Journal of Electrical Engineering, Vol. 75, No. 4, 2024, pp. 304-316

sciendo

PAPERS

Multi-sensor fusion for robust indoor localization of industrial UAVs using particle filter

Eduard Mráz[1], Adam Trizuljak[2], Matej Rajchl[1], Martin Sedláček[1], Filip Štec[1], Jaromír Stanko[1], Jozef Rodina[1]

Robotic platforms including Unmanned Aerial Vehicles (UAVs) require an accurate and reliable source of position information, especially in indoor environments where GNSS cannot be used. This is typically accomplished by using multiple independent position sensors. This paper presents a UAV position estimation mechanism based on a particle filter, that combines information from visual odometry cameras and visual detection of fiducial markers. The article proposes very compact, lightweight and robust method for indoor localization, that can run with high frequency on the UAV's onboard computer. The filter is implemented such that it can seamlessly handle sensor failures and disconnections. Moreover, the filter can be extended to include inputs from additional sensors. The implemented approach is validated on data from real-life UAV test flights, where average position error under 0.4 m was achieved.

Keywords: UAV, UAS, sensor fusion, particle filter, localization

1 Introduction

One of the fundamental challenges in robotics is to determine the position of the robot within the environment. This is a significant task for mobile robots, especially among flying robots (unmanned aerial vehicles – UAV, drones) in industrial applications. In these applications, autonomous behaviour helps to speed up various processes, but it also places very high requirements on the UAV in terms of robustness, reliability and operational safety. It is crucial to estimate the pose of the UAV continuously, because discrete behaviour (such as sudden position jumps) can cause system instability. This can cause the UAV to crash, which poses a risk to the operating personnel and can lead to damage to the UAV or the operating environment. Moreover, it is desirable to utilize multiple independent sources of position estimation in order to increase the accuracy of localization, as well as to provide redundancy in the case of failure of some of the sensors [1].

The required accuracy of the system usually depends on many factors, such as what type of indoor space is required for the UAV to navigate. How many and what type of obstacles are in this space and various other factors. Therefore, it is difficult to set up general rules for the required accuracy or performance.

In this article, warehouse inventory is simulated. This is the environment where all the tests take place and where the whole system was developed. Thus, it is the main application of proposed system, but the system is not limited to it. Warehouse inventory is a perfect task to be done by an UAV [2] because it moves vertically with more ease than humans. However, it might be difficult to achieve high repeatability with such a system. High repeatability in this case means minimizing possibility of failures and accidents. Otherwise, the system would not be considered reliable.

There are multiple requirements that need to be satisfied with such applications for UAV's. Since the UAV performs flights in 3 dimensions, the localization system needs to provide a 3D pose estimate. The pose estimate must be accurate and must be provided in real-time, so that it can be used by the UAV navigation system for real-time position control. It is intended to use the system in visually well-structured industrial environments – warehouses, industrial halls etc. Many solutions such as [3] and [4] rely on a pre-constructed map. Every solution must use some type of map, unless the facility is equipped with very accurate indoor positioning system such as VICON [5]. In our application, the mapping problem is aided by using fiducial markers positioned at known locations

[1] Faculty of Electrical Engineering and Information Technology at the Slovak University of Technology, Ilkovičova 3, 841 04 Bratislava 1, Slovak Republic
[2] Photoneo s.r.o., Plynárenská 6, 821 09 Bratislava, Slovak Republic
eduard.mraz@stuba.sk, trizuljak@photoneo.com, matej.rajchl@stuba.sk, martin.sedlacek@stuba.sk, filip.stec@stuba.sk, jaromir.stanko@stuba.sk, jozef.rodina@stuba.sk

https://doi.org/10.2478/jee-2024-0037, Print (till 2015) ISSN 1335-3632, On-line ISSN 1339-309X

throughout the environment. This process is expected to require only a one-time effort during installation, when the positions of markers would be accurately measured by standard tools like laser measuring tool.

Using multiple sensors requires the use of a fusion mechanism to produce a combined state estimate. This paper proposes a solution in the form of a particle filter (PF). PF is used to fuse multiple position estimation sources into one final, robust and reliable estimate of a UAV's pose. PF allows the fusion of multiple sensors/methods with various qualities, in this case position estimates from two Intel RealSense T265 visual odometry cameras, and position estimates from visual detections of fiducial markers. The core idea is to combine the real-time visual odometry measurements, which can exhibit drift over time, with accurate absolute position information from detections of fiducial markers, which are sparsely positioned in the operating environment. The novelty of this approach relies on using two Intel T265 cameras. To this date, authors are not aware of any localization system, which was designed specifically to fuse two Intel T265 cameras for one final position estimate. Also, in the analysed sensor-fusion articles, there is no focus on sensor disconnection or malfunction. On the other hand, our work focuses on solving the sensor disconnection/malfunction scenarios, which explicitly emphasizes higher level of robustness. Another advantage of our proposed approach is that it can use ArUco markers, but the system is also fully usable without them.

The implemented filter is intended to be used as the main real-time position estimator for a UAV in indoor industrial applications. The implementation of the system allows it to run on a small form factor on-board computer, which also encourages the miniaturization of the UAV itself. The price of the sensors and hardware used in this work is a fraction of the price compared to the works analyzed in the Related work section. The final cost is often omitted in research papers, but it is proven to play an important role in industrial applications, such as warehouse inventories [2].

2 Related work

The interest to develop a fully autonomous UAV is obvious in the robotic community. Numerous systems rely on SLAM (simultaneous localization and mapping) methods for localization, such as ORB-SLAM [6], RTAB-MAP [4] and SOFT-SLAM [7]. SLAM methods solve a problem, where no map exists prior to the localization process, so the map is being continually constructed during the simultaneous calculation of the robot's position. The output from SLAM methods can then be used as a localization source for the robot. Notably, SLAM algorithms tend to be computationally expensive, which can prohibit their use on platforms with limited computational resources.

In response to this issue, several devices have been introduced to the market in the recent years that present a more integrated position estimation solution. A prime example is the Intel RealSense T265 [8]. It includes a stereo camera pair coupled with an inertial measurement unit (IMU), and the data is processed by an embedded application-specific processor that executes a highly optimized visual-inertial odometry algorithm. Thanks to that the device has a low power consumption, low computational requirements and is easy to use and integrate into robotic platforms. However, its proprietary visual odometry algorithm may suffer from drift and occasional discontinuities in pose estimation.

A common approach to robot localization is Monte Carlo Localization (MCL). It utilizes the particle filter technique, which uses a large set of randomly sampled weighted particles to represent a posterior distribution of the possible system states given the current sensor observations and prior state probabilities. Thus, the particles (samples) are concentrated in the area of high probability [9]. This is more efficient than the Markov localization algorithm, which maintains a probability distribution over the entire state space [9]. Another advantage of the PF is that it can represent even highly non-linear, non-Gaussian systems with almost arbitrary process and sensor models [9-11]. In contrast, the Kalman filter (KF), which is another popular approach to localization and sensor fusion, relies on a linear system model.

Notable examples of popular PF-based localization systems are AMCL [12] and AMCL-3D [3], that can be used to localize a robot in a known environment map. AMCL stands for Adaptive Monte Carlo Localization. In principle, Monte Carlo mechanism is used to fuse multiple sources (sensors) to produce one output - robot position estimate in GNSS denied environments using particle filtering principles. In the case of AMCL, a scan from a 2D laser sensor is aligned with a known 2D map of the environment, which is used for position estimation in 2D. AMCL-3D extends this approach into three dimensions. It uses an RGB-D sensor to match a point cloud with a previously constructed 3D map of the environment to estimate the robot's position in 3D. As an addition, AMCL-3D uses stationary beacon (such as UWB radio) measurements to satisfy robustness and improve accuracy of the system.

A disadvantage of PF is that depending on the model complexity and size of the state space, it may require a very large number of particles to achieve accurate results [13], which in turn makes it computationally expensive. However, the weight of each particle can be computed independently of other particles, which means that these computations can be highly parallelized. Thus, modern GPU acceleration techniques can be used to

dramatically improve filter performance [14]. As far as sensor-fusion methods are concerned, there were 4 recent articles analysed [15-18].

The work [18] contains a standard approach to the fusion of several sensors. It achieves interesting results, but also provides several reasons for the existence of the system proposed in this article. The work deals with the issue of real-time UAV state estimation in various environments using the Extended Kalman Filter (EKF) algorithm. EKF is used for the fusion of data from several heterogeneous sensors. These sensors include IMU, magnetometer, barometer, GNSS receiver, optical flow sensor, LiDAR and RGB-D camera. Interestingly, the work presents a hybrid architecture of multi-sensor data fusion (MSDF), which combines local program nodes for primary data fusion and a secondary program node for global fusion using the EKF algorithm. The accuracy in the results reached an error of 10 cm and less in the position and ±3° in the heading of the UAV. Although the system brings high accuracy, the use of expensive LiDAR, the high weight of the drone (8.5 kg) and the dependence on a standard PC represent significant disadvantages. These sensors cost thousands of euros. In addition, the work uses a full-fledged Intel i5 series processor. Nevertheless, the estimate ran with a frequency of only 10 Hz.

The paper [17] presents an adaptive and robust sensor fusion approach for indoor UAV localization. The main goal is to improve the estimation of the state (position) of the UAV using the Moving Horizon Estimator (MHE) algorithm in combination with ArUco markers. This approach uses Gaussian Mixture Models (GMMs) to model sensor uncertainty, thereby increasing localization accuracy. The system was tested and compared with the Vicon reference system. In these tests, it achieves error of 6 cm and less. The problems arise at several points. For example, several ArUco markers are used to simulate several sensors. However, these markers are captured by only one camera – that is, one sensor. This pre-empts the use of heterogeneous sensors, or at least does not verify the possibility of their use. Since ArUco markers represent the only estimate of the system's position, it is a condition that they are always in the camera's sight. Such a restriction markedly limits the range of spaces where such localization may take place.

The work [15] deals with data fusion using EKF. The work presents a system of visual-inertial odometry (VIO) based on AprilTag visual markers and data from the IMU. Proposed system allows for a high density of markers, which minimizes the loss of information at higher UAV speeds. The dense distribution of markers ensures that even at higher speeds, several markers are always within range, thus increasing the reliability of localization. The experimental results show that the average localization error is lower than 11 cm. The plus point is that the system has been verified on two different UAV platforms (quadrotor and hexrotor) in different flight conditions. The results show good localization consistency, indicating that the system is adaptable to different types of UAVs and to different flight scenarios. The problem is the absolute dependence on visual markers. Additionally, these markers were placed on the ground, which may be unacceptable in some industrial environments. In addition, the marker map proposed in the article occupies a large area.

The article [16] deals with the problem of locating UAVs indoors, where GPS and magnetometer data are unavailable or unreliable. The proposed solution uses a variation of the EKF algorithm - error state extended Kalman filter (ES-EKF) for sensor fusion. ES-EKF aims for accurate and robust position and orientation estimations by exploiting the linear dynamics of the error state to optimally predict and update the state error covariance. Simply put, this more advanced version of the EKF additionally includes individual sensor errors in the prediction and update steps. The authors declare that in this way it is ensured that only measurements with a smaller error are included in the fusion. The article presents a system for the fusion of a relatively large number of heterogeneous sensors and location sources: the Ultra-wideband (UWB) Pozyx system, visual odometry from the Intel RealSense T265 camera and the SLAM algorithm from the LiDAR sensor. The limitation for the LiDAR used is the same as for the work [18]. UWB limits system deployment due to the need to deploy active electronic devices across an industrial space.

Analysing strict sensor fusion methods confirms recent scientific efforts to provide robust and accurate solution for indoor localization. As shown in analysis of related work, many of these efforts provide these qualities. However, it has been proven that these efforts are often not price effective, or it is not possible to deploy them in industrial environments due to the aforementioned limitations. The system proposed in this article aims to solve the problems pointed out in the analysis.

The rest of this paper is structured as follows. First, section 3 gives an overview of the utilized UAV platform, describing the available sensors and their data outputs. Next, section 4 details the implementation of the particle filter fusion mechanism, including the pre-processing of sensor data, particle weight update, resampling and state estimation, as well as sensor failure handling. The implemented filter is then evaluated in the section 5 using data from real-life flight tests.

3 Platform description

Development and testing were carried out on the Discovery quad-rotor UAV developed by Airvolute s.r.o. It is a modular robotic UAV solution intended for various industrial applications such as warehouse inventories. Thanks to its modular design, it is also appropriate for educational, development, and testing purposes. It is powered by the NVIDIA Jetson Xavier NX compute module [19] with a 6-core ARM CPU and a GPU with 384 CUDA cores. This compute module offers large opportunities for the use of deep learning and GPU-accelerated parallel algorithms. The software stack of this platform is based on the Robot Operating System (ROS) and allows for quick deployment of user applications. Although the platform allows to use any ROS version compatible with OS Ubuntu 20.04 LTS, the algorithms described in this article are implemented on ROS1 Noetic version.

The Discovery UAV is equipped with multiple redundant sensors to increase robustness in case of hardware or software failure. The system obtains raw position estimates using two independent Intel RealSense T265 visual odometry cameras and pose estimation based on ArUco fiducial markers detection. Additionally, the UAV carries an Intel RealSense D455 depth and RGB camera and a lidar sensor that measures the height above ground. These sensors are also used for reactive navigation in the environment. The following subsections describe the on-board sensors and their data outputs, that are relevant for this work.

Fig. 1. The Airvolute Discovery UAV used to perform the flight tests in this work

3.1 Flight controller

The UAV is controlled by The Cube autopilot module [20] with a modified version of the Ardupilot firmware. Ardupilot runs the complete control loop including trajectory tracking and position, velocity and angular rate PID controllers. The firmware was modified to publish the target velocity signal

$$\mathbf{u}_{\mathrm{B}} = [\,\mathbf{v}_{\mathrm{B}}\ \omega_{\psi}\,]^{\mathrm{T}},$$

which consists of the target linear velocity vector $\mathbf{v}_{\mathrm{B}} = [v_x\ v_y\ v_z\,]^{\mathrm{T}}$ in the body frame B of the UAV and a target yaw rate ω_{ψ} .

3.2 Visual odometry

The Intel RealSense T265 [8] is a camera which is able to compute visual-inertial odometry based on visual feature tracking and a built-in IMU. It features a stereo fisheye camera pair, which is used for feature detection. The camera outputs a 200 Hz stream with estimated position vector $\mathbf{p}_{\mathrm{O}}^{\mathrm{W}} = [p_x\, p_y\, p_z]^{\mathrm{T}}$, orientation quaternion $\mathbf{q}_{\mathrm{O}}^{\mathrm{W}} = [q_x\, q_y\, q_z\, q_w]^{\mathrm{T}}$, linear velocity $\mathbf{v}_{\mathrm{O}} = [v_x v_y v_z]^{\mathrm{T}}$ and angular velocity $\boldsymbol{\omega}_{\mathrm{O}} = [\omega_x\ \omega_y\ \omega_z]^{\mathrm{T}}$. All quantities are estimated relative to a gravity-aligned global frame W, with its origin and initial heading being initialized to zero upon camera startup. Importantly, as with any visual odometry algorithm, the T265 pose estimate is subject to drift and it may also exhibit discontinuities.

