

Energy-Sensitive Vision-Based Autonomous Tracking and Landing of a UAV

Georgios Zamanakos, Adam Seewald, Henrik Skov Midtiby, and Ulrik Pagh Schultz
SDU UAS Center, Mærsk Mc-Kinney Møller Institute, University of Southern Denmark
Contact email: ups@mmmi.sdu.dk

Abstract

In this paper, we present a robust, vision-based algorithm for autonomous tracking and landing on a moving platform in varying environmental conditions. We use a novel landing marker robust to occlusions to track the moving platform and the YOLOv3-tiny CNN to detect ground-based hazards in an agricultural use case. We perform all computations onboard using an NVIDIA Jetson Nano and analyse the impact on the flight time by profiling the energy consumption of the landing marker detection algorithm and YOLOv3-tiny CNN. Experiments are conducted in Gazebo simulation using an energy modeling tool to measure the energy cost as a function of QoS. Our experiments test the energy efficiency and robustness of our system in various dynamic wind disturbances. We show that the landing marker detection algorithm can be run at the highest QoS with only a marginal energy overhead whereas adapting the QoS level of YOLOv3-tiny CNN results in a considerable power saving for the system as a whole. The power saving is significant for a system executing on a fixed-wing UAV but only marginal if executing on a standard multirotor UAV.

I. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) are increasingly used for applications such as monitoring, surveillance, transportation of small payloads, and agricultural applications [1], [2]. One of the major constraints of such applications is their limited level of autonomy due to battery limitations. Extending the flying time of a UAV is normally done by having it land in order to replace or charge the battery before continuing the mission. Performing landings autonomously can however be challenging depending on the environment and whether the landing platform is stationary or mobile. Moreover, relying solely on the availability of a Global Positioning System (GPS) signal for autonomous precision landing is not considered safe, since GPS signals can be temporarily lost or even tampered with. As an alternative, in this paper we investigate the use of a novel vision-based autonomous landing system and evaluate its robustness towards environmental conditions such as visual disturbances and wind.

Extension of the flight time can be also achieved by using *energy-sensitive algorithms* that can reduce energy consumption by reducing the Quality of Service (QoS). With this approach, energy-costly computations such as computer vision are adapted by selecting the desired quality of service to match the available energy [3]. By combining energy-sensitive algorithms with autonomous landing capabilities, we aim to increase the total availability of the UAV to perform operations, by extending the flight time and using autonomous recharging when needed.

The main contribution of this paper concerns the experimental study of a robust vision-based algorithm for autonomous tracking and landing in varying environmental conditions. The algorithms are executed on an NVIDIA Jetson Nano companion computer controlling a simulated drone. The vision-based tracking and landing algorithms provide novel capabilities in terms of tolerance to visual disturbance and varying environmental conditions such as wind. Our experiments are based on an agricultural use-case where a multirotor UAV performs visual identification of ground-

58 based hazards while tracking and landing on a moving platform.

59 II. STATE OF THE ART

60 Vision-based autonomous landing on a marker has been extensively
61 studied by many researchers. Key distinctions include whether the marker
62 is on a moving platform, the type of the marker, the algorithms used to
63 detect it, as well as the mounted sensors on-board of the UAV.

64 For stationary platforms, one of the first experiments with vision-based
65 autonomous landing was conducted by Saripalli et al. [4]. Here, a heli-
66 copter with a color camera facing vertically towards the ground would
67 land on an "H"-shape pattern (similar to ones found on a helipad) using a
68 hierarchical behavior-based control architecture. In physical tests a marker
69 of 122×122 cm size was detected for a maximum altitude of 10 m. A
70 landing marker inspired by a QR code but consisting of three artificial
71 markers is demonstrated by Yuan et al. [5], and was shown to provide a
72 6 –Degree Of Freedom (DOF) pose over an altitude range of 0-20 m. Our
73 work is however focused on the ability to land on moving platforms.

74 Saripalli et al. [6] also demonstrated the use of a Kalman Filter to
75 track a moving platform. However, all the computations were performed
76 offline. Similarly, an ArUco marker was used as a landing marker by
77 Lee et al. [7] to detect a moving platform. The control of the UAV is
78 performed based on the error provided by the vision algorithm but all the
79 computations were performed off-board. Arrar, et al. [8] focus on extending
80 the detection range by using an AprilTag [9] as a landing marker. Again all
81 the computer vision algorithms were also executed off-board. Conversely,
82 a crucial aspect of our application is to perform all the computations
83 on-board, and to evaluate them according to their energy efficiency as
84 a function of QoS.

85 The design of the marker and choice of sensors can facilitate doing the
86 computations onboard. Chen et al. [10] utilized a marker consisting of
87 a circle and rectangles of different colors along with a LiDAR scanning

range finder for height estimation. The marker was detected by performing color segmentation on the incoming image frame. By fusing the height measurement from the LiDAR into the vision measurement, a relative pose of the UAV from the moving platform was obtained. A color segmentation approach was also implemented by Lee et al. [11]. A red rectangle was used as a landing marker while the detection was done by a vertically facing camera with a fish-eye lens. The setup accounted for a successful landing from an altitude of 70 m. In the above two cases an on-board companion computer is used to perform all the computations on the UAV. However, we in this work do not consider a color segmentation approach as a safe option, since for a realistic (outdoor) case it would be difficult, if not impossible, to ensure that the landing marker will be the only object of a specific color in the scene.

The use of a hybrid camera system consisting of a fish-eye IR camera and a stereo camera was demonstrated by Yang et al. [12]. An ArUco marker was used to mark the moving platform and a convolutional neural network (CNN) YOLO v3 was trained specifically for marker detection. A similar approach concerning the detection of a landing marker was demonstrated by Nguyen et al. [13] in which a specific CNN was trained to detect a landing marker pattern: successful detection of a $1\text{ m} \times 1\text{ m}$ marker size was demonstrated from a distance of 50 m. An AprilTag marker was used as a landing marker by Kyritsis et al. [14]. The identification of the AprilTag marker was performed through Graphics Processing Unit (GPU). In the above three cases, the researchers have utilized the companion's computer GPU to detect the landing marker. In the agricultural use-case addressed in this paper, the GPU is however needed for a CNN to detect ground hazards, and since the GPU cannot normally run different algorithms simultaneously, the CPU should be used for detecting the landing marker. By doing so, a different QoS can be chosen for each algorithm.

