Unmanned Systems, Vol. 10, No. 4 (2022) 329–342

DOI: 10.1142/S2301385022410011

UNMANNED SYSTEMS
https://www.worldscientific.com/worldscinet/us

Warehouse Inspection with an Autonomous Micro Air Vehicle

Jose Martinez-Carranza*, Leticia Oyuki Rojas-Perez

Instituto Nacional de Astrofísica
Óptica y Electrónica, Puebla, Mexico

In this work, we present an approach to address the problem of warehouse inspection using a MicroAir Vehicle (MAV) that performs autonomous flight by following a set of waypoints in a GPS-denied environment. During the navigation, a second onboard camera is used to observe the inspection area where it is expected to observe QR codes attached to packages placed on shelves. To this end, we use a metric monocular simultaneous localization And Mapping (SLAM) system to estimate the MAVs position in metres, avoiding the need for an external positioning system. The onboard computer executes the detection and decoding of QR codes and the SLAM system. The MAV can also communicate with a Ground Control Station (GCS) to display telemetry, the MAVs position, images from the inspection camera with detected QR codes, and a list of the found packages. This approach was used to compete in the indoors competition of the International Micro Air Vehicle Competition (IMAV) 2019, where we received the special award: Best Flight Performance.

Keywords: Warehouse inspection; micro air vehicles; visual SLAM.

US

1. Introduction

Warehouse inspection using Unmanned Aerial Systems or drones has received wide attention in the industry sector, seeking to automatize part of the logistic process in package management and also to reduce accidents involving human operators [1].

However, for drones to be deployed in a warehouse facility, some considerations have to be considered. For instance, vehicles must be equipped with sensors to detect package labels in the form of QR or Bar codes and a data management system to register the information [2]. The first approach is for a human pilot to control the drone to perform the inspection [2–5]. However, this may require a qualified pilot whose ability may be prone to failures if the operation becomes repetitive and tiresome. This calls for a solution where human intervention is as minimum as possible, meaning that the drone must be capable of some level of autonomy.

Motivated by the above, we present an approach to achieve autonomous flight to carry out warehouse inspection in this work. We propose to use a Micro Air Vehicle (MAV) equipped with a few sensors. It is true that a large drone may carry more sensors, which adds redundancy, and may also provide a longer flight time. However, in the event of crashing, material losses may be considerable, let alone the high risks of injuries to humans in the scene.

For the reasons mentioned before, we argue for the use of small drones such as MAVs with a size less than 40 cm in diameter, which may be easier to repair after a crash than with larger drones. However, onboard processing limits the number of sensors to be used since battery life will be drained in proportion to this number.

Therefore, to achieve autonomous navigation, we make use of a monocular metric SLAM system proposed in [6, 7]. This enables the drone to use a monocular camera only to estimate its position and orientation in metres, without requiring any external positioning system or for the environment to be populated with visual markers such as it is done for other works [8, 9].

For the inspection, we use a lightweight second camera whose images are processed to detect and decode QR codes used to identify the packages. Both, the SLAM and the QR

Received 8 February 2022; Revised 4 April 2022; Accepted 4 April 2022; Published 20 May 2022. This paper was recommended for publication in its revised form by Co-Editor-in-Chief, Ben M. Chen.
Email Address: *carranza@inaoep.mx*

Fig. 1. Warehouse inspection using drones was the main theme in the IMAV 2019 indoors competition. In this work we describe our lightweight solution consisting of an autonomous MAV with full onboard processing, capable of self-localization using a metric monocular SLAM, and QR detection and decoding using a side inspection camera.

detection and decoding, are executed by the an onboard computer attached to the MAV. Communication with a GCS is also possible to display telemetry, images from the inspection camera, the current MAVs position with respect to the flight plan, and the list of scanned packages.

We have evaluated the performance of our system in a test facility. In addition, we have also used this approach to compete in the indoors competition of the International Micro Air competition (IMAV) 2019, see Fig. 1. Our performance in the warehouse inspection task of the competition granted us the special award: *Best Flight Performance*. In sum, the contributions of this work are outlined as follows:

(1) We present a hardware/software system that enables a MAV to perform autonomous flight for warehouse inspection with full onboard processing without depending upon external sensors.
(2) We present a navigation system that uses a metric monocular SLAM system to estimate the MAVs position, which enables to fly a trajectory given as waypoints with coordinates in metres.
(3) We present an efficient QR detection method that uses a second onboard light weight camera, which also operates fully onboard.

To present our approach, this work has been organized as follows. Section 2 discusses relevant related work. Section 3 describes the main system components of our approach. Section 4 presents our experimental framework, including a description of our participation in the IMAV 2019; finally, Sec. 5 outlines our conclusions.

2. Related Work

This section describes the most representative works related to warehouse inspection. Some vision-based solutions for inspection use convolutional neural networks (CNNs) to detect the location of pallets [10], or damage to racks [11]. Likewise, in [12] the authors employ a CNN to recognize alphanumeric characters and barcodes. However, these solutions do not perform real-time scanning, as they use information from videos captured previously with a drone. In contrast, in [13] the authors present a blockchain-based system to receive the inventory data recorded from the detection of RFID tags in the warehouse with a drone, which are validated and sent to their system to real-time query. However, these solutions depend on the pilot's ability to avoid collisions.

On the other hand, some works use localization systems to perform inspection autonomously, using tags or markers in the environment to obtain the drone's position in the facility. For example, a more straightforward approach uses 2D barcodes detected with an IR camera to estimate the drone's position in the warehouse [14]. Other works use a CNN to detect 4 QR codes placed on the ground to localize the drone in an area of 4×4 m [9] or use WhyCon markers on each product box to generate a flight path through the inventory [8]. Similarly, work on [15] uses a bottom camera to detect lines that indicate the direction, which is processed in the Ground Control Station (GCS) to calculate the drone's position.

Other works present the combination of two robotic platforms. For example, in [16], authors describe an autonomous robotic system consisting of an underground vehicle (UGV), which uses a LIDAR-based SLAM to localize itself and determine the pallet's location in 2D space. In addition, this UGV has infrared (IR) markers on its top, which are observed with an IR camera placed on the bottom of the drone. In this way, the drone can use the position of the UGV to locate itself. Furthermore, the drone simultaneously executes barcode detection, with which it generates a code map to calculate the drone's position and perform the flight path. Similarly, the work on [17] uses a UGV to deliver the drone to the desired rack. The UGV uses LIDAR to perform the location between the shelves, and the drone also uses a lower camera to follow the UGV, using it as a reference for the location. The drone scans the barcodes and transmits them to the GCS during this process.

Another approach is presented in [18], where the authors build an initial map by flying the drone manually before the autonomous flight and use a LIDAR to perform SLAM and for package scanning, they equip the drone with a camera to recognize AprilTags and an RFID reader. On the other hand, in [19] the authors combine a LIDAR and IMU to perform SLAM and three onboard cameras. The first camera (forward-facing) to recognize AprilTag markers, the second camera (upward-facing) to perform visual SLAM on the ceiling, and the third camera (downward-facing) to perform lane recognition. This proposal seems quite robust; however, the solution requires information from multiple sensors; consequently, this approach employs a large drone, measuring 74 $\times$ 71 cm in the horizontal plane and weighing 3.2 kg.