The T265 camera uses a proprietary visual odometry algorithm, which does not directly provide covariance matrices for the pose estimates. Instead, a four-level estimate confidence value $c \in \{0,1,2,3\}$ is provided, where 0 indicates low confidence (i.e., bad tracking quality) and 3 indicates high confidence. This value is therefore used by the *realsense-ros* package [21] to compute an "approximate" odometry covariance value

$$\sigma_{\mathrm{O}} = \sigma_0 \times 10^{3-c}$$

where σ_0 is a baseline covariance parameter, typically set to 0.01 m.

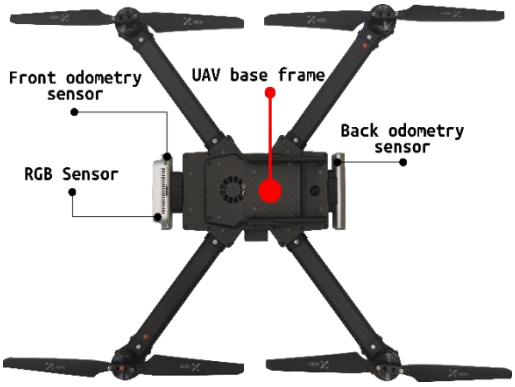

Fig. 2. Sensor placement of Airvolute Discovery UAV

In our platform, two T265 sensors are used with the goal of increasing system robustness. One sensor is placed in the front of the drone and another one on the back side, see Fig. 2. This not only helps with making hardware redundancy possible but also each camera can

see different parts of the environment. This can help with visibility issues, e.g., when one camera is partially obstructed, blinded by over-exposure or when it faces a low-texture environment. In such cases the estimate confidence of this camera is decreased, but the estimate from the other camera can still be used. In the rest of this paper, measurements from the front and back camera will be denoted by subscripts F and B, and the respective world frames will be denoted F and B.

3.3 Localization with visual markers

The T265 cameras estimate their pose relative to their initial position, however the transformations between their respective reference frames and the global frame of the environment are initially unknown. To address this problem, multiple ArUco visual markers [22, 23] are placed throughout the environment at known locations along the expected flight trajectory of the UAV.

Figure 3 illustrates the coordinate frames used by the system. *W* stands for World frame, *M* stands for Marker frame and *U* stands for UAV frame. Ultimately, the position of the UAV in the World frame needs to be determined. Orientation of the *M* frame is identical to the *W* frame. The transform of a single marker in the world frame $\mathbf{T}_{\mathrm{M}}^{\mathrm{W}}$ is set by placing the marker on well measured positions in space where localization takes place. This transform is assumed to be constant and since the position of the marker in the global frame is also known this makes the process known correspondence problem. Using the known marker positions, it is possible to find the transform (position and orientation) $\mathbf{T}_{\mathrm{U}}^{\mathrm{W}}$ between world and the UAV usign the $\mathbf{T}_{\mathrm{U}}^{\mathrm{M}}$ (UAV to marker) transform. This transform is determined by a camera pose estimation process based on ArUco markers described in [24]. Whenever a marker is detected by an on-board camera, this method is used to estimate the position and rotation (heading) of the UAV with respect to the world frame *W*, defined by the vector $\mathbf{p}_{\mathrm{A}}^{\mathrm{W}} = \left[p_x\, p_y p_z\right]^{\mathrm{T}}$ and the quaternion $\mathbf{q}_{\mathrm{A}}^{\mathrm{W}} = \left[q_x\, q_y\, q_z\, q_w\right]^{\mathrm{T}}$, which is then converted to Euler angles representation to obtain the global heading $\psi_{\mathrm{A}}^{\mathrm{W}}$. The covariance of this detection is σ_{A}.

One marker is positioned such that it is in view of the UAV immediately after takeoff. Position estimate obtained from the initial (starting) marker is only accepted, when it is considered stable. Stability is achieved when the variance of pose estimate drops below a specific threshold [24]. This initial detection is then used to obtain the initial position of the UAV $\mathbf{p}_{\mathrm{A0}}^{\mathrm{W}}$. The initial rotation is set to $\psi_{\mathrm{A0}}^{\mathrm{W}} = 0$.

4 Particle filter

Particle filter is a sequential Monte Carlo method that uses a large number of weighted particles to statistically represent possible system states. In general, a particle filter works by first propagating the particle states using a system model and the current control command. Afterwards, all particles are weighted based on how well they correspond with the current sensor measurements. The state estimate is then computed as a weighted mean of all particles. Lastly, the set of particles is resampled according to their calculated weights.

The decision to use a particle filter as the sensor fusion mechanism was based on several factors. Firstly, the RealSense T265 camera does not provide a true covariance value, as discussed in section 3.2. This, when combined with various non-linearities of the UAV system, would make it difficult to use as an input to a more traditional Kalman filter, which relies on a linear system model and accurate covariance values. While the Extended and Unscented Kalman filter techniques can be applied to non-linear systems, the particle filter can achieve better performance [10]. Moreover, the particle filter is able to represent arbitrary, non-analytical distributions [11]. A disadvantage is the increased computational load due to the high number of particles and the associated weight update calculations, however these could be massively parallelized using GPU acceleration [14], leveraging the power of the on-board Jetson Xavier NX processor. The current version of the particle filter presented in this paper is implemented in the Python language and uses only the CPU for calculations, however it still achieves sufficient real-time performance.

In this work, the goal of the particle filter is to estimate the state of the UAV

$$\hat{\mathbf{x}} = \left[\,\hat{\mathbf{p}}\ \hat{\psi}\,\right]^{\mathrm{T}}$$

consisting of the global 3D position vector $\hat{\mathbf{p}} = \left[\,\widehat{p_x}\ \widehat{p_y}\ \widehat{p_z}\,\right]^{\mathrm{T}}$ and global heading $\hat{\psi}$. In the algorithm, the estimation of roll and pitch angles was omitted because the UAV primarily follows planar trajectories, and the onboard autopilot system, Ardupilot, already delivers sufficiently accurate estimates for these angles. Heading is however more complicated, since indoor use of compass can prove difficult, requiring calibration for each of the different industrial environments which can change dramatically over the course of flight (e.g., flying from open space to an aisle between the racks). Therefore, to keep the system robust heading needs to be estimated using different approach and so it is estimated as one of the states of the PF. Our particle filter is tailored towards high-level position control and trajectory tracking, where estimation of global heading is critical.

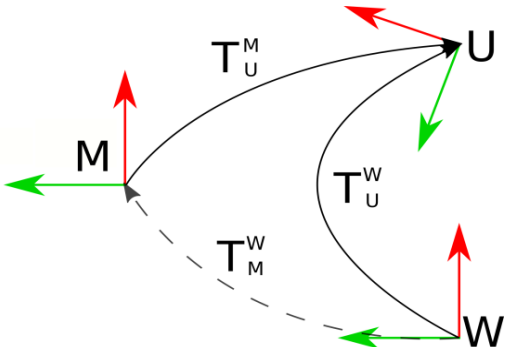

Fig. 3. Basic coordinate frames of the proposed PF localization system.