To account for the energy modeling of computer vision algorithms, we considered the work previously carried by Nardi et al. [15]. The au-

thors present SLAMBench, a framework that investigates Simultaneous Localisation and Mapping (SLAM) algorithms configuration alternatives for energy efficiency. In our work, we use `powprofiler`, a generic energy modeling tool [16]. This tool enables measuring the energy impact of different configurations of the ROS-based system implementing the agricultural use-case. The `powprofiler` tool is part of the TeamPlay toolchain, which aims to make tradeoffs between energy and other non-functional properties accessible to the developer. In this paper, we present extensions to `powprofiler` that facilitates the initial exploration of the energy usage of complex ROS-based systems.

Other approaches to energy modeling, such as the mission-based energy models studied by Sadrpour et al. [17], [18], focus mostly on ground-based autonomous vehicles instead of the UAVs. Morales et al. [19] extensively investigated the relation between motion and energy in a robot, but do not account for the energy required for computation. Energy modeling of mobile robots as carried by Mei et al. [20]–[22] has provided the ground for the concept of modeling computation for energy-sensitive algorithm design. Indeed, the approach employed in this paper has evolved from an energy-efficient motion planning technique in [22], a design strategy that allows accounting for motion and computations separately in [21], to an energy-efficient deployment algorithm in [20].

The battery in our system is considered in the context of a drone being able to perform its mission while accounting for the eventuality of a battery shortage; to this end, we investigated the approach presented by Berenz et al. [23], where a battery management mission-based dynamic recharge approach is presented. A set of recharge stations are used, along with self-docking capable robots. Our approach similarly allows landing on a moving platform for recharging, which is in the context of this paper is considered in the proximity of the drone. The actual landing is handled by the proposed algorithm, and we also account for the energy required for executing this algorithm during landing.

150 Taking into account varying environmental conditions and unpredictable
 151 movements of the platform to land on is relevant for the use of landing
 152 in outdoor, mobile scenarios. Regarding wind conditions, an AprilTag
 153 marker was used by Feng et al. [24] with a constant wind speed of 5 m/s
 154 as an external disturbance in a simulation environment. Nevertheless, a
 155 fluctuation in the wind's magnitude and direction is likely to happen in
 156 realistic cases. Concerning estimation of the moving platform's position
 157 and velocity, similar to our approach a Kalman Filter or Extended Kalman
 158 Filter (EKF) has been used for the estimation [8], [24], [25], whereas
 159 Yang et al. [12] constructed a velocity observer algorithm by calculating
 160 the actual moving distance of the moving platform over a period of time.

161 III. ENERGY-SENSITIVE MISSION DEPLOYMENT

162 A. Overall approach

163 The energy-sensitive design is a mission-oriented concept that adjusts
 164 the computations to the mission being performed while taking into account
 165 energy requirements, including energy consumed by actuation, computa-
 166 tion, and the presence of a limited power source. Specifically, in our agri-
 167 cultural use-case, the concept is employed to profile and eventually adapt
 168 the computationally heavy algorithms performing autonomous tracking,
 169 landing, and hazard detection. This adaptation enables energy-sensitivity,
 170 in the sense that QoS parameters can be modified to enable the mission
 171 to be completed at the highest possible QoS level that does not exceed
 172 the available energy budget. Tradeoffs between QoS parameters can be
 173 performed by an end-user, i.e., trading the robustness towards wind during
 174 landing for precision of hazard detection.

175 The energy-sensitive design using `powprofiler` relies on empirical
 176 experiments to measure the actual power consumption on the robot hard-
 177 ware [16]. In this paper, we focus on the initial profiling using of energy
 178 usage of the companion computer, which from the point of view of energy
 179 consumption can be studied independently from the specific drone it is

180 mounted in.

181 First, the developer specifies the maximum and minimum QoS level
 182 for each algorithm running on the system. During mission execution the
 183 levels are statically defined: automatic adaptation during different phases
 184 of the mission is currently being investigated and is considered future
 185 work. Then, the developer executes the system to empirically determine
 186 the power consumption. This can be done in two different ways:

187 1) Automatically using `powprofiler` to control the experiment execu-
 188 tion [16]. For a ROS-based system, we assume that the algorithms
 189 are wrapped as ROS nodes, and we require the developer to specify
 190 the QoS parameters using a ROS configuration. We use a config-
 191 uration file in a key-value pair format which is then interpreted by
 192 `powprofiler`, enabling `powprofiler` to iterate through all possible
 193 combinations and empirically sample the energy consumption of each
 194 combination of QoS parameters. Once all combinations have been
 195 iterated through, `powprofiler` automatically combines the energy
 196 consumption data into a complete model.

197 2) Semi-automatically using `powprofiler` to sample energy and com-
 198 bine the results of all experiments, but allowing the developer to
 199 control all aspects of the experiment execution. This approach is new
 200 and is described in more detail later in this section. Basically, a ROS
 201 node interfaces to `powprofiler` and is used by the developer to
 202 start/stop sampling in a given configuration. Once all experiments
 203 have been completed, `powprofiler` is invoked by the developer to
 204 combine the energy consumption data into a complete model.

205 Regardless of the approach, `powprofiler` builds a single model map-
 206 ping QoS to total system energy consumption. Coarse-grained sampling
 207 is employed to reduce the number of experiments, and missing values are
 208 automatically inferred from the others by the means of a multivariate linear
 209 interpolation.

210 In the context of this paper, sampling experiments are iterated in a

211 simulated environment with different configurations. For example, the au-
 212 tonomous tracking allows changing the tracking algorithm QoS in terms
 213 of frequency, the landing algorithm in terms of frequency, and hazard
 214 detection QoS in terms of frequency.

