Autonomous Landing, Obstacle Avoidance and Block Recovery for a Quadrotor Drone

Matteo Fei*, Marcos Linares Bayón†, Andrei Muresan Radu‡, Wonhee Lee§, Leandro R. Lustosa
ISAE-SUPAERO, Université de Toulouse, France

ABSTRACT

This technical report presents our proposed solution to the IMAV 2023 Drone Competition. The competition suggests an autonomous block recovery mission using a quadcopter with obstacle-avoidance capabilities. Our drone hardware and software architectures are discussed herein, and, in particular, we introduce a ROS-less custom configuration for intermachine communication between a high-level, low-sampling-rate Companion Computer and a low-level, high-sampling-rate Flight Controller. In addition, the developed controllers and state machines to tackle the different phases of the competition are presented. Finally, the necessary computer vision algorithms for relative navigation amongst obstacles, windows, and blocks are explored.

1 INTRODUCTION

This paper aims to present the drone brought to the IMAV 2023 indoor competition by the ION**LAB** team. The competition challenge involves collecting and stacking cone blocks with known dimensions and weight. Different path alternatives are to be followed to fetch the blocks, each with its corresponding complexity level and associated score. Each team can decide which path to follow. This paper focuses on the path that involves landing on a rotating platform and passing through a window with a moving obstacle.

This article is organized as follows. Section 2 explains the architecture of the drone hardware while Sec. 3 explores its software counterpart. Section 4 presents the simulations to validate our strategy. Section 5 introduces the state machine of the drone. Section 6 explains the computer vision tasks necessary for relative navigation in a GNSS-denied environment. Finally, Sec. 7 summarises the work presented in the paper and provides findings.

2 HARDWARE ARCHITECTURE

Our platform builds on a pre-built Holybro X500 V2 frame equipped with Pixhawk 6X Autopilot Flight Controller hardware. Except for configuration parameters (i.e., propeller number and geometry, transmitter mapping, and calibration data for inertial sensors and magnetometer), the corresponding driving PX4 software code has not been modified from the community development tree[1]. Instead, an Nvidia Jetson Nano Companion Computer with in-house tailored behavior for guidance, altitude control, and computer vision broadcasts attitude and thrust setpoints to the Pixhawk board. Additionally, stereo and monocular cameras retrieve visual information from the surroundings (see Fig. 13). Finally, a downward-facing LiDAR sensor measures the relative altitude from the ground. Figure 1 summarizes the drone's avionics.

The mechanism chosen to pick the blocks is two 3D-printed pins fixed between the legs of the quadrotor. The pins are attached to a cartesian mechanism that enables them to slide recovering or releasing the cone depending on the section of the competition. The sliding mechanism will be controlled by a servomotor through a circular-to-linear mechanism, being this motor directly connected and controlled to the Pixhawk board.

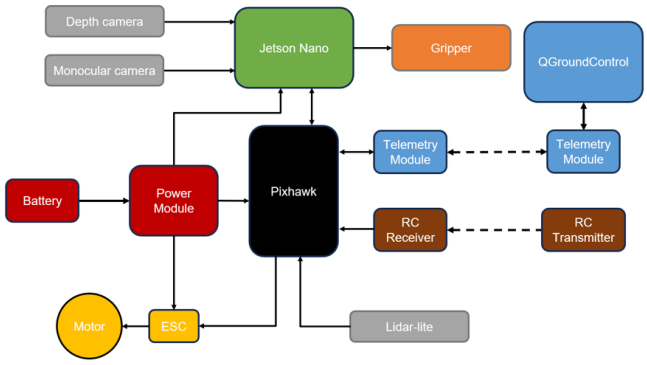

Figure 1: Avionics architecture.

3 SOFTWARE ARCHITECTURE

This section describes the distributed control routines operating asynchronously in the Flight Controller (i.e., Pixhawk) and the Companion Computer (i.e., Jetson Nano) boards. The Flight Controller provides low-level high-frequency (i.e., sampling rate up to 400Hz) sensing, stabilization, and control. At the same time, the Companion Computer provides high-level lower frequency (around 50Hz) guidance routines. The latter also includes computer vision through a

*Corresponding Author: matteo.fei@student.isae-supaero.fr
†marcos.linares-bayon@student.isae-supaero.fr
‡andrei.muresan-radu@student.isae-supaero.fr
§wonhee.lee@student.isae-supaero.fr

[1]More specifically, commit hash 3303323971f02cff6eb2fc029562bc4.

dedicated Graphics Processing Unit (GPU) and the state machine that manages mission phases and respective guidance laws switching conditions and mechanisms.

3.1 The Flight Controller Flight Stack

The PX4 software consists of two layers: the flight stack, an estimation and flight control library, and the middleware, a general robotics layer that supports many platforms (e.g., fixed-wing aircraft, rotary-wing copters, rovers), providing communication interfaces and hardware abstractions. The flight control architecture is dependent on the platform. In a quadcopter, for instance, the control flight stack consists of a cascade of Proportional-Integral-Derivative (PID) controllers according to Fig. 2.

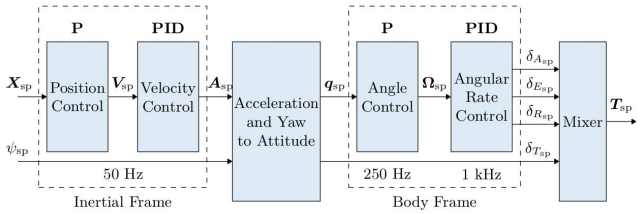

Figure 2: PX4 quadcopter cascaded controller. (Adapted from the official PX4 online documentation [1].)

Each PID controller is a separate process in the Pixhawk 6X board, and all PX4 processes share the same memory space (through NuttX operating system calls). The PID controllers exchange data internally (i.e., in Flight Controller scope) through the uORB messaging middleware. For intermachine communication, the μXRCE-DDS middleware exposes Flight Controller uORB topics through a UDP socket interface.

3.2 The Intermachine Communication Bridge

While the commonplace strategy to deploy intermachine communication between the Pixhawk Flight Controller and a Companion Computer is to install the Robot Operating System (ROS) and rely on the native μXRCE-DDS client on PX4 and respective predeveloped μXRCE-DDS ROS agent modules on the Companion Computer side, we have decided to develop our own μXRCE-DDS agent, namely, the **PIL**ION project, circumventing the need of ROS (see Fig. 3). We avoid ROS usage in our projects due to portability issues: some of our drones operate QNX and other Real-Time operating systems incompatible with ROS. Thankfully, μXRCE-DDS is compatible with QNX, for instance. Nevertheless, our Companion Computer in this drone does operate Linux Ubuntu.

The Companion Computer broadcasts the desired control setpoints every 20ms to the Flight Controller. In case of intermachine communication failure, a Flight Controller timeout (natively included in PX4) automatically switches to an autonomous hold mode for safety.

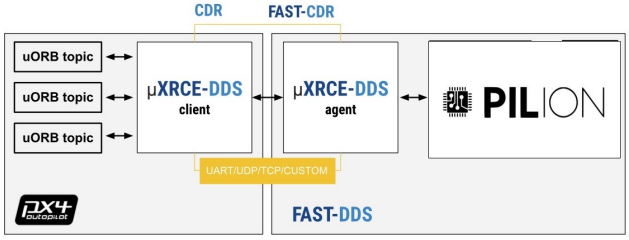

Figure 3: Intermachine interface. (Adapted from the official PX4 documentation [2]).

PX4 provides a set of uORB messages for setting desired setpoints to the drone. The setpoint abstraction level choices go from high-level input, such as desired position, down to desired torque and thrust inputs. According to the chosen setpoint abstraction level, higher-level PID controller modules should be turned off in the Flight Controller to avoid message conflict due to multiple setpoint masters. For example, referring to Fig. 2, if the Companion Computer periodically sends acceleration setpoints, the first two PID controllers of the cascade should be neglected. To give a position, velocity, or acceleration setpoint, PX4 requires a Global Navigation Satellite System (GNSS) receiver. Since the drone participates in an indoor competition, decent GNSS coverage is not guaranteed. Consequently, we chose to have the Companion Computer send attitude setpoints through the uORB message `Vehicle Attitude Setpoint`. This message requires a quaternion setpoint and a normalized collective thrust input. For that to occur, the internal Flight Controller PX4 flight mode should be set to `Offboard`. The PX4 flight mode dictates the PID controller setpoints' origin. In our case, we neglect pilot and ground control setpoints through the offboard mode). Fast switching to stabilized mode (where pilot inputs are taken as velocity setpoints instead) is available for safety through a pilot transmitter switch.

3.3 The Companion Computer Controller

Figure 4 shows a schematic of the overall Companion Computer controller architecture. It achieves position control through a series of PD controllers for each axis. The z-axis controller drives the drone's thrust input, while the x-axis and y-axis controllers drive the pitch and roll angles, respectively. Implementing a yaw controller is unnecessary since this angle is unrelated to any positional input. Finally, roll, pitch, and yaw reference angles are wrapped together as a quaternion setpoint before being sent to the Flight Controller.