At a given time, step *i*, the *i*-th particle contains a position vector $\mathbf{p}_t^{(i)}$ and heading $\psi_t^{(i)}$:

$$\mathbf{x}_t^{(i)} = \left[\, \mathbf{p}_t^{(i)} \; \psi_t^{(i)} \,\right]^{\mathrm{T}}$$

The state of the UAV is statistically represented by a set of *N* weighted particles

$$\mathbf{S}_t = \{\mathbf{x}_t^{(i)};\; w_t^{(i)}\},\; i \in 1 \ldots N,$$

where the individual particles $\mathbf{x}_t^{(i)}$ are random samples from the state space. The weight of the *i*-th particle $w_t^{(i)} \in (0; 1)$ describes the posterior probability that this particle accurately represents the true system state given its prior probability and current sensor inputs. The implemented particle filter algorithm is summarized in Algorithm 1. The following sub-sections provide detailed description of each step of the computation.

Algorithm 1 Particle filter summary

Input: Number of particles *N*, Initial particle position $\mathbf{x}_0$ and variance σ_0
Initialize particle set $\mathbf{S} = \{\mathbf{x}^{(i)};\; w^{(i)}\},\; i \in 1..N, \mathbf{x}^{(i)} \sim \mathcal{N}(\mathbf{x}_0, \sigma_0),\; w^{(i)} = 1/N$
loop
  Input: Pose estimates $\mathbf{p}_\mathrm{F}, \mathbf{p}_\mathrm{B}, \mathbf{p}_\mathrm{A}$ in World frame, Velocity command $\mathbf{u}$
  for *i* = 1 to *N* **do** » Prediction
    Sample process noise $\mathbf{u}^* \sim \mathcal{N}(0, \sigma_\mathrm{u})$
    $\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i)} + (\mathbf{u} + \mathbf{u}^*)\mathrm{d}t$ » Move *i*-th particle by velocity command + noise
  end for
  for *i* = 1 to *N* **do** » Selective weight update
    if $\mathbf{p}_\mathrm{F}, \psi_\mathrm{F}$ is available **then** » Update w.r.t. front odometry estimate
      $w^{(i)} \leftarrow w^{(i)} * W\left(\mathbf{p}_\mathrm{F}, \psi_\mathrm{F}, \mathbf{x}^{(i)}\right)$
    end if
    if $\mathbf{p}_\mathrm{B}, \psi_\mathrm{B}$ are available » Update w.r.t. back odometry estimate
      $w^{(i)} \leftarrow w^{(i)} * W\left(\mathbf{p}_\mathrm{B}, \psi_\mathrm{B}, \mathbf{x}^{(i)}\right)$
    end if
    if $\mathbf{p}_\mathrm{A}, \psi_\mathrm{A}$ are available » Update w.r.t. ArUco pose estimate
      $w^{(i)} \leftarrow w^{(i)} * W\left(\mathbf{p}_\mathrm{A}, \psi_\mathrm{A}, \mathbf{x}^{(i)}\right)$
    end if
  end for
  if $N_{eff} < N$- th
    Resample $\mathbf{S}_t$ given $w^{(i)}$
  end if
  Normalize weights s. t. $\sum w^{(i)} = 1$
  $\hat{\mathbf{x}} \leftarrow estimate\ (\mathbf{S}_t)$ » Compute state estimate
end loop

4.1 Input processing

The acquired sensor data described in section 3 needs to be pre-processed before it can be used by the particle filter. This consists mainly of transforming all measurements into a common global frame and is detailed in the following subsections. This processing occurs before every particle filter update step, i.e., at 20 Hz.

The desired velocity command $\mathbf{u}_\mathrm{B} = \left[\mathbf{v}_\mathrm{B}\ \omega_\psi\right]^{\mathrm{T}}$ in the UAV body frame *B* is published by the flight controller at 20 Hz. This velocity must first be transformed into the world frame *W* using the quaternion $\mathbf{q}_\mathrm{A}$ from the ArUco marker detection:

$$\mathbf{u}_\mathrm{W} = \left[\, \mathbf{v}_\mathrm{W}\ \omega_\psi \,\right]^{\mathrm{T}} = \left[\, \mathbf{q}_\mathrm{A}^{\mathrm{W}}\, \mathbf{v}_\mathrm{B}\ \ \mathbf{q}_\mathrm{A}^{\mathrm{W}^{-1}}\ \omega_\psi \,\right]^{\mathrm{T}},$$

where $\mathbf{q}_\mathrm{A}^{\mathrm{W}^{-1}} = \left[-q_x\ -q_y\ -q_z\ q_w\right]^{\mathrm{T}}$ represents the quaternion conjugation operation. This becomes the control input that is used to propagate the particles in the filter predict step. To keep the nomenclature concise, the control input in the global frame, the control input in the global frame will be further labelled as $\mathbf{u}$, omitting the World frame subscript.

The front and back visual odometry cameras provide estimates of odometry poses, consisting of position vectors $\mathbf{p}_\mathrm{F}, \mathbf{p}_\mathrm{B}$ and orientations $\psi_\mathrm{F}, \psi_\mathrm{B}$. These measurements are transformed into the world frame using the world frame estimate from ArUco, and static transforms that describe the position of the sensor with respect to the center of gravity of the UAV. For instance, by placing an odometry sensor on the back of the UAV, there is a static transform representing X, Y, Z position offset and yaw rotation offset by 180 degrees.

In the first stage, it is necessary to rotate all those coordinate frames to match the global orientation of the UAV. Translating is not necessary, because only position increments are used instead of the absolute position estimate. For this purpose, the initial position and orientation – $\mathbf{p}_{A0}^{W}$ and ψ_{A0}^{W} respectively – detected from initial ArUco camera pose estimate is used. So, to transform odometry measurements $\mathbf{p}_F, \mathbf{p}_B, \psi_F, \psi_B$ to the global frame, rotation transforms $\mathbf{R}_F^W$ and $\mathbf{R}_B^W$ need to be applied to both position vectors $\mathbf{p}_F, \mathbf{p}_B$ and orientations ψ_F, ψ_B. These transformations are obtained by multiplying the odometry yaw measurements with the initial ArUco yaw transform ψ_{A0}^{W}

$$\mathbf{R}_F^W = \psi_{A0}^W \, \psi_F^{-1},$$

$$\mathbf{R}_B^W = \psi_{A0}^W \, \psi_B^{-1}.$$

For the system to become immune to sensor malfunction, the PF process cannot hang on sensor disconnection/malfunction event. For this reason, a fail-safe mechanism was implemented, which periodically checks the newest received message from every sensor. If the message is older than a specified threshold, the source is discarded from the state estimation process. The threshold value depends on the frequency that the sensor is working with.

From the particle filter perspective this means that the sensor data from this sensor will not have its' weight function evaluated and the specific step will be skipped. This is one of the advantages of using particle filter since not all of the weighting functions need to be evaluated at each iteration of the algorithm. This helps to deal with situations in which the measurements do not come synchronized but appear at different intervals or different rates.