3. Proposed System

In this section, we present the components of our system. We begin by describing the visual SLAM system used to localize the MAV in the environment where it is assumed that no GPS or external positioning systems exit. Next, we describe the hardware regarding the drone, onboard computer and inspection camera, the software system based on the Robotic Operating System (ROS) [20], and finally, our outer controller using the pose estimates provided by the visual SLAM system.

3.1. *Metric monocular SLAM*

For MAV localization, we use the metric monocular SLAM method proposed in [6, 7]. This method uses ORB-SLAM in its RGB-D version [21]. RGB-D ORB-SLAM requires chromatic and depth images to estimate the camera pose with metric (in metres); however, our MAV uses the Bebop's onboard camera as a unique camera to perform SLAM, this is, there is no RGB-D or stereo camera that can provide a depth image. Instead, the metric monocular SLAM method exploits the fact that the Bebop's camera can change its viewing angle in the pitch angle with respect to the MAVs body frame. This angle can take values from -83° to 20° and when set to -83°, the camera view is almost perpendicular to the ground. In addition, the MAVs altimeter is also used to provide height measurements of the MAV with respect to the ground.

Figure 2 illustrates our approach coupled to RGB-D ORB-SLAM, which expects image pairs of chromatic and depth images. With our proposed synthetic depth image generation method, we can provide a chromatic image obtained from Bebop's camera and its corresponding depth image to RGB-D SLAM. This makes it possible to obtain pose estimates for translation (t) and orientation (quaternion q) that can be used to localize the MAV, also used by the controller to command the MAV to navigate towards the waypoints in the flight plan.

For the sake of clarity, we summarize the required steps to compute a synthetic depth image given a chromatic image captured by the Bebop's onboard camera. Under the assumption that the ground is planar, Fig. 3 shows the geometric configuration used by the metric monocular SLAM method to compute a synthetic depth image with respect to the camera's coordinate system; this figure depicts the camera's geometry seen from a side view. In this configuration, we use the left-hand convention for the camera's coordinate system, indicated with purple lines in the same figure (X_c, Y_c, Z_c).

To compute a depth value for each pixel of the camera image, let us assume that the camera has been set to an angle θ (highlighted in orange in Fig. 3). From the camera's center of projection (origin of the camera's coordinate system), we can draw a vector parallel to the gravity vector and perpendicular to the ground. Assuming the ground has no inclination, this parallel vector can be calculated by knowing its magnitude a (which can be measured with the MAVs altimeter) and the complementary angle $\alpha = -(90 + \theta)$, depicted in yellow in the same figure. This vector is the normal vector $\mathbf{n}_p$ of the ground:

$$\mathbf{n}_p = \begin{bmatrix} 0 \\ a \sin\alpha \\ a \cos\alpha \end{bmatrix}. \tag{1}$$

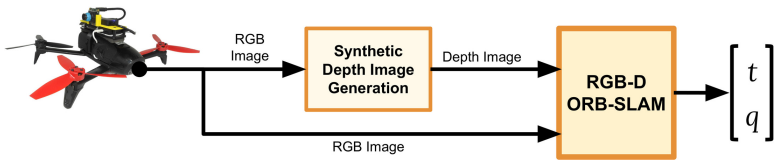

Fig. 2. General overview of our metric RGB-D SLAM approach based on RGB-D ORB-SLAM [21], which uses the chromatic image from the Bebop's camera and the synthetic depth image generated with our method using the altimeter and the planar ground assumption. ORB-SLAM returns a translation vector t and a quaternion q as estimates of the camera pose, which are used by the controller to command the MAV towards each waypoint in the flight plan.

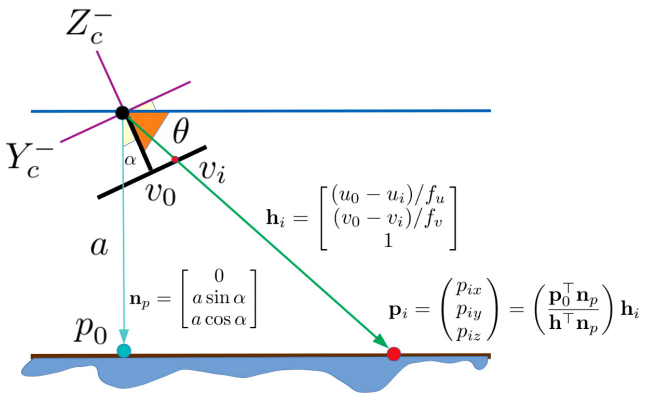

Fig. 3. Geometric configuration to compute the synthetic depth image using the MAVs altitude a, the camera's angle θ for each pixel (u_i, v_i), whose depth will correspond to p_{iz} in the camera's coordinate system (X_c, Y_c, Z_c).

Furthermore, a point on the ground plane is equivalent to $\mathbf{p}_0 = \mathbf{n}_p$.

If we know the normal and a point on the ground plane, we can calculate the intersection of a ray $\mathbf{h}_i$ with the plane, departing from the center of projection and crossing the camera plane in the pixel coordinates (u_i, v_i). To calculate $\mathbf{h}_i$, we use the optical center (u_0, v_0) and the focal length (f_u, f_v) as follows:

$$\mathbf{h}_i = \begin{bmatrix} (u_0 - u_i)/f_u \\ (v_0 - v_i)/f_v \\ 1 \end{bmatrix}. \tag{2}$$

Using the line-plane intersection equations, we can estimate the intersecting point $\mathbf{p}_i$ as follows:

$$\mathbf{p}_i = \begin{pmatrix} p_{ix} \\ p_{iy} \\ p_{iz} \end{pmatrix} = \left(\frac{\mathbf{p}_0^\top \mathbf{n}_p}{\mathbf{h}^\top \mathbf{n}_p}\right)\mathbf{h}_i. \tag{3}$$

Thus, p_{iz} is the corresponding depth value for the pixel (u_i, v_i).

If the camera image has radial distortion, (u_i, v_i) can be replaced by its corresponding undistorted image coordinate [22]. We mention this because ORB-SLAM does not expect undistorted images; if distortion exists, it computes the undistortion only for those image points to be used.

With the method described above, for a given camera image, a synthetic depth image can be computed with the caveat that the quality of the depth image depends upon the precision of the MAVs altimeter and how close the planar ground assumption holds, which we deem reasonable for warehouses where usually the floor is regular with no inclinations. An evaluation of ORB-SLAM using this method has been conducted in [23], showing a percentage error with respect to of 1.5% for a trajectory of 13 m.

As a final point, in this work, the complementary angle is -7°, resulting in a view almost parallel to the ground. However, this small angle may produce a slight rotation affecting the camera pose and orientation (quaternion) $(\mathbf{t}, \mathbf{q})$ estimated with ORB-SLAM. This is easily corrected by rotating these estimates in the opposite angle α and over the X-axis:

$$\mathbf{R}_X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \tag{4}$$

$$\mathbf{t}_f = \mathbf{R}_X^\top \mathbf{t}, \tag{5}$$

$$\mathbf{R}_f = \mathbf{R}_X^\top R(\mathbf{q}), \tag{6}$$

$$\mathbf{q}_f = q(\mathbf{R}_f), \tag{7}$$

where $R(\cdot)$ and $q(\cdot)$ are conversions to rotation matrix and quaternion, respectively.

3.2. *Hardware and software*

To carry out the warehouse inspection, we use the Bebop 2.0 Power Edition (referred to as Bebop from now onward), the Intel Compute Stick as the host computer and the uEye as a side inspection camera, as seen in Fig. 4 this vehicle has a diagonal diameter of 0.44 m and weighs 700 g.

