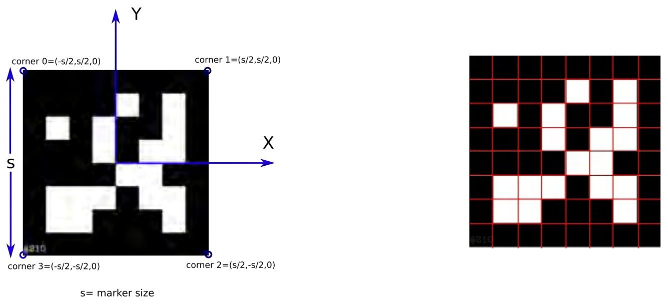
MARKER SENSOR SUBSYSTEM

This chapter presents the implementation of the marker sensor mechanism. We start by briefly presenting the basic technology used to detect visual/fractal markers. The material is based on the following sources: [39, 40]. Then, we discuss how we have used this technology to implement the corresponding sensor subsystem and integrate it into the ArduPilot framework.

4.1 MARKER DETECTION

Special visual markers are used in several scientific and commercial applications. In particular, markers are often used in robotics as special landmarks, which can be easily identified by a robot in order to properly navigate itself within the environment.

Markers typically have an external black border and an inner region that encodes a binary pattern (Figure 5b). The binary pattern is unique and identifies each marker. Depending on the dictionary (which is a set of visual markers with equal bits), there are markers with more or less bits. The more bits, the more words in the dictionary, and the smaller chance of miss-detection. However, a larger number of bits and more complex internal structure also requires a higher resolution to correctly detect the marker.



(a) Coordinate system: X-right, Y-up and Z-forward.

(b) Border and inner part.

Figure 5: Visual marker composition and reference system [41].

In our work, marker detection is achieved using the ArUco library [39], which, in turn, relies on OpenCV [42]. The ArUco library detects rectangle markers and extract their binary code through a stepwise process, briefly described below.

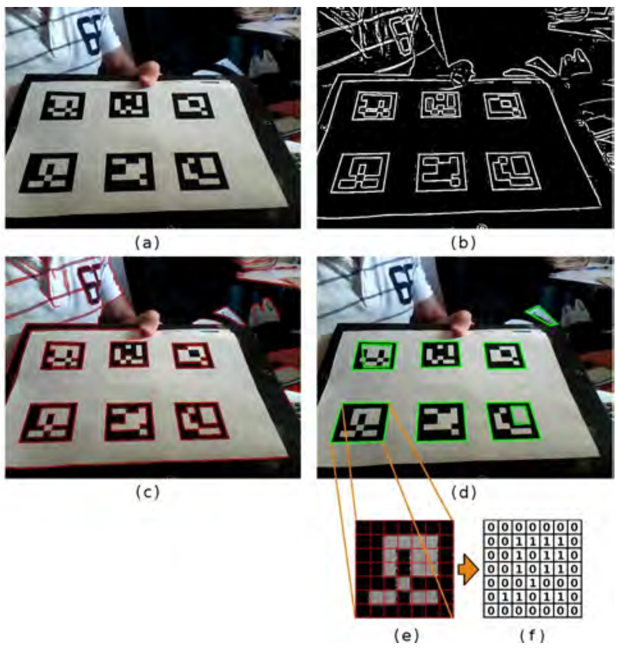


Figure 6: Steps of the marker detection process [39, 41].

- **Image Segmentation.** Image segmentation is typically used to locate objects and boundaries, such as lines and curves, in images. Since a marker is designed to have an external black border surrounded by a white space, its borders can be found by segmentation (Figure 6b). The library achieves this using adaptive thresholding.

- **Contour Extraction and Filtering.** Subsequently, a contour extraction and filtering step is applied to identify the boundaries of the marker(s) in the image. Firstly, using the Suzuki and Abe algorithm [43], a set of contours is obtained from the thresholded image. However, it is possible to have irrelevant background elements in the output image (Figure 6c), so an additional filtering step is applied in order to remove them. For this purpose, a polygonal approximation is performed using the Douglas-Peucker algorithm [44]. Since markers are squares, the detection process only focuses on contours that can be approximated as four-corner polygons, discarding any other elements. Finally, contours that are too small are discarded, leaving only the external ones. The resulting polygons identified via this process are the contours of the marker(s) (Figure 6d).

- **Code Extraction.** In order to determine which of the remaining contours are valid markers, the perspective projection is removed to obtain a frontal view of each of the rectangles, using the homography technique. Then, a thresholding of the image is performed using the Otsu method [45], to provide the optimal image threshold value (Figure 6e). The resulting thresholded image is divided into a regular grid, and each element is assigned the value 0 or 1 depending on the values of the majority of the surrounding pixels, producing a binarized image (Figure 6f).

- **Matching.** For each marker candidate in the image, the library determines whether it belongs to the set of predefined valid markers or if it is a uninterested background element. For example, if we divide the binarized image into a $6x6$ grid, the $5x5$ cells contains the identifier information, while the remaining ones correspond to the external black border. For this purpose, a check is made to confirm the presence of the external black border, and then the internal cells are read to see if they provide a valid code. There are four possible identifiers that can be obtained for each candidate, which correspond to the four possible rotations of the canonical image. If any of the these identifiers belong to the set of valid markers, then the markers are accepted and the detection is completed.
- **Corner Refinement.** The last step of the ArUco process consists in estimating the location of the corners with subpixel accuracy, in case some corners lie "between" pixels. The technique that is used computes all the intersections of the lines of the marker sides that employing all the contour pixels.

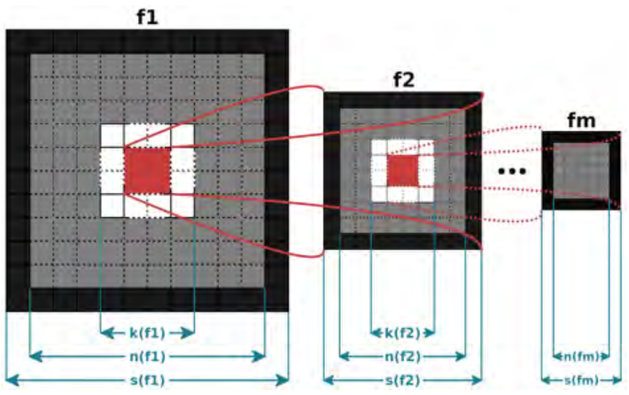


Figure 7: Configuration of a fractal marker [46].

The ArUco library also supports the detection of so-called fractal markers [46], built by embedding smaller markers within larger ones in a recursive manner (Figure 7). This makes it possible to tolerate errors due to partial or total occlusions of individual markers, as well as to support detection of a wider range of distances.

4.2 MARKER SENSOR SUBSYSTEM

The Marker sensor subsystem is integrated into the ArduPilot framework following the usual front-end driver and back-end scheme (see Section 3.2). Figure 8 illustrates the implementation approach.

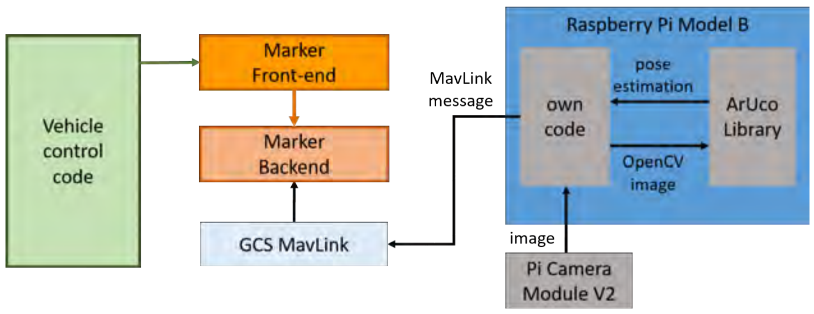


Figure 8: Marker sensor subsystem design.

The Marker sensor subsystem uses an on-board System-on-Chip (SoC), based on Raspberry Pi model B with a low-cost camera module v2 configured at a resolution of $640x480$ (480p) with a rate of 30 fps. The Pi runs the ArUco library [39] in order to detect the marker in the frames that are captured by the camera. Each time the marker is detected in a new frame, the estimated camera position relative to the center of the marker are calculated and are sent to the rest of the system. The frame processing, marker detection and offset calculation takes on average about 50 milliseconds.

The marker detection SoC sends the position estimation information to the back-end of the marker sensor subsystem over a serial connection via the MAVLink protocol (for this purpose we introduce a new MAVLink message). Each time the back-end receives such a message, it calculates the vehicle's relative position and distance to the center of the marker and updates the timestamp of the most recently received measurement. Finally, this information is retrieved from the corresponding front-end, each time it is invoked from the main control logic of the autopilot. The front-end interface of the marker sensor subsystem is practically identical to that of IRLock. One important difference is that the marker sensor subsystem also offers a method for retrieving the distance of the vehicle to the center of the marker. As a consequence, when using the marker sensor there is no need to have an extra sensor that provides altitude information (e.g., a Range-finder). Furthermore, one can use the altitude information obtained via the marker sensor subsystem in combination with the horizontal position information provided by the IRLock sensor subsystem.

5 FUSED SENSOR SUBSYSTEM

This chapter presents the implementation of the fused precision landing sensor mechanism. The objective of this mechanism is to increase the robustness of the autopilot's precision landing operation, by combining the IRLock and marker sensor subsystems in a way that is transparent for the rest of the autopilot software.

5.1 SOFTWARE DESIGN

The fused sensor mechanism is integrated into the ArduPilot framework following the usual approach, via a corresponding front-end library that hides all the back-end details. In this particular case, the front-end does not have its own back-end but instead connects to the back-ends of the IRLock and marker sensors. Figure 9 illustrates the approach.

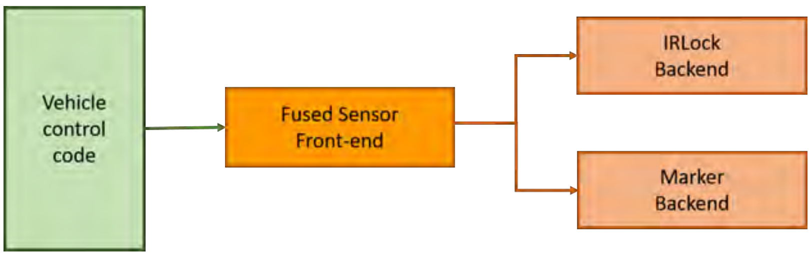


Figure 9: Approach for supporting a fused precision landing sensor.

The main control logic of the autopilot uses the fused sensor front-end in the same way as it would use any other precision landing sensor front-end (such as the IRLock and marker sensors). In turn, the front-end invokes the IRLock and marker back-ends to obtain the corresponding position information, which is then fused to provide the information towards the main control logic of the vehicle.

The fusion sensor front-end component is created and initialized by the precision landing controller of the ArduPilot framework on start-up. In turn, the front-end initializes the IRLock and marker sensor back-ends. Throughout the landing operation, the ArduPilot code periodically polls the fusion sensor front-end to check whether new positioning information is available, and if so to retrieve it. Each such invocation is internally converted to invocations towards the IRLock and marker back-ends in order to retrieve the position information that is produced by each sensor and to combine this information into a fused value. The fusion logic is discussed in more detail in the sequel.

5.2 FUSION LOGIC

As already mentioned, the autopilot periodically invokes the front-end during the precision landing procedure. In turn, the front-end utilizes the information it retrieves from the IRLock and marker back-ends in order to produce the position information that will be returned to the autopilot. Each time a new value/measurement is retrieved from the IRLock and/or marker sensor back-end, the front-end updates the value to be returned and stores it internally together with a timestamp. This timestamps is used to determine (in subsequent iterations) the freshness of the stored value.

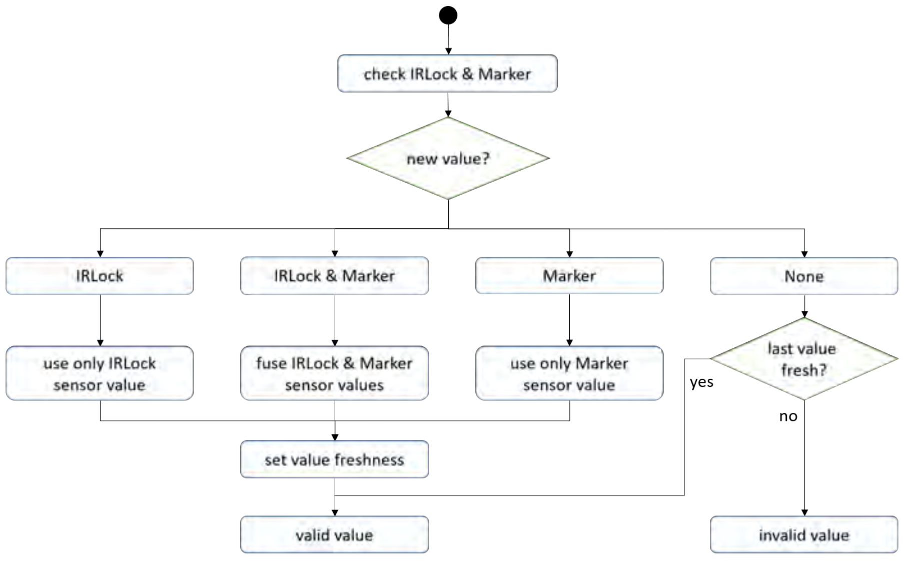


Figure 10: Flow diagram of the fusion logic.

The internal operation of the fused sensor is illustrated in Figure 10. More concretely, each time the front-end attempts to retrieve new values from the IRLock and marker sensor back-ends, there are four possible cases: (i) both the IRLock sensor and the marker sensor produce a new value; (ii) only the IRLock sensor produces a new value; (iii) only the marker sensor produces a new value; (iv) neither the IRLock sensor nor the marker sensor produce a new value.

If both sensors produce a new value, the two values are fused in order to produce the value that will be returned to the autopilot. We implement a simple fusion approach where the final value is calculated as the weighted average of the two individual sensor values. For more flexibility, the weights are configurable (via a corresponding MAVLink message/command). Note that this only applies to the horizontal position information as the IRLock sensor does not report any ground distance information.

If only one of the sensors produces a new value, the front-end keeps and returns this value to the autopilot. In this case, no attempt is made to fuse the newly acquired value with an older value coming from the other sensor, as this would merely distort the freshly acquired sensor value.

Finally, if no sensor produces a new value, the front-end returns the most recently stored value, provided this is considered to be sufficiently fresh – in our current prototype, the freshness threshold is set to 1 second, as this is the case in the IRLock and marker sensor subsystems. Else, if the stored value is not fresh, the front-end reports that it cannot provide a valid value.

Of course, the autopilot is able to handle the case where the employed precision landing sensor does not produce a sufficiently fresh value – this can also happen when using the IRLock sensor or the marker sensor, individually. In fact, the precision landing control logic of ArduPilot *ignores* old (stored) sensor values that were already retrieved in a previous iteration, and only takes into account newly acquired sensor values. As a consequence, the freshness setting has no real effect on the control of the precision landing process. Nevertheless, as done in all other sensor subsystems, we adopt the notion of freshness and the standard freshness threshold in the fused sensor front-end too.

6 REFINED PRECISION LANDING

The default operation of ArduPilot during precision landing is to continue the descend towards the ground even if the landing target is lost, i.e., despite the precision landing sensor failing to provide new/valid measurements. However, this behavior is not always desirable. For instance, the vehicle may need to land on a battery recharging/switching platform or inside a hangar, in which case it is important to accurately guide the vehicle during the entire landing procedure.

With this motivation, we have introduced a new mode of operation for precision landing, where the vehicle can repeat the landing attempt multiple times, before taking the decision to perform an un-guided (blind) vertical descent. The main difference is in the logic that controls the vertical movement of the vehicle during precision landing.

6.1 DEFAULT OPERATION

In each iteration of the precision landing control loop, the autopilot checks the distance to the ground as well as the horizontal displacement with respect to the landing target, and adjusts the descent accordingly. In a nutshell, if the ground distance is above a high threshold or below a low threshold, the desired rate of descent is set to a pre-specified maximum and minimum value, respectively. Else, it is calculated as a function of a pre-specified landing speed and the horizontal displacement from the target landing position (larger deviations lead to a reduction of the desired rate of descent). Finally, the desired rate of descent is fed into the vertical position controller, which converts it to the desired acceleration and throttle value, which, in turn, is sent to the attitude control library that controlling the motors/actuators of the vehicle.

If the landing target is lost during the precision landing procedure, the autopilot will not have any input regarding the horizontal displacement of the vehicle with respect to the target landing position. In this case, the autopilot simply continues the descent, at the default landing speed. In other words, once the autopilot is placed in the land mode, it performs a steady descent towards the ground, irrespective of whether the precision landing sensor can track the landing target.

6.2 REPEATED LANDING ATTEMPTS

There are several reasons why the landing target can be lost during descent. For instance, it might be temporarily occluded, or fall outside of the field of view of the sensor, or the sensor may have malfunctioned. Note that such incidents can be transient rather than permanent. In such cases, it can be meaningful to repeat the landing attempt instead of continuing the descent in a blind way as in the default operation.

To this end, we introduce a so-called *cautious* precision landing operation, where the landing attempt is repeated if the landing target is lost during descent. Figure 11 illustrates the control flow.

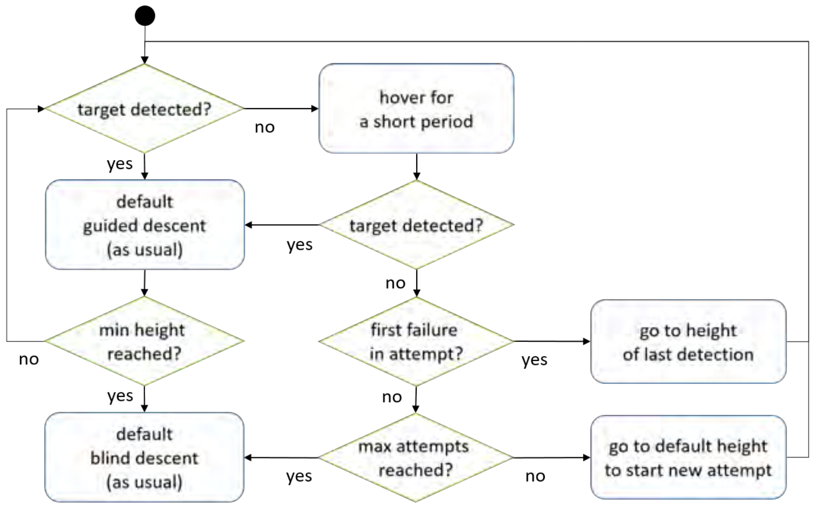

target detected?
no
hover for a short period
yes
default guided descent (as usual)
target detected?
yes
no
min height reached?
no
first failure in attempt?
yes
go to height of last detection
yes
no
default blind descent (as usual)
max attempts reached?
yes
no
go to default height to start new attempt

Figure 11: Cautious precision landing control flow.

Each time the target landing position is lost (the precision landing sensor used does not report new displacement values), the autopilot is instructed to pause the descent and keep the vehicle steady for a short amount of time. During this pause period, the sensor is polled to check if the landing target is detected. If this is the case, the landing procedure continues as usual, taking into account the displacement information produced by the sensor. Else, the autopilot is instructed to go back to the position where the target was last detected successfully during the current attempt, or (if this fails too) to a default height from where it starts another landing attempt from scratch.

Obviously, it is not desirable to perform attempts ad infinitum – besides introducing a large delay, at some point the vehicle will run out of energy. For this reason, we set a limit on the number of landing attempts that can be performed. When this limit is reached, the autopilot performs a blind descent as usual.

Note that the pause period, the default height from where repeated landing attempts are started and the pause period during which the autopilot waits to detect the landing target, are all configurable parameters which can be set before starting a mission.

7 FAULT INJECTION

Testing the precision landing support that was described in the previous chapters is non-trivial. The reason is that failures are rare during real operation Also, they are hardly reproducible. In order to enable testing in a simple and flexible way, a simple but practical mechanism was developed, which makes it possible to inject artificial faults to the individual precision landing sensor subsystems.

7.1 FAULT-INJECTION (DROP) MODES

The currently supported fault-injection results in sensor values being dropped (there is no distortion of sensor values). This way it is possible to model sporadic or temporary malfunctions of a given sensor subsystem. This includes the case where a given sensor functions properly but still fails to detect the target, e.g., because of external interference.

Two different drop modes are supported:

- **Random drop.** In this mode, a newly acquired value is dropped with some probability.
- **Periodic drop.** In this mode, a number of newly acquired values are periodically dropped. Thus, there is a phase where all newly acquired sensor values are kept as usual, followed by another phase where all new sensor values are systematically dropped.

The implementation approach is discussed below.

7.2 IMPLEMENTATION

The fault injection functionality is introduced by modifying the front-ends of the precision landing sensor subsystems (IRLock sensor, marker sensor and fused sensor) so that they drop the values produced by the respective back-ends, according to the current mode. The mode can be set, at runtime, via a command issued to the autopilot. Table 1 shows the main structure of the respective MAVLink message.

The *target* field indicates the sensor subsystem where fault-injection should be applied. The *r_drop* field is used in the random drop mode. It takes values between 0 and 1 that indicate the probability for dropping

Field	**Type**	**Description**
target	uint8_t	Target sensor subsystem (IRLock: 1, marker: 2, both: 3)
r_drop	float	Probability for dropping a new sensor value (random mode)
p_keep	uint16_t	Number of consecutive new sensor values to keep (periodic mode)
p_drop	uint16_t	Number of consecutive new sensor values to drop (periodic mode)

Table 1: Main part of the fault-injection MAVLink message.

a new sensor value that is acquired from the respective back-end. If $r\_drop > 0.0$ the other fields of the message are ignored. If $r\_drop = 0.0$, the rest of the fields are used to specify the behavior of the periodic drop mode. More specifically, *p_keep* specifies the number of consecutive new sensor values that should be kept before starting the drop phase, and, similarly, *p_drop* is the number of new values that should be dropped in sequence before reverting to the keep phase. Note that $r\_drop = 1.0$ is equivalent to $r\_drop = 0.0 \wedge p\_keep = 0 \wedge p\_drop > 0$ yielding a permanent failure of the respective sensor subsystem. Also, a message with all fields set to zero ($r\_drop = 0.0 \wedge p\_keep = 0 \wedge p\_drop = 0$) will de-activate fault-injection for the respective target sensor subsystem.

Each time the ArduPilot command/message handler receives this message, it simply forwards it to the active front-end of the sensor subsystem that is used to guide precision landing (only one front-end can be active: for the IRLock, marker or fused sensor). The active front-end driver, in turn, decodes the message and applies the corresponding fault injection to the values that are acquired from the corresponding back-end driver(s).

Note that the IRLock and marker front-ends can only apply fault-injection to the values received from their own back-ends, and thus will ignore any commands that target another subsystem. Of course, the fused sensor front-end can handle fault-injection commands targeting any of the two sensor subsystems.

8

EVALUATION

This chapter presents the evaluation of the newly developed precision landing mechanisms. Initially, several tests were performed using a simulated setup in order to debug the system in a fast and safe way. Once sufficient confidence was gained in the robustness of the implementation, additional experiments were performed using a real drone to confirm the additional precision landing functionality.

8.1 SIMULATION EXPERIMENTS

Simulation-based experiments are a typical step in the development of autonomous vehicles, before attempting to perform tests in the field. A good simulated setup makes it possible to debug critical pieces of code, especially different corner cases that may occur rarely in reality, without having safety concerns or risking to damage the equipment. This is crucially important for unmanned aerial vehicles where even a simple bug can lead to severe crashes and accidents.

8.1.1 *Gazebo*

In our work, we use the open-source Gazebo [47] simulator, a 3D dynamic simulator with the capability of accurately and efficiently simulating robots in complex indoor and outdoor environments. Gazebo offers physics simulation at a high degree of fidelity, a variety of different sensors as well as interfaces for both users and external programs.

Notably, Gazebo allows the user to define robot kinematics and dynamic attributes, sensors, joint frictions and many more properties. These are described in the XML-based Simulation Description Format (SDF), where the two main elements are: (i) models, which may represent a wide range of both static and active objects, from simple and passive 3D shapes to complex and mobile robots, and (ii) a world, which is a collection of models together with a variety of physics properties, such as gravity, wind, light, etc.

These elements can be customized through plugins, which are written in the C++ language and are compiled as shared libraries. Such plugins can be inserted as attributes of the target element in the SDF file. Through plugins it is possible to programmatically alter the simulation itself, e.g., generating stronger wind or increasing the ambient light, move models to points of interest, retrieve sensor data, respond to events, or to dynamically insert new models under certain preconditions.

8.1.2 *Simulation setup*

For the APM autopilot, we use a software-in-the-loop (SITL) configuration, which is provided by the ArduPilot software suite. This is illustrated in Figure 12. The configuration is derived from the same code base that targets the Pixhawk and other boards that are used in real drones.

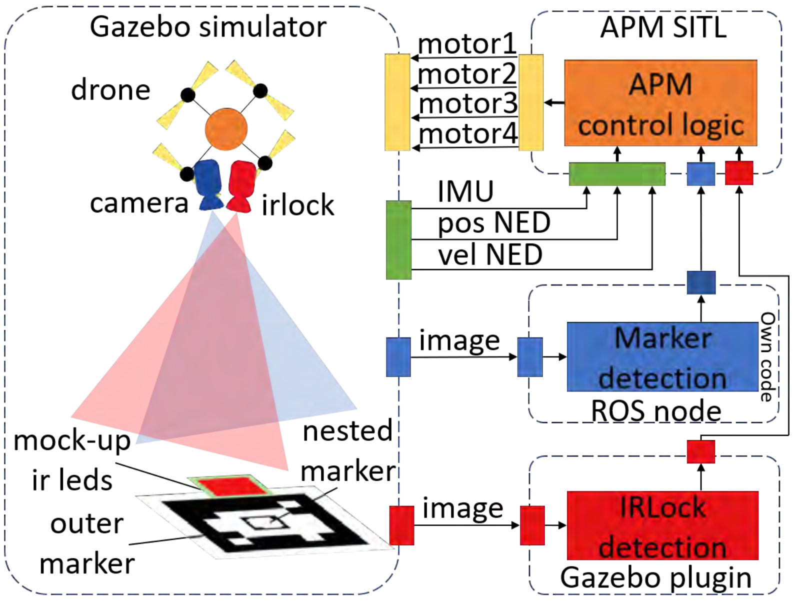


Figure 12: SITL configuration setup.

Gazebo acts as an external flight-dynamics simulation engine for the autopilot, providing it with the vehicle's position and velocity vectors in north-east-down (NED) frame, as well as IMU angular velocity, IMU linear acceleration and IMU quaternion orientation. The autopilot processes these inputs in practically the same way (running almost the same code) as when it runs on a real drone (where it receives this data from real sensors/IMU). As a result of this processing, the autopilot generates the control outputs for the drone's motors, which are fed back into the Gazebo simulation environment (instead of the motors of a real drone).

The model for the drone, written in SDF format, the linkage between the standard build-in IMU sensor (provided by the Gazebo software suite) and the drone as well as the plugin for the vehicle's logic/behavior within the simulation environment, are all derived from the officially recommended code repository [48]. The communication between the

Gazebo plugins and the ArduPilot autopilot components is done via the UDP protocol.

8.1.3 *IRLock subsystem*

The Gazebo environment also provides an IRLock plugin and a camera attached to the drone model (red camera in Figure 12), which can be used to perform precision landing using the IRLock subsystem. More specifically, the camera sensor produces images within Gazebo, and a plugin that is attached to the camera object receives and processes these images with the goal to recognize a red object within the simulation environment (which is a proxy for the IR beacon). Each time the sensor detects this specific object, it sends position information to the autopilot system. Note, however, that this is basically a mock-up, rather than a real simulation of the IRLock mechanism (as described in Section 3.4).

8.1.4 *Marker detection subsystem*

In order to execute the marker-based precision landing via Gazebo, we have developed a simple box model with its top side featuring a fractal marker pattern. Also, we customized the available drone model (SDF file) by linking it with an additional camera sensor (blue camera in Figure 12). This sensor grasps raw images from the simulation environment and sends them to the marker detection software component in order to detect the fractal marker.

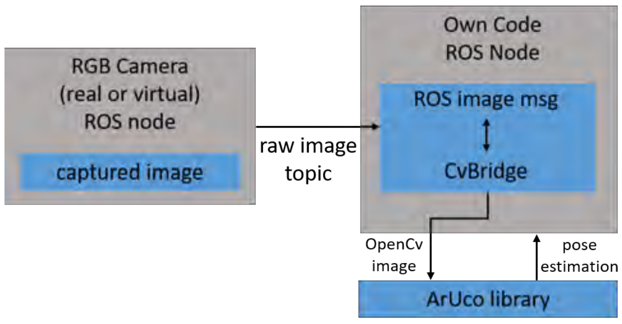


Figure 13: Implementation of marker detection component on top of ROS.

An illustration of the implementation approach is shown in Figure 13. To achieve a more flexible configuration, the marker detection code was implemented on top of the ROS middleware [49]. The communication between the camera sensor and the marker detection component is achieved via the pub/sub support of ROS – the camera publishes each image as a separate message to a specific topic, and the marker detection component subscribes to that topic in order to receive these images. Using the CvBridge ROS routine function, each raw image is converted to the OpenCV image format so that it can be processed by the ArUco library.

The marker detection component is designed to support two different configurations, for simulation-based and real-world operation, respectively. The desired configuration is chosen when starting the program, via configuration parameter. The difference between the two configurations is how the results of market detection (pose estimation vectors) are sent to the autopilot subsystem. In the simulation-based configuration this is done via UDP (in accordance to the convention of the Gazebo environment), whereas in the real-world configuration this information is communicated to the autopilot via the MAVLink protocol.

8.1.5 *Precision landing modes*

The ArduPilot software suite already support an *IRLock_Gazebo* mode in order to allow precision landing experiments using the IRLock mechanism (mockup) in the simulations.

In the same spirit, we introduce additional precision landing modes *Marker_Gazebo* and *Fused_Gazebo* so that the newly introduced precision landing sensor subsystems can be used in the Gazebo simulator. Also, corresponding back-ends and front-ends (referring to these back-ends) were developed. The implementation is essentially identical to the one used for real-world operation. The main difference is that the back-end version for the simulation receives data through the UDP protocol instead of MAVLink.

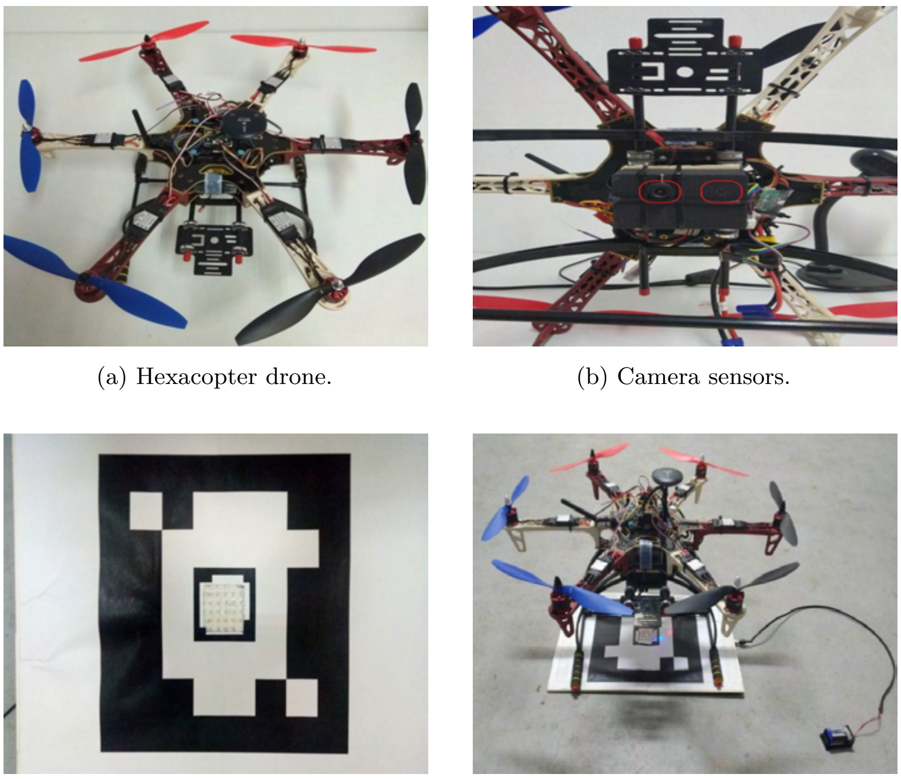
8.1.6 *Validation*

The functionality of new precision landing capabilities was validated in an extensive way using the SITL configuration of the ArduPilot in the Gazebo simulation environment. Numerous experiments were performed in order to test: (i) the behaviour of each of the supported precision landing modes; (ii) the fault-injection logic for each of the sensor subsystems; (iii) the refined precision landing operation.

After some debugging, we confirmed the successful implementation/integration into the ArduPilot autopilot software and a robust operation of the newly developed functionality for a wide range of scenarios. We then proceeded to perform experiments in the field using a real drone.

8.2 FIELD EXPERIMENTS

Several experiments were performed in the field in order to measure the actual accuracy of the individual precision landing mechanisms and verify the robustness of the fused sensor in practice. In the following, we provide an overview of the drone hardware setup, present indicative experiments and discuss the obtained results.


(a) Hexacopter drone. (b) Camera sensors.

(c) Marker with nested MarkOne. (d) Drone on the landing target.

Figure 14: Drone and the landing target.

8.2.1 *Hardware Setup*

The vehicle used for the field experiments is a custom-built hexacopter, shown in Figure 14a. The autopilot software runs on *CUAV V5 Nano*, a 32-bit flight controller, based on the Pixhawk FMUv5 design standard hardware platform, designed by CUAV in collaboration with the PX4 team. The platform features the following build-in sensors: (i) a ICM-20689 6-axis motion tracking device that combines a 3-axis gyroscope and a 3-axis accelerometer, (ii) a ICM-20602 6-axis motion tracking device including a 3-axis gyroscope and a 3-axis accelerometer, (iii) a BMI055 6-axis inertial measurement unit (IMU) consisting of a digital, 3-axis acceleration sensor and a 3-axis gyroscope, (iv) a IST8310 3-axis digital magnetometer, and (v) a MS5611 barometric pressure sensor. The drone also features the Neo v2 GPS/Compass sensor and an on-board Raspberry Pi model B, which communicates with the CUAV V5 Nano board via the MAVLink protocol on top of a serial UART interface.

As shown in Figure 14b, a downwards facing IRLock sensor camera (left red box) and Raspberry Pi camera module v2 (right red box), used for marker detection, are attached at adjacent positions in the frame's center. The IRLock sensor camera features the Pixy vision sensor with a 3.6 millimeter lens, and runs the IRLock filter that generates data at

50 Hz which are sent to the autopilot controller via the I2C protocol. The Raspberry Pi camera is connected to the on-board Raspberry Pi board and is configured to generate images at a resolution of $640x480$ ($480p$) at a rate of 30 fps.

The landing pad, shown in Figure 14c, consists of a fractal ArUco marker and a MarkOne IR beacon. The beacon is placed so that it lies inside the nested marker of the fractal marker. The fractal marker was designed so that most of its inner part is white. The inner part of the nested marker is cut out and replaced with the IR beacon. Finally, the beacon's surface is covered with a white duct tape (except the IR LEDs) in order for the inner part of the nested marker to remain white and be correctly detected by the ArUco library.

8.2.2 *Mission planning*

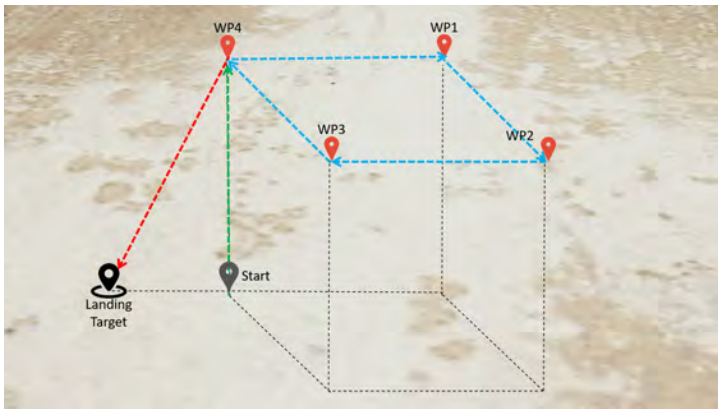
In all experiments, the drone follows a standard mission specified through a script using the DroneKit environment [50]. An illustration of the mission is given in Figure 15a showing all major positions and waypoints. The photo shows the site where all tests where performed (flat open space with no obstacles).

Initially, the drone is placed at a start position, 2 meters away from the landing pad. It is then armed and instructed to take off at a certain altitude, which depends on the precision landing mode and sensor used in each experiment/run. More specifically, in the IRLock and Fusion mode, the take-off altitude is set to 10 meters, whereas in the Marker mode it is set to 6 meters because the marker cannot be detected reliably from higher altitudes.

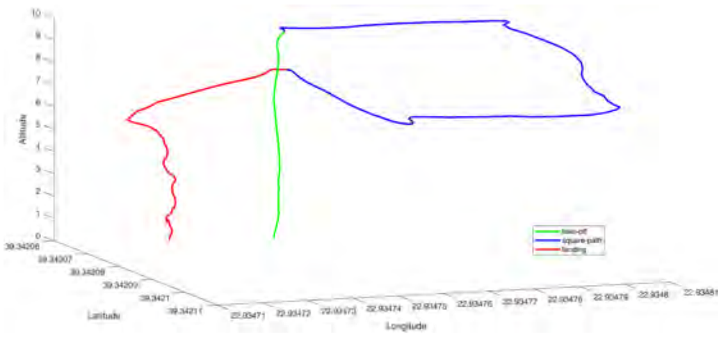
When the drone reaches the target take-off altitude, it marks its current GPS position and then follows a horizontal square path, moving away from the landing target and then returning back to the recorded GPS position. The path is defined via four waypoints (WP1, WP2, WP3, WP4) with the last one being the position recorded once take-off completes. To conserve the drone's battery and increase the number of runs that can be performed without recharging, the distance between two consecutive waypoints is set to 2 meters.

Finally, when the drone reaches the last waypoint, it is put in one of the supported land modes (Normal, IRLock, Marker or Fused). As a result, it starts the landing approach and eventually completes the mission. If a precision landing mode is enabled, the drone will try to detect and land on the center of landing target, else it will perform a normal landing approach towards the initial start position.

Figure 15b illustrates the actual path that is followed by the drone in one of the tests. In this case, a precision landing modes is enabled, so that the drone lands on the target (rather than on the initial start position). The green line represents the trajectory followed during take-off, the blue line marks the square path followed by the drone,



(a) Planned mission and test site.



(b) Indicative drone trajectory.

Figure 15: Mission and indicative flight path.

and the red line is the trajectory followed during the landing approach towards the target.

8.2.3 *Configuration options*

We performed a wide range of experiments using the different precision landing sensors (via the corresponding land modes) and for various failure scenarios for the individual sensor subsystems. The different configuration options are summarized in Table 2.

Landing Sensor	**Sensor Failure**
IRLock Marker Fused	None (no drops) Random X (drop with probability X) Permanent (drop during the entire landing approach) Below X (drop only if $<=$ X meters above target) Above X (drop only if $>=$ X meters above target)

Table 2: Experiment configuration options.

To keep the number of experiments reasonable, we experiment with the following sensor failure scenarios: (i) *Random*, where each value that is produced by the sensor in question is dropped with a certain probability; (ii) *Permanent*, where all sensor values are dropped during the entire landing approach; (iii) *Below X*, where all sensor values are dropped when the drone is below a certain height from the landing position; (iv) *Above X*, where all sensor values are dropped when the drone is above a certain height from the landing position.

In the Random failure scenarios, we set the failure probability to 0.75 for the IRLock sensor, and to 0.50 for the Market sensor (which has a lower sampling rate). In the height-based sensor failure scenarios (Below and Above), the height threshold is set to 3 meters, so that sensor values are dropped systematically as long as the drone is below or above 3 meters from the landing position, respectively.

Note that the above failure scenarios can be applied to the individual IRLock and Marker sensor subsystems, interchangeably or at the same time. The fused sensor simply uses the values that are provided by the two sensor subsystems, if any.

In each experiment, we record the landing position of the drone and measure the distance of the drone's camera(s) from the center of the landing pad (landing error). In the Normal land mode, where the drone does not employ any precision landing sensor, the distance (and error) is measured with respect to the initial take-off position, instead of the center of the landing pad.

8.2.4 *Results for the individual precision landing sensors*

In a first set of experiments, we evaluate the landing accuracy when using each of the IRLock and Marker sensor subsystems individually. We perform tests for the case where these sensors function properly without any (artificial) failure as well as for different scenarios of sensor failures. As a reference, we use the Normal land mode.

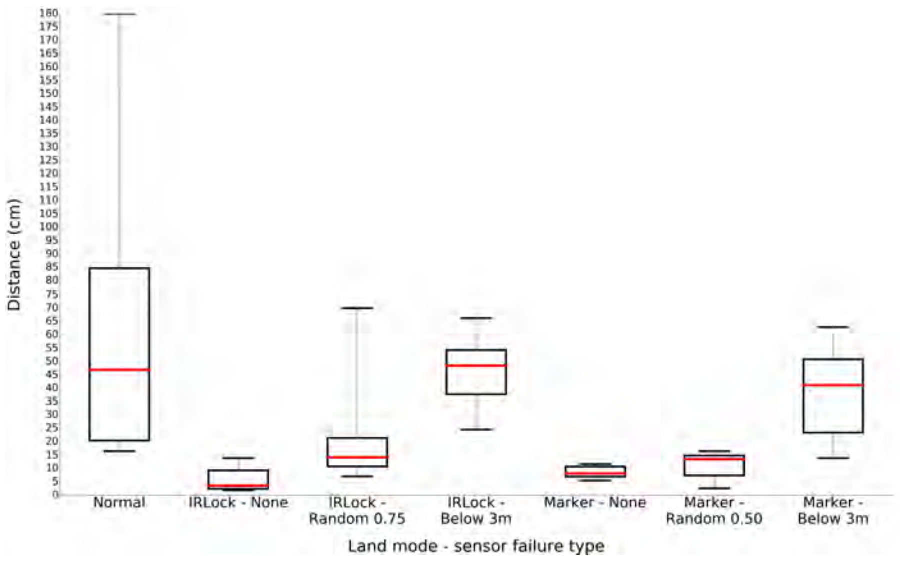
Each scenario was tested 10 times. During these tests, the weather conditions where practically ideal (no wind, no rain, clear visibility). Table 3 reports the average and maximum values, while Figure 16 provides a statistical and actual view of the final landing positions in these trials. The detailed measurements are given in Appendix A, Appendix B and Appendix C.

Initial Height	**Landing Sensor**	**Sensor Failure**	**Avg Error (cm)**	**Max Error (cm)**
10m	None	None	67.15	180.00
10m	IRLock	None	05.79	13.86
10m	IRLock	Random 0.75	21.86	70.00
10m	IRLock	Below 3m	47.10	66.30
6m	Marker	None	08.51	11.60
6m	Marker	Random 0.50	11.05	16.50
6m	Marker	Below 3m	39.32	62.90

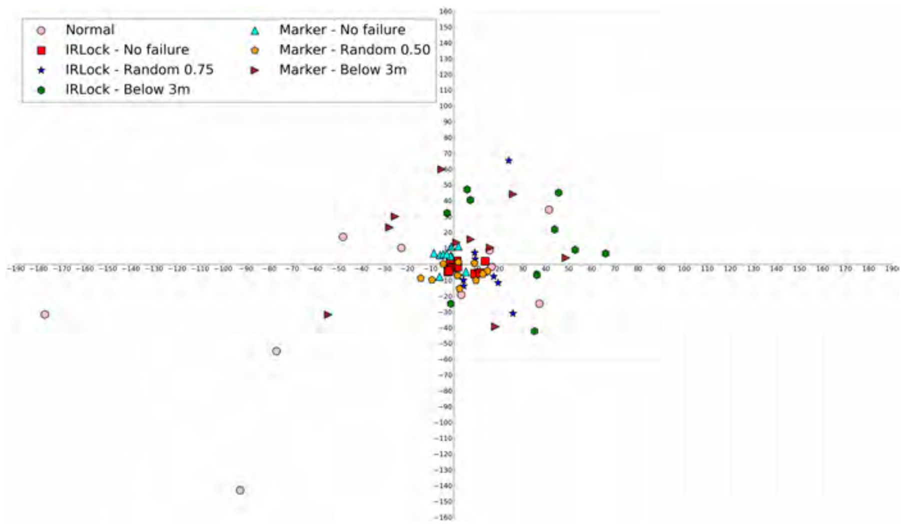
Table 3: Results for the Normal, IRLock and Marker land modes.

When using the Normal land mode, as expected, the result is heavily affected by the the GPS error [13], with a a recorded average deviation of 67.15 cm and a worst case of 180 cm (almost 2 meters) away from the initial position. Obviously, this inaccuracy is unacceptable for applications where the drone must land accurately, e.g., on a platform or inside a hangar. The landing accuracy improves substantially when using the IRLock or Marker sensor subsystem, in which cases the drone lands with just a small error, on average 6.79 and 8.51 cm with a worst case of just 13.86 and 11.60 cm, respectively.

In the same set of experiments, we also investigate the behavior when the individual precision landing sensor subsystems fail. We perform tests for random failures (IRLock Random 0.75, Marker Random 0.5) and more systematic failures (Below 3 meter failure mode). As expected, failures have a negative impact on the landing accuracy. Still, most random failures are tolerated fairly well, leading to an average landing error of 21.86 cm and 11.50 cm for the IRLock and Marker sensor, respectively. In contrast, the error increases quite significantly when the sensors fail systematically below 3 meters, as the drone performs the last phase of the descent based exclusively on its IMU and dead-reckoning ability. These results show that it is crucial to tolerate sensor malfunctions especially during the last phase of the landing approach.



(a) Distance of the landing position from the center of the landing target. The red line is the median, the box is the interquartile range, and the whiskers are the maximum/minimum values.



(b) Landing positions relative to the center of the landing target (units in cm).

Figure 16: Landing accuracy when using the IRLock or Marker sensor subsystem individually, with/without failures.

8.2.5 *Results for the fused precision landing sensor*

In a second set of experiments, we evaluate the robustness of the fused precision landing sensor, which combines the IRLock and Marker sensor subsystems. We also apply suitable failure combinations for the individual sensor subsystems to verify the robustness of the mechanism. More specifically, we let the two sensors fail in a complementary way (IRLock Above 3 m - Marker Below 3 m, and vice versa) as well as in a random way (IRLock Random 0.75 - Marker Random 0.5) where both sensors may occasionally fail at the same time.

As above, each scenario was tested 10 times. The results are summarized in Table 4 and Figure 17. The detailed measurements are given in Appendix D. It is important to note that, in this case, the test site was exposed to regular wind gusts during the tests – unlike the previous experiments, which were conducted under ideal conditions. For this reason, it is not meaningful to directly compare these results with the ones that were obtained in the previous set of experiments. Nevertheless, Figure 17 also shows the results of the Normal land mode (under ideal conditions) as an optimistic proxy for the best result that could be achieved under the non-optimal conditions without using any precision landing sensor or (equivalently) when using a single sensor that fails systematically during the entire landing approach.

Initial Height	**IRLock Failure**	**Marker Failure**	**Avg Error (cm)**	**Max Error (cm)**
10m	None	None	13.62	24.60
10m	Below 3m	Above 3m	14.66	20.60
6m	Above 3m	Below 3m	13.74	20.40
10m	Random 0.75	Random 0.50	29.40	46.10

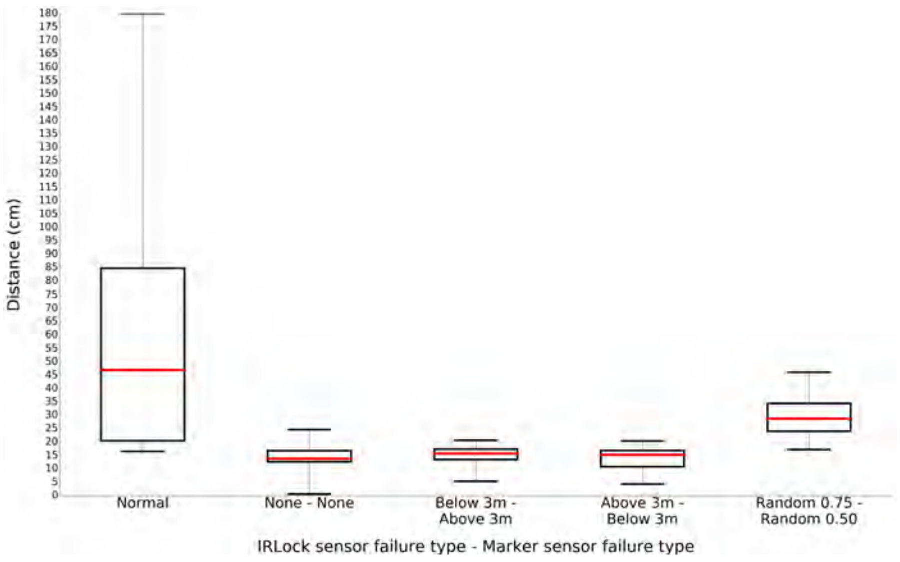
Table 4: Results for the Fused land mode.

The robustness of the fused precision landing sensor can be clearly seen by comparing the case where both the IRLock and Marker sensor subsystems operate without any failure (IRLock None - Marker None) with the cases where only one of these subsystems operates normally while the other fails systematically (IRLock Above 3 m - Marker Below 3 m, and IRLock Below 3 m - Marker Above 3 m). Namely, all cases yield practically the same average error of 13-15 cm with a worst case of 20-21 cm. Also note that the worst case remains an entire order of magnitude lower than what would be expected if the drone had to land without a (working) precision landing sensor.

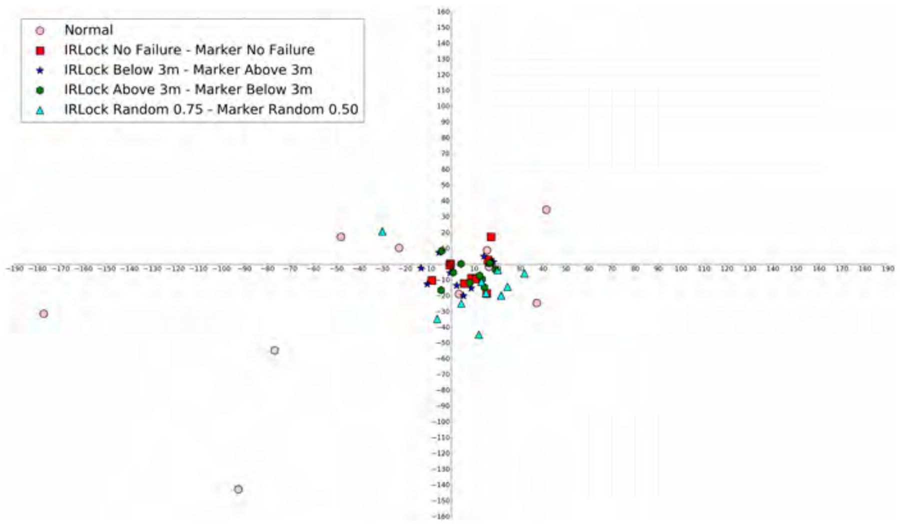
The occasional simultaneous failures in both sensor subsystems (IRLock Random 0.75 - Marker Random 0.5) lead to occasional (total) failures of the fused precision landing sensor. Of course, this has a clear impact on the landing accuracy. In this case, the average error jumps to about 30 cm and the worst case to more than 46 cm, a deterioration of roughly $2x$ compared to the scenarios where at least one of the sensor subsystems works properly at any point in time and the fused sensor is able to provide reliable measurements during the entire landing approach.

8.2.6 *Results for the cautious landing approach*

In a final set of experiments, we evaluate the developed support for a more cautious landing approach, where the landing attempt is repeated



(a) Distance of the landing position from the center of the landing target. The red line is the median, the box is the interquartile range, and the whiskers are the maximum/minimum values.



(b) Landing positions relative to the center of the landing target (units in cm).

Figure 17: Landing accuracy when using the Fused sensor, with/without failures of the individual IRLock and Marker sensor subsystems.

if the precision landing sensor fails to provide measurements during descent. We use the Marker sensor and inject a systematic malfunction once the drone reaches a height less than 3 meters from the landing target (Below 3 m failure mode). Recall that, in the default approach, the autopilot continues the landing procedure even if it does not receive any measurements from the precision landing sensor. In contrast, in the cautious approach, the drone repeats the attempt, initially by hovering for a short while and then returning to a higher altitude in order to re-detect the landing target. In this second attempt, we remove the artificial failure and the Marker sensor works properly during the entire landing approach.

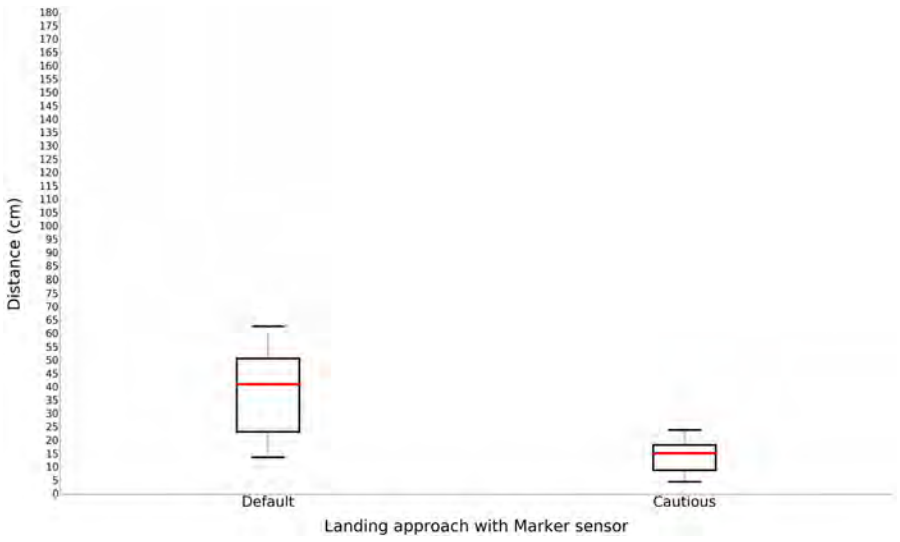
Each test is repeated 10 times. The results are summarized in Table 5 and Figure 18. In Table 5, we also report the average time that was needed for the drone to land. The detailed measurements of the cautious landing approach are given in Appendix E. During the tests, the weather conditions were once again non-optimal, comparable to the ones in the second set of experiments.

Landing Approach	**Avg Error (cm)**	**Max Error (cm)**	**Avg Time (sec)**
Default	39.32	62.90	17.43
Cautious	14.37	24.10	41.46

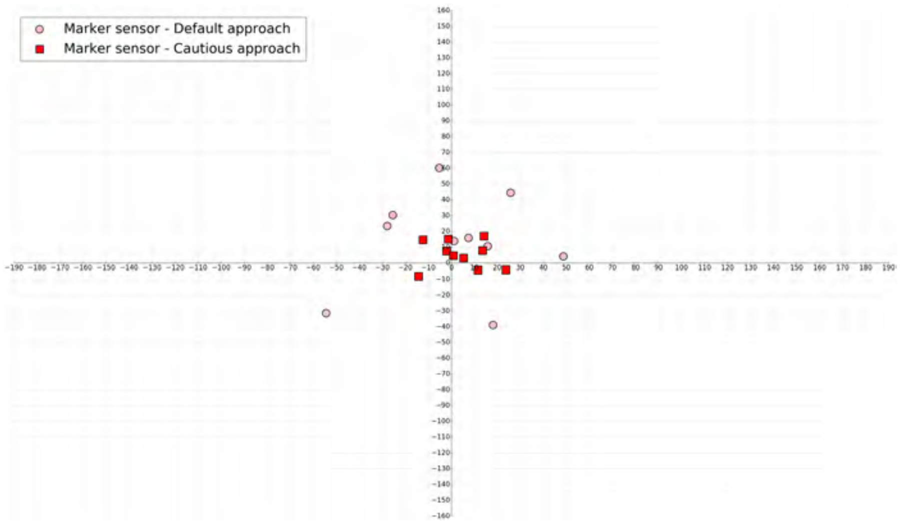
Table 5: Results for the cautious landing approach, for the case where the precision landing sensor (Marker) fails below 3 m during the first landing attempt.

As can be seen, the cautious landing approach significantly increases robustness, achieving an average and worst case error of 14.37 cm and 24.10 cm, respectively. Without this precaution, the drone lands with a much larger error, more than $2x$ compared to the cautious approach.

Note that the results of the cautious landing approach are very close to those in the scenarios where at least one of the precision landing sensors function properly already during the first attempt (under comparable conditions, in the previous experiments). Of course, the additional landing attempt performed in the cautious approach comes at an increased delay, taking more than $2x$ compared to the case where the drone continues the landing approach despite the sensor failure. This extra delay (a few tens of seconds) is typically perfectly tolerable in order to ensure a more accurate landing.



(a) Distance of the landing position from the landing target. The red line is the median, the box is the interquartile range, and the whiskers are the maximum/minimum values.



(b) Landing positions relative to the center of the landing target (units in cm).

Figure 18: Landing accuracy when using the default and cautious landing approach, with a temporary sensor malfunction during the first landing attempt.

9

CONCLUSIONS

In this thesis, we have developed a precision landing sensor subsystem based on the detection of visual markers, and we have introduced support for its combined use together with an existing infrared-based sensor subsystem through a fused sensor that can tolerate individual failures and malfunctions of any of the two subsystems as long as these do not occur at the same time. We have also introduced a more refined landing approach, whereby the autopilot repeats the landing procedure in case the landing target is lost during descent.

The new precision landing capabilities are evaluated using the Gazebo [47] simulator, extending the functionality of the officially recommended code repository [48] of the SITL ArduPilot platform, as well as by conducting a wide range of flight tests in the field. In both cases, we experiment with different scenarios by injecting artificial faults/malfunctions in the individual sensor subsystems at runtime in a controlled way. Our results show that the proposed approach indeed achieves the desired tolerance to independent failures of any single sensor subsystem at any point in time, while maintaining a satisfactory accuracy compared to the Normal land mode.

Our work opens-up the way for the smooth integration of additional precision landing sensor mechanisms, which can be based on completely different technologies, such as magnetic materials or ultrasound signals. Such technologies could be used individually and/or be combined with the current sensor subsystems to further improve the landing accuracy for applications that have even stricter requirements in this respect. Given the very significant overhead of performing real experiments in the field, it would also be important to introduce realistic simulation support for these sensor subsystems in the Gazebo environment.

Appendices

A

NORMAL LAND MODE

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	95	$-76.85, -54.48$	13.35
2	170	$-92.58, -142.57$	10.73
3	180	$-177.26, -31.25$	15.1
4	54	$41.36, 34.71$	12.37
5	51	$-47.92, 17.44$	13.11
6	43	$37.23, -21.5$	9.21
7	18	$15.58, 9$	14.62
8	16.5	$16.43, -1.43$	11.23
9	19	$3.29, -18.71$	12.76
10	25	$-22.65, 10.56$	13.81

Table 6: Normal land mode.

B

IRLOCK LAND MODE

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	2.7459	$1.5, 2.3$	21.15
2	2.1931	$-1.6, 1.5$	23.1
3	10.6850	$9.1, -5.6$	23.67
4	12.4390	$11.3, -5.2$	28.11
5	13.86	$13.7, 2.1$	22.9
6	1.8027	$0.1, -1.8$	20.62
7	2.3259	$-1, -2.1$	24.3
8	5.1478	$-2.5, -4.5$	21.43
9	4.3416	$-2.1, -3.8$	21.77
10	2.4083	$1.8, -1.6$	20.17

Table 7: IRLock land mode, no sensor failure.

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	70	$23.94, 65.77$	29.3
2	40	$25.71, -30.64$	31.89
3	11.729	$9.1, 7.4$	27.19
4	9.902	$9.3, 3.4$	32.6
5	14.5165	$13.7, -4.8$	30.29
6	13.914	$4.4, -13.2$	30.72
7	7.0214	$2.1, -6.7$	24.72
8	10.4278	$4.3, -9.5$	27.21
9	22.3	$19.31, -11.15$	29.32
10	18.8308	$17.4, -7.2$	31.17

Table 8: IRLock land mode, random 0.75 sensor failure.

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	24.5	$-1.28, -24.46$	31.23
2	36.6	$36.04, -6.35$	27.82
3	64.3	$45.47, 45.47$	27.7
4	49.1	$43.75, 22.29$	25.32
5	66.3	$65.93, 6.93$	32.36
6	53.5	$52.69, 9.29$	33.41
7	41.5	$7.21, 40.87$	22.8
8	54.6	$35.09, -41.82$	24.88
9	47.9	$5.84, 47.54$	23.29
10	32.7	$-2.85, 32.58$	23.4

Table 9: IRLock land mode, below 3m sensor failure.

C

MARKER LAND MODE

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	11.1	$-8.5, 7.13$	23.34
2	8.4	$-5.93, 5.93$	26.01
3	7.9	$-4.53, 6.47$	24.99
4	9.5	$-6.1, -7.28$	25.12
5	5.7	$-0.99, 5.61$	25.29
6	5.5	$-1.42, 5.31$	25.17
7	7.2	$-3.04, 6.52$	27.67
8	11.3	$-0.98, 11.25$	23.67
9	11.6	$2.01, 11.42$	27.09
10	6.9	$5.28, -4.43$	21.56

Table 10: Marker land mode, no sensor failure.

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	13.7	$9.68, -9.68$	21.91
2	2.5	$2.05, 1.43$	22.57
3	15.1	$14.59, -3.91$	24.83
4	9.1	$9.06, 0.79$	21.42
5	13.9	$12.59, -5.87$	20.31
6	6.8	$1.76, -6.57$	32.01
7	16.5	$-14.28, -8.25$	31.02
8	4.5	$-4.48, 0.39$	24.96
9	15.1	$2.62, -14.87$	25.13
10	13.3	$-9.4, -9.4$	31.5

Table 11: Marker land mode, random 0.50 sensor failure.

Trial	**Landing Error (cm)**	**x, y (cm)**	**Landing Time (s)**
1	60.5	$-5.28, 60.27$	17.13
2	51.5	$25.75, 44.6$	17.87
3	13.9	$1.21, 13.84$	16.73
4	17.5	$7.39, 15.86$	18.67
5	36.4	$-27.88, 23.39$	17.05
6	39.6	$-25.45, 30.33$	17.62
7	62.9	$-54.47, -31.45$	19.17
8	42.9	$18.13, -38.88$	16.84
9	19.1	$15.83, 10.68$	15.98
10	48.9	$48.71, 4.26$	17.26

Table 12: Marker land mode, below 3m sensor failure.

D

FUSED LAND MODE

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	16.9	$16.83, 1.47$	25.31
2	16.2	$15.95, 2.81$	36.84
3	24.6	$17.39, 17.39$	34.71
4	12.4	$8.76, -8.76$	39.33
5	0.5	$-0.38, 0.32$	36.65
6	0.7	$-0.6, -0.35$	37.39
7	13.2	$-8.48, -10.11$	40.24
8	24.1	$15.49, -18.46$	34.21
9	13.4	$5.66, -12.14$	32.13
10	14.2	$10.87, -9.13$	34.47

Table 13: Fused land mode, IRLock no failure – Marker no failure.

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	13.4	$-13.17, -2.32$	36.51
2	5.3	$-0.46, -5.28$	36.85
3	20.6	$5.33, -19.89$	37.9
4	18.4	$18.33, 1.6$	31.32
5	9.3	$-5.33, 7.61$	33.5
6	15.1	$14.19, 5.16$	34.05
7	17.3	$17.23, 1.51$	36.18
8	16.3	$-10.47, -12.48$	38.05
9	13.5	$2.34, -13.29$	34.61
10	17.4	$8.7, -15.07$	32.74

Table 14: Fused land mode, IRLock below 3m failure – Marker above 3m failure.

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	20.4	$14.42, -14.42$	36.43
2	16.4	$13.43, -9.41$	34.2
3	4.3	$4.28, 0.37$	37.62
4	16.7	$16.67, 0.87$	26.75
5	19.6	$19.3, -3.4$	37.2
6	14.2	$12.29, -7.1$	33.3
7	14.1	$8.09, -11.55$	25.1
8	5.2	$0.9, -5.12$	31.3
9	16.9	$-4.37, -16.32$	38.1
10	9.6	$-4.05, 8.7$	34.52

Table 15: Fused land mode, IRLock above 3m failure – Marker below 3m failure.

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	46.1	$11.93, -44.53$	32.82
2	28.2	$24.42, -14.1$	34.26
3	17.2	$13.17, -11.05$	34.41
4	32.3	$31.81, -5.61$	34.56
5	20.5	$20.19, -3.56$	34.52
6	25.1	$4.36, -24.72$	36.07
7	29.3	$21.75, -19.61$	39.74
8	23.6	$15.17, -18.08$	38.72
9	36.6	$-29.98, 20.99$	33.99
10	35.1	$-6.09, -34.57$	40.42

Table 16: Fused land mode, IRLock random 0.75 failure – Marker random 0.50 failure.

E

CAUTIOUS LANDING APPROACH

Trial	Landing Error (cm)	x, y (cm)	Landing Time (s)
1	6.1	$5.28, 3.05$	40.5
2	22.1	$14.21, 16.92$	42.3
3	24.1	$23.73, -4.18$	39.9
4	15.2	$-1.32, 15.14$	45.1
5	7.9	$-2.04, 7.63$	41.31
6	4.7	$0.87, 4.62$	41.48
7	15.7	$13.59, 7.85$	38.9
8	19.2	$-12.34, 14.71$	39.73
9	12.3	$11.56, -4.21$	43.67
10	16.4	$-14.2, -8.2$	41.7

Table 17: Cautious landing with a repeated landing approach.

BIBLIOGRAPHY

[1] *Cinematography and advertising with Drones.* https://flymotionus.com. Accessed: 2020-09-24.

[2] *Coverage of social and sports events with Drones.* https://strigroup.com/environments-sector/drone-services. Accessed: 2020-09-24.

[3] *Drone Delivery Market Map 2019.* https://www.droneii.com/project/the-drone-delivery-market-map. Accessed: 2020-09-24.

[4] *Courier and delivery services with Drones.* https://flytrex.com. Accessed: 2020-09-24.

[5] *Agriculture Drones.* https://www.postscapes.com/agriculture-robots/#drones. Accessed: 2020-09-24.

[6] S. Sankarasrinivasan, E. Balasubramanian, K. Karthik, U. Chandrasekar, and R. Gupta. "Health Monitoring of Civil Structures with Integrated UAV and Image Processing System". In: *Procedia Computer Science* 54 (2015), pp. 508 –515.

[7] *Drone emergencies and protection.* https://www.dronebydrone.com/en/drones-emergencies-protection.php. Accessed: 2020-09-24.

[8] *Drone solutions on search and rescue operations.* https://www.dronesolutions.gr/ereuna-diasosi.php. Accessed: 2020-09-24.

[9] *Surveillance Drones.* https://skilledflyer.com/surveillance-drones-forsale. Accessed: 2020-09-24.

[10] K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh. "Programming micro-aerial vehicle swarms with karma". In: *9th Conference on Embedded Networked Sensor Systems.* 2011.

[11] C. Pinciroli, A. Lee-Brown, and G. Beltrame. "Buzz: An Extensible Programming Language for Self-Organizing Heterogeneous Robot Swarms". In: *arXiv:1507.05946* (2015). arXiv: 1507.05946.

[12] M. Koutsoubelias and S. Lalis. "TeCoLa: A Programming Framework for Dynamic and Heterogeneous Robotic Teams". In: *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services.* 2016.

[13] *GPS accuracy.* https://www.gps.gov/systems/gps/performance/accuracy. Accessed: 2020-09-24.

[14] M. R. Hayajneh and A. R. E. Badawi. “Automatic UAV Wireless Charging over Solar Vehicle to Enable Frequent Flight Missions”. In: *Proceedings of the 2019 3rd International Conference on Automation, Control and Robots*. ICACR 2019. Prague, Czech Republic: Association for Computing Machinery, 2019, 44–49. ISBN: 9781450372886.

[15] *Drone landing into a hangar*. https://www.airscort.me. Accessed: 2020-09-24.

[16] J. Janousek and P. Marcon. “Precision landing options in unmanned aerial vehicles”. In: *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. 2018, pp. 58–60.

[17] *Ardupilot autopilot software suite*. https://ardupilot.org/ardupilot/. Accessed: 2020-09-24.

[18] S. Lin, M. Garratt, and A. Lambert. “Monocular vision-based real-time target recognition and tracking for autonomously landing an UAV in a cluttered shipboard”. In: *Autonomous Robots* 41.4 (2019), pp. 881–901.

[19] C. Patruno, M. Nitti, A. Petitti, E. Stella, and T. D’Orazio. “A Vision-Based Approach for Unmanned Aerial Vehicle Landing”. In: *Journal of Intelligent & Robotic Systems* 95.2 (2019), pp. 645–664.

[20] S. Yang, S. Scherer, and A. Zell. “An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle”. In: *Journal of Intelligent & Robotic Systems* 69.1 (2013), pp. 499–515.

[21] F. Cocchioni, V. Pierfelice, A. Benini, A. Mancini, E. Frontoni, P. Zingaretti, G. Ippoliti, and S. Longhi. “Unmanned Ground and Aerial Vehicles in extended range indoor and outdoor missions”. In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2014, pp. 374–382.

[22] R. Barták, A. Hraško, and D. Obdržálek. “A controller for autonomous landing of AR.Drone”. In: *The 26th Chinese Control and Decision Conference (2014 CCDC)*. 2014, pp. 329–334.

[23] L. Wei, W. J. Dong, and L. M. Yang. “A Vision-Based Attitude/-Position Estimation for the Automatic Landing of Unmanned Helicopter on Ship”. In: *Proceedings of the International Conference on Watermarking and Image Processing*. ICWIP 2017. Paris, France: Association for Computing Machinery, 2017, 1–5. ISBN: 9781450353076.

[24] M. Al-Sharman, B. Emran, M. Jaradat, H. Najjaran, R. Al-Husari, and Y. Zweiri. “Precision landing using an adaptive fuzzy multi-sensor data fusion architecture”. In: *Applied Soft Computing* 69 (2018), pp. 149–164.

[25] B. Herissé, T. Hamel, R. Mahony, and F. Russotto. “Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow”. In: *IEEE Transactions on Robotics* 28.1 (2012), pp. 77–89.

[26] G. Wang, Z. Liu, and X. Wang. “UAV Autonomous Landing Using Visual Servo Control Based on Aerostack”. In: *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*. CSAE 2019. Sanya, China: Association for Computing Machinery, 2019. ISBN: 9781450362948.

[27] D. Lee, T. Ryan, and H. J. Kim. “Autonomous landing of a VTOL UAV on a moving platform using image-based visual servoing”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 971–976.

[28] J. S. Wynn and T. W. McLain. “Visual Servoing with Feed-Forward for Precision Shipboard Landing of an Autonomous Multirotor”. In: *2019 American Control Conference (ACC)*. 2019, pp. 3928–3935.

[29] Z. Li, Y. Chen, H. Lu, H. Wu, and L. Cheng. “UAV Autonomous Landing Technology Based on AprilTags Vision Positioning Algorithm”. In: *2019 Chinese Control Conference (CCC)*. 2019, pp. 8148–8153.

[30] X. Liu, S. Zhang, T. Jiayi, and L. Longbin. “An Onboard Vision-Based System for Autonomous Landing of a Low-Cost Quadrotor on a Novel Landing Pad”. In: *Sensors (Basel)* 19.21 (2019), pp. 4703–4722.

[31] K. Wenzel, P. Rosset, and A. Zell. “Low-Cost Visual Tracking of a Landing Place and Hovering Flight Control with a Microcontroller”. In: *Journal of Intelligent & Robotic Systems* 57.1 (2010), pp. 297–311.

[32] K. Wenzel, A. Masselli, and A. Zell. “Automatic Take Off, Tracking and Landing of a Miniature UAV on a Moving Carrier Vehicle”. In: *Journal of Intelligent & Robotic Systems* 61.1 (2011), pp. 221–238.

[33] W. K. Holmes and J. W. Langelaan. “Autonomous Ship-board Landing using Monocular Vision”. In: 2016.

[34] L. Wang and X. Bai. “Quadrotor Autonomous Approaching and Landing on a Vessel Deck”. In: *Journal of Intelligent & Robotic Systems* 92.1 (2018), pp. 125–143.

[35] J. S. Wynn and T. W. McLain. “Visual Servoing for Multirotor Precision Landing in Daylight and After-Dark Conditions”. In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2019, pp. 1242–1248.

[36] *IRLock target tracking system*. https://irlock.com. Accessed: 2020-09-24.

[37] *Pixy vision sensor*. https://pixycam.com/pixy-cmucam5/. Accessed: 2020-09-24.

[38] *IRLock target tracking system components*. https://irlock.readme.io/docs. Accessed: 2020-09-24.

[39] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer. "Speeded Up Detection of Squared Fiducial Markers". In: *Image and Vision Computing* 76 (June 2018).

[40] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer. "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming". In: *Pattern Recognition* 51 (Oct. 2015).

[41] *ArUco library documantion*. https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit. Accessed: 2020-09-29.

[42] *OpenCV library*. https://opencv.org. Accessed: 2020-09-29.

[43] S. Suzuki and K. Abe. "Topological structural analysis of digitized binary images by border following". In: *Comput. Vis. Graph. Image Process.* 30 (1985), pp. 32–46.

[44] D. H. Douglas and T. K. Peucker. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographical: The International Journal for Geographic Information and Geovisualization.* 10 (1973), pp. 112–122.

[45] N. Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics.* 9 (1979), pp. 62–66.

[46] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer. "Fractal Markers: a new approach for long-range marker pose estimation under occlusion". In: *IEEE Access* PP (Nov. 2019), pp. 1–1.

[47] *Gazebo open-source 3D robotics simulator*. http://gazebosim.org. Accessed: 2020-09-24.

[48] *ArduPilot-Gazebo plugin repository*. https://github.com/khancyr/ardupilot_gazebo. Accessed: 2020-09-24.

[49] *Robot Operating System - ROS*. https://www.ros.org. Accessed: 2020-09-24.

[50] DroneKit. *Developer tools for drones*. https://dronekit.io/. Accessed: 2020-09-24.