Another condition that needs to be handled is the reconnection of the sensor in the middle of the process. The problem is that after reconnecting, the visual odometry estimate usually starts in its own local frame from zero position. After it has been detected that the sensor is successfully working again, the transform from local sensor frame to the world frame needs to be applied. Again, only rotation transform is needed, because only position and heading increments are used, so translation is not needed. The heading of the last PF pose estimate is used as a world frame orientation in this case. Reconnect rotation transform is set by computing a difference between the PF heading (yaw angle) $\hat{\psi}$ and the current heading of the odometry sensor ψ – which was reset after reconnection. Calculating reconnect rotation transform $\mathbf{R}^R$ is symbolically denoted by using a function *QuaternionFromEuler* which converts single euler angles (roll, pitch, yaw) to one complete quaternion.

$$\mathbf{R}^R = \mathrm{QuaternionFromEuler}\left(0,0,\hat{\psi} - \psi\right)$$

Finally, after the reconnection event, the odometry position estimate is transformed to the world frame as follows:

$$\mathbf{p}_S^W = \mathbf{p}\mathbf{R}^W\mathbf{R}^R$$

The last transform consists of rotating a static translation vector $\mathbf{t}^S$, which represents translation of the odometry (or any other) sensor frame with respect to UAV base frame. Illustration of configuration of the UAV used in this work is displayed in Fig. 2. To transform $\mathbf{t}^S$ into the world frame, it needs to be in the world frame of the sensor's position estimate $\mathbf{p}^W$. Define $\mathbf{R}_C^W$ as a rotation holding current yaw estimate ψ of a specific sensor. Furthermore, it needs to include a static rotation $\mathbf{R}^S$ of the sensor with respect to the UAV base frame. It should be noted that $\mathbf{t}^S$ and $\mathbf{R}^S$ depend on the physical sensor placement on the UAV and need to be measured prior to flight. Final transform of $\mathbf{t}^S$ into a world frame is then defined as

$$\mathbf{t}^W = \mathbf{t}^S \, \mathbf{R}_C^W \, \mathbf{R}^S$$

Then, it is possible to add this vector to the current position estimate coordinates of a specific sensor

$$\mathbf{p}^W = \mathbf{p}_{S(x,y,z)}^W + \mathbf{t}^W$$

Calculating odometry increments is done by calculating the distance between two consecutive measurements from T265 visual odometry sensors

$$\triangle \mathbf{p} = \mathbf{p}_t^W - \mathbf{p}_{t-1}^W,$$

$$\triangle \psi = \mathrm{atan2}\left(\sin(\psi_t - \psi_{t-1}),\, \cos(\psi_t - \psi_{t-1})\right)$$

Odometry increments are then used to obtain an estimated global position and heading from the sensor

$$\widehat{\mathbf{p}_{t+1}} = \widehat{\mathbf{p}_t} + \triangle \mathbf{p},$$

$$\widehat{\psi_{t+1}} = \left(\widehat{\psi_t} + \triangle \psi\right) \bmod 2\pi$$

These calculations are applied to both the front and back T265 measurements, which produces the estimated global position from the front camera $\widehat{\mathbf{p}_F}$, $\widehat{\psi_F}$ and back camera $\widehat{\mathbf{p}_B}$, $\widehat{\psi_B}$. This method is specific for sensors used with the proposed PF mechanism. The process of increment calculation may vary depending on different hardware.

The reason behind using odometry increments instead of absolute position estimates from odometry sensors is to avoid biasing the particle filter with accumulated odometry error. Error accumulation is fairly common in computing visual odometry. Currently used odometry sensors – T265 cameras – are closed systems which perform optimization to some extent. To follow the proposed PF architecture, it is desired to use the simplest form of odometry measurements, in other words – most unprocessed data which is possible to obtain. Then, after obtaining such data, state estimation optimization is performed inside the PF algorithm.

To keep the notation clear, this superscript *W* will be omitted in the rest of this paper, thus **p** will denote a position measurement already transformed into the World frame.

4.2 Filter initialization

After takeoff, the filter waits until the initial estimate of ArUco position $\boldsymbol{p}_{A0}$ and heading ψ_{A0} is available. The positions and rotations of all particles $i \in 1 \ldots N$ are then initialized by generating samples from normal distributions that are centered about this initial position estimate:

$$\boldsymbol{p}^{(i)} \sim N(\boldsymbol{p}_{A0}, \sigma_{A0})$$
$$\psi^{(i)} \sim N(\psi_{A0}, \sigma_{\psi A0})$$

The covariance parameters σ_{A0} and $\sigma_{\psi 0}$ control the initial spread of the particles, which expresses the uncertainty of the initial pose estimate. Since the initial position of the UAV within the environment is well known, relatively (w.r.t. to the size of the environment) low values $\sigma_{A0} = 1.0$ m and $\sigma_{\psi 0} = \pi/4$ were used. All particles are assigned a uniform weight $w_t^{(i)} = 1/N$.

4.3 Prediction

Once the filter is initialized, its update process is executed upon every received velocity command $\mathbf{u} = \begin{bmatrix} \mathbf{v} \ \omega_\psi \end{bmatrix}^{\mathrm{T}}$, which occurs at 20 Hz. At a given time step *t*, the position of every particle $\mathbf{x}_t^{(i)} = \begin{bmatrix} \mathbf{p}_t^{(i)} \ \psi_t^{(i)} \end{bmatrix}^{\mathrm{T}} \in \mathbf{S}_t$ is predicted using a linear motion model

$$\mathbf{p}_{t+1} = \mathbf{p}_t + (\mathbf{v} + \mathbf{v}^*)\,\mathrm{d}t$$

where $\mathbf{v}^* \sim \mathrm{N}(0, \sigma_{\mathrm{up}})$ is a sample of the Gaussian process noise with covariance σ_{up}, which describes the uncertainty in tracking of the desired velocity signal. The heading angle prediction follows a similar principle

$$\psi_{t+1} = \left(\psi_t + \left(\omega_\psi + \omega_\psi^*\right)\mathrm{d}t\right) \bmod 2\pi$$

where the sample of the Gaussian process noise $\omega_\psi^* \sim N(0, \sigma_{\mathrm{u}\psi})$ with covariance $\sigma_{\mathrm{u}\psi}$ models the uncertainty of yaw rate tracking. The modulo operation ensures that the predicted angle remains within the interval $\langle 0;\ 2\pi \rangle$.

4.4 Selective weight update

The next step is to update the weight of each particle with respect to multiple measurement sources. Through the incremental calculation described in section 4.1, the front and back T265 cameras provide the estimated positions $\widehat{\mathbf{p}_{\mathrm{F}}}, \widehat{\mathbf{p}_{\mathrm{B}}}$ and headings $\widehat{\psi_{\mathrm{F}}}, \widehat{\psi_{\mathrm{B}}}$ with corresponding covariances $\widehat{\sigma_{\mathrm{F}}}, \widehat{\sigma_{\mathrm{B}}}$**.** If an ArUco marker is currently being detected, it provides the estimated pose $\widehat{\mathbf{p}_{\mathrm{A}}}$, heading $\widehat{\psi_{\mathrm{A}}}$ and covariance $\widehat{\sigma_{\mathrm{A}}}$. The idea is to selectively incorporate these measurements only when they are available or reliable.

We define a function *W* that computes the likelihood of a particle $\mathbf{x}_t^{(i)} = \left[\mathbf{p}_t^{(i)}\ \psi_t^{(i)}\right]^{\mathrm{T}}$ based on the position observation **p** with covariance σ_p, heading observation ψ with covariance σ_ψ. This function is divided into two partial functions W_p and W_ψ that separately handle the position and heading update

$$W\left(\mathbf{p}, \sigma_p, \psi, \sigma_\psi, \mathbf{x}_t^{(i)}\right) = W_p\left(\mathbf{p}, \sigma_p, \mathbf{x}_t^{(i)}\right) W_\sigma\left(\psi, \sigma_\psi, \mathbf{x}_t^{(i)}\right)$$

The position likelihood function compares the position observation **p** with the position of the particle $\mathbf{p}_t^{(i)}$

$$W_p\left(\mathbf{p}, \sigma_p, \mathbf{x}_t^{(i)}\right) = p\left(\mathbf{p} \middle| \mathbf{p}_t^{(i)}\right)$$

where $p(\cdot)$ is the probability density function of a normal distribution with mean $\boldsymbol{\mu} = \mathbf{p}_t^{(i)}$ and covariance σ_p.