The Bebop from the Parrot Company offers an SDK that allows the communication to receive stream, pilot, save and download photos and videos, and send and playback flight plans on autopilot. The Bebop has an altimeter that combines readings from an ultrasonic sensor and a barometer. It also uses a Visual-optical Positioning System (VPS) to maintain hovering, an Inertial Measurement Unit (IMU) that includes a three-axis gyroscope, a three-axis accelerometer, and a three-axis magnetometer, and finally, a 14-megapixel wide-angle camera with 3-axis digital image stabilization system.

Although the Bebop has a dual-core Cortex 9 CPU dedicated to the vehicle's operation, consequently, it cannot run

Fig. 4. Side view of the Bebop carrying the Intel Computer Stick m3 and uEye Camera.

custom onboard programs. Therefore, we use the Intel Compute Stick m3 as a host computer, which connects via USB-ethernet to the Bebop to receive streams at a resolution of 640×368 pixels at 30 fps and send flight commands.

The host computer (Intel Compute Stick m3) has a Core M3-6Y30 processor with 4 GB RAM and 64 GB eMMC storage, on which we installed the operating system Ubuntu 16.04 LTS and ROS Kinetic Kame. It also has Wifi-AC and Bluetooth wireless connectivity and is powered by a voltage of 5 V at 3.1 A. To connect the Bebop and the inspection camera (uEye), we incorporate a 3.0 hub because the Intel Compute Stick has only one USB 3.0 port.

For the inspection, we used the uEye industrial camera model UI-3481LE-M-GL, manufactured by IDS Imaging Development Systems GmbH. The uEye is equipped with a 4.92 megapixel CMOS Mono sensor and offers a maximum resolution of 2560×1920 pixels at 15.2 fps. However, we configured the uEye to obtain a resolution of 900×400 at 93 fps through the development software.

To carry out our experiments, we used an Alienware 15 r3 Laptop as GCS, see Fig. 5 and we used the local wireless network of the Bebop to communicate the Bebop, the Intel Compute Stick, and the GCS. Also, we configured the Intel Compute Stick in the ROS environment as ROS Master and the GCS as ROS Client to receive the position of the Bebop, the display of the image of the inspection camera (uEye) and the information of the identified and decoded packages. In addition, this communication system allows enabling or cancelling the autonomous flight and piloting the Bebop manually from the GCS in case of risk of collision.

Our approach uses ROS [20] as a communication base, with which it is possible to run on the central computer (Intel Compute Stick) four independent nodes oriented to the following tasks: (1) Bebop Autonomy, driver that enables the transmission and reception of information between ROS and Bebop; (2) Metric RGB-D SLAM, which receives the image from the bebop camera at a resolution of 640×368 at 30 fps; (3) waypoint-based navigation control, which receives the set of waypoints from the GCS, it is worth mentioning that we send these waypoints before executing the inspection; (4) uEye Camera & QR Decoder, in this node the uEye is linked to the Intel Compute Stick via USB to acquire the image at a resolution of 900×400 at 93 fps, and we used Zbar Library [24] to detect and decode the QR code, then the information of the packages found and the image of the uEye at low resolution is published. Figure 6 shows an overview diagram of the ROS nodes implemented in our system. This diagram indicates which nodes run on the onboard computer, the Intel Compute Stick (orange boxes), and which run on the GCS (purple box). The green arrows indicate the data published by the nodes. Red arrows indicate data consumed by other nodes.

3.3. *Outer controller*

The Bebop vehicle offers a Software Development Kit (SDK) to communicate the vehicle with a computer program that sends control signals to the Bebop's attitude inner controller. We implemented a proportional controller to control the vehicle to fly from one waypoint to another, following the imaginary line connecting these two waypoints.

For the inspection, we want the vehicle to fly forward towards the inspection area and once there, the vehicle has to fly upwards and downwards, moving forward between these transitions. This strategy seeks two things: (1) allow visual features in SLAM to be persistent; this is, if the drone

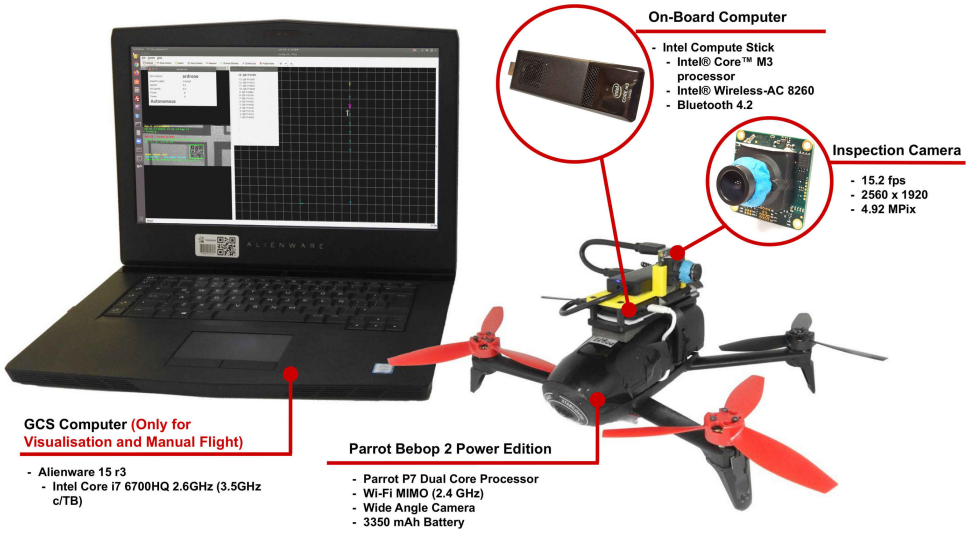

Fig. 5. GCS and the Bebop with the intel Compute Stick and the uEye camera.

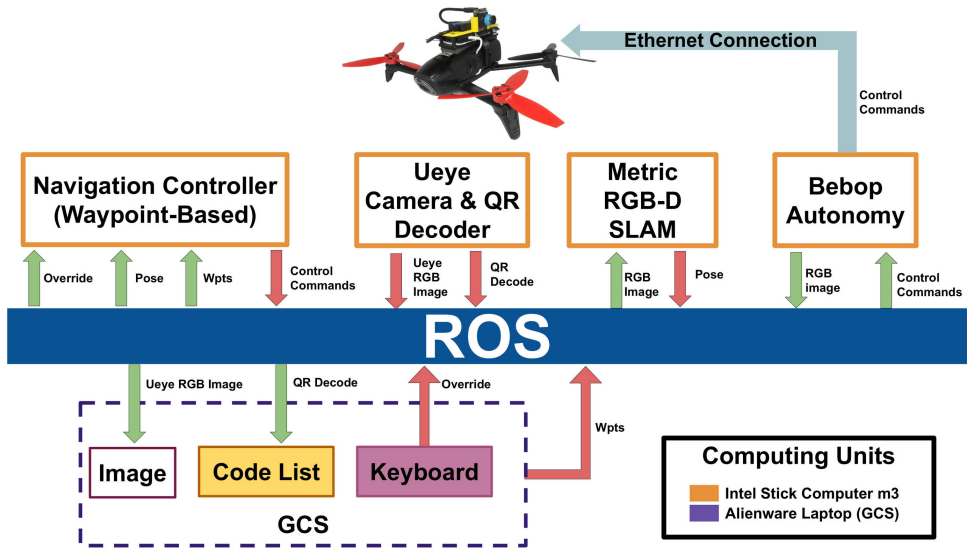

Fig. 6. (Color online) Communication system based on the ROS. Nodes running on the Intel Compute Stick are marked in orange. Nodes running on the GCS are marked in purple. Green arrows indicate the messages consumed by the nodes, whereas red arrows indicate the messages published by the nodes. The GCS is used for visualization only.

goes up or down, given that the camera is looking downwards (at $-83^{\top}$, visual features will be tracked for a longer period of time, thus enabling a map refinement of their corresponding 3D points, which benefits the camera pose estimation; (2) the resolution of the inspection camera larger in width than in height, a vertical inspection with upwards/downwards motion will increase the possibilities of finding QR codes; in contrast, horizontal flights at different highs may miss some codes, especially if the controller does not maintain the height in a stable manner.