215 *B. Semi-automatic energy profiling*

216 A ROS node has been developed for the purposes of the semi-automatic
 217 approach used in this paper. This node allows automatic generation of the
 218 basic energy models that map time to the instantaneous power consump-
 219 tion. To activate this functionality, the developer simply publishes on a
 220 ROS topic to start the model generation, with `powprofiler` accounting
 221 for the invocation of an asynchronous thread which collects data from the
 222 energy sensors. Similarly, the developer publishes on another ROS topic
 223 to stop the model generation, while `powprofiler` finalizes collecting
 224 data from sensors, builds the basic energy model, and stores it for later
 225 processing.

226 Once all the basic energy models for the desired QoS ranges have been
 227 collected, QoS ranges are specified in a configuration file: the developer
 228 defines what QoS configuration corresponds to which basic model (instan-
 229 tantaneous power consumption as a function of time). Running `powprofiler`
 230 using this configuration file as a parameter generates the complete model
 231 that maps QoS to energy consumption.

232 IV. VISION-BASED AUTONOMOUS TRACKING AND LANDING

233 The vision-based autonomous tracking and landing can be split into
 234 four main sub-problems: detection of the moving platform, navigation,
 235 guidance, and control of the UAV. From an energy-sensitive design ap-
 236 proach, our focus is on the computer vision algorithms used to detect the
 237 moving platform and the parameterization by a QoS influencing energy
 238 consumption and performance, as described in Section IV-A. Furthermore,
 239 the navigation block is designed to increase the robustness and overall
 240 performance of the system, as described in Section IV-B. Last, a model

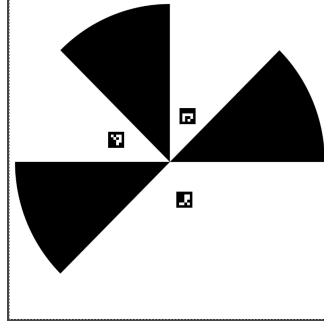


Fig. 1. The landing marker

of dynamically changing wind disturbances is analysed and described in Section IV-C, allowing the system to be tested in a more realistic simulation.

A. Detection of the moving platform

To mark the moving platform a special pattern is constructed, consisting of an n-fold marker [26] along with three ArUco markers [27] with different ids. This pattern will be referred to as the *landing marker* and can be seen in Figure 1. The n-fold marker is primarily used to detect the moving platform from a high altitude, while the ArUco markers are used as extra landmarks in case the marker is partially visible in the image frame.

To evaluate the computer vision algorithm for detecting the landing marker, real images of 640×480 pixel size were captured with an Intel RealSense D435 camera. In the Gazebo simulation a color camera with the same distortion coefficients as the Intel camera is used to output a 640×480 pixel image at 10 frames per second (fps).

To extract the pixel coordinates of the tip of the n-fold marker, a kernel size of 13×13 pixels consisting of a real and imaginary part is created. For every pixel in the image, a convolution is performed with this kernel and the magnitude of the convolution is stored. The pixel with the highest magnitude is considered as a candidate tip of the n-fold marker. For that

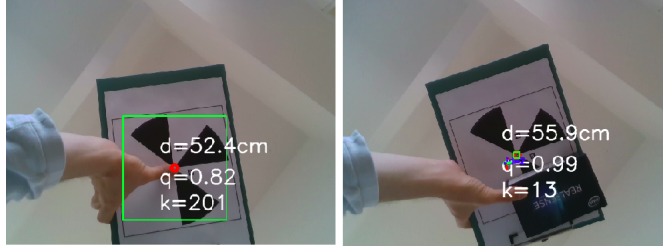


Fig. 2. Detection of the landing marker under different occlusions. On the left an occlusion on the tip of the n-fold marker and on the right an occlusion on a sector of the n-fold marker.

261 candidate pixel only, an estimation of the orientation/phase of the marker
 262 is made and an overall normalized quality score between 0.0 and 1.0 is
 263 calculated. If the score is above a desired threshold value then the pixel is
 264 accepted as the tip of the n-fold marker. If an n-fold marker is detected,
 265 the result will be the pixel coordinates of the tip of the n-fold marker along
 266 with its orientation/phase.

267 Since a convolution is a computationally expensive process, an increase
 268 in the kernel size would also increase the computation time and therefore
 269 the energy consumption. However, a higher computation time and energy
 270 consumption is preferred over a non-detection of the n-fold marker. To
 271 balance between energy consumption and effective marker detection, an
 272 adaptive kernel selection function is created to ensure the selection of a
 273 proper kernel size based on a threshold quality score value. In Figure 2
 274 an Intel RealSense D435 Depth camera is used to capture the image and
 275 measure also the distance d from the n-fold marker. Based on a desired
 276 quality q , the proper kernel size k is selected. It is seen that an occlusion on
 277 the tip of the n-fold marker results in a significant increase in the selected
 278 kernel size.

279 To detect the ArUco markers the standard OpenCV library is used and
 280 if any ArUco markers are detected, their central pixel coordinates and pose
 281 are stored.

282 The next step is to convert these pixel coordinates into a real-world

relative position $[X, Y, Z]$ according to a local coordinate frame. The origin of the local coordinate frame $[0, 0, 0]$ is defined as the center of the landing marker and alignment according to the North, East, Down (*NED*) frame. The available sensor measurements and sensor fusion algorithms from the flight controller are used in this process. The PX4 flight controller outputs through mavlink messages the altitude and the attitude of the UAV (roll, pitch, yaw). For the Z component, the altitude of the UAV from the flight controller's EKF is used. To obtain the X, Y components, an algorithm is constructed to convert pixel coordinates into real world X, Y coordinates in meters:

- 1) The pose of the camera in UAV's *BODY* frame is calculated by utilizing the roll and pitch IMU data.
- 2) The normalized coordinates of the four image corners, according to the camera's horizontal, vertical field of view and the camera's pose from step 1, are calculated.
- 3) The coordinates of the four image corners (in meters), with respect to the UAV's *BODY* frame, are determined by using the normalised coordinates from step 2 along with the UAV's altitude. The result is a projection plane of the image corners on the ground.
- 4) The perspective homography matrix is calculated between the two planes, the image plane and the world plane from step 3.
- 5) The homography matrix from step 4 is used to convert the pixel coordinates of the tip of the n -fold marker from the image plane, into coordinates (in meters) in the UAV's *BODY* frame.
- 6) The coordinates from step 5 are in respect to the UAV's *BODY* frame. To convert them into the UAV's *NED* frame, the yaw IMU data from the flight controller is used.
- 7) The coordinates from step 6, are converted from UAV's *NED* frame into the landing site's local coordinate frame.
- 8) For ArUco markers only, an offset vector in the x, y axis is added depending on the distance of each ArUco marker from the tip of the

314 n-fold marker. It is assumed that this vector is prior known.