Finally, we developed a State Machine in the Companion Computer to adequately tackle the different mission phase requirements. Accordingly, depending on the mission phase, the State Machine decides if relative target position measurements should come from the forward-facing or downward-facing camera computer vision modules.

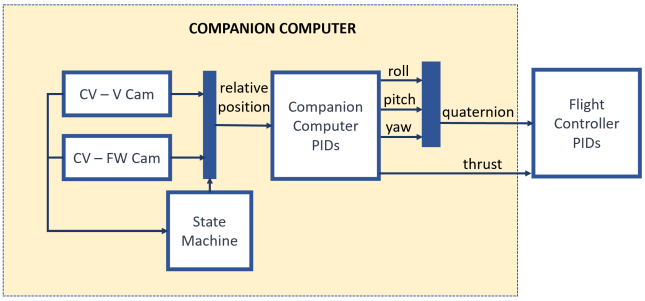

Figure 4: Companion Computer control diagram.

4 SIMULATION

The PX4 development environment provides a Software-In-The-Loop (SITL) simulation environment using the Gazebo Engine to simulate the exact Flight Controller code in a simulated environment with visualization and debugging interfaces available and the possibility of connecting to QGroundControl. The later open-source application provides ground station functionalities. However, a MATLAB model simulating PX4 behavior (drone dynamics plus low-level PID control) was conceived for controller tuning to analyze the Companion Computer PID controllers' response through MATLAB practical functions and toolboxes.

4.1 Tuning Gains for the Companion Computer Controllers

The Companion Controller PID controller gains, which takes a target relative position as input and outputs quaternion and thrust setpoints for the Flight Controller PID controllers, are computed through the pole placement technique. Figure 6 shows the step response outcome on the x-axis.

The dynamical response is satisfactory regarding overshoot, being less than 4%. The performance on the other axes has similar behavior.

4.2 Landing on a Moving Platform

Once the Companion Computer controller is designed, a MATLAB simulation is performed to test its behavior when landing on a moving platform. A state machine manages the landing decision-making process, inspired by others seen in literature [3, 4]. Let the error vector be the difference between the drone's position on the x-y plane – namely, the plane of the platform's rotation – and the platform's position. The drone will track the target (i.e., the platform's position) while keeping a constant altitude. The drone starts descending while following the target when the error norm is below a predefined threshold. If the target is lost, meaning that the error norm becomes higher than the predefined threshold, the drone will return to a given altitude and repeat the process until the landing is fully archived. For this simulation, no computer vision module was implemented, and the platform's position for landing is assumed to be known. Also, the position of the drone itself is considered to be known: in the competition environment, this is not true since GNSS signal is not available, and instead, the relative difference between the drone and the target is considered. The Simulink model used to archive this simulation is shown in Fig. 5. Finally, the competition platform rotates in a 50 centimeter radius at an angular velocity of 10^o/s.

Figure 8 shows the simulation results. The drone starts seeing the platform 20 seconds from the start of the simulation and starts to follow the platform. Complete landing is archived in 5 seconds, as shown in Fig. 7. The final error at the moment of landing is 10 centimeters, which is low enough for the competition.

5 STATE MACHINE

We subdivide the proposed complex mission into several tasks that the drone performs in an orderly fashion. This division allows for Companion Computer orderly instructions determination at each point and is implemented as a state machine. Figure 9 illustrates the state machine at the highest level. Starting from the drop zone, the drone flies through a window while avoiding a moving obstacle, then reaches the pickup zone and picks up a cone-shaped load. Then, it returns to the original stack zone while avoiding the same obstacle and places the block on a pile of previous ones or the ground, in the case of a first iteration.

We further break down each one of the blocks of the high-level state machine into programmable commands. Figure 10 depicts the logic for moving through the transit zone. A series of references are established in order of priority for the drone to detect and follow. First, it looks for the window and approaches it. If the window is not found, it goes towards the QR code behind it. If the latter is not detected, it follows the line on the ground. If the line is not found, it flies toward the known magnetic trajectory of the target while still searching for any of the previous visual references. When the window is detected and approached, it stops at a 1 m distance, where the forward camera can see the whole screen. It waits in front of it until the moving obstacle is cleared, and it advances following the guideline. While crossing the window, altitude is no longer read with the Lidar range sensor but with the barometer to avoid noise caused by sudden attitude changes due to the window's frame. After passing the window, a similar cascade of references approaches the QR code at the end of the transit lane.