From the strategy described before, the MAV flies on a horizontal plane when flying forwards or vertical motions when going up or down. Thus, we can dismiss the height coordinate when controlling the forward motion and use it only when controlling changes in height.

For the next equations, we assume that the pose estimates returned by ORB-SLAM are converted to the right hand coordinate system. Thus, given the translation and orientation of the camera pose $(\mathbf{t}, \mathbf{q})$ and the corresponding rotation matrix $R(\mathbf{q})$, we define a vector to represent the orientation of the MAV as

$$\mathbf{d} = R(\mathbf{q}) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \tag{8}$$

Thus, suppose we have two waypoints, a start waypoint $\mathbf{w}_s$ and the goal waypoint $\mathbf{w}_g$, which defines the imaginary line $\mathbf{l} = \mathbf{w}_g - \mathbf{w}_s$ to be traversed by the MAV; assume that $\mathbf{R}_\mathbf{l}$ is its corresponding rotation matrix.

Then we compute the MAVs position relative to $\mathbf{w}_s$:

$$\mathbf{r} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} = \mathbf{R}_\mathbf{l}^{\top}(\mathbf{t} - \mathbf{w}_s). \tag{9}$$

Thus, we can compute the control signals for the angle rates of the MAV with respect to its body frame, which are pitch s_θ (moving forward/backwards), roll s_ϕ (lateral motion), altitude s_h (elevation), and yaw s_ψ (rotation to the left/right), using proportional controllers with gains K_{p_θ}, $K_{p_\phi}, K_{p_h}, K_{p_\psi}$ as set by Eqs. (10)–(14).

To explain these controllers, we should remember that $\mathbf{r}$ is relative to the departing waypoint $\mathbf{w}_s$. Thus, for s_θ, we use the magnitude of the imaginary line $\mathbf{l}$ as reference to compare with r_x, which will increase as the MAV moves forwards. For s_ϕ, we expect r_y to be zero if the MAV flies centered over the imaginary line $\mathbf{l}$, thus the reference is the zero value. For s_ψ, we calculate the angle between $\mathbf{l}$ and the MAVs orientation vector $\mathbf{d}$. Ideally, this angle should approximate to zero as the MAVs orientation rotates to align its direction with $\mathbf{l}$; hence, the reference is zero. However, to know the sign of the angle, this is, whether the MAV is rotating towards $\mathbf{l}$ from the right or the left, we calculate the dot product $\mathbf{n}$ between $\mathbf{l}$ and $\mathbf{d}$. Thus, the sign of the n_z component of $\mathbf{n}$ will determine the sign of the angle, see Eqs. (13) and (15). For s_h, the goal waypoint w_{gz} of w_g is used as height reference.

$$s_\theta = K_{p_\theta}(\|\mathbf{l}\| - r_x), \tag{10}$$

$$s_\phi = K_{p_\phi}(-r_y), \tag{11}$$

$$s_h = K_{p_h}(w_{gz} - r_z), \tag{12}$$

$$\mathbf{n} = \mathbf{l} \times \mathbf{d}, \tag{13}$$

$$s_\psi = K_{p_\psi} \text{sign}(\mathbf{n}) a \cos\left(\frac{\mathbf{l} \cdot \mathbf{d}}{\|\mathbf{l}\|\|\mathbf{d}\|}\right), \tag{14}$$

where sign is defined as

$$\text{sign}(\mathbf{n}) = \begin{cases} 1: & n_z \geq 0, \\ -1: & n_z < 0, \end{cases} \tag{15}$$

and $\mathbf{w}_g = [w_{gx}, w_{gy}, w_{gz}]^\top$ and $\mathbf{n} = [n_x, n_y, n_z]^\top$.

4. Experiments and Results

In this section, we present the experimental framework to evaluate our approach for warehouse inspection using an autonomous MAV. First, we present an evaluation carried out in a test facility. Then, we discuss the performance of our approach in the indoors competition of the IMAV 2019. A video summarizing our results can be watched at https://youtu.be/vyOiIXkq9KA.

Table 1 summarizes the configurations for the two cameras on board the drone, including the camera angles with respect to the principal axis (right-hand coordinate system), image resolutions and frequency operation. For our visual SLAM method, described in Sec. 3.1, we use the Bebop's camera with a pitch angle set to $-83°$. The inspection camera (uEye) is placed at $-90°/90°$ in the yaw angle (depending on which side the inspection area is), thus enabling the inspection during a forward flight. The Bebop's camera captures images at a frequency of 30 Hz. Note that the SLAM operates at 12 Hz using 2000 visual features and running on the Intel Compute Stick. This frequency is enough to achieve a flight speed of 0.25 m/s.

The inspection camera is set to work with an image resolution of 900×400 pixels, capturing images at a frequency of 93 Hz on the Intel Compute Stick. Although the QR detector and decoder run at the much slower frequency of 9 Hz, the uEye is programmed to maintain a small image queue; thus, when the decoder requests an image from the uEye, it will be working with images showing the current inspection area with the advantage that, due to the high frequency, these images will have no blur. This is useful, especially when this camera is not gyro-stabilized; blur may occur for conventional cameras (30 Hz or less) due to vibrations of the MAV during the flight.

For our experiments and during the competition and for safety reasons, the pilot only intervenes to activate the take-off and landing manoeuvres already available in the Bebop's SDK, although it can cancel the autonomous flight at any time to take over and pilot the MAV manually. The flight trajectory is loaded beforehand, and it is defined as a list of waypoints with coordinates in metres. This list can be easily created with our GCS user interface or written manually in the settings file. The maximum MAVs flight is also set in the settings file before taking off.

4.1. *Experiments in test facility*

In preparation for the competition, our approach was tested in a facility resembling an industrial warehouse, see Fig. 7. We were not able to place any racks at it, but the facility had a narrow corridor with a wall where we could lay out QR codes to test our inspection strategy. The size of the QR codes is also indicated in the same figure. Some columns stand out from the walls, leaving an entrance and navigation gap of 1.45 m. We deemed this narrow space to be a good test scenario to evaluate our autonomous navigation controller. In particular because in the competition arena, two racks were placed in parallel, thus forming a corridor where the MAV was also expected to navigate to perform the inspection.

In our approach, the MAV performs forward navigation, carrying the inspection with the inspection camera set to $90°$ in yaw. We argue that this also helps the 3D mapping since visual features will tend to map the corridor's floor rather than the walls if the MAV is facing towards the inspection area. This would also compromise the assumption that what is being observed has to be planar.