315 The mean measurements from the detected n-fold and/or ArUco markers
316 are used to determine the position and orientation of the landing marker.

317 The result is an $[X, Y, Z]$ relative position of the UAV from the moving
318 platform along with the yaw orientation of the landing marker.

319 *B. Navigation*

320 The purpose of the navigation block is to provide an accurate prediction
321 for the state of the UAV at any given time. This is important because it
322 will allow us to process images at different fps according to a desired
323 QoS. Furthermore, the overall robustness of the system is increased in
324 case the moving platform is not detected in every image frame. A velocity
325 estimator for the moving platform will also be implemented as a part of
326 the navigation block.

327 The variables of interest that describe the state of the UAV for this
328 project are:

- 329 • The relative position of the UAV from the moving platform, x_k , .
- 330 • The attitude of the UAV $[roll, pitch, yaw]$, obtained from the flight
331 controller's IMUs.
- 332 • the velocity of the UAV, \dot{x}_k , in *NED* frame, obtained from the flight
333 controller's EKF.
- 334 • the acceleration of the UAV, \ddot{x}_k , in *NED* frame, obtained by differ-
335 entiating the velocities.

336 The altitude, attitude, velocity and acceleration variables of the UAV's
337 state are obtained from the flight controller's onboard sensors and already
338 implemented sensor fusion algorithms (EKF). To fuse those state variables
339 with the obtained position measurements from Section IV-A, a Kalman
340 Filter is implemented. The measurements from the flight controller are
341 used in the prediction step.

I think we might have used a bad name for this value. It currently overlaps with the estimated position of the UAV relative to the landing target.

These values are also described on the next page.

Prediction Step:

$$\hat{x}_k = F \cdot \hat{x}_{k-1} + G_k \cdot u_k, \quad (1a)$$

$$P_k = F \cdot P_{k-1} \cdot F^\top + Q_k, \quad (1b)$$

where $x_k \in \mathbb{R}^2$ is the position of the UAV, $u_k = [\dot{x}_k \quad \ddot{x}_k \quad \dot{x}_k^l]^\top$ is its control with x_k, x_k^l being the positions of the UAV and the moving platform respectively. Given $I := I_2$, a 2×2 identity matrix, the matrix $F = I$, the noise covariance matrix $Q_k = \Delta t_k \cdot \sigma_{\text{IMU}}^2 \cdot I$, and $\sigma_{\text{IMU}}^2 \in \mathbb{R}$ is the variance of the velocities retrieved from the flight controller's data. Further, Δt_k is the time interval the flight controller output the data (usually around 33 ms). The input matrix is given by:

Henrik, little lengthy with implicitly saying "is a 2 by 2 identity matrix"; what about this one?

$$G_k = \begin{bmatrix} \Delta t_k & 0 & \frac{\Delta t_k^2}{2} & 0 & -\Delta t_k & 0 \\ 0 & \Delta t_k & 0 & \frac{\Delta t_k^2}{2} & 0 & -\Delta t_k \end{bmatrix},$$

Is this the right word?
[Ad3:] actually no/yes—
diatribes between
controls people stating
that the model of the
system is perfect since is
a model [cite Hector]

Ad3: Actually G_k might still go easily inline. Something like (commented because latex generates error). Up to you

Ad3: the itemize doesn't really add any information the reader doesn't know so far (takes space also)... Makes sense in a thesis, report, not in a paper.

- x_k is the position of the UAV in the landing site's local coordinate system (aligned with NED frame),
- Δt_k is the time interval the PX4's EKF outputs the v_x, v_y data, usually around 33ms,
- \dot{x}_k is the linear velocity of the UAV in NED frame, estimated from the PX4's EKF,
- \ddot{x}_k is the linear acceleration of the UAV in NED frame, estimated on the companion computer from differentiating the velocities.

- \dot{x}_k^l is the linear velocity of the moving platform in NED frame, estimated from the velocity estimator for the moving platform,
- σ_{IMU}^2 is the variance of the velocities based estimated by the PX4's EKF.

Ad3: from here to windspeed, I just apply my earlier math suggestion. Henrik, Georgios please check/change/revise.

In the correction step, the observed measurements from the downward looking camera are used to correct and update the estimated position of the UAV \hat{x}_k . However due to the computation time needed to detect the landing marker, the incoming measurement is delayed by a time $\Delta t_k^{\text{obs}} \in \mathbb{R}_{\geq 0}$. Physically, let us define the displacement $\tilde{x}_k^{\text{obs}} \in \mathbb{R}$ as

$$\tilde{x}_k^{\text{obs}} := \bar{x}_k \Delta t_k^{\text{obs}}, \quad (2)$$

where $\bar{x}_t \in \mathbb{R}$ is the mean velocity measured empirically. Such displacement is later employed in the observed measurements. The correction step of the Kalman Filter is

$$K_k = P_k \cdot H^T (H \cdot P_k \cdot H^T + R_k)^{-1}, \quad (3a)$$

$$\hat{x}_k^f = \hat{x}_k + K_k (x_k^{\text{obs}} + \tilde{x}_k^{\text{obs}} - \hat{x}_k), \quad (3b)$$

$$P_k^f = (I - K_k \cdot H) P_k, \quad (3c)$$

where $H = I$, and x_t^{obs} is the position measured empirically at time t . Let us further define $R_t := \sigma_{\text{obs}}^2 I$ where $\sigma_{\text{obs}}^2 \in \mathbb{R}$ is the variance of the velocities measured empirically.