Figure 11 shows the algorithm followed to pick up the block. Once the forward camera sees the ArUco code, the drone approaches it until it is visible to the vertical down-facing camera. At that point, the drone starts following the circular motion of the platform until it is consistently over the code, a point at which it starts descending. The drone then lands on the platform, the cone is mounted manually, and the drone takes off again to a predefined mission altitude. In case of loss of QR code visuals, the drone returns to mission altitude and repeats the process.

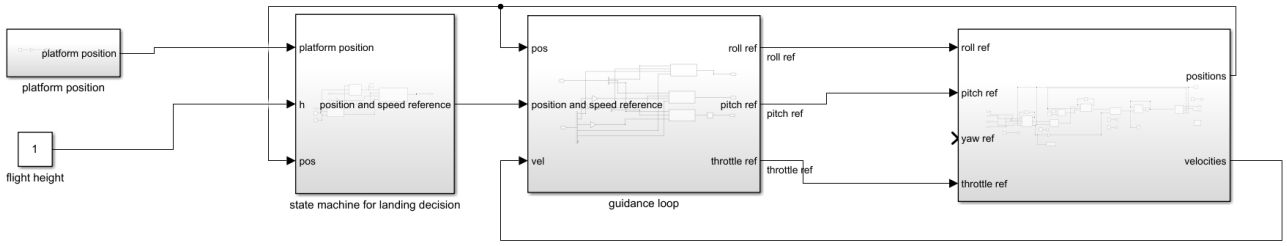

Figure 5: The Simulink model of the Companion Computer control diagram. On the left side, the state machine and the guidance laws represent the Companion Computer controller, while the block on the right represents Flight Controller controllers and the drone dynamics.

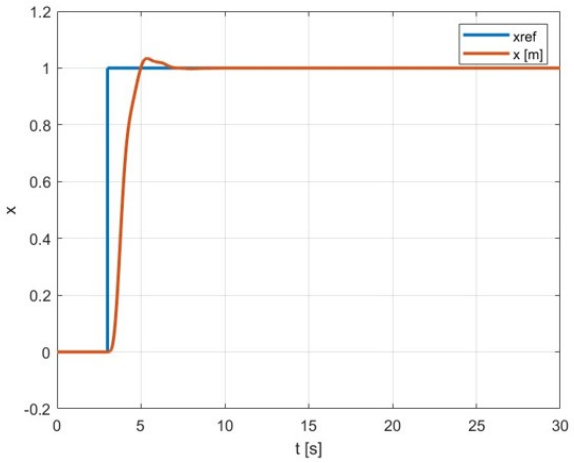

Figure 6: Step response on x-axis.

The last of the high-level blocks of the state machine is the release of the cone on top of the cone pile. At the end of the transit back, the drone searches for the QR code of the drop zone, reaches the position on top of it, and slides slightly left. For the first block, it simply descends to an altitude calculated based on the cone height and releases it. Then, it takes off again at the right altitude and yaw and proceeds towards fetching the next cone. For the consecutive ones, the drone approaches the QR code, looks for the stack, and centers itself on top using the vertical camera. The drone descends to a height calculated from the number of cones stacked and releases the load. Safety protocols are implemented in case of loss of target visuals or centering.

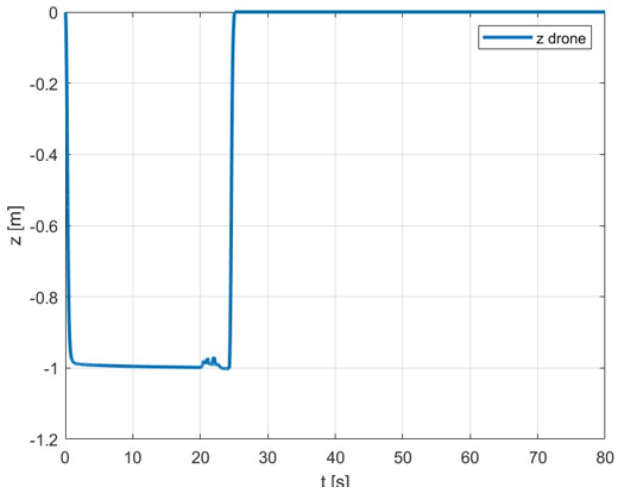

Figure 7: z-axis plot.