Note that there is some tape placed randomly on the ground. This helps to create visual texture, which benefits the Bebop's onward camera used by the inner controller to perform optical flow to maintain stability during hovering; otherwise, the Bebop would drift. There is some tape on the columns, but that is not used by the SLAM system in this

Table 1. Image resolution and operation frequencies used in our approach.

	Camera angle	Image resolution	Frequency (Hz)
Bebop's Camera	$83°$ in Pitch	640×380	30
uEye	$-90°/90°$ in Yaw	900×400	93
SLAM (2000 Features)	—	640×380	12
QR Detector/Decoder	—	900×400	9

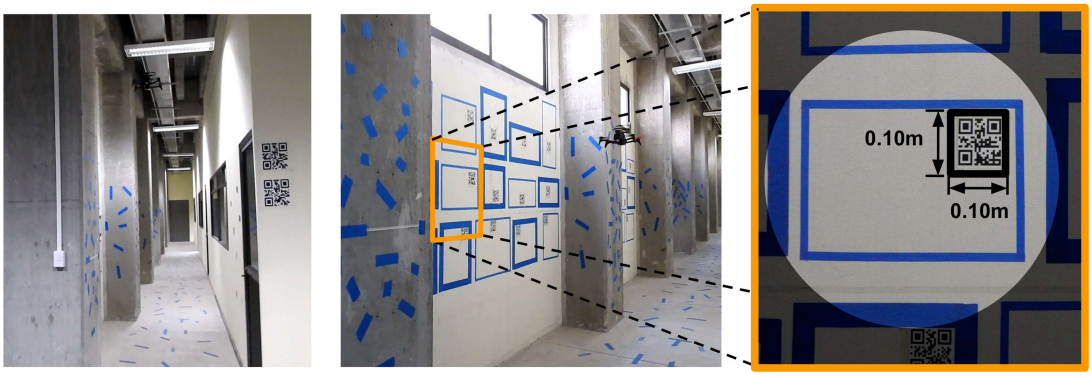

Fig. 7. Test facility used in the experiments and QR codes laid out on the test scenario.

work. To emulate packages on the walls, we made box-like shapes with tape as well, but again, that does not help the mapping, although it is useful to assess whether the QR detector returns false positives detected on the tape.

At the outset, the MAV takes off from a place away from the corridor. The MAV has to fly forward and turn towards the corridor to enter it and follow a set of waypoints that will make it fly up and down while the inspection is carried out with the uEye camera, see Fig. 8. A 3D map and the camera's trajectory estimated with the visual SLAM system, followed during the autonomous flight, are shown in Fig. 9. This map is created from scratch, and the mapping is triggered automatically when the MAV reaches 1 m in height after taking off. In this case, the height measured with the altimeter is added up to the camera translation estimate in the Z-axis (after being converted to the right-hand coordinate system). We have considered mapping the scene beforehand and loading it when required to perform a flight. However, we were interested in evaluating the mapping consistency when using our metric monocular SLAM approach. Saving and re-using the map is something to be done in our future work.

We carried out five runs to evaluate the inspection in this facility. The corridor was inspected for a horizontal length of 6 m, a minimum height of 1.4 and a maximum height of 2.5 m. The camera's trajectory for each run is shown in Fig. 10 in a top view and a 3D view.

The former helps to appreciate that the MAV follows the set flight trajectory in a consistent manner. After taking off and towards the coordinate $(6, 0)$, the trajectory waves from left to right (with respect to the X-axis). This is attributed to a drift that occurred after taking off. We believe that even when the take-off area had some texture, the illumination was low and affected the Bebop's optical flow performance. However, our outer controller tried to bring the vehicle to fly over the virtual line between the first waypoint at $(0, 0)$ and the turn waypoint at $(6, 0)$. After the turn, the flight became smoother except when entering the

Fig. 8. External views of the MAV performing autonomous flight and inspection of the corridor in the test facility, the MAV is indicated with a green circle. A video of these experiments can be watched at https://youtu.be/vyOiIXkq9KA.

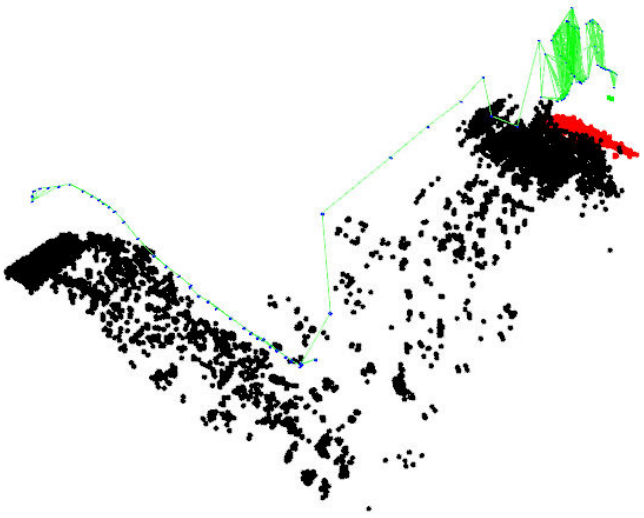

Fig. 9. Skewed view of camera poses (green) and map estimated (black and red dots) with RGB-D ORB-SLAM using the Bebop's monocular camera and the synthetic depth image method in [6, 7].

corridor, where there are some sideway shifts. We attribute this to the wall effect, given the narrow space in the corridor, which had some effect when flying upwards and downwards. Nevertheless, the 3D view in the same image shows that the MAV flew the same trajectory consistently during all the runs. This means that, as expected, even when the visual SLAM ran from scratch for every run, the camera pose estimates remained consistent in scale, meaning that the computation of the synthetic depth was successful.

Figure 11 shows one of the runs to illustrate the effect of the controller on the MAVs position for the first and second waypoints. In this figure, we plot the relative position of the MAV $\mathbf{r}$ (see Eq. (9)), with respect to these two waypoints and the direction angle, this is, the angle between the vector $\mathbf{l}$ and the MAVs orientation vector $\mathbf{d}$ (see Eq. (14)), indicated as *dir. ang.* These variables are shown in red, whereas the references are shown in blue. We should point out that the MAV reaches the first waypoint in the step 460, after which, the MAV has to rotate 90° to face towards the second waypoint for which the distance and height references

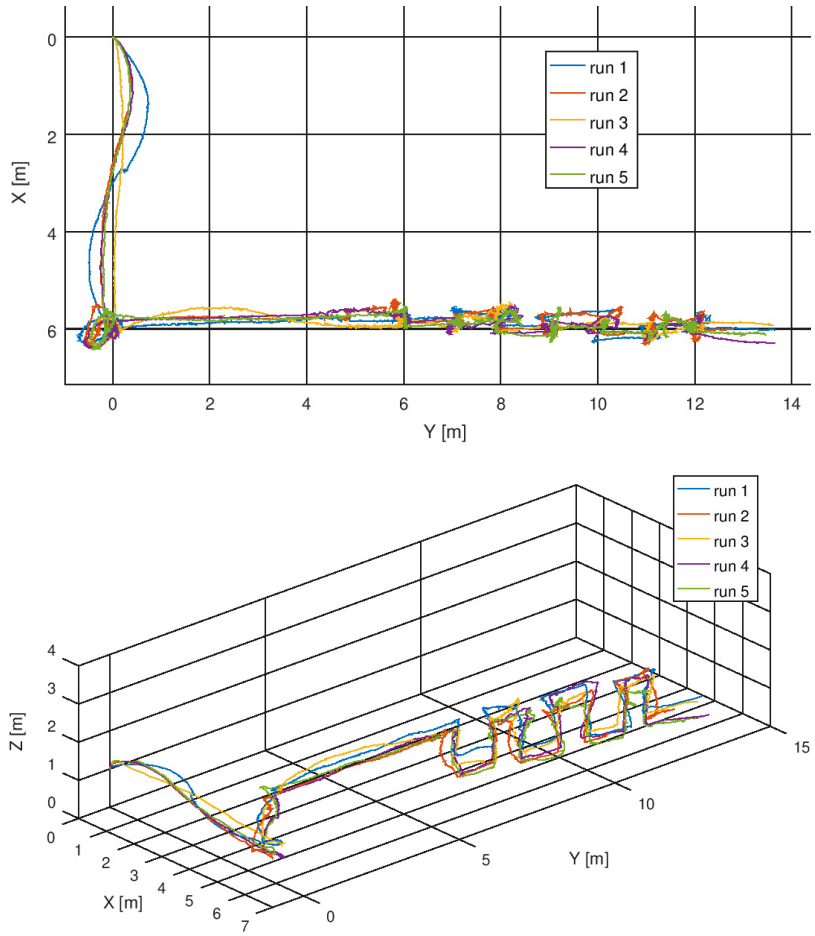

Fig. 10. Top and 3D view of the trajectories obtained with the metric monocular SLAM during autonomous flight, we use the right-hand coordinate system.

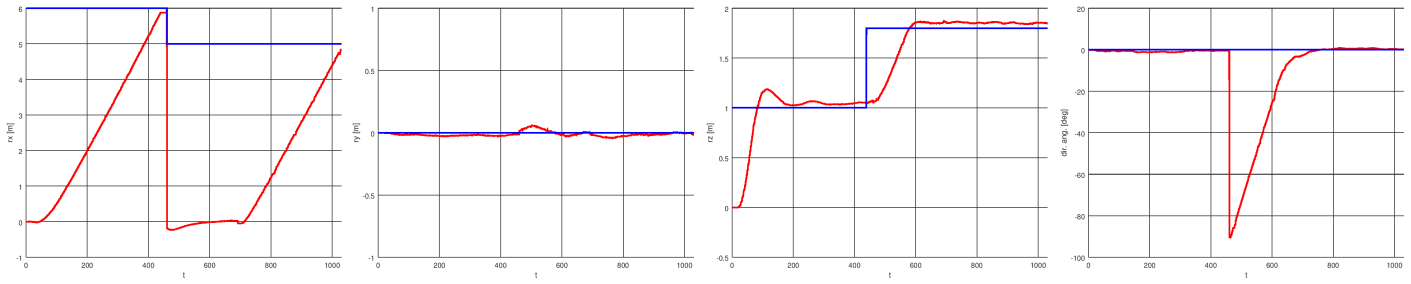

Fig. 11. Plots for one run of the MAVs relative position with respect to the first and the second way points of the flight trajectory in the test facility experiments. The relative position is given by the vector $\mathbf{r} = [r_x, r_y, r_z]$. We also show the direction angle calculated in Eq. (14). The references are depicted in blue. These variables change according to the signals sent to the MAV by the pitch, roll, height and yaw controllers defined in Eqs. (10)–(12). Note that the variables reach the references as expected.

change for r_x and r_z. However, before the pitch and roll controller begin to work, the yaw controller rotates the MAV seeking to reduce the direction angle, while the height controller seeks to reach the height reference. Once the direction angle is close to zero under some threshold (we set it to be less than $2°$) then the pitch and roll controller start to work.

Regarding the inspection, a total of 24 QR codes had to be detected and decoded. On average, 90% of the packages were detected and correctly decoded. Figure 12 shows a snapshot of the GCS in one of the runs. The inspection camera images are shown to the left in low resolution to save transmission time. A top view of the flight plan is shown to the right, with the waypoints indicated as cyan arrows. The MAVs position is indicated as a white arrow. The next waypoint and the final waypoint (for landing) are shown in purple and yellow, respectively. The list of found packages is overlaid on the flight plan. The keyboard for manual control is also shown in the left-top corner. Note that it indicates when the MAV is performing an autonomous flight.

4.2. *Performance in the IMAV 2019 competition*

The approach described in this work was tested in the indoors competition of the IMAV 2019, whose main theme was *warehouse inspection using drones*. Besides the inspection of shelves, the competition included additional tasks such as object picking, object delivery and autonomous landing. We focused on the inspection task, although we developed some strategies for object delivery and autonomous landing [25]. Furthermore, inspired by the object picking task, after the competition, we also developed an approach for autonomous object picking [26].

For the warehouse inspection task, a set of eight labels were given to each team right before their turn to enter the

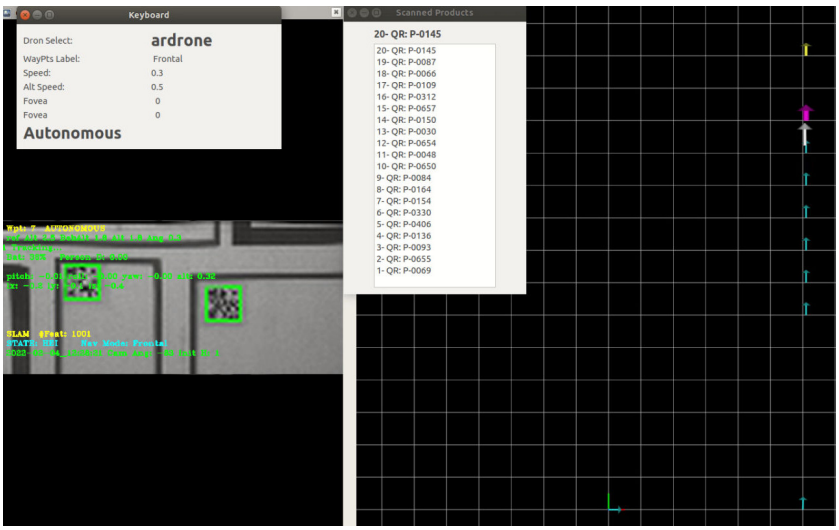

Fig. 12. View of the GCS showing a list of detected QR codes. The camera view shows a low-resolution image of the inspection camera with detected QR codes. A top view of the flight trajectory (cyan arrows) is also shown with the MAVs position shown in real time (white arrow).

arena. These labels were chosen randomly and corresponded to target package labels placed on any of the two racks in the arena; of course, there were many other packages with different labels. There were eight shelves for each rack. Each shelf also had a label that could be read on a tag laid out at the bottom of the shelf. A QR code was also placed next to the tag, containing the same label information. Teams could use QR codes or tags to know the label of the shelf. Therefore, the goal was to indicate on what shelf a given target package was located.

Package labels were alphanumeric strings of five characters. Shelf labels started with a two-digit number and a single letter, giving a length of three characters. We decided to use QR detection to find both, package and shelf labels. Every time a QR code was detected and decoded, we kept two lists based on the decoded information, one for labels corresponding to found target packages and one for labels corresponding to shelves. For each package or shelf, we also stored their 3D position, which was easy to calculate given that we knew the position of the inspection camera in the world (easy to calculate with respect to the Bebop's camera used for SLAM), and given the image position of the detected QR code, we can calculate a 3D vector departing from the camera's center of projection, crossing this image position and with a given depth value, which in this case we know since we calculated the flight trajectory to have a distance of 1 m between the MAV and the rack. Thus, if one of the decoded labels corresponded to a target package, we sought out in a list of shelf labels the one whose 3D position was closer to the 3D position of the found target package. In the case that a shelf label was detected, then it was stored in the second list, and we ran an update of the closes shelf label associated with each target package.

Figure 13 shows some pictures of our MAVs performance during the inspection task. From take-off to landing, the flight had a duration of approximately 12 min. The

Fig. 13. Snapshots of our MAV performing autonomous flight to carry out the warehouse inspection flight. The order is from left to right, top to bottom. In the first image, the MAV takes off and flies towards rack B, and after inspecting it, it flies towards rack A, flying in between the racks to continue with the inspection, the MAV is indicated with a green circle. A video of this performance can be watched at https://youtu.be/vyOiIXkq9KA.

Table 2. List of target packages to be found during the inspection and list of packages found with our autonomous MAV.

Target packages	Found packages	Shelf label	Valid	Physically found at
R972Y	G842J	11B	Yes	Rack B
M177F	V645K	22B	Yes	Rack B
Z632O	E223W	24B	Yes	Rack B
X324V	W233P	14B	Yes	Rack B
V645K	R972Y	21B	No	Rack A
E223W	X324V	11B	No	Rack A
G842J	Z632O	13B	No	Rack A
W233P				

dimensions of the arena were given in advance to teams; however, the exact position of the racks was given right at the moment the team entered the arena. Thus, once we obtained the position of the racks, we outlined a flight trajectory based on waypoints and loaded it into the MAVs onboard computer. Given that the racks contained packages facing the arena's left side, the inspection camera's angle was set to $-90°$. In the same figure, it can be seen that the MAV took off from the designated location and flew towards the first rack. Like our experiments in the test facility, once the MAV was in front of the first rack, it performed upward and downward. When the MAV reached the end of the rack, the drone turned around (180°) and navigated to the beginning of the rack to turn to the left and then continued forward towards the second rack. In this case, it flew into the corridor formed by the two racks and once there, it performed its upward/downward flight. The flight ended when the MAV reached the end of the second rack.

When we entered the arena, the judges reshuffled the packages on the racks and randomly selected a list of target packages to be found by our MAV. These packages are listed in the first column of Table 2. The second column of the same table shows the packages found during the inspection and, according to our system, their corresponding shelve label, shown in the third column of the table. When a QR code was detected, it was shown on the camera view of the GCS, with a green square outlining the image area where the QR was detected. If the QR corresponded to a target package and a shelf could be associated, then it was shown on the package list also shown on the GCS. This is shown in Fig. 14; the image at the bottom right shows the full list of the target packages found during the inspection.

The competition judges validated that the first four packages were correctly associated to the shelf location, in the first rack label as B, see the fourth column of Table 2. Unfortunately, the other three packages were not located

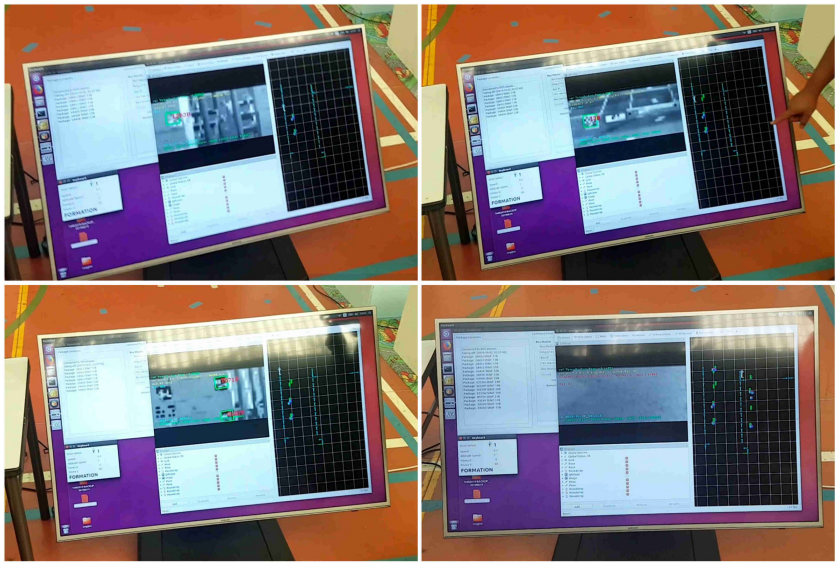

Fig. 14. Snapshots of the GCS, showing images of the inspection camera transmitted by the MAVs onboard computer. The flight plan and the current MAVs position is also shown. The list of found products is shown to the left of the screen.

correctly, even when they were correctly detected on the second rack label as A. This was due to a bug in our association algorithm concerning the 3D distance rule. Nevertheless, in terms of the QR codes, our system managed to correctly detect the QR for seven out of eight target packages, and it also detected and decoded all the QR codes for all the shelf labels. Due to the performance of MAV during this task, we were granted the special award: *Best Flight Performance*.

5. Conclusions

We have presented our approach to addressing the warehouse inspection task in the indoors competition of the IMAV 2019. We have proposed to use an autonomous MAV to perform the inspection. To achieve autonomous flight, we have used a metric monocular SLAM system based on RGB-D ORB-SLAM and a method to compute a synthetic depth image using the MAVs height, measured with the altimeter, and the angle of the MAVs onboard camera. A second camera onboard the MAV is used as an inspection camera. Images captured with this camera are processed by the onboard computer, the Intel Stick Computer, to detect and decode QR codes observed on the inspection area. The SLAM system also runs on the Intel Stick. In this sense, our approach performs full onboard processing, establishing a connection with a GCS for visualization purposes only, showing telemetry, images from the inspection camera and the list of packages found during the inspection. Our results indicate that our approach is functional and can be used not only on warehouse inspection but also in similar tasks where visual inspection is required.

There are some aspects we want to improve in our future work. The first one is to add a module for online waypoint generation for when a map of the inspection area is already available similar to the work in [27]. In addition, we are keen on investigating novel localization algorithms that enable the MAV to fly faster [28]. We are also interested in incorporating efficient deep learning models that can run onboard in order to perform broader visual inspection tasks. Finally, we will also investigate methods based on collective robotics to enable two or more MAVs to distribute the inspection task.

Acknowledgments

The second author is thankful to Consejo Nacional de Ciencia y Tecnología (CONACYT) in Mexico for the scholarship No. 924254.

We are also thankful to the Consorcio en Inteligencia Artificial sponsored by CONACYT, for the financial support received to attend the IMAV 2019 under the grant project FORDECYT 296737.

References

[1] D. B. Urević, N. Pavlov and M. Božić, The role of warehouse in e-business, in *E-Business Technologies* (Belgrade, Serbia, June, 10–11, 2021), p. 7.

[2] T. M. Fernández-Caramés, O. Blanco-Novoa, I. Froiz-Míguez and P. Fraga-Lamas, Towards an autonomous industry 4.0 warehouse: A UAV and blockchain-based system for inventory and traceability applications in big data-driven supply chain management, *Sensors* **19** (10) (2019) 2394.

[3] Company Airvant, INVENTARIOS DE ALMACÉN, https://www.airvant.es/cadenadesuministro.

[4] Company Drone Scan, Stock Take, Physical Inventory, Cycle Counting, website https://www.dronescan.co/info.

[5] S. M. Bae, K. H. Han, C. N. Cha and H. Y. Lee, Development of inventory checking system based on UAV and RFID in open storage yard, in *2016 Int. Conf. Information Science and Security (ICISS)* (IEEE, 2016), pp. 1–2.

[6] L. O. Rojas-Perez and J. Martinez-Carranza, Metric monocular slam and color segmentation for multiple obstacle avoidance in autonomous flight, in *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)* (IEEE, 2017), pp. 234–239.

[7] H. Moon *et al.*, Challenges and implemented technologies used in autonomous drone racing, *Intell. Serv. Robot.* **12**(2) (2019) 137–148.

[8] A. Manjrekar, Jha, Dr. Swati, Khapare, Siddhi, Jagtap, Pratiksha and Yadav, Vinay, Warehouse inventory management with cycle counting using drones, in *Proceedings of the 4th International Conference on Advances in Science & Technology (ICAST2021)*. 7 May 2021, eds. S. Khapare, P. Jagtap and V. Yadav (2021). Available at https://ssrn.com/abstract=3869512 or http://dx.doi.org/10.2139/ssrn.3869512.

[9] B. Yoon, H. Kim, G. Youn and J. Rhee, 3d position estimation of drone and object based on QR code segmentation model for inventory management automation, in *2021 IEEE Int. Symp. Safety, Security, and Rescue Robotics (SSRR)* (IEEE, 2021), pp. 223–229.

[10] R. Keßler, C. Melching, R. Goehrs and J. M. Gómez, Using camera-drones and artificial intelligence to automate warehouse inventory, in *Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021)*, Vol. 2846 (Stanford University, Palo Alto, California, USA, March 22–24, 2021).

[11] F. Farahnakian, L. Koivunen, T. Mäkilä and J. Heikkonen, Towards autonomous industrial warehouse inspection, in *2021 26th Int. Conf. Automation and Computing (ICAC)* (IEEE, 2021), pp. 1–6.

[12] A. De Falco, F. Narducci and A. Petrosino, An uav autonomous warehouse inventorying by deep learning, in *Int. Conf. Image Analysis and Processing* (Springer, 2019), pp. 443–453.

[13] T. M. Fernández-Caramés, O. Blanco-Novoa, M. Suárez-Albela and P. Fraga-Lamas, A UAV and blockchain-based system for industry 4.0 inventory and traceability applications, *Multidiscip. Digit. Publ. Inst. Proc.* **4**(1) (2018) 26.

[14] H. Cho, D. Kim, J. Park, K. Roh and W. Hwang, 2d barcode detection using images for drone-assisted inventory management, in *2018 15th Int. Conf. Ubiquitous Robots (UR)* (IEEE, 2018), pp. 461–465.

[15] D. Cristiani, F. Bottonelli, A. Trotta and M. Di Felice, Inventory management through mini-drones: Architecture and proof-of-concept

implementation, in *2020 IEEE 21st Int. Symp "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)* (IEEE, 2020), pp. 317–322.

[16] I. Kalinov, A. Petrovsky, V. Ilin, E. Pristanskiy, M. Kurenkov, V. Ramzhaev, I. Idrisov and D. Tsetserukou, Warevision: CNN barcode detection-based uav trajectory optimization for autonomous warehouse stocktaking, *IEEE Robot. Autom. Lett.* **5**(4) (2020) 6647–6653.

[17] F. Guérin *et al.*, Towards an autonomous warehouse inventory scheme, in *2016 IEEE Symp. Series on Computational Intelligence (SSCI)* (IEEE, 2016), pp. 1–8.

[18] M. Beul, D. Droeschel, M. Nieuwenhuisen, J. Quenzel, S. Houben and S. Behnke, Fast autonomous flight in warehouses for inventory applications, *IEEE Robot. Autom. Lett.* **3**(4) (2018) 3121–3128.

[19] W. Kwon, J. H. Park, M. Lee, J. Her, S.-H. Kim and J.-W. Seo, Robust autonomous navigation of Unmanned Aerial Vehicles (UAVS) for warehouses' inventory application, *IEEE Robot. Autom. Lett.* **5**(1) (2019) 243–249.

[20] M. Quigley *et al.*, Ros: An open-source robot operating system, in *ICRA Workshop on Open Source Software*, Vol. 3 (3.2) (Kobe, Japan, 2009), p. 5.

[21] R. Mur-Artal, J. M. M. Montiel and J. D. Tardos, Orb-slam: A versatile and accurate monocular slam system, *IEEE Trans. Robot.* **31**(5) (2015) 1147–1163.

[22] R. Swaminathan and S. K. Nayar, Nonmetric calibration of wide-angle lenses and polycameras, *IEEE Trans. Pattern Anal. Mach. Intell.* **22** (10) (2000) 1172–1178.

[23] L. O. Rojas-Perez and J. Martinez-Carranza, Flight coordination of drones in GPS-denied environments using a metric visual slam, in *11th Int. Micro Air Vehicle Competition and Conf.*, ed. P. Campoy (Madrid, Spain, 2019), pp. 200–205.

[24] The ZBar Bar Code Reader is free software. Available: http://zbar.sourceforge.net/.

[25] A. A. Cabrera-Ponce and J. Martínez-Carranza, Onboard CNN-based processing for target detection and autonomous landing for MAVS, in *Mexican Conf. Pattern Recognition* (Springer, 2020), pp. 195–208.

[26] M. López García and J. Martínez Carranza, A first CNN-based approach towards autonomous flight for object lifting, *Comput. Sist.* **24** (3) (2020) 1219–1228.

[27] Á. Martínez Novo, L. Lu and P. Campoy, Fast RRT* 3d-sliced planner for autonomous exploration using MAVs, *Unmanned Syst.* **10**(2) (2022) 175–186.

[28] S. Yuan, H. Wang and L. Xie, Survey on localization systems and algorithms for unmanned systems, *Unmanned Syst.* **09**(2) (2021) 129–163.

Jose Martinez-Carranza is a Full-Time Principal Researcher B (equivalent to Associate Professor) in the Computer Science Department at the Instituto Nacional de Astrofisica Optica y Electronica (INAOE). In 2015, he was awarded the highly prestigious Newton Advanced Fellowship and currently, he holds an Honorary Senior Research Fellowship in the Computer Science Department at the University of Bristol in the UK. His team has won international competitions such as First Place in the IEEE IROS 2017 Autonomous Drone Racing competition and 1st Place in the Regional Prize of the OpenCV AI Competition 2021. In 2021, he served as General Chair of the International Micro Air Vehicle conference, the IMAV 2021.

Leticia Oyuki Rojas-Perez graduated in Mechatronics Engineering at the Instituto Tecnologico Superior de Atlixco and obtained a Master of Science degree in Computer Science from the Instituto Nacional de Astrofisica, Optica y Electronica (INAOE). She is currently a PhD student in the Department of Computer Science at INAOE under the supervision of Professor Jose Martínez-Carranza. Her master thesis won 1st place in the National Competition for the Best Master Thesis in AI, awarded by the Mexican Society of Artificial Intelligence (SMIA). She has participated in several international drone competitions.