The values of the estimated state \hat{x}_t^f , and error covariance matrix P_t^f , are used as input to the next iteration of the prediction step \hat{x}_{t+1}, P_{t+1} .

A velocity estimator is also constructed to determine the magnitude and direction of the moving platform's velocity vector. The magnitude is calculated by differentiating two sequential X, Y positions of the tractor, obtained by detecting the n-fold marker from the vertically facing camera as explained in section IV-A and taking into consideration the UAV's *NED* velocities according to the PX4's EKF. A low-pass filter is used to

377 provide a smooth estimation of the velocity's magnitude by filtering out
 378 high frequency noise. High frequency noise can be caused by oscillations
 379 of the UAV along with a fast update rate in the landing marker detection
 380 algorithm.

Ad3: confusing a bit; if we differentiate to estimate the velocity (dx/dt if x is the position) how we use low pass filter to estimate the same velocity we differentiated?

382 In the agricultural use-case the moving platform is likely to change its
 383 direction up to 180 degrees. Furthermore, it is assumed that the moving
 384 platform is a nonholonomic system, like a tractor. To compensate for
 385 sudden turns, the moving platform's yaw orientation will be taken into
 386 account. Based on the velocity's magnitude and moving platform's yaw
 387 orientation, the moving platform's velocity (\dot{x}_k^l) in *NED* frame can be
 388 obtained.

389 C. Wind Disturbances

Ad3: the force is F . Why are we then calling it *force* later on?

391 The exact and accurate estimation of the applied wind forces on a body
 392 is a complex matter studied by the field of fluid dynamics. We use a
 393 simplified approach, as follows. We assume that the wind will be applied
 394 on an area of 0.09 m^2 . Such an area is emulating a UAV with extra payload
 395 attached on its frame. Two different wind speeds of 8 m/s and 12 m/s will
 396 be used to calculate the applied wind forces on that area. The wind forces
 397 are considered to be applied on the center of gravity of the UAV with
 398 direction parallel to the ground.

399 The magnitude of the applied force, corresponding to a certain wind
 400 speed is calculated by the following equations [28], [29]:

$$\begin{aligned} F &= p_d \cdot A \\ p_d &= \frac{\rho \cdot v^2}{2} \end{aligned} \tag{4}$$

Ad3: can be one line, and actually, I would trust the reader to search for this one. Is really basic physics.

where F is the force, p_d is the dynamic pressure, A is the area of the applied pressure, ρ is the density of the air (around 1.2 kg/m^3), v is the wind velocity in m/s.

By solving the above equations the applied force on the UAV is found to be 3.45 N for an 8 m/s wind speed, and 7.76 N for a 12 m/s wind speed. The wind forces will be applied on the UAV in Gazebo simulation, to test the performance of the whole system. This will be done by constructing a program that will apply the wind forces on the virtual UAV.

To simulate a wind pattern the wind direction and magnitude must be defined. The wind direction is assumed to remain the same for the whole duration of the experiments. That direction vector is defined as $[0.8, 0.2]$ according to Gazebo's (x, y) axes. The magnitude of the wind is calculated as follows:

- An initial wind force of either 3.45 N or 7.76 N is chosen.
- An update cycle of 5.5 s is chosen between two different applied forces.
- A random float number between 0.9 and 1.2 is chosen at every update cycle. This number is defined as *Random_noise*.
- The applied wind force is calculated as:

Ad3: this is just $F_k = w \cdot F \cdot r$? where w, r is the direction vector you defined earlier $[0.8, 0.2]$ and r a random number.

$$force_x = -x_{norm} * force * Random_noise$$

$$force_y = -y_{norm} * force * Random_noise$$

where $x_{norm} = 0.8$, $y_{norm} = 0.2$

For an altitude of 6.0 m and above, $force = force_{init}$ where $force_{init}$ is either 3.45 N or 7.76 N

- For a UAV altitude of 6.0 m and below, the wind force decreases as:

431 $force = force_{init} * (altitude/6.0)$

432 • For a UAV altitude of 3 m and below, $force = 0.5N$.

433 This model is created to simulate different scaling in the wind's magnitude
434 and will be used for all simulated tests.

435 V. EVALUATION

436 We evaluate our approach in terms of the quality of the overall func-
437 tionality and the energy efficiency of the algorithms. All algorithms are
438 executed on an embedded companion computer interfaced to a simulation
439 running on a standard computer.

440 A. Use-case: agricultural safety

441 We evaluate our approach based on a simulated use-case where a multi-
442 rotor UAV identifies hazardous objects around a moving platform, and
443 lands on the moving platform to recharge. No communication link is
444 considered between the moving platform and the UAV and no GNSS
445 positioning is assumed to be available. The system can thus be considered
446 as a fallback for fault-tolerance.

447 Object detection and classification is performed by feeding the input
448 image from the downward facing camera into a *YOLOv3-tiny* CNN [30]
449 implemented in ROS [31]. Four different classes are selected: cars, humans,
450 tractors, and cows. Based on the CNN's predictions and onboard sensors,
451 the UAV maps the detected objects onto a 2D map.

452 A simulated field is created in Gazebo with objects placed in random
453 positions and orientations as seen in Figure 3.

454 Pre-trained weights for *YOLOv3-tiny*, were initially used but the per-
455 formance on detecting objects from a downward facing camera was not
456 satisfactory. Since a dataset for detecting the above four classes from a
457 top view was not available, we created an artificial dataset based on the
458 Gazebo models. The dataset consists of 1200 images, 300 for each class.
459 We trained the *YOLOv3-tiny* by its default training parameters for 5000
460 epochs, for an input image size of 416×416 pixels.