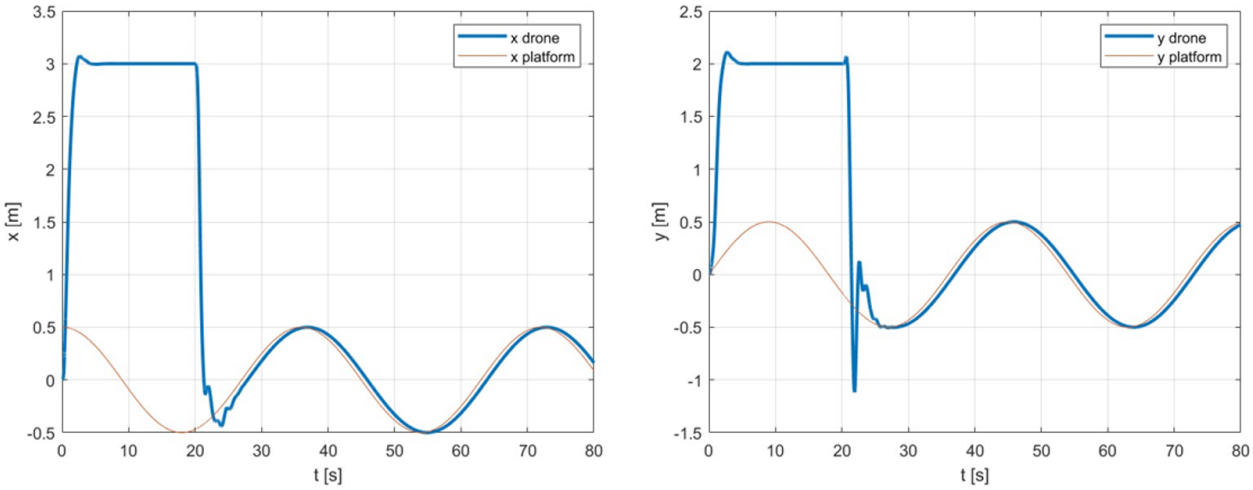

Figure 8: x-axis and y-axis plot.

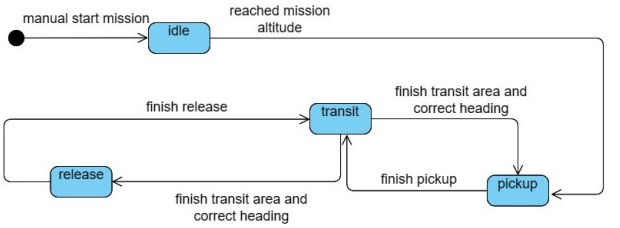

Figure 9: High level state machine.

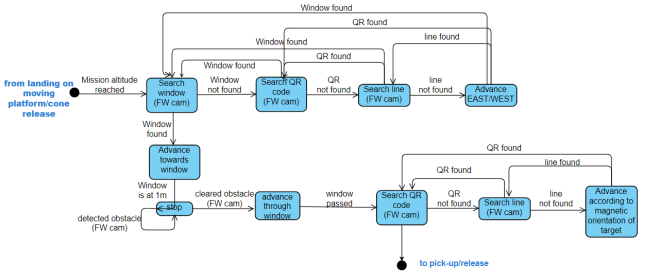

Figure 10: Transit state machine.

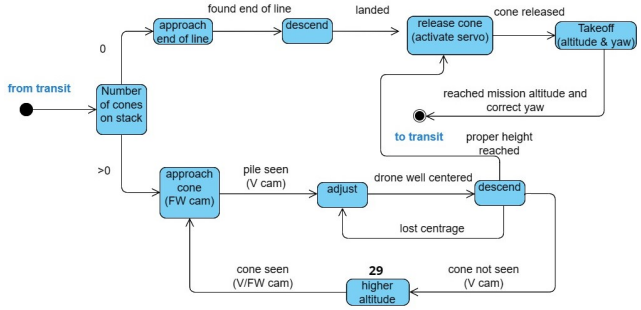

Figure 11: Block collection state machine.

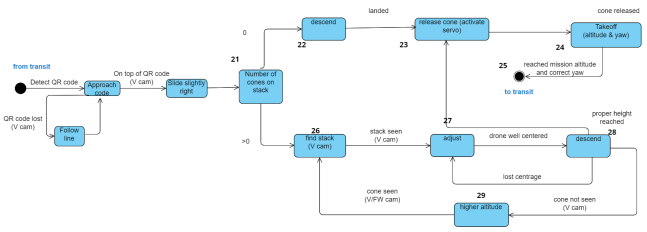

Figure 12: Block release state machine.

