Working Paper

Scale-Corrected Monocular-SLAM for the AR.Drone 2.0

Author(s):
Gehrig, Daniel; Göttgens, Maximilian; Paden, Brian; Frazzoli, Emilio

Publication Date:
2017

Permanent Link:
https://doi.org/10.3929/ethz-a-010897472

Rights / License:
In Copyright - Non-Commercial Use Permitted
Abstract—This paper describes an open source software package in the Robot Operating System (ROS) framework for conducting robotics research with the Parrot AR.Drone 2.0. The software builds upon existing tools by adapting the ORB-SLAM library with a ROS driver for the drone so that a scaled monocular-SLAM based map and accurate pose estimate is available for navigation and control processes. A focus of the discussion is on robust scale estimation for the monocular SLAM generated map using sensor data from the drone’s on-board sensors. The purpose of this package is to provide an easy to use platform for experimental testing of autonomous navigation and control techniques for micro aerial vehicles.

I. INTRODUCTION

In recent years micro aerial vehicles (MAV) have greatly increased in popularity due to their versatility and relatively low cost. Some promising applications include package delivery [1], [2], search and rescue [3] and industrial inspection [4]. The increase in popularity is due in part to their high flexibility, maneuverability and sensing capabilities. These are results of advances in embedded control and sensor fusion algorithms which are becoming more viable as the cost for on-board sensing and computation decreases. These are vital for autonomously operated robotic systems, which must run a wide range of processes to interpret sensor data and take appropriate action to accomplish a task.

Typical processes include navigation modules to determine safe motions through an environment, feedback control modules to faithfully execute reference motions, and mapping and localization modules which provide a map of the obstacles in the environment as well as an estimated pose relative to that map.

A driving force for the development of such algorithms is open source software development and low-cost robotics hardware which is streamlining collaboration in robotics research and creating exciting educational opportunities. A good example for this is the AR.Drone 2.0. This quadcopter has gained a lot of interest as an educational tool and research testbed due to its low cost and large variety of on-board sensors. As such, existing developments for the AR.Drone 2.0 include a driver [5] that provides an interface to the Robot Operating System (ROS) framework, a package providing tools for manual flight [6], and a pose for pose estimation and basic control functionality [7].

The software package providing pose estimation, described in [8], performs high fidelity drone localization in GPS denied environments thanks to state of the art pose tracking algorithms which combine visual and inertial data derived from on-board cameras and inertial sensors. In particular visual pose tracking is performed using monocular simultaneous localization and mapping (SLAM) based on parallel tracking and mapping (PTAM) [9], [10]. However, monocular SLAM algorithms suffer from the well known limitation that they can determine pose and landmarks only up to an unknown scale factor. The solution presented in [8] is based on a maximum likelihood estimate that incorporates inertial data to determine an appropriate scale factor. While the implementation of this approach, available in [7], works well, there is potential for improvement:

1) The PTAM routine has to be initialized (bootstrapped) manually which limits autonomy
2) Mapping information is not provided prohibiting path-planning or other obstacle avoidance applications
3) PTAM is limited to local operation because it performs limited continuous mapping

In this work we utilize the maximum likelihood estimator proposed in [7], but replace the PTAM library with the more recently developed ORB-SLAM library to overcome the issues described above. In doing so, we also provide a reproducibility study of the maximum likelihood estimator’s performance with an alternative mapping tool.

ORB-SLAM is based on the ORB descriptor [11] and presented in [12]. The ORB descriptor has the advantage of being invariant to large view point changes [11], a feature that is not shared with the features from accelerated segment test (FAST) descriptor which is used in PTAM. ORB-SLAM additionally features automatic bootstrapping [12] and large-scale operability through continuous mapping of the surroundings. This is possible thanks loop-closures which avoid map duplication, eliminate drift, and lead to more efficient data storage. In general it has been shown that ORB also performs better in relocalization [12] leading to a more robust system.

The remainder of the paper is organized as follows: Section II presents an overview of system hardware and software. Section III reviews the scale estimator proposed in [7] as well as practical implementation details not described in [7]. In Section IV we present the experimental test results evaluating the performance of the system and modifications to the scale estimation scheme.

II. SYSTEM OVERVIEW

The system hardware is composed of an AR.Drone 2.0 unit along with an external computer for data processing and computations. A connection between both devices is established via a WLAN network. The drone’s on-board sensor suite consists of a forward facing camera, an
ROS Framework
node defined in the package ardrone_orb interface to the AR.Drone 2.0 quadcopter. Similarly, the node from the ardrone_autonomy package provides the ROS processes through channels called topics. The communication tools. Messages are passed between processes simultaneously and provides easy-to-use interprocess communication tools. Messages are passed between processes through channels called topics. The ardrone_driver node from the ardrone_autonomy package provides the ROS interface to the AR.Drone 2.0 quadcopter. Similarly, the ORB_mono node defined in the package ardrone_orb interfaces the ORB-SLAM library with the AR.Drone 2.0. While there is already ROS support for ORB-SLAM package, this node provides an interface specific to the AR.Drone 2.0. This includes necessary transformations to align the AR.Drone 2.0 driver’s coordinate frame with the ORB-SLAM frame, and publishes an unscaled pose estimate and point cloud of the detected features. The scale estimator node defined in the package ardrone_orb estimates a scale between the ORB-SLAM and the drone driver coordinate frames. Together, these nodes provide a scaled pose estimate of the drone.

A. Coordinate Frames for the AR.Drone 2.0

Accurate pose estimation requires keeping track of several transformations between coordinate frames. The drone driver publishes a base coordinate frame odom over the navdata topic according to ROS REP 105. This signal is subject to drift over time. Drift can be eliminated from this estimate by incorporating localization information published by the ORB-SLAM wrapper node.

Three features of the AR.Drone 2.0/ORB-SLAM have to be considered for a transformation from the driver’s odom frame to the drone’s body fixed frame (pose estimate).

1) The first pose estimated by ORB-SLAM is not equal to the origin of the ORB-SLAM baseframe
2) ORB-SLAM data is expressed in a camera reference coordinate frame ([13]) for the front camera whereas the driver pose estimations are expressed in a odom-aligned frame
3) Upon take-off, a rotation in the base-link transformation published by the drone driver occurs

To incorporate these properties, we establish a transformation chain, which is initialized directly after ORB-SLAM initialization. Once the transformation chain is successfully set, all static transformations in the chain can be consolidated into one fixed transformation. As a result, only one transformation multiplication is needed in order to get the final transformation of the pose (odom → orb_pose_unscaled ). However, to achieve this, all parts of the transformation chain have to be collected successively at the correct time. The complete transformation tree is illustrated in Figure 2.

1) odom to second_keyframe: Upon ORB-SLAM initialization at time $t_{\text{init}}$ (successful bootstrapping), the frames second_keyframe_base_link and second_keyframe_cam are defined. Call these $S$ and $S^*$. Together with the odom frame, $O$, these define the transformation $T_{OS} \in SE(3)$ which transform data in frame $S$ to frame $O$ when multiplied the vector from the left. They have the same rotation relative to $O$ as ardrone_base_link and ardrone_base_frontcam respectively at initialization. Note that $S$ and $S^*$ have the same origin but different orientations according to [13]. We will switch between camera and odom-aligned frames with the transformation $R = T_{A4}$ such that $T_{OA^*} = T_{OA}R$ where $A$ is an arbitrary frame.

2) second_keyframe to first_keyframe: The first pose returned by ORB-SLAM is the pose of the second keyframe of the bootstrapping process. However, all poses and landmarks from ORB-SLAM are expressed with respect to the first keyframe. This means that at initialization we can only determine the first ORB-SLAM pose with respect to a frame first_keyframe_cam, denoted $F^*$ which was generated at an unknown time point in the past. This is because ORB-SLAM sets the first keyframe dynamically and dependent on whether it can initialize well or not. The first received pose defines the transformation $T_{S^*F^*}$.

The transformation to the desired frame $F^*$ can be obtained through the multiplication $T_{DF^*} = T_{DS^*}T_{S^*F^*}$.

3) first_keyframe to pose_unscaled: The drone pose and point cloud can now both be expressed in $F^*$. The pose is defined over its body fixed frame orb_pose_unscaled_cam, denoted $B^*$, which can be reached through pose information gathered from ORB-SLAM of the form $T_{B^*F^*}(t)$ and depends on the time $t$ since it is a moving frame. These time varying transformations define the frames $B^*$ and $B^*_o$, orb_pose_unscaled, over the multiplication $T_{OB^*_o} = T_{OF^*}T_{B^*F^*}$ and $T_{OB} = R^{-1}T_{OB^*_o}T_{F^*B^*}$.

Lastly a quirk of the the drone driver node is that it introduces a rotation around the z-axis of...
ardