2014 International Conference on Unmanned Aircraft Systems (ICUAS)
May 27-30, 2014. Orlando, FL, USA

Autonomous Navigation, Landing and Recharge of a Quadrotor using Artificial Vision

Francesco Cocchioni[1], Adriano Mancini and Sauro Longhi

Abstract— **In this paper a solution to UAV reduced endurance and autonomous flight is proposed. With a complete on-board solution, based on artificial vision, the developed system is able to autonomously take off, navigate and land, recharging its battery by using a dedicated landing platform, both in indoor and outdoor scenarios. The landing platform includes a passive centering system to correct the landing error of the UAV, with a novel design wich reduce cost and increase the safety (thanks to small and isolated electrical contacts) without invasive hardware changes on the drone. The developed vision algorithm provides a fast and accurate measurement of UAV position with respect to the landing platform using a visual target, but at the same time it is able to automatically switch to an estimation of position that is independent from the visual target. This aspect is used during navigation or when the tracking of the target fails, ensuring a continuous position measurement feed to the controllers. The developed control system manages all the different phases of a mission (motor turning on/off, take off, navigation, landing, ...) with low control error, ensuring a landing over the landing platform with an error that is lower than 5cm for both x and y axis. The developed software in ROS environment is modular and provides input/output interfaces to receive command, or send data.**

I. Introduction

Autonomous flight of an Unmanned Aerial Vehicles (UAVs) is an open problem not yet definitively solved in every scenario. In contrast with Unmanned Ground Vehicle (UGV), where the robot navigates on the ground, an UAV must constantly faces with the effect of gravity force and external disturbances, like wind gust. The fast and intrinsically unstable dynamics of a quadrotor requires at least two control layers to ensure a complete autonomous flight. The inner control layer ensures the stability of drone attitude, and is generally achieved using an Inertial Measurement Unit (IMU) installed on-board with a linear or non linear controller. The outer control layer provides the stabilization of drone position in the space, achieved with a similar controller and a sensor providing the drone position with respect to a reference system. Currently there is not a definitive solution for UAV localization that works in all the possible usage scenarios. Most drones are equipped with a Global Positioning System (GPS) used for autonomous flight in outdoor scenario. Indoor flight, or particularly accurate maneuver, requires a precise and fast measurement of drone position not provided by a GPS, due to the lack of signal in indoor area, the low accuracy of the estimated position, and the slow update rate of this sensor. This work focuses on the development of a system for autonomous vision based UAV navigation, landing and battery recharge. The system operates both in indoor and outdoor scenarios, using only hardware that can be installed on-board, so that no external sensor(s) or computational unit(s) are required. This work has been developed as a demonstrator for the European project R3-COP (Resilient Reasoning Robotic Co-operating Systems) [1], involving a collaboration with an Unmanned Ground Vehicle during the mission.

*All authors are with Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche, Ancona, ITALY.
[1]Contact author: francesco.cocchioni@gmail.com

II. System Configuration

The UAV used for this work is a quadrotor manufactured by Ascending Technology: the Asctec Pelican. The sophisticated design of the drone allows to install different kind of sensors and processing units up to a maximum suggested payload of 650g. At maximum payload the drone endurance is limited to 12-14min using a 6Ah LiPo battery. A 8Ah battery, which can be installed, significantly reduces the remaining payload for the final hardware configuration used in this work. The drone is sold ready to fly (using a remote radio control) with an integrated attitude controller. In outdoor environments the integrated GPS module can be used for a complete autonomous flight, that is take off, navigation and landing. The system is not able to autonomously perform these tasks in indoor or GPS-denied environments, and is strictly limited by the accuracy of GPS measurement, which can have an error up to 6 meter near the ground or in presence of canopy.

The on-board micro-controller, the Asctec Autopilot, includes two different ARM7 processors. The low level processor manages the data fusion of IMU measurements and the attitude controller by using a closed source firmware. The high level processor can be freely programmed by the user. The attitude controller is based on a PD controller that works at 1kHz using the attitude data available from the data fusion at the same rate. The motors and their controllers are specifically designed to work at high frequency.

The UAV has been customized with an on-board computer, the Asctec Mastermind based on the Intel Core2Duo SL9400 (dual-core 1.86 GHz) with 4GB of RAM. This platform provides the necessary computational power for all the algorithms developed to achieve a fully autonomous vision based flight. The camera selected to acquire images is the Matrix Vision BlueFox MLC200wC, with a resolution of 752x480 pixel and a maximum acquisition rate of 90Hz. The installed lens is a 2.8mm, that ensures a good field of view even when the drone is near the ground. The drone is also equipped with the Maxbotix XL-MaxSonar-EZ MB1320 ultrasonic sensor,

978-1-4799-2376-2/14/$31.00 ©2014 IEEE

with a resolution of 1cm and a maximum update rate of 10Hz. Both camera and ultrasonic are installed on the lower part of the drone, facing down toward the ground.

The drone is also equipped with a small protection for the propellers to increase safety in case of hard landings, and with a foot extension module (characterized by a rectangular design with four small windows) allowing the installation of Asctec Mastermind. Fig. 1 shows the drone used in this work with all the installed modules, except for the camera and the ultrasonic sensor.

The software has been developed in C++ as ROS (Robot Operating System) [2] package, using the well know OpenCV libraries [3] to process the images. The interface between the Asctec Autopilot and ROS to read sensors data and write control command(s) to the low level attitude controller (using ROS topics), is handled by the Asctec MAV Framework [4]. This ROS package overcomes some limitation of the serial communication speed (at the time of the development of this work) of the Asctec Pelican. The Asctec MAV Framework provides other useful features specifically written for the Asctec Pelican but in this work it was used only as a communication layer between ROS and Asctec Autopilot.

III. Landing Platform

The reduced endurance of UAVs does not allow a real continuous autonomous fly. Usually the replacement of an empty battery with a new one is performed by a human operator. This aspect reduces the set of the possible scenarios, like flying for a long time along a closed path for surveillance or inspection.

A specific landing platform has been developed to overcome this limitation. The developed platform provides not only a safe place to land, but also a system to recharge UAV battery extending the operating time. The battery recharge system has been chosen instead of a replacing one [5], [6], to reduce the development time, cost and complexity of the platform. It is based on a slip-ring installed both on the platform and on the drone, respectively connected to the battery charger and the UAV battery. A passive centering system has been developed to reduce the size of slip-rings, minimizing the risk related to large exposed electrical surface. The platform brings the drone once landed in a specific position, where the slip-rings are small in size, correcting the landing error. The gravity based passive centering system has been chosen instead of an active one (with actuators) for the above mentioned reasons. The landing platform has been initially modeled in a 3D CAD environment. The Asctec Pelican base has been modeled in a 1:1 scale, and has been used to validate the ideas of the various versions of the platform. The developed landing platform for the final system (shown in Fig. 2) has four upside down hollow cones used as passive centering system.

The idea behind our design is that the drone has to land, with four specifically designed feet, inside the cones. The gravity force will move the drone toward the center of the landing platform. The maximum error that the landing platform is able to correct is bounded to the radius of the four cones. The height of each cone is directly bounded (if the radius is fixed) to the slope of the internal sliding surface, increasing or decreasing the probability to slide. A larger radius for each cone, or a larger height, increases the performance of the centering system. The dimensions of cones directly influence the structure of the drone since its feet should slide on the inner cone surface. The radius of each cone is upper bounded to half the side of the square generated by the four feet. The feet height must be at least equal or greater than the cone height to avoid any contact between the landing surface and the drone base. With a 2D mathematical model it is possible to find the maximum landing error that a specific centering system (with fixed cone radius) is able to compensate. In general, once the UAV is landed, it may have an error with respect to the ideal position (the drone is perfectly centered with respect to the landing platform, with its foot exactly on the center of each cone) that can be at the same time both translational and rotational.

To simulate a generic landing, the UAV ideal foot position is multiplied by a roto-translation planar matrix (with a defined translation and rotation error), checking if each foot is still inside its respective cone. In this case the landing platform compensates the error, passively bringing the drone to the ideal landing position. Table I summarizes the obtained results with a centering system with cone of 10cm radius, showing with steps of one centimeter in the translation error

Fig. 1. Asctec Pelican in the original configuration.

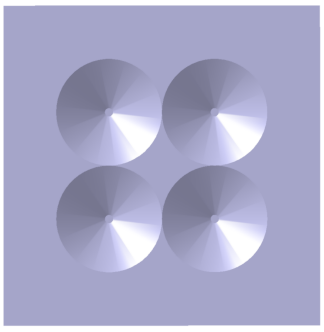

Fig. 2. 3D model of the developed landing platform.

(along both axis) the maximum allowable rotation error (the table can also be read on the opposite side). Fig. 3 shows a graphical representation of 5cm translation error (along both X and Y axis) and 10° rotation error with foot radius of 0.5cm. The original Asctec Pelican planar base, with foot extension (visible on the bottom of Fig. 1), does not fit with our centering system based. For this reason we designed a new landing foot. Instead of redesigning from scratch a new base, an additional feet have been developed such that they could be installed on the original Asctec foot extension, without any modification. Fig. 4 shows the developed landing foot, which is able to slide on the inner cone surface, without collision between landing platform and UAV base. The arch design increases the size of the UAV base (so that larger cone radius can be used, and larger landing error can be compensated), providing also a better impact dissipation in case of abrupt landing.

The external surface has been hollowed to remove unnecessary material reducing the weight of each landing foot. The inner part of the foot has been drilled to host the necessary cables to recharge the system. The upper part of the landing foot, between the two screw hole, has been deeply excavated not only to reduce the weight, but also to create a weak structural point where the foot can break in case of very hard crash dissipating the impact to the main UAV structure.

The performed analysis for the maximum landing error (along x and y) that can be compensated with four cones of 10cm radius demonstrated a particular affordable combination of the maximum translation and rotation landing error, that is 5cm along X and Y axis and 10 degree with respect to North, South, East, or West orientation. This is a good margin for the maximum landing error that should be achieved with the final navigation controller. A larger centering system could provide more margin, but at the same time it will require a larger landing platform and larger landing foot. The original Asctec Pelican base with foot extension, indeed, creates a square with side of approximately 15.5cm, which can be easily extended to 20cm with the non invasive landing foot shown in Fig. 4. The cones of 10cm radius also allow small cone height achieving a good sliding surface slope. With an height of 5cm the internal surface of each cone has a slope of 27 degree, which, in addition to a small landing foot contact surface (0.5cm of radius), allows a perfect UAV sliding for each possible landing position inside the cones. For these reasons the final landing foot additional height has been chosen to be the minimum one required (5cm). The final landing foot, with an overall weight of 7g is shown in Fig. 4. Fig. 5 shows the 3D model of the final landing foot installed on the original Asctec Pelican base, together with the final landing platform of 60x60x7cm. The slip-rings for the recharge system are installed on the bottom of two upside down cone and connected to a LiPo battery charger. The drone is equipped with two contacts installed on the bottom of two landing foot connected to its LiPo battery. The developed landing feet shown in Fig. 4 have been manufactured using a 3D printer, and integrated into our UAV shown in Fig. 1. The two slip rings connecting battery poles have been installed in the two front landing

TABLE I

MAXIMUM LANDING ERROR ALONG X AND Y AXIS FOR A LANDING PLATFORM WITH CONE OF 10CM RADIUS

X and Y Axis Error	Maximum Rotational Error
0cm	40°
1cm	33°
2cm	27°
3cm	22°
4cm	15°
5cm	10°
6cm	4°

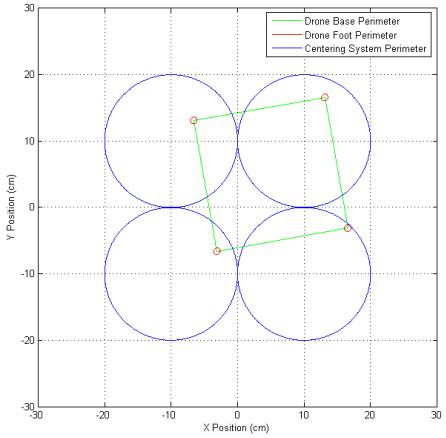

Fig. 3. Depiction of drone landing foot position in the centering system, in presence of a landing error of 5cm along X and Y axis and 10° along the reference direction.

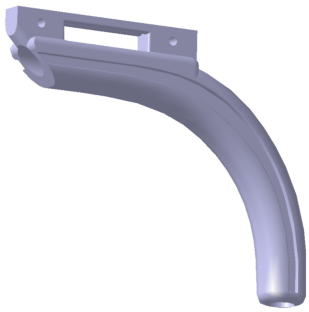

Fig. 4. 3D model of the developed landing foot.

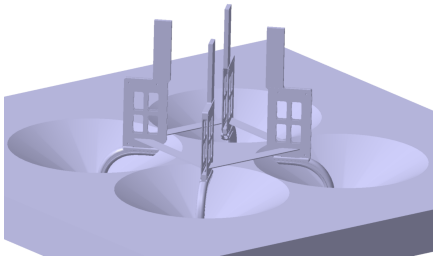

Fig. 5. 3D model of Asctec Pelican base with landing foot installed, once landed inside the landing platform.

foot, and a fuse has been introduced to avoid any short circuit. Fig. 6 shows the right side of the drone used in the demonstrator with all the described components. The cones in the real landing platform shown in Fig. 7 have been also produced using 3D printer connecting the drone to a LiPo battery charger with a maximum charging current of 20A, allowing a fast 3C recharge of our battery (6000mAh, max charge 3C). Only two of the four slip rings are connected to the battery charger.

IV. Landing Vision System

The designed landing platform sets the following constraint: the drone should land with a translation error with respect to the center of the landing platform which is $\leq$ 5cm along both x and y axis, and a maximum rotation error of 10 degree. To compensate the fast dynamics of an UAV and, at the same time, provide an accurate landing, the navigation controller must rely on accurate and fast measurement of drone position with respect to the landing platform. The measurements have to be available in every flight scenarios, indoor or outdoor, without a priori structured environment. For these reasons, a fast and accurate vision algorithm has been developed to compute the UAV position, with respect to a visual target installed on the landing platform.

The landing target has been designed to optimally fit the planar space available on the final landing platform, shown in Fig. 2. The target should be visible both at low and high altitudes. As shown in Fig. 7, the target has a redundant design. The small inner part is visible both from low and high altitudes, while the outer part provide more accuracy at high altitudes. This target allows to estimate the 6DOF of the drone. The larger visible circle provides the estimation of 3D position, roll and pitch angle of the drone, while the larger visible triangle provides the estimation of the yaw angle. The dimensions of each target element are summarized in Table II. The developed algorithm processes the images acquired by the Matrix Vision BlueFox MLC200wC camera, which is installed on the lower side of the drone facing down toward the ground. It is written in C++ as a dedicated ROS package, also using the well known OpenCV libraries. In order to achieve maximum performance every image is processed considering the minimum set of operations, using efficient implementation for each ones. Fig. 8 shows the different operations to estimate the drone 6DOF.

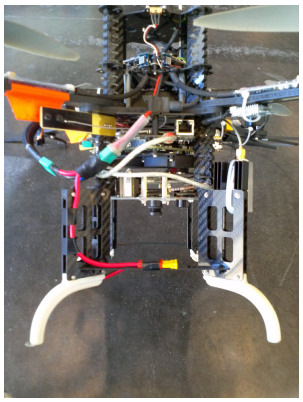

Fig. 6. UAV used for the final demonstrator, with all the hardware changes.

Images are acquired using a specifically developed ROS package which integrates the driver provided by the manufacturer inside the ROS environment, and performs the conversion of image data into ROS data type. The camera has been configured to acquire 8 bit gray-scale image (mono8) at the maximum speed available (90Hz), using automatic exposure time (with an upper bound of 11111us to ensure 90Hz in every situation) compensating different illuminations (this is a critical aspect when moving from indoor to outdoor area).

The threshold algorithm has been specifically developed for this work to be computational efficient, providing at the same time accurate binarization results. Since the drone has to land both in indoor and outdoor environments with different environmental conditions, the developed threshold algorithm is adaptive. It is based on the adaptive threshold algorithm developed by Wellner [7], in which the image is explored only one time row by row, calculating for each pixel the moving average of the previous S pixel explored. If a pixel is at least T percent darker than the moving average, than in the binary image it will be black, white otherwise. Denoting with p_n the intensity of the n-th pixel, and with q_n the average intensity of previous S pixel, the associated

TABLE II
Landing target elements dimensions.

Position	Ring	Triangle
Inner	R=3cm r=2cm	b=2.728cm l_1=l_2=2.027cm
Outer	R=27cm r=25cm	b=14cm l_1=l_2=10.63cm

Fig. 7. Landing platform, with landing target, built for the final system.

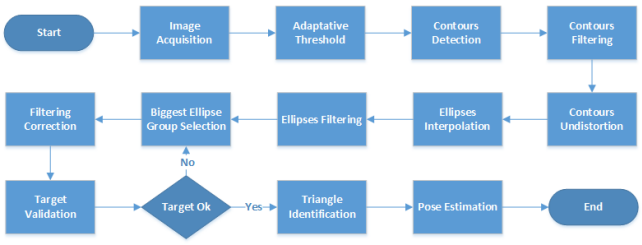

Fig. 8. Flow diagram of the vision algorithm to estimate the drone pose during takeoff and landing over the landing platform.

value to the n-th pixel in the binary image, called O is:

$$O(n) = \begin{cases} 0 & if\ p(n) < g(n) \cdot (1 - T) \\ 1 & otherwise \end{cases} \quad (1)$$

where 0 is a black pixel and 1 a white one. Since the algorithm is influenced by the direction of row exploration, the author suggests to explore row alternatively from left to right and from right to left, using also the information about the moving average of each pixel on the previous line. In the previous formulation $g(n)$ is substituted by $h(n)$, which is:

$$h(n) = \frac{g(n) + g(n - w)}{2} \quad (2)$$

where w is the number of pixel for each row (image width). The moving average for the n-th pixel of a row is replaced by the average between the n-th pixel moving average and the moving average of the pixel directly above in the previous line. To improve computational performance, the author suggests to use an iterative moving average formula instead of a exact one. The exact moving average calculation has been replaced in our algorithm with the approximation provided by the exponential moving average, which can be written as:

$$g(n) = \alpha \cdot p(n) + (1 - \alpha) \cdot g(n - 1) \quad (3)$$

where α is a forgetting factor between 0 and 1. In our case $\alpha = 1/S$. The Wellner algorithm so far described provides good binarization results but does not match with our constraints. The on-board Asctec Mastermind (Intel Core2Duo SL9400) performs the binarization within 8.971ms which could produce an overrun in the execution time considering the set of operations shown in Fig. 8.

Our requirement is to process the image provided by the camera (752x480 pixel) within 11.11ms (90Hz). Most of the time required to binarize an image using Wellner algorithm is spent to multiply and divide floating point numbers. The performance of the algorithm have been improved with prior bufferization of all the possible operations, known once fixed S and T. Since each pixel intensity varies from 0 to 255, the approximation of each calculation with the first decimal digit achieves excellent performance. This is done using online only sums or difference of integer number used as index (conveniently shifted to avoid negative indexing) for the buffers which store all the calculation done at program initialization. The computation time achieved with this solution is not subject to the value of S and T: it depends only on the size of the input image. The average required time to process an image using our algorithm is 1.997ms.

The original algorithm has been further improved increasing the quality of the binarized image. Equation (1) has been modified so that also the moving average of the last S_b pixel defined as black is taken into account, as:

$$O(n) = \begin{cases} 0 & if\ p(n) < g(n) \cdot (1 - T) \\ or\ if\ p(n) < j(n) \cdot (1 + T_b) \\ 1 & otherwise \end{cases} \quad (4)$$

where $j(n)$ is the exponential moving average for the last S_b pixel defined as black, while T_b is a value between 0 and 1. Pixels in the input image that are at least the T_b percent less dark than the average intensity of last S_b pixel defined as black, are classified as black on the binary image.

This improves the quality of the output image, reducing the incorrect binarization in case of area with an high number (greater than the averaging window) of adjacent black pixels; this change requires two more buffer, bringing to a little drawback in performance with an average execution time of 2.156ms. The binarized images with our adaptive threshold algorithm has a better performance in terms of computational time and results, with respect to OpenCV algorithms. The computational time is close to Otsu global threshold [8], which in our system takes on average 0.991ms, while regarding quality our algorithm provides better results in presence of different light conditions and shadows.

In the OpenCV adaptive threshold the chosen parameter directly defines the computational time required to process an image. In our solution it depends only on the image size. At the same time in OpenCV adaptive threshold the parameter used for image acquired at low altitude does not work well with image acquired at higher altitudes. Our solution with a fixed set of parameters for thresholding (T, S, T_b, S_b) correctly processes images taken at different altitudes with different light and shadows conditions. The computational performances of the OpenCV adaptive threshold are also worst, requiring on average in the faster implementation (with a simple mean over a window, kept small, of pixel) more than 6ms in our system.

The binarized image is then processed using Suzuki algorithm [9] searching for contours. All the contours are classified with a tree hierarchy to find the target and its different elements without using further time consuming operations. The algorithm is executed on average in 2.402ms. Considering the chosen pattern of our target, most of the contours can be ignored, like the ones which have not both a parent and a child contour, or the ones whose area is too small. After this preliminary filtering, the remaining contours are processed to remove distortions introduced by camera lens (2.8mm) using the camera matrix obtained with the camera calibration process proposed by Zhang [10] and Bouguet [11]. Removing the distortions on some specific contours, instead of processing the entire image, reduces the computational time required by this operation (to remove the distortions on entire image, our system takes on average 5.5ms using efficient matrix mapping method). Once the filtered contours are undistorted, the ellipse that interpolates each one is found. The prospective projection of a circle in the image plane is generally an ellipse.

All these operations require on our system on average 0.758ms. After that, all the ellipses are processed to find the ones associated to the target removing the outliers. A custom clustering algorithm has been developed to quickly detect ellipses whose centers are sufficiently close to be assumed concentric. The probability that in the image there is a group of concentric ellipse which is larger in number with respect to the one associated to target is very small. For this reason we assume that the largest group of concentric ellipses is

the one associated to target. The creations of all the possible cluster takes in our system on average 0.039ms. After a set of ellipse which are probably associated to the target is detected, a check ensure that no ellipses have been wrongly filtered in the previous operations. In that case the missing ellipses associated to target are added back. This operation has almost no computational cost, taking in our system on average 0.0006ms.

The last filtering phase takes into account the ratio between various real dimensions of the target (see Table II). The ratio remains almost the same on the plane image, and can be used to validate if the group of ellipses isolated from the others is really associated to the target, understanding if the target in his entirely or partially (inner or outer) is seen. If the check of the proportions fails (in the rare case in which the largest number of concentric ellipses visible are not associated to the target), the second largest cluster of concentric ellipses is taken into account checking for proportions, and so on. This operation is also no time consuming, taking on average 0.0002ms.

Once the ellipses associated to the target have been definitely isolated, it is possible to search inside target contours, using the hierarchy previously calculated, the one associated to the largest triangle visible. The triangle is identified looking for the contours which can be approximated with three point, using Douglas-Peucker algorithm [12]. This contours must also satisfy a check of the ratios calculated with its real dimensions, shown in Table II. This phase requires on average 0.118ms. The overall algorithm processes in the worst situation an image with an average time of 5.5ms, with the camera at an altitude that guarantees the full visibility of the target and background. Table III summarizes the computational time required by each module of the landing vision algorithm.

Our algorithm processes all the images provided by the camera at its full speed (90Hz); it could be able to process even further image per seconds, if a faster image acquisition system is available. Figures 9, 10, 11 show the robustness of our algorithm to external disturbance. Figures 9, 10 show the effectiveness of our adaptive threshold algorithm, which binarizes images acquired at different altitudes and exposed to different kind of light and shadows, using the same set of configuration parameter (which is not possible with OpenCV integrated adaptive threshold which required different sets of parameter for the two example images). Fig. 11 shows how the filtering algorithm correctly discriminates the contours associated to the target even in presence of some noisy element with the same shape and size of the target. It clearly shows the correction of the distortion introduced by the 2.8mm lens.

The 6DOF of the camera with respect to the landing platform are estimated from the largest visible circle and triangle. The estimation of drone pose (with the exception of yaw angle) is based on the work proposed by Chen [13]. A generic ellipse (which, in the image plane, is the perspective projection of a circle) can be expressed using the following conic equation:

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0 \qquad (5)$$

or in the quadratic form as:

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} A & B & D \\ B & C & E \\ D & E & F \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0 \qquad (6)$$

where f is the focal length of the camera. It is possible to write the image plane as $z = -f$. A number of lines going

TABLE III

AVERAGE COMPUTATION TIME REQUIRED TO PERFORM EACH PART OF THE LANDING VISION ALGORITHM.

Operation	Average Time	Standard Deviation
Adaptative Threshold	2.156ms	0.551ms
Contour Detection	2.402ms	0.296ms
Ellipses Interpolation	0.758ms	0.122ms
Ellipses Filtering	0.024ms	0.005ms
Filtering Correction	0.006ms	0.002ms
Target Validation	0.002ms	0.001ms
Triangle Identification	0.118ms	0.026ms
Total	5.467ms	0.551ms

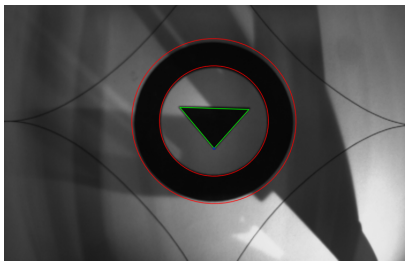

Fig. 9. Landing target detection at low altitude with external light/shadow disturbance.

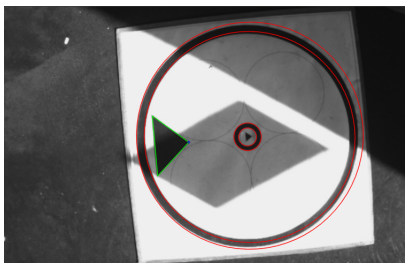

Fig. 10. Landing target detection at high altitude with external light/shadow disturbance.

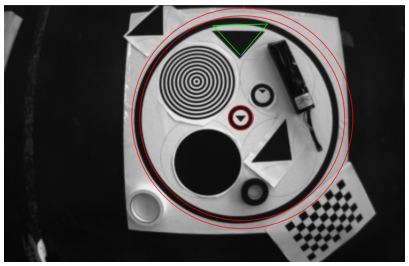

Fig. 11. Landing target detection in presence of similar geometric disturbance element.

through the optical center of the camera to the perimeter of the ellipse describes an oblique elliptical cone of equation:

$$P = k \begin{pmatrix} x & y & -f \end{pmatrix}^T \tag{7}$$

where k is a scale factor. From (6) and (7) it is possible to write:

$$P^T Q P = 0 \tag{8}$$

with

$$Q = \begin{pmatrix} A & B & -\frac{D}{f} \\ B & C & -\frac{E}{f} \\ -\frac{D}{f} & -\frac{E}{f} & \frac{F}{f^2} \end{pmatrix} \tag{9}$$

Denoting with λ_1, λ_2, λ_3 the eigenvalues of Q (with $\lambda_3 < 0 < \lambda_2 \leq \lambda_1$ for Kanatani [14]), and with v_1, v_2, v_3 the associated eigenvectors, it is possible to calculate the position of the center of the circle (C) and the unit normal vector (N) to the supporting plane (which contains the circle), with respect to camera coordinate system, as:

$$\begin{cases} z_0 = S_3 \frac{\lambda_2}{\sqrt{-\lambda_1 \lambda_3}} r \\ C = z_0 \left[S_2 \frac{\lambda_3}{\lambda_2} \sqrt{\frac{\lambda_1 - \lambda_2}{\lambda_1 - \lambda_3}} v_1 - S_1 \frac{\lambda_1}{\lambda_2} \sqrt{\frac{\lambda_2 - \lambda_3}{\lambda_1 - \lambda_3}} v_3 \right] \\ N = S_2 \sqrt{\frac{\lambda_1 - \lambda_2}{\lambda_1 - \lambda_3}} v_1 - S_1 \sqrt{\frac{\lambda_2 - \lambda_3}{\lambda_1 - \lambda_3}} v_3 \end{cases} \tag{10}$$

where S_1, S_2, S_3 are three undetermined signs, and r is the radius of the circle seen by the camera. Both C and N have multiple solutions due to the three undetermined signs. Since the center of the circle is in front of the camera, and choosing with N the normal vector to the surface seen by the camera, two of the three undetermined signs can be calculated solving:

$$\begin{cases} N^T \cdot (0\ 0\ 1)^T > 0 \\ C^T \cdot (0\ 0\ 1)^T < 0 \end{cases} \tag{11}$$

Equation (10) shows two different solutions (only one if $\lambda_1 = \lambda_2$) depending on the value of the unknown remaining sign; with the help of an embedded IMU it is possible to discriminate the last sign calculating the unique solution for (10). The roll and pitch angle provided by IMU are used to calculate the unit normal vector of the supporting plane, expressed in camera coordinate system (with the assumption that the supporting plane of the circle is parallel to the ground), and it is compared with the ones calculated by the vision algorithm. The last sign can be found looking for the vision algorithm normal vector that is closer to the normal vector calculated with the IMU. Once the unique solution for (10) has been found, it is possible to calculate from normal vector N the roll and pitch angle of drone. C expresses the position of the center of the circle with respect to the camera coordinate system. Since this position is not unique, depending on the roll and pitch angle of the camera, it is possible to calculate the position of the camera with respect to a world coordinate system defined on the landing platform.

Using the roll and pitch angle calculated with the largest visible circle (or the one provided by the IMU) and the yaw angle calculated with the largest visible triangle (finding the angle generated by its unique top vertex), it is possible to calculate a rotation matrix that bring the measurements from the camera coordinate system to the world reference system defined on the landing platform; the position measurements are then derived to calculate drone speed. The accuracy of the estimated pose has been verified with a series of different tests. Since a ground truth positioning system was not available the estimation of accuracy has been evaluated with static tests.

The camera has been placed in a fixed position with respect to the center of the target, using a laser pointing system and a distance measurement system. The estimated position has been compared with the real one calculated by hands. A wide set of different positions, both along x, y and z axis have been tested, and the results are summarized in Table IV. The results show that the error increases with the distance due to the loss of accuracy for each pixel in the landing target considering the low resolution of camera sensor, and the use of a wide angle lens (2.8mm). In our case this was not a problem, since to perform a landing the drone will get very close (less than 50cm) to the landing platform, where the position estimation accuracy is $\leq 0.5cm$. The accuracy of roll and pitch angle has been calculated using the angle provided by the IMU as ground truth. The yaw angle estimated by the IMU shows a large error of 20° over a single rotation of 90°. For these reasons the yaw angle that is estimated by the vision system has been manually validated. Fig. 12 shows a comparison of the roll and pitch angles estimated by the vision algorithm and IMU, while the drone oscillates over the target. Table V summarizes the average error for roll and pitch angle in the experiments, and also shows the average error of yaw angle.

TABLE IV

AVERAGE POSITION ESTIMATION ERROR, USING THE LANDING TARGET AT DIFFERENT HEIGHT.

Distance	X Axis Error	Y Axis Error	Z Axis Error
25cm	0.13cm	0.08cm	0.35cm
50cm	0.42cm	0.30cm	1.94cm
100cm	1.32cm	1.49cm	3.74cm
200cm	1.81cm	1.98cm	5.86cm

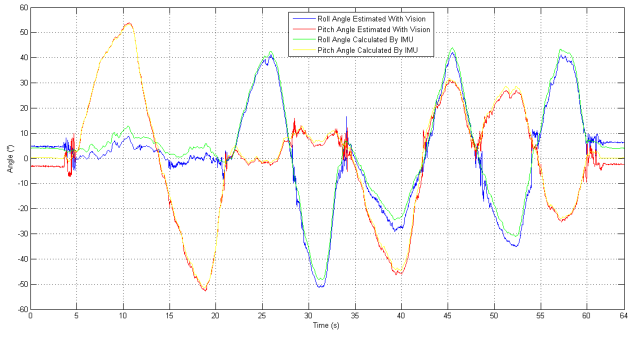

Fig. 12. Roll and pitch angles estimated by the landing vision system, compared to the IMU measurements.

V. Vision System for Navigation

The estimated position of the drone with respect to the world reference system defined on the landing platform can not be used for its navigation, since the landing vision algorithm requires that the visual target is always in the field of view of the camera.

An additional vision algorithm has been developed to estimate the position of the drone without using the landing target. This algorithm is based on the optical flow to estimate the odometry of the drone. A set of features are tracked between temporally close images estimating the movements of the camera. This vision algorithm has been developed in C++ using OpenCV library sharing the same ROS application as the landing vision algorithm. Fig. 13 shows the different operations performed by vision system for navigation. Since the navigation vision system uses different algorithms which require to entirely process each image we decided to resize (376x240) each image to reduce the computation time. This operation takes on average 1.7ms on our system. The resized image is then processed removing the distortions introduced by the lens. Since the entire image must be processed by the next algorithms, the distortion removal must be globally performed on the resized image, requiring in our system on average 1.45ms.

The best features to track are found in the previous image using the algorithm proposed by Shi [15], upper bounded to 50. This time consuming operation is not performed on the second to last image for every new one available, but it is done every fifth time, propagating in the meanwhile the feature found the first time from image to image using the optical flow algorithm, improving the efficiency of the software. If the system is not able to propagate some features from image to image, and their total number becomes lower than 20, the feature search is performed no matter the time passed since the last one performed.

The search for the best features to track in our system takes on average when performed 9.495ms. The found features are tracked in the following image using the pyramid implementation of Lucas-Kanade optical flow [16] algorithm, proposed by Bouguet [11], which provides for each found feature in the previous image its position on the new one; this operation requires on average 5.754ms. With the previous and current position for each feature it is then possible to calculate, for each couple, the movement vector that defines the movement sensed by the camera in the image plane for the specific feature.

Since the tracking algorithm may provides a wrong new position for some features (e.g., a similar group of pixel in the image), all vectors can not be considered valid to estimate the movement done by the camera. A filtering algorithm, based on the clustering described in the previous section, processes all the movement vectors and extrapolates the largest group of them which share approximately the same intensity and direction.

It is supposed that the tracking algorithm provides more correct tracking results rather than incorrect ones. All the resulting vectors from the filtering algorithm are then averaged defining a new single moving vector, with center in the origin of the image; the filtering takes on average 0.357ms. Fig. 15 shows the results of the filtering algorithm for the moving vectors obtained by the previous image shown in Fig. 14. The direction of each vector is defined with a circle. The red vectors marks the ones that are discarded by the filtering algorithm in favor of blue ones. The green vector shows the resulting one took into account to estimate camera movement. As well as in the landing vision algorithm, lot of effort has been spent to reduce the computation time required by the navigation vision algorithm. In contrast with the previous one, this algorithm requires to process the entire gray-scale image more times, increasing the required time for each operation. The 90Hz goal could not be reached in our system without loosing too much in accuracy. The reduction to half the original size of the image and the

TABLE V
AVERAGE ANGLE ESTIMATION ERROR USING THE LANDING TARGET.

Angle	Error
Roll	2.66°
Pitch	1.45°
Yaw	1.27°

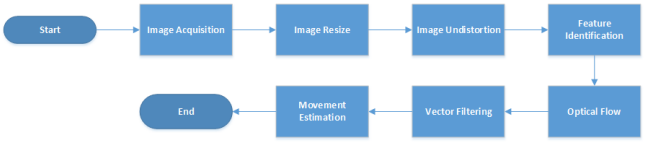

Fig. 13. Flow diagram of the vision algorithm used for drone position estimation during navigation.

Fig. 14. Image at time k for the optical flow algorithm.

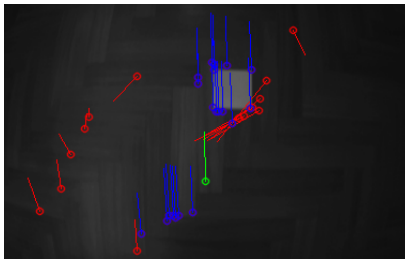

Fig. 15. Image at time k+1 for the optical flow algorithm, with moving vector filtering (red ones are discarded, blue ones are accepted, and green one is the average of accepted ones).

use of an efficient outliers moving vector filtering system, allows to process each image in less than 20ms, which means a position estimation performed at 50Hz. Table VI summarizes the computational time required by each module of the navigation algorithm.

The slowest task is the identification of best features to track. Without it (for example when the features are propagated from image to image) the system processes 90 images per second, but at least every five images new features must be found to ensure good accuracy and avoid to end with no more feature to track (the tracking algorithm is not always able to track all the feature in the new image). For these reasons a safe margin of 20ms per image has been taken into account, reducing to 50Hz the frequency of position estimation with the navigation vision algorithm. The resulting vector shown in Fig. 15 only defines in the image plane the movement sensed by the camera. This must be converted from pixel to a real distance unit by using the pin hole model and the distance of camera from the ground (in our case provided by the ultrasonic sensor). Since the camera is fixed on the drone frame, an oscillation of the drone during its normal operation produces in the image plane a movement vector which is not associated to a real camera translation movement. For this reason the vector provided by the filtering algorithm must be processed to remove the component introduced by the rotational movement of the camera. This is performed using the solution proposed by Rondon [17].

Denoting with OF_x and OF_y the velocity vectors in the image plane along x and y axis, calculated dividing the x-y component x_p and y_p of the moving vector for the time (t_i) between the two images, it is possible to write OF_x and OF_y as the sum of two components:

$$\begin{bmatrix} OF_x \\ OF_y \end{bmatrix} = T_{OF} + R_{OF} \tag{12}$$

where T_{OF} is the translational component, and R_{OF} the rotational one. T_{OF} can be written as

$$T_{OF} = \frac{1}{h} \begin{bmatrix} -f & 0 & x_p \\ 0 & -f & y_p \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{13}$$

where v_x, v_y, v_z are respectively the linear velocity of the camera along x, y and z axis, f is the focal length of the camera obtained with camera calibration process (in our case it will be half the value due to image resize), and h is the distance of the camera from the ground (obtained from the ultrasonic sensor). R_{OF} can be written as

$$R_{OF} = \begin{bmatrix} R_{OF_x} \\ R_{OF_y} \end{bmatrix} = \begin{bmatrix} \frac{x_p y_p}{f} & -(f + \frac{x_p^2}{f}) & y_p \\ (f + \frac{y_p^2}{f}) & -\frac{x_p y_p}{f} & -x_p \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{14}$$

where ω_x, ω_y, ω_z are respectively the angular velocity of the camera along x, y and z axis (obtained by the IMU). It is possible to calculate the linear velocity of the camera along x and y axis, denoting (13) and (14) into (12), as

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} -(OF_x - \frac{v_z x_p}{h} - R_{OF_x}) \frac{h}{f} \\ -(OF_y - \frac{v_z y_p}{h} - R_{OF_y}) \frac{h}{f} \end{bmatrix} \tag{15}$$

Fig. 16 shows a comparison between the position estimation performed removing the rotational component, and without removing it. The camera initially kept still on a position, has been tilted along its roll and pitch angle generating moving vectors not associated to a real translational movement. As shown in Fig. 16, without removing the rotational components the estimated position follows the rotation angle, with a peak of 30cm. The removal of the rotational component mitigates the effect of rotation, with the estimated position kept inside a boundary of $\pm$5cm.

It has not been possible to calculate the accuracy of the estimated position provided by the navigation vision algorithm, since there was no ground truth localization system. A work around test has been used to validate the estimated position. The drone has been commanded to autonomously fly in a closed path by using the control algorithm and the high-level logic discussed in the following of this paper, starting over the landing target which defines the origin of our reference system. Once the drone returns in sight of the landing target a comparison between the accurate value provided by the landing vision algorithm and the one provided by the navigation vision algorithm is performed, calculating the error for x and y axis. Table VII shows the results for three different autonomous flight done on three paths of different length, which are performed, respectively from the shortest to the longest, indoor, outdoor and both indoor and outdoor.

TABLE VI

AVERAGE COMPUTATION TIME REQUIRED TO PERFORM EACH PART OF THE NAVIGATION VISION ALGORITHM.

Operation	Average Time	Standard Deviation
Image Resize	1.710ms	0.342ms
Image Undistortion	1.455ms	0.321ms
Feature Identification	9.495ms	1.688ms
Optical Flow	5.754ms	1.414ms
Vector Filtering	0.357ms	0.514ms
Total	18.242ms	1.932ms

VI. CONTROL SYSTEM

The navigation controllers are three PID which control x, y and z position of the drone. A fourth PID stabilizes the yaw angle. The attitude PD controller is embedded inside the Asctec Autopilot. The dynamic model of the drone used to

TABLE VII

POSITION ESTIMATION ERROR USING THE NAVIGATION VISION ALGORITHM, AFTER CLOSED PATH OF DIFFERENT LENGTH.

Path Lenght	X Axis Error	Y Axis Error
13m	0.49cm	6.07cm
37m	38.8cm	21.6cm
63m	136cm	57.3cm

develop the control system is based on the work of Freddi [18] based on the Lagrangian approach. The six second order equations of the dynamics are the following:

$$\begin{cases} \ddot{x} = \frac{1}{m}(\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi))u_f \\ \ddot{y} = \frac{1}{m}(\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi))u_f \\ \ddot{z} = \frac{1}{m}(\cos(\phi)\cos(\theta)u_f) - g \\ \ddot{\phi} = \tilde{\tau}_\phi \\ \ddot{\theta} = \tilde{\tau}_\theta \\ \ddot{\psi} = \tilde{\tau}_\psi \end{cases} \tag{16}$$

were ϕ, θ and ψ are the roll, pitch, and yaw angle of the drone, m is its mass, and τ is the generalized moments vector; g is the gravitational force and u_f is the sums of all rotors thrusts. The low level attitude controller is already embedded, so there is no need to design it from scratch, since the one provided by Asctec work very well for our needs. For this reason the last three equations, describing the rotational dynamics, can be omitted. Using small angle approximation for the first two equations, the first three equations of (16) can be rewritten as:

$$\begin{cases} \ddot{x} = \frac{1}{m}(\phi\cos(\psi) + \theta\sin(\psi))u_f \\ \ddot{y} = \frac{1}{m}(\phi\sin(\psi) - \theta\cos(\psi))u_f \\ \ddot{z} = \frac{1}{m}(\cos(\phi)\cos(\theta)u_f) - g \end{cases} \tag{17}$$

Supposing a null yaw angle, the system in (17) can be written as:

$$\begin{cases} \ddot{x} = \frac{1}{m}\phi u_f \\ \ddot{y} = -\frac{1}{m}\theta u_f \\ \ddot{z} = \frac{1}{m}(\cos(\phi)\cos(\theta)u_f) - g \end{cases} \tag{18}$$

Equation (18) can then be rewritten removing the non-linearity as:

$$\begin{cases} \ddot{x} = u_x \\ \ddot{y} = u_y \\ \ddot{z} = u_z \end{cases} \tag{19}$$

where $u_x = \frac{1}{m}\phi u_f$, $u_y = -\frac{1}{m}\theta u_f$ and $u_z = \frac{1}{m}(\cos(\phi)\cos(\theta)u_f) - g$.

In order to define drone accelerations we used three PID controllers, one for each axis in (19). Then it is possible to calculate roll and pitch reference angle for the low level attitude controller, and the reference thrust, using the inversion of the dynamics as:

$$\phi_r = \frac{m}{u_{f_r}}u_x \tag{20}$$

$$\theta_r = -\frac{m}{u_{f_r}}u_y \tag{21}$$

$$u_{f_r} = \frac{u_z + g}{\cos(\phi)\cos(\theta)}m \tag{22}$$

The three PID controllers have been tuned initially in a simulation environment, then refined with a large set of experimental tests on the real system. Tables VIII and IX summarizes the error of position control calculated in different test scenario.

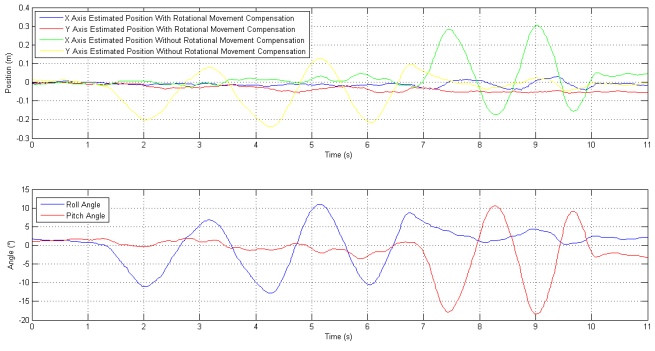

Fig. 16. Comparison between the positions estimated by the navigation vision algorithm, with and without rotational movement compensation, while the drone is hold in a specific position and rotated along roll and pith angle.

VII. Overall Software Design

The developed system composed by software and hardware modules. Software side, the vision algorithms, the control system, the camera interface, and the ultrasonic sensor interface, have been developed as different ROS packages ensuring better modularity, code reuse, and stability.

The two vision algorithms share the same package, and are implemented with two separated thread inside the same module. Since the drone needs a constant feedback about its position to fly, it is necessary that both the algorithms provide a measurement of position if the other is not able to perform its task. The ensure minimum delay during the switch between the two algorithm, the acquired image is sent to the same modules, which internally selects the best algorithm. The estimated position with the target is limited to the times it is in sight of the camera; on the other side the position provided by the optical flow, even if less accurate, can be used without any predefined visual landmark.

From the acquired images if it is not possible to find the landing target (and consequently estimate drone position) for more than 50ms (approximately 5 image in a raw with the target not visible) the vision algorithm automatically processes the image, and the following ones, with the optical

TABLE VIII

Control error using position estimated with target, indoor (left) and outdoor with 15-20km/h wind speed (right).

Axis	RMS
X	5.34cm
Y	3.65cm
Z	6.65cm

Axis	RMS
X	8.76cm
Y	11.46cm
Z	6.57cm

TABLE IX

Control error using position estimated with optical flow, indoor (left) and outdoor with 15-20km/h wind speed (right).

Axis	RMS
X	6.14cm
Y	5.83cm
Z	8.80cm

Axis	RMS
X	12.31cm
Y	14.18cm
Z	7.63cm

flow position estimation algorithm, initializing the position with the last one available from the landing target if available, from zero otherwise.

During optical flow position estimation, every 100ms the image acquired is processed in parallel to the navigation algorithm searching, using the landing vision algorithm, if the target is in sight of the camera; if the target is visible for at least 500ms the following images are processed using the landing vision system, deactivating the navigation one.

This approach ensures that a position measurement is constantly provided to the navigation control system, allowing the drone to take off over the target and navigate to a specific position, automatically managing the best available measurement to stabilize its position.

The vision package has been designed to provide an input/output interface for a high level navigation system, which assigns a mission and coordinates the UAV mission reading its state and commanding it for a specific task.

The developed ROS control system package embeds the discretized PID controller (using trapezoidal method for the integrative part and backward Euler method for the derivative part), safety check, ..., managing the different phases of flight (turn on/off motors, take off, landing, ...). This package provides the interface to receive position, turn on/off motors, and land command. Each position command is performed with a ramp trajectory. This trajectory is defined by a reference generator which ensures to reach a desired position (with the pseudo speed defined in the position command), reducing the overshoot of the system during movements. Take off maneuver can be performed (if the drone has been turn on with the specific command) regardless of the presence or not of the target, sending a positive z position command to the drone; drone rotors (that spins at minimum speed when on ground) will progressively spin faster till reaching the take off value, avoiding an abrupt change in the generated thrust.

The landing maneuver depends by the presence of the landing target in the field of view of the camera. If the target is not present or not detected, the drone lands on the ground using the position estimated by the optical flow. In this case it is sufficient to send a zero position command to the drone, so that the z PID controller brings it down to ground. If the drone has to land on the target to recharge its battery, a more precise maneuver is required to keep the target on sight.

Knowing camera parameters it is possible to calculate the area visible by the camera at a specific altitude, checking if the drone is moving too far away from the center of the landing platform, where even the smaller inner ring will not be visible (forcing a switch to optical flow measurements). During the landing procedure over the target, if the control error is less than a specific percentage of the maximum visible area the drone reduces its altitude till a fixed value over the target, otherwise it gains altitude improving its field of view.

Once reached the minimum altitude the drone waits to be sufficiently stable to land inside the passive centering error (landing error must be less than 5cm for both x and y axis). When this happens it performs a structured reduction of rotor thrust which is independent from altitude controller. Knowing the rate of decrease of the thrust, and the current drone height and thrust, it is possible to calculate from the third equation in (18) the required time to touch the ground. Knowing the speed of the drone provided by the vision algorithm, its current position and the time to reach ground, the system checks if it falls (calculating the landing error at ground) inside the centering system even in presence of position oscillation due to stabilization.

VIII. Conclusion and Future Works

This paper tries to solve two main challenges of quadrotors: flight endurance and autonomous indoor/outdoor navigation. The developed landing platform is able to entirely remove landing error, along x-y axis and yaw angle. Both the slope of the inner cones surface, and the reduced landing foot base area, allow a flawless centering of the drone with the respect to the landing platform, using only the gravity force, without any problem due too high friction between drone feet and the centering system.

The centering system brings the slip ring contacts installed on the bottom of drone feet (connected to its battery) over the landing platform ones (connected to the LiPo battery charger). The available planar space on the upper surface of the landing platform has shown to be suitable to include a visual target used to estimate drone pose with respect to the landing platform.

The developed vision algorithm correctly estimates the drone pose, at the maximum admissible acquisition speed of the camera when the target is detectable. If the target is not detectable the system switches to an alternative position estimation algorithm, based on optical flow. This switching approach ensures a continuous position estimation that works both in indoor and outdoor scenarios.

The pose estimation over the target demonstrated excellent computation performance, requiring only 5.5ms for a complete execution, providing the estimation of drone 6DOF with an accuracy at low altitudes lower than 1cm for the position and 3° for the angle estimation. Position estimation done using optical flow provided good results.

The optical flow estimated position suffers of incremental drifts due to the optical odometry involved. This error can be easily removed putting different landing target along drone path at specific waypoints. In the case of repetitive fly along the same path (for example during surveillance), it is possible to perform long fly with optical flow position estimation without the problem of growing estimation error (which will reset with targets at each waypoint), providing at the same time different landing platform where the drone can recharge its battery to perform its task indefinitely.

The control system based on PID controllers demonstrated a small control error, and it was able to manage with success all different tasks as take off, navigation and safe landing. Using the landing platform, both take off and landing have the maximum performance (thanks to the accurate measurements), granting a land inside the centering system for 95%

of times (so with a landing error lower than 5cm along both x and y axis, and 10° with respect to the orientation defined by the triangle); the system is able to take off and land in any position of the flight area using the measurements provided by the optical flow, avoiding the use of the landing platform if the recharge is not required. The control system worked fine also in presence of external disturbances, like wind (tested with gust up to 25km/h but with larger oscillations), allowing to take off, navigate and land without problem for the most part of the test (with strong wind landing over target shown reduced performance on its final phase, when the drone perform the final landing maneuver near the target).

It is possible to improve the performance of the control system when the drone is in sight of the target designing a controller that work at 90Hz (instead of 50Hz used in the final system), using all the estimated position provided by the vision algorithm.

The overall system passed the final revision of the European project R3-COP. The drone was able to autonomously fly both indoor and outdoor, recharging its battery in cooperation with a ground vehicle used as a mobile landing and recharge station. The difference in height introduced by the ground robot was compensated avoiding any problem to the UAV, thanks to the specific reference signal generator used in the control system; the UAV is able to approach to the UGV and landing on it using the ultrasonic sensor to calculate drone height, without any abrupt control effort and the consequent drone oscillation.

The first part of the video in [19] shows the developed drone that take-off from the visual target installed on the UGV that is moving through its waypoints; the UAV follow both the position and the orientation of the UGV. Finally it land over the UGV that is still moving. In the second part of the video the UAV takes-off from the target reaching a waypoint which is 2.70 meters at its left. In this case the UAV automatically switches to the optical flow measurements when target is no more detected. After the first waypoint is reached the UAV changes its position meters ahead.

REFERENCES

[1] "R3COP," 2013. [Online]. Available: http://www.r3-cop.eu

[2] "ROS," 2013. [Online]. Available: http://wiki.ros.org

[3] "OpenCV," 2013. [Online]. Available: http://opencv.org/

[4] "Asctec MAV Framework," 2013. [Online]. Available: http://wiki.ros.org/asctec_mav_framework

[5] T. Toksoz, J. Redding, M. Michini, B. Michini, J. P. How, M. Vavrina, and J. Vian, "Automated Battery Swap and Recharge to Enable Persistent UAV Missions," 2011. [Online]. Available: http://acl.mit.edu/papers/infotech-recharge-2011.pdf

[6] K. Swieringa, C. Hanson, J. Richardson, J. White, Z. Hasan, E. Qian, and A. Girard, "Autonomous battery swapping system for small-scale helicopters," 2010. [Online]. Available: http://diyhpl.us/~bryan/papers2/paperbot/bd5390c058ee5a7a3ca4f80865eeafaf.pdf

[7] P. D. Wellner, "Adaptive Thresholding For The DigitalDesk," Tech. Rep., 1993. [Online]. Available: http://xrce.fr/content/download/16283/117540/file/EPC-1993-110.pdf

[8] O. Nobuyuki, "A Threshold Selection Method From Gray-Level Histograms," 1979. [Online]. Available: http://dx.doi.org/10.1109/TSMC.1979.4310076

[9] S. Satoshi and A. Keiichi, "Topological Structural Analysis Of Digitized Binary Images By Border Following," 1985. [Online]. Available: http://dblp.uni-trier.de/db/journals/cvgip/cvgip30.html#SuzukiA85

[10] Z. Zhang, "A Flexible New Technique For Camera Calibration," 2000. [Online]. Available: http://research.microsoft.com/~zhang/Papers/TR98-71.pdf

[11] J.-Y. Bouguet, "Camera Calibration Toolbox For Matlab," 2008. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

[12] D. Douglas and T. Peucker, "Algorithms For The Reduction Of The Number Of Points Required To Represent A Digitized Line Or Its Caricature," 1973. [Online]. Available: http://utpjournals.metapress.com/content/FM576770U75U7727

[13] C. Qian, W. Haiyuan, and W. Toshikazu, "Camera calibration with two arbitrary coplanar circles," 2004. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24672-5_41

[14] K. Kanatani and W. Liu, "3D Interpretation Of Conics And Orthogonality," 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1049966083710430

[15] J. Shi and C. Tomasi, "Good Features To Track," 1994. [Online]. Available: http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf

[16] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique With An Application To Stereo Vision," 1981. [Online]. Available: http://www.ces.clemson.edu/~stb/klt/lucas_bruce_d_1981_1.pdf

[17] E. Rondon, I. Fantoni-Coichot, A. Sanchez, and G. Sanahuja, "Optical Flow-Based Controller For Reactive And Relative Navigation Dedicated To A Four Rotor Rotorcraft," 2009. [Online]. Available: http://dx.doi.org/10.1109/IROS.2009.5354483

[18] A. Freddi, A. Monteriu, and S. Longhi, "A Model-Based Fault Diagnosis System For A Mini-Quadrotor," 2009. [Online]. Available: http://www.issi.uz.zgora.pl/ACD_2009/program/Papers/58_ACD_2009.pdf

[19] "Video Of The Developed System," 2013. [Online]. Available: http://www.youtube.com/watch?v=Q5WWU84TGaI