Fig. 3. On the left, a top view of the Gazebo scene. On the right, a view of the UAV attempting a landing on the moving platform

B. Experimental setup

All experiments are performed in Gazebo simulation under Ubuntu 18.04 and ROS Melodic on a i7-8550U 1.8GHz (4.0GHz Boost), 8GB DDR4 laptop. The *PX4* Software In The Loop (SITL) Firmware v1.10.2 is used as the flight controller and the *IRIS* quadcopter is used as the UAV platform. A vertically facing RGB camera is placed on the UAV providing a 640×480 pixel image at 10 fps. An *Nvidia Jetson Nano* with Ubuntu 18.04 and ROS Melodic is used as the UAV's companion computer. All the computer vision, guidance, and control algorithms execute on the *Nvidia Jetson Nano*, similar to how they would be deployed if the Nano was a companion computer in a physical drone. Energy profiling is performed directly on the *Nvidia Jetson Nano* using `powprofiler` as outlined in Section III-B.

Two groups of experiments are conducted. The first group evaluates the energy consumption and QoS of the *tracking* mode and the second group evaluates the energy consumption and QoS of the *landing* mode. For both groups, the experiments start with the tractor moving at a constant speed of 0.3 m/s, according to a square path similar to that of a plowing tractor, and the UAV taking off and hovering at an altitude of 25 m. After reaching the desired altitude, the UAV starts searching for the landing marker in the image frame. Once the landing marker is detected, the UAV commences its actions.

483 In *tracking* mode, the UAV will follow the moving platform at a fixed
 484 altitude and use its vertically facing camera to map the environment,
 485 while in *landing* mode the UAV will follow the moving platform and
 486 gradually lower its altitude until it lands on it. Both *tracking* and *landing*
 487 modes are tested under three different cases of no wind disturbances, wind
 488 disturbances of 8 m/s and wind disturbances of 12 m/s according to the
 489 wind model described in Section IV-C.

490 C. Results

491 The first group of experiments was conducted to test the *tracking* mode
 492 and evaluate its energy efficiency and QoS. For the energy evaluation,
 493 eight tests were executed for different fps rates for the *YOLOv3-tiny* ROS
 494 node (4fps, 1fps, 0.5fps, 0.1fps) and the *landing marker* detection ROS
 495 node (10fps, 0.5fps) as seen in Figure 4. For a 4fps update rate for
 496 *YOLOv3-tiny*, a power consumption of 6.30 W is observed while for a
 497 1fps and 0.1fps update rates, the power consumption drops to 4.8 W
 498 and 3.9 W accordingly. By reducing the update rate for *landing marker*
 499 detection, from 10fps to 0.5fps, a further power saving of 0.15 W- 0.19 W
 500 is achieved.

501 For the QoS evaluation, twelve tests were executed for different fps
 502 rates for the *YOLOv3-tiny* ROS node (4fps, 0.1fps) and for the *landing*
 503 *marker* detection ROS node (10fps, 0.5fps) under three different cases
 504 of wind disturbances. The QoS is determined as the number of correctly
 505 detected objects. An object is considered to be correctly detected if it is
 506 detected within a distance of 2 m from the its actual position and classified
 507 with the correct class. The detection results can be seen in Figure 5. The
 508 best results were obtained for a high fps update rate in *YOLOv3-tiny* and
 509 *landing marker* detection, under no wind disturbances, since 28 out of the
 510 32 objects were detected. Nevertheless, for the same high fps values but
 511 with a wind speed of 12 m/s, only 18 out of 32 objects were correctly
 512 detected.

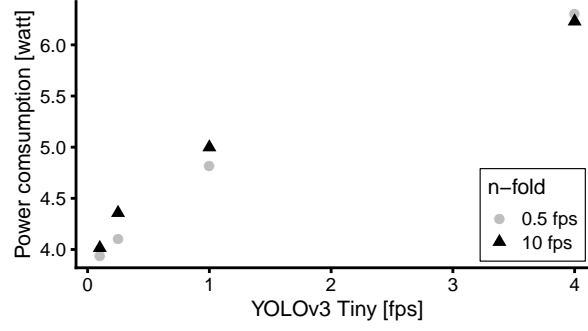


Fig. 4. Power consumption during tracking mode. YOLOv3 Tiny fps: 0=0.1fps, 1=0.5fps, 2=1fps, 3=4fps.

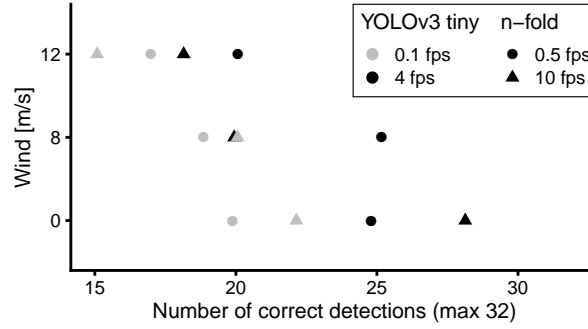


Fig. 5. Number of correctly detected objects under different conditions. Grey color denotes a 0.1fps and black color denotes a 4fps update rate in *YOLOv3-tiny* ROS node. Circles denote a 0.5fps and triangles denote a 10fps update rate in the *landing marker* detection ROS node.

513 The second group of experiments was conducted to test the *landing*
 514 mode and evaluate its energy efficiency and QoS. For the energy evalua-
 515 tion, eight tests were executed for different fps rates for the *landing marker*
 516 detection ROS node (10fps, 2fps, 1fps, 0.5fps) using two different cases.
 517 In the first case, the kernel size remains fixed at 22×22 pixels and in the
 518 second case an adaptive selection kernel size algorithm is used. The landing
 519 time is also taken under consideration as seen in Figure 6. The largest
 520 difference in the power consumption is 0.14 W and is observed between

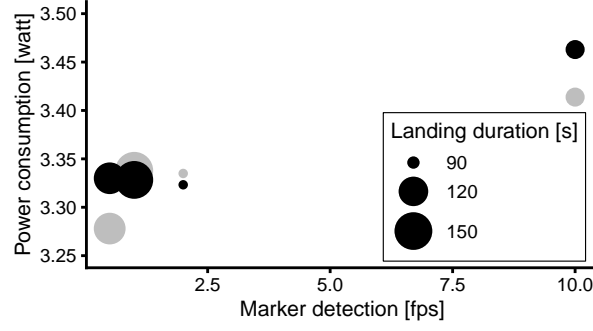


Fig. 6. Power consumption during landing mode. The black circles denote a fixed kernel size while the grey circles denote an adaptive kernel size.

the 0.5fps and the 10fps update rate for the *landing marker* detection. However the landing time is reduced by 30 s when using the 10fps update rate. The adaptive kernel size in most cases outperforms the fixed kernel size by 0.11 W. Furthermore the adaptive kernel size can compensate for marker occlusions which will increase the overall robustness of the system.