When calculating the angular difference $\triangle\psi^{(i)}$ between a heading measurement ψ with the heading of a particle $\psi_t^{(i)}$, the wrap-around between 0 and 2π must be handled. This is accomplished by defining an angular difference function $D(\alpha_1, \alpha_2)$ that yields the smaller angle between the two headings, which is in the interval $\langle -\pi;\ \pi \rangle$

$$D(\alpha_1, \alpha_2) = \mathrm{atan2}\left(\sin(\alpha_1 - \alpha_2),\ \cos(\alpha_1 - \alpha_2)\right),$$
$$\triangle\psi^{(i)} = D\left(\psi, \psi_t^{(i)}\right)$$

It assumes that particles with near-zero values of $\triangle\psi^{(i)}$ must be close to the angle observation ψ, thus their weight should be high, and vice versa. The heading likelihood function W_ψ performs this comparison by computing the probability density function $\mathrm{p}\left(\triangle\psi^{(i)}\right)$ of a normal distribution with mean $\mu = 0$ and and covariance σ_ψ.

$$W_\sigma\left(\psi, \sigma_\psi, \mathbf{x}_\mathbf{t}^{(\mathbf{i})}\right) = \mathrm{p}\left(\triangle\psi^{(i)}\right)$$

The weight update function *W* is then used to update the prior particle weights $w_t^{(i)}$ with respect to all available sensor measurements

$$w_{t+1}^{(i)} = w_t^{(i)} \cdot W\left(\widehat{\mathbf{p}_{\mathrm{F}}}, \widehat{\sigma_{\mathrm{F}}}, \widehat{\psi_{\mathrm{F}}}, \widehat{\sigma_{\mathrm{F}}}, \mathbf{x}_t^{(i)}\right) \cdot W\left(\widehat{\mathbf{p}_{\mathrm{B}}}, \widehat{\sigma_{\mathrm{B}}}, \widehat{\psi_{\mathrm{B}}}, \widehat{\sigma_{\mathrm{B}}}, \mathbf{x}_t^{(i)}\right) \cdot W\left(\widehat{\mathbf{p}_{\mathrm{A}}}, \widehat{\sigma_A}, \widehat{\psi_{\mathrm{A}}}, \widehat{\sigma_{\mathrm{A}}}, \mathbf{x}_t^{(i)}\right)$$

However, not all three functions are always included in the calculation. The ArUco update $W(\widehat{\mathbf{p}_{\mathrm{A}}}, \ldots)$ occurs

only if a marker is currently being detected and a pose estimate is available. The visual odometry updates $W(\widehat{\mathbf{p}_\mathrm{F}}, \ldots)$ and $W(\widehat{\mathbf{p}_\mathrm{B}}, \ldots)$ are skipped, if a failure of the front or back T265 camera is detected, as described in section 4.1. This method of weight computation also allows us to incorporate additional sensor inputs into the estimation process.

Finally, all particle weights are normalized to obtain a probability distribution such that $\sum w^{(i)} = 1$:

$$w_{t+1}^{(i)} = \frac{w_{t+1}^{(i)}}{\sum_{i=1}^{N} w_{t+1}^{(i)}}$$

In the edge case that none of the updates are available the whole weight update step is skipped, and previous weights are used to compute the state estimate – the filter can estimate the state just based on the model of the system accurately only for a couple of seconds (mainly because the model is just approximate, and the true state of the system will deviate from this model).

4.5 Resampling

By repeatedly applying the prediction step (section 4.3), the particles would gradually spread out over time, until only a few particles remain near the true system state. This is called the filter degeneracy problem [10]. To address it, the particles must be periodically re-sampled. During this process, the particles with large weights are replicated, and particles with low weights are diminished.

Resampling does not have to occur at every iteration of the particle filter. The metric of number of effective particles [10], [11] can be used to estimate the number of particles that meaningfully contribute to the filter estimate

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} \left(w_t^{(i)}\right)^2}$$

The resampling is triggered when N_{eff} falls below a threshold N_{th}, which was chosen to be $N/4$. The new set of particles $\mathbf{S}_{t+1}$ is generated using the systematic resampling method [25], which was selected based on its good algorithmic complexity $O(\mathrm{N})$.

4.6 State estimation

The final step is to compute the state estimate $\widehat{\mathbf{x}_t} = \left[\widehat{\mathbf{p}_t}\ \widehat{\psi_t}\right]^\mathrm{T}$ from the current particle set $\mathbf{S}_t$. The position estimate $\widehat{\mathbf{p}_t}$ with variance $\widehat{\sigma_\mathrm{p}}$ is obtained as weighted mean of positions of all particles

$$\widehat{\mathbf{p}_t} = \sum_{i=1}^{N} w_t^{(i)} \mathbf{p}_t^{(i)}$$

$$\widehat{\sigma_p} = \sum_{i=1}^{N} w_t^{(i)} \left(\mathbf{p}_t^{(i)} - \widehat{\mathbf{p}_t}\right)^2$$

The heading estimate $\widehat{\psi_t}$ is computed as weighted circular mean

$$a_1 = \sum_{i=1}^{N} w_t^{(i)} \sin\left(\psi_t^{(i)}\right),\ a_2 = \sum_{i=1}^{N} w_t^{(i)} \cos\left(\psi_t^{(i)}\right)$$

$$\widehat{\psi_t} = \mathrm{atan2}(a_1, a_2) \bmod 2\pi$$

When computing the weighted heading variance $\widehat{\sigma_\psi}$, the particle heading $\psi_t^{(i)}$ and the heading estimate $\widehat{\psi_t}$ can not be directly subtracted, as this would introduce large values near the warp-around between 0 and 2π. Instead, the angular difference function *D* previously defined in chapter 4.4 is used

$$\widehat{\sigma_\psi} = \frac{\sum_{i=1}^{N} w_t^{(i)} D\left(\psi_t^{(i)}, \widehat{\psi_t}\right)^2}{\sum_{i=1}^{N} w_t^{(i)}}$$

Afterwards, the filter continues with the next iteration.

5 Evaluation

The experimental evaluation is set up to mimic a simple warehouse inventory monitoring scenario. A map sketch of the testing environment and the desired trajectory is shown in Fig. 4. The environment consists of two rows of warehouse racks. The UAV takes off in front of marker 1. Then UAV continues its path heading towards RACK A. At the end of the RACK A, it does 180 degrees turn towards RACK B and continues on the trajectory back to the starting point. An initial ArUco marker is placed in front of the UAV take off position. One primary ArUco marker is placed approximately in the middle of each rack (two markers in total – green markers in Fig. 4). The position of all markers in the environment is known accurately. In a real-world scenario, these markers will be spread throughout the environment and provide accurate local position information for the UAV.

To obtain a reference position estimate, additional (secondary) ArUco markers (marked blue in Fig. 4) are positioned at known locations along the expected UAV flight path with focus on maximizing the number of visible markers over the main course of the flight. It is important to mention that only primary markers are used by the particle filter in the estimation process, the secondary markers are ignored.

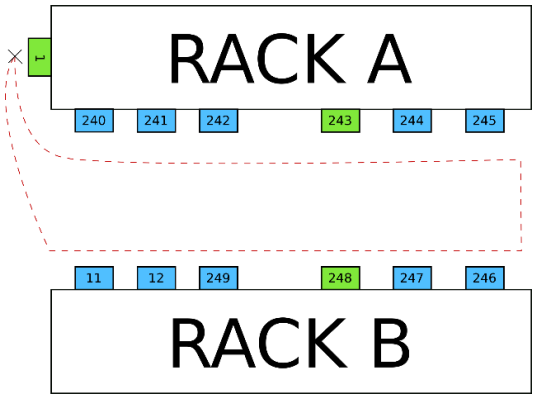

Fig. 4. Sketch of the testing environment. Green rectangles represent primary ArUco markers used by the PF update, blue rectangles are secondary markers used as ground truth position reference. The dashed line shows the desired flight trajectory of the UAV.

The UAV performs a single test flight in this environment. During this flight, Ardupilot is configured to directly use the pose estimate from the front T265 camera as position measurements via the *mocap* topic. We record the pose estimates from the front and back T265 cameras, ArUco pose estimates, Ardupilot velocity commands and the output of the internal Ardupilot Kalman filter pose estimator. The particle filter algorithm is then executed offline using the collected data. The filter parameters used in this test are summarized in Table 1.

Table 1. Particle filter parameters

Parameter		**Value**
N	Number of particles	2000
N_{th}	Resample threshold	$N/4$
σ_{up}	Process model position covariance	[0.125, 0.125, 0.125] m
$\sigma_{u\psi}$	Process model heading covariance	2.5 rad

5.1 Results

Results are displayed in Figs. 5, 6 and 8. In Fig. 5, the particle filter (blue line) is able to provide a continuous smooth estimate of position. When a primary ArUco marker is detected, the odometry measurement jumps towards the ArUco position (see green markers in Fig. 4). The global PF position estimate slowly converges towards the ArUco position, instead of jumping immediately. The black dots represent the ground truth UAV position estimate obtained from the secondary

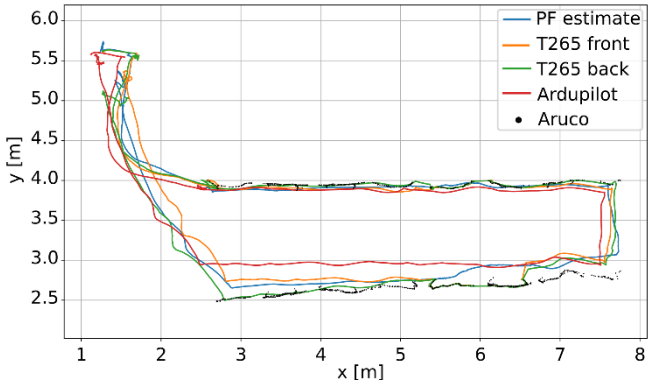

Fig. 5. Comparison of multiple position estimates. This result represents an ideal scenario – with no hardware failure.

ArUco markers. The red line shows the position estimated by the Ardupilot EKF using the position measurements from the front T265 camera and IMU measurements. Figure 6 shows the estimated positions and headings after each PF update, as well as the covariance of the PF position estimate. Figure 8 shows the RMS error between the PF position estimate and the ground truth position from ArUco detections.

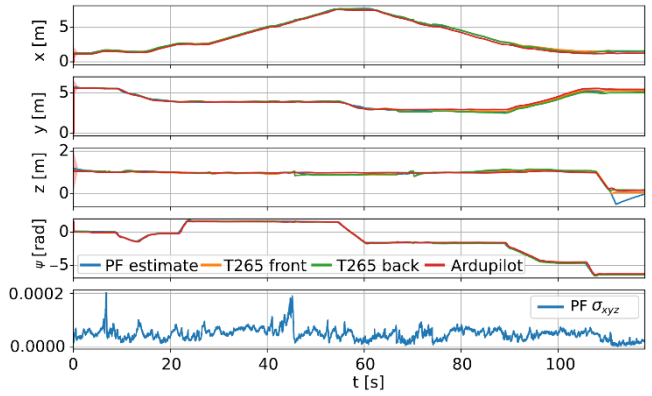

Fig. 6. Results displayed in a single axis including X, Y, Z and YAW angle. Figure also includes variance of the PF. This result represents an ideal scenario – with no hardware failure.

The only unexpected error which occurs is a bad Z axis estimate during landing. This is visible in Fig. 6. The issue arises because target velocity commands with a negative Z velocity are being sent even after the UAV is on the ground steadily, which causes particles of the PF to continue moving downwards. This continues for several seconds, until Ardupilot detects the landing and commands a zero velocity. This could be addressed by including the height sensor measurements into the PF state estimation process, which is expected to be implemented in future work.

The position estimation error (Fig. 8) is only computed when reference ArUco markers are in sight of the RGB camera. Notice the spike the error (Fig. 7) around the 60 s mark. This is caused by the manoeuvre the UAV performs. The UAV performs 180 degrees turn and therefore no reference position data is available during this time and so the error is not computed. Also, no green markers are available as well and so only the odometry data is fused during this turn and so the inaccuracy spikes.

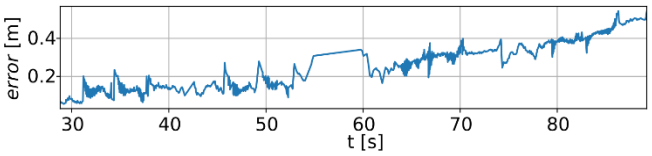

Fig. 7. Error of the ArduPilot fusion mechanism in comparison to reference ArUco visual markers reference

To compare the proposed PF algorithm with another sensor fusion solution, RMSE and position error between reference ArUco markers and the existing ArduPilot sensor fusion mechanism was also evaluated, its position error is displayed in Fig. 7.

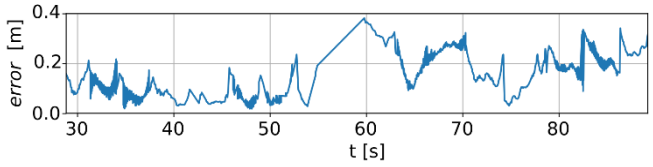

Fig. 8. RMS error of the particle filter in comparison to the reference from ArUco visual markers. This result represents the ideal scenario – with no hardware failure.

Figures 7 and 8 start from around the 30th second of the flight. This is caused by the fact, that the reference ArUco markers are only visible after that time.

Additional test to prove robustness and handling sensor malfunction was simulated by turning back odometry T265 camera off. The camera was turned off around timestamp 32 seconds and the average error rose to the value of 0.4202 m.

To compare our results with similar state-of-the-art methods a selection of the current manuscripts has been done. To make the comparison as fair as possible, the manuscript in which authors perform similar experiments are mainly based around visual odometry and ArUco markers.

Table 2 shows the results of this comparison. Other state-of-the-art methods outperform our system from the standpoint of accuracy. However, shortcomings of these state-of-the-art methods (which were stated in the Related work section) have been overcome by our system. For example, the price of all cameras on UAV used in our system is lower than a single Velodyne LiDAR. As far as robustness is concerned, occlusion or disconnection of one camera sensor would not cause failure of the whole process. Opposed to the [16] or [18] where occlusion of markers or camera would cause malfunction of the system. Finally, sensor disconnection was addressed theoretically by using selective weight update and then it was also tested experimentally.