6 COMPUTER VISION

The purpose of the computer vision is to provide a setpoint for an active control loop, based on the visual information acquired by cameras. The setpoint is a 3D relative pose given by the special Euclidean group (SE(3)) or its component.

6.1 Architecture

The drone uses two cameras; one forward-looking Intel Realsense D455 depth camera, and one down-looking fisheye monocular camera. The field of views (FoV) of the camera when looked from the side are depicted in Fig. 13.

The depth camera allows to acquire the depth data of the scene directly without using computationally costly algorithms. The fisheye camera reduces the blind spot between the two cameras.

The computer vision software module comprises of five main classes. The name and output of each class is provided in Table 1.

To allow various camera types, each class is independent of the camera model. Different camera models are dealt by

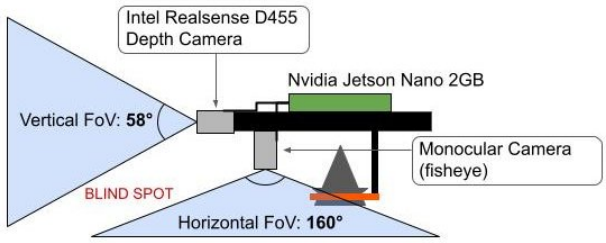

Figure 13: Drone architecture from computer vision perspective and side field of views (FoV) of cameras.

Table 1: Computer vision module classes with their outputs.

Class	Output
ArUco_Detector	Transform matrix (SE3) from marker to camera (T_{cm})
Window_Detector	Transform matrix (SE3) from window to camera (T_{cw})
Cone_Detector	3D position of cone in camera coordinate system.
Release_Decider	3D position of cone in camera coordinate system. Boolean which is true when it is decided that the cone is safe to be released.
Line_Tracer	Approximate relative bearing of the line with respect to the drone.

the abstract camera representation as used by some visual odometry (VO) or simultaneous localization and mapping (SLAM) algorithms such as ORB-SLAM3 [5]. Considering the FoVs of the camera used, the depth camera is modelled by the Brown[6]-Conrady[7] model, and the fisheye monocular camera by the Kannala-Brandt [8] model.

Figure 14 shows the essential state machine with active classes for each mission phase.

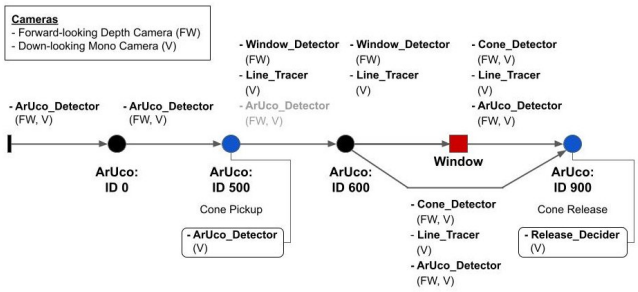

Figure 14: High-level state machine from computer vision perspective.

6.2 *Marker Detection and Pose Estimation*

The goal of the marker detection is to provide the drone with the relative 6 DoF pose of the drone with respect to the marker. The perspective-n-point (PnP) algorithm estimates the pose of the camera, given n number of 3D points of an object in the camera coordinate system and corresponding 2D pixel points in the image.

Since a marker is planar, a special type of PnP algorithms called infinitesimal plane-based pose estimation [9] (IPPE) can be used. In this case, minimum 4 pairs of points are needed, and many open-source libraries such as OpenCV allows to acquire positions of 4 vertices of a marker as well as its ID. IPPE can yield at most two possible solutions.

Along with the solution ambiguity of IPPE, other factors may affect the pose estimation quality, such as the distance from the marker to camera with respect to its focal length[10]. To improve the robustness on estimation, three enhancement methods were applied:

1) initial pose guess is obtained by solving PnP with random sample consensus[11] (RANSAC) to remove outliers.

2) if the difference between the guess obtained from 1) and the previous pose is larger than a threshold, the guess is ignored.

3) if the pose difference is less than the threshold, the guess obtained from 1) is iteratively refined using Levenberg-Marquardt[12][13] method.

Figure 15 shows the visual result of the ArUco marker pose estimation.

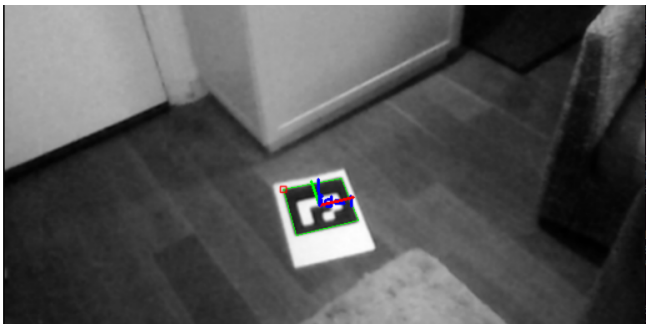

Figure 15: Detected ArUco marker and its estimated pose represented by the axes.

6.3 *Cone Detection, Tracking and Pose Estimation*

The goal of cone detection is to provide the relative 3D position of the target cone which the drone aims to approach. Two approaches were considered to detect the cone: 1) traditional method based on the image processing using geometric characteristics and extracted features, 2) deep learning method based on the neural network trained with custom dataset. Both approaches have their own weaknesses. The former is more prone to change in the scene such as perspective, and the latter is more dependent on the dataset quality.

The mean average precision (mAP) is a metric to evaluate the performance of an object detection algorithm. The mAP of deep learning algorithms surpass most traditional algorithms [14]. Therefore, You Look Only Once (YOLO) object detector was selected as the basis for the cone detection. Specifically, YOLOv4-tiny was chosen to achieve faster inference under limited computing resources.

In addition to detection, tracking is required to follow only one cone out of multiple cones, if they exist. Hence, after the detection of cones, one cone is locked for tracking based on a certain criteria, such as the distance from the image center. Then, a (Channel and Spatial Reliability Tracking) CSRT

[15] tracker is used to track the one. When tracking is lost, a new cone is locked. Figure 16 shows the cone detection and tracking result.

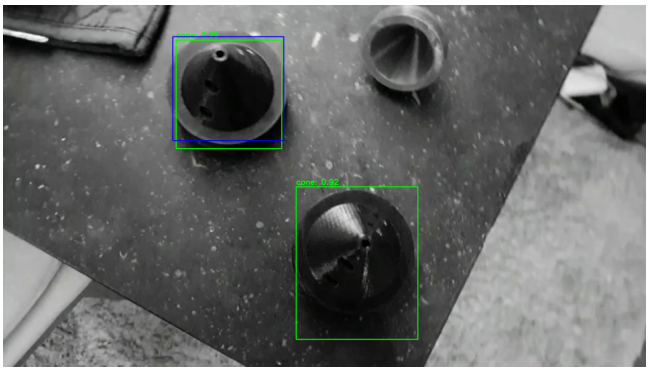

Figure 16: Detected cones (Green) using YOLOv4-tiny trained with custom cone image dataset, and the locked cone (Blue) for tracking using CSRT tracker.

During the tracking, the relative position of the cone can be estimated by unprojection. For the depth camera, the needed depth is directly obtained. For a monocular camera, the known dimensions of the cone can be used to infer the depth.

6.4 *Cone Release*

The goal of cone release decision is to signal the drone when it is believed to be safe to release the cone. First, the desired 3D position of the cone is projected to the image. The bounding box of the projected cone will be named as the target box. The bounding box of the cone output by the tracker will be named as the tracker box. Then, the cone is released when the intersection over union (IoU) between the target box and the tracker box is over a certain threshold. Figure 17 shows the cone before and after the release decision.

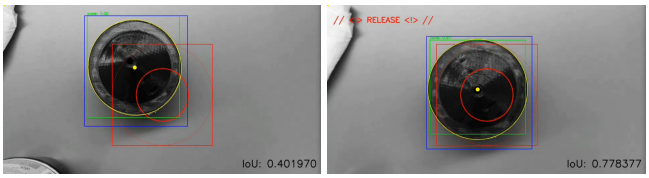

Figure 17: Cone release decision based on the IoU between the target and tracking bounding boxes. Before release (left), after release (right).

6.5 *Window Detection and Pose Estimation*

Window detection can be classified as an object detection problem in computer vision. Since the exact shape of the window is not provided, the image processing approach was chosen over the deep learning one.

The depth image obtained by the depth camera is first thresholded by the depth boundaries from 0.3 to 4 m to get the foreground. This image is binarized with Canny edge detection [16], then quadrilaterals are found using the Ramer–Douglas–Peucker algorithm [17, 18]. Figure 18 shows the scene with a window, and the detected window.

Figure 18: Scene with a window (left), and detected window using depth camera (right).

However, the this method has following expected limitations.

1) if the window is not perfectly quadrilateral, the possibility of detection is scarce.

2) the frequent noise in the depth image near the window leads to 1).

3) depending on the threshold used for the Canny edge detection and on the lighting condition, some edges might not be detected. This case also leads to 1).

To overcome these limitations, a more robust algorithm is required.

Once the contour of the window is determined, the relative pose between the window and the camera can be found using the PnP algorithm. If the orientation is not needed, the relative translation can be found by unprojecting the centroid of the contour.

6.6 *Colored Line Tracing*

The goal of line detection is to provide the relative bearing of the line with respect to the drone. In addition to the bearing, the approximate relative 2D position of the line also needs to be provided to keep the drone lined up.

To extract only the blobs of a specific color from an image, thresholding based on color can be used. However, color is heavily dependent on the lighting condition. Therefore, to improve the robustness [19], red-green-blue (RGB) color space is first converted to hue-saturation-value (HSV) color space. Still, the line can be detected as fragmented blobs due to the varying lighting condition. The reduce this fragmentation, morphological transformations are applied. Next, contours having reasonable areas are sought, along with smallest rectangles that enclose them. The rectangle that has the maximum fineness ratio in the image is assumed as the guiding rectangle. Finally, the relative bearing is set to the angle between the longer side of the rectangle and the vertical axis of the image. The geometric center of the guiding rectangle is used to estimate the relative 2D position of the line with respect to the camera. Figure 19 shows the detected red line and the center and relative bearing derived from the guiding rectangle.

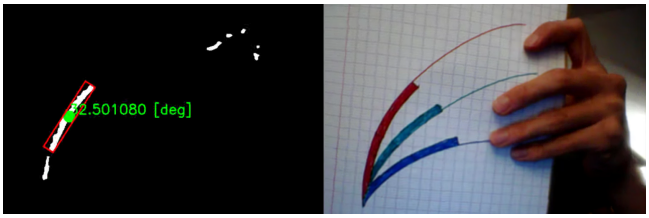

Figure 19: Detected red line and its computed relative bearing (left) in the image of hand-drawn lines of three different colors (right).

7 CONCLUSION

This draft paper intends to give some insight into the work process that went into preparing the drone for the IMAV 2023 competition. In it several aspects of the project are explained, from the architecture, to the dynamics involved, to the avionics that goes with it. Given the degree of autonomy required for the drone, computer vision is also a fundamental part of the project. Additionally, an oversight of the logical process represented by the state machine is also explained, corresponding to the actions performed in the different phases of the mission. It is worth noting that the paper is not a complete walk-through of the work doen since as of the day of the delivery, the project is not finished.

REFERENCES

[1] https://docs.px4.io/main/en/flight_stack/controller_diagrams.html#fixed-wing-attitude-controller.

[2] https://docs.px4.io/main/en/ros/ros2_comm.html.

[3] Alvika Gautam, Mandeep Singh, Pedda Baliyarasimhuni Sujit, and Srikanth Saripalli. Autonomous quadcopter landing on a moving target. *Sensors*, 22(3), 2022.

[4] Pengyu Wang, Chaoqun Wang, Jiankun Wang, and Max Q.-H. Meng. Quadrotor autonomous landing on moving platform. *Procedia Computer Science*, 209:40–49, 2022. Proceedings of the 2022 International Symposium on Biomimetic Intelligence and Robotics (ISBIR).

[5] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardos. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, dec 2021.

[6] Dean Brown. Decentering distortion of lenses. 1966.

[7] A. E. Conrady. Decentred Lens-Systems. *Monthly Notices of the Royal Astronomical Society*, 79(5):384–390, 03 1919.

[8] J. Kannala and S.S. Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1335–1340, 2006.

[9] Toby Collins and Adrien Bartoli. Infinitesimal plane-based pose estimation. *International Journal of Computer Vision*, 109(3):252–286, 2014.

[10] Denis Oberkampf, Daniel DeMenthon, and Larry Davis. Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, 63:495–511, 05 1996.

[11] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[12] Kenneth Levenberg. A method for the solution of certain non – linear problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[13] Donald W. Marquardt, E. I. duPont, Bennett R, and George C Burrell. Marquardt d w. an algorithm for least-squares estimation of nonlinear parameters. j. soc. indust. appl. math. 11:431-41, 1963. 2004.

[14] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2023.

[15] Alan Lukežič, Tomáš Vojíř, Luka Čehovin Zajc, Jiří Matas, and Matej Kristan. Discriminative correlation filter tracker with channel and spatial reliability. *International Journal of Computer Vision*, 126(7):671–688, jan 2018.

[16] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[17] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.

[18] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[19] Aisha Ajmal, Christopher Hollitt, Marcus Frean, and Harith Al-Sahaf. A comparison of rgb and hsv colour spaces for visual attention models. *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6, 2018.