For the QoS evaluation, twelve tests were executed for different fps rates for the *landing marker* detection ROS node (10fps, 2fps, 1fps, 0.5fps) under three different cases of wind disturbances. The QoS is determined as the mean squared error (MSE) between the predicted position of the moving platform, from the navigation block, and the moving platform's actual position. Four different altitude bins were used as seen in Figure 7. A large MSE of around 3 m^2 is observed for an altitude greater than 20 m for a 0.5fps rate, while an MSE close to zero is observed for an altitude of less than 5 m for an update rate of 10fps. Furthermore, wind disturbances do not seem to have an influence on the MSE. We believe the larger MSE for the y coordinates compared to the x coordinates is caused by a sudden change of the moving platform's direction on the y axis.

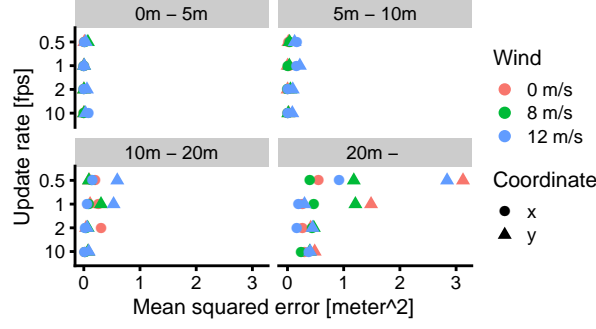


Fig. 7. Position error during landing and how it depends on the marker detection rate at different altitudes and wind disturbances. Circles denote errors in the x direction and triangles errors in the y direction.

D. Discussion

The experiments show that both tracking mode and landing mode are supported by the system, in a simulated environment with a moving platform and random wind conditions. Moreover, the performance of both modes is sensitive to the QoS, with a high success rate of both modes at high QoS levels, and significantly lower performance at lower QoS levels.

The potential energy savings from having an energy-sensitive algorithm that can adapt its QoS by changing the fps values for the *YOLOv3-tiny* and *landing marker* ROS nodes should be seen in relation to the total energy consumption of the UAV. As a concrete example, consider a DJI Phantom 4 multirotor and a Sky-Watch Cumulus fixed-wing (the fixed-wing would need to circle while tracking and would need VTOL capabilities to land, but we nevertheless include it for comparison). We estimate¹ that the Phantom uses roughly 140 W while cruising whereas the Cumulus uses roughly 40 W while cruising. The maximal saving gained from changing the *YOLOv3-tiny* rate is $6.30 \text{ W} - 3.9 \text{ W} = 2.4 \text{ W}$ whereas the maximal

¹From information on the respective product pages regarding battery capacity and maximal flight time.

554 saving gained from changing the *landing marker* rate is 0.2 W. For the
 555 Cumulus, there is thus a 6.5% potential energy saving, whereas the po-
 556 tential energy saving only is 1.9% for the Phantom. For the Cumulus this
 557 saving is considered large enough to significantly impact the flying time
 558 of the drone, with a total energy saving of 23.4 kJ. For the Cumulus, the
 559 potential saving from adapting the *landing marker* QoS is however only
 560 0.5%.

561 For the tracking mode, changing the *landing marker* rate provided a
 562 minor saving of 0.14 W, but at the cost of increased landing time. There-
 563 fore, although the higher-QoS computer vision algorithm is marginally
 564 more expensive by 0.14 W, the UAV will overall save energy because of
 565 a reduced flight time.

566 VI. CONCLUSION AND FUTURE WORK

567 In this paper we presented a robust, energy-sensitive, vision-based al-
 568 gorithm for autonomous tracking and landing in varying environmental
 569 conditions, by experimentally executing all the necessary algorithms on
 570 the *Nvidia Jetson Nano* companion computer. Our experiments show that
 571 the proposed computer vision algorithms for detecting the moving plat-
 572 form can be run at the highest QoS level with only a marginal energy
 573 overhead, whereas adapting the QoS level of *YOLOv3-tiny* CNN results in
 574 a considerable power saving for the system as a whole. This power saving
 575 is significant if the system was executing on a fixed-wing UAV, but only
 576 marginal if executing on a multirotor UAV.

577 In terms of future work, we are interested in automatically adapting
 578 the QoS level to the available battery, and in testing this approach on a
 579 physical drone.

580 ACKNOWLEDGMENT

581 This work is supported and partly funded by the European Union's Hori-
 582 zon2020 research and innovation program under grant agreement No. 779882
 583 (TeamPlay).