Table 2. RMSE comparison

Method	RMS Error
PF (Our method)	0.394 m
MHE and ArUco [17]	0.697 m
VIO method based on AprilTag and EKF [15]	0.137 m
ES-EKF [16]	0.142 m

The average execution time of the PF update function with 2000 particles is 15.485 ms. This measurement is based on the average of five runs of the particle filter on a Jetson Xavier NX processor, with other software components such as the camera driver and ArUco detection running concurrently. The relatively high execution time is due to the Python CPU-only implementation of the PF. Performance improvements could be achieved by re-implementing the filter in a compiled language (such as C++) and utilizing GPU acceleration to parallelize certain aspects of the computation [12]. Despite this, the filter still achieves satisfactory real-time performance. To substantiate this claim, we compare our method's execution times with those reported in related works that use onboard computers for computations. For instance, the ES-EKF method presented in [16] achieves approximately 20 ms per execution step on an i5-based platform. The VIO method based on AprilTag visual tags [15] executes in 250 ms, with the final EKF fusion running at unspecified higher frequencies. Similarly, the work [18] uses Intel i5 series processors, achieving execution times under 100 ms.

The results show that the developed particle filter localization method is able to achieve satisfactory accuracy with adequate updates rates. Unfortunately, the exact accuracy of the proposed system cannot be measured at the present time due to the absence of a very accurate reference localization system (such as VICON etc.). Instead of using a reference system with exactly measured accuracy, it was decided to use the camera pose estimate with the use of ArUco markers, which are widely used in various types of applications in robotics [26, 27]. Using ArUco markers as a reference produced satisfying results in terms of error - distance between reference and the PF estimate and RMSE. Putting

calculated errors aside, it is safe to assume that the proposed system was not “lost” during flights and there were no significant spikes in the pose estimation, which could cause a fatal crash of the UAV. The absence of need to have an extremely accurate (order of millimetres) localization system is based on the fact that an autonomous flying UAV must utilize a robust reactive navigation system which should prevent fatal crashes. What should not happen is the absolute loss of position estimate, which can happen due to hardware failures. This scenario is minimized by utilizing the input from multiple position estimation sensors.

6 Conclusion

This paper presented a localization method for a UAV based on a particle filter. The filter fuses position estimates from two visual odometry cameras and from visual detections of fiducial markers. The method was evaluated on data from real-life test flights, which shows that the filter achieves acceptable accuracy in a sense that it would not cause a serious accident. To conclude, the work meets its requirements stated at the start by authors, which is all in all considered as a step towards a fully autonomous solution for UAVs operating indoors.

Future work will focus on deploying the developed algorithm to the UAV, to be used as the main real-time pose estimator. This will include evaluating the filter accuracy using an accurate position measurement system and performing multiple flights to mitigate stochasticity of the filter. The architecture of the particle filter allows for more sensor inputs to be included in the estimation process. Additionally, the Jetson Xavier NX GPU could be utilized to parallelize the particle filter computation, which would improve the speed and efficiency of the computation.

Acknowledgement

This work was supported by projects APVV-21-0352 “Navigation stack for autonomous drones in industrial environment” and MVPBLP - Mapping of indoor spaces using unmanned aerial vehicles. The authors would like to thank Airvolute s.r.o. for providing the necessary UAV hardware and support.

References

[1] N. El-Sheimy and Y. Li, "Indoor navigation: state of the art and future trends," *Satellite Navigation,* vol. 2, no. 1, 5 2021.

[2] C. Malang, P. Charoenkwan and R. Wudhikarn, "Implementation and Critical Factors of Unmanned Aerial Vehicle (UAV) in Warehouse Management: A Systematic Literature Review," *Drones,* vol. 7, no. 2, 2023.

[3] F. J. Perez-Grau, F. Caballero, A. Viguria and A. Ollero, "Multi-sensor three-dimensional Monte Carlo localization for long-term aerial robot navigation," *International Journal of Advanced Robotic Systems,* vol. 14, 2017.

[4] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics,* vol. 36, pp. 416-446, 2019.

[5] Vicon Motion Systems Ltd UK, "Award Winning Motion Capture Systems | Vicon," [Online]. Available: https://www.vicon.com/. [Accessed 15 07 2024].

[6] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics,* vol. 33, p. 1255–1262, October 2017.

[7] I. Cvišić, J. Ćesić, I. Marković and I. Petrović, "SOFT-SLAM: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles," *Journal of Field Robotics,* vol. 35, pp. 578-595, 2018.

[8] Intel Corporation, *Intel® RealSense™ Tracking Camera T265.*

[9] S. Thrun, D. Fox, W. Burgard and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence,* vol. 128, pp. 99-141, 2001.

[10] N. Yang, W. F. Tian, Z. H. Jin and C. B. Zhang, "Particle filter for sensor fusion in a land vehicle navigation system," *Measurement Science and Technology,* vol. 16, p. 677–681, February 2005.

[11] R. R. Labbe, *Kalman and Bayesian Filters in Python.*

[12] D. Fox, W. Burgard, F. Dellaert and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI,* vol. 1999, p. 2–2, 1999.

[13] C. Snyder, T. Bengtsson, P. Bickel and J. Anderson, "Obstacles to High-Dimensional Particle Filtering," *Monthly Weather Review,* vol. 136, pp. 4629-4640, 2008.

[14] B. Yan, J. Xin, M. Shan and Y. Wang, "CUDA Implementation of A Parallel Particle Filter for Mobile Robot Pose Estimation," in *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2019.

[15] M. Bertoni, S. Michieletto, R. Oboe and G. Michieletto, "Indoor Visual-Based Localization System for Multi-Rotor UAVs," *Sensors,* vol. 22, no. 15, 8 2022.

[16] L. Markovic, M. Kovac, R. Milijas, M. Car and S. Bogdan, "Error State Extended Kalman Filter Multi-Sensor Fusion for Unmanned Aerial Vehicle Localization in GPS and Magnetometer Denied Indoor Environments," in *2022 International Conference on Unmanned Aircraft Systems, ICUAS 2022*, 2022.

[17] S. Sina, B. Jeremy, J.-S. Farrokh and M. Iraj, "A Robust and Adaptive Sensor Fusion Approach for Indoor UAV Localization," in *2023 International Conference on Unmanned Aircraft Systems, ICUAS 2023*, 2023.

[18] D. Hao, W. Wei, C. Xu, R. Xiao and C. Sun, "Real-time onboard 3D state estimation of an unmanned aerial vehicle in multi-environments using multi-sensor data fusion," *Sensors (Switzerland),* vol. 20, no. 3, 2 2020.

[19] NVIDIA Corporation, *Jetson Xavier NX.*

[20] Ardupilot, *The Cube Overview.*

[21] IntelRealSense, *ROS Wrapper for Intel® RealSense™ Devices.*

[22] F. J. Romero-Ramirez, R. Muñoz-Salinas and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and Vision Computing,* vol. 76, pp. 38-47, 2018.

[23] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming," *Pattern Recognition,* vol. 51, pp. 481-491, 2016.

[24] E. Mráz, J. Rodina and A. Babinec, "Using fiducial markers to improve localization of a drone," in *2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR),* 2020.

[25] T. Li, M. Bolic and P. Djuric, "Resampling Methods for Particle Filtering: Classification, implementation, and strategies," *Signal Processing Magazine, IEEE,* vol. 32, pp. 70-86, May 2015.

[26] A. Babinec, L. Jurišica, P. Hubinský and F. Duchoň, "Visual Localization of Mobile Robot Using Artificial Markers," *Procedia Engineering,* vol. 96, December 2014.

[27] D. Avola, L. Cinque, G. L. Foresti, C. Mercuri and D. Pannone, "A practical framework for the development of augmented reality applications by using ArUco markers," in *International Conference on Pattern Recognition Applications and Methods*, 2016.

Received 25 April 2024