REFERENCES

- [1] F. G. Costa, J. Ueyama, T. Braun, G. Pessin, F. S. Osório, and P. A. Vargas, "The use of unmanned aerial vehicles and wireless sensor network in agricultural applications," in *2012 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2012, pp. 5045–5048.
- [2] E. Salami, C. Barrado, and E. Pastor, "Uav flight experiments applied to the remote sensing of vegetated areas," *Remote Sensing*, vol. 6, no. 11, pp. 11 051–11 081, 2014.
- [3] A. Seewald, H. Garcia de Marina, H. S. Midtiby, and U. P. Schultz, "Mechanical and computational energy estimation of a fixed-wing drone," in *Proceedings of the 2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2020, p. to appear.
- [4] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme, "Vision-based autonomous landing of an unmanned aerial vehicle," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 3. IEEE, 2002, pp. 2799–2804.
- [5] H. Yuan, C. Xiao, S. Xiu, W. Zhan, Z. Ye, F. Zhang, C. Zhou, Y. Wen, and Q. Li, "A hierarchical vision-based localization of rotor unmanned aerial vehicles for autonomous landing," *International Journal of Distributed Sensor Networks*, vol. 14, no. 9, 2018.
- [6] S. Saripalli and G. S. Sukhatme, "Landing on a moving target using an autonomous helicopter," in *Field and service robotics*. Springer, 2003, pp. 277–286.
- [7] D. Lee, T. Ryan, and H. J. Kim, "Autonomous landing of a vtol uav on a moving platform using image-based visual servoing," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 971–976.
- [8] O. Araar, N. Aouf, and I. Vitanov, "Vision based autonomous landing of multirotor uav on moving platform," *Journal of Intelligent & Robotic Systems*, vol. 85, no. 2, pp. 369–384, 2017.
- [9] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3400–3407.
- [10] X. Chen, S. K. Phang, M. Shan, and B. M. Chen, "System integration of a vision-guided uav for autonomous landing on moving platform," in *2016 12th IEEE International Conference on Control and Automation (ICCA)*. IEEE, 2016, pp. 761–766.
- [11] H. Lee, S. Jung, and D. H. Shim, "Vision-based uav landing on the moving vehicle," in *2016 International conference on unmanned aircraft systems (ICUAS)*. IEEE, 2016, pp. 1–7.
- [12] T. Yang, Q. Ren, F. Zhang, B. Xie, H. Ren, J. Li, and Y. Zhang, "Hybrid camera array-based uav auto-landing on moving ugv in gps-denied environment," *Remote Sensing*, vol. 10, no. 11, p. 1829, 2018.
- [13] P. H. Nguyen, M. Arsalan, J. H. Koo, R. A. Naqvi, N. Q. Truong, and K. R. Park, "Lightdenseyolo: A fast and accurate marker tracker for autonomous uav landing by visible light camera sensor on drone," *Sensors*, vol. 18, no. 6, p. 1703, 2018.
- [14] S. Kyristis, A. Antonopoulos, T. Chaniakakis, E. Stefanakis, C. Linardos, A. Tripolitsi-

- 624 otis, and P. Partsinevelos, “Towards autonomous modular uav missions: The detection,
625 geo-location and landing paradigm,” *Sensors*, vol. 16, no. 11, p. 1844, 2016.
- 626 [15] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. Kelly, A. J. Davison,
627 M. Luján, M. F. O’Boyle, G. Riley *et al.*, “Introducing slambench, a performance and
628 accuracy benchmarking methodology for slam,” in *2015 IEEE International Conference
629 on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5783–5790.
- 630 [16] A. Seewald, U. P. Schultz, E. Ebeid, and H. S. Midtiby, “Coarse-grained computation-
631 oriented energy modeling for heterogeneous parallel embedded systems,” *International
632 Journal of Parallel Programming*, pp. 1–22, 2019.
- 633 [17] A. Sadrpour, J. Jin, and A. G. Ulsoy, “Experimental validation of mission energy pre-
634 diction model for unmanned ground vehicles,” in *2013 American Control Conference*.
635 IEEE, 2013, pp. 5960–5965.
- 636 [18] —, “Mission energy prediction for unmanned ground vehicles using real-time
637 measurements and prior knowledge,” *Journal of Field Robotics*, vol. 30, no. 3, pp.
638 399–414, 2013.
- 639 [19] J. Morales, J. L. Martinez, A. Mandow, A. J. García-Cerezo, and S. Pedraza, “Power
640 consumption modeling of skid-steer tracked mobile robots on rigid terrain,” *IEEE
641 Transactions on Robotics*, vol. 25, no. 5, pp. 1098–1108, 2009.
- 642 [20] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, “Deployment of mobile robots with energy
643 and timing constraints,” *IEEE Transactions on robotics*, vol. 22, no. 3, pp. 507–522,
644 2006.
- 645 [21] —, “A case study of mobile robot’s energy consumption and conservation tech-
646 niques,” in *ICAR’05. Proceedings., 12th International Conference on Advanced
647 Robotics, 2005*. IEEE, 2005, pp. 492–497.
- 648 [22] —, “Energy-efficient motion planning for mobile robots,” in *IEEE International
649 Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*, vol. 5.
650 IEEE, 2004, pp. 4344–4349.
- 651 [23] V. Berenz, F. Tanaka, and K. Suzuki, “Autonomous battery management for mobile
652 robots based on risk and gain assessment,” *Artificial Intelligence Review*, vol. 37, no. 3,
653 pp. 217–237, 2012.
- 654 [24] Y. Feng, C. Zhang, S. Baek, S. Rawashdeh, and A. Mohammadi, “Autonomous landing
655 of a uav on a moving platform using model predictive control,” *Drones*, vol. 2, no. 4,
656 p. 34, 2018.
- 657 [25] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, “Vision-
658 based autonomous quadrotor landing on a moving platform,” in *2017 IEEE International
659 Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, 2017, pp. 200–207.
- 660 [26] Henrik Skov Midtiby, “N-fold marker tracker repository,” [https://github.com/
661 henrikmidtiby/MarkerLocator](https://github.com/henrikmidtiby/MarkerLocator), 2015.
- 662 [27] OpenCV, “Detection of ArUco markers,” [https://docs.opencv.org/3.4/d5/dae/tutorial_
663 aruco_detection.html](https://docs.opencv.org/3.4/d5/dae/tutorial_aruco_detection.html), 2020.

- 664 [28] NASA, “Dynamic pressure (NASA),” [https://www.grc.nasa.gov/WWW/K-12/airplane/](https://www.grc.nasa.gov/WWW/K-12/airplane/dynpress.html)
665 [dynpress.html](https://www.grc.nasa.gov/WWW/K-12/airplane/dynpress.html), 2020.
- 666 [29] J. D. Anderson Jr, *Fundamentals of aerodynamics*. Tata McGraw-Hill Education, 2010.
- 667 [30] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv*, 2018.
- 668 [31] M. Bjelonic, “YOLO ROS: Real-time object detection for ROS,” [https://github.com/](https://github.com/leggedrobotics/darknet_ros)
669 [leggedrobotics/darknet_ros](https://github.com/leggedrobotics/darknet_ros), 2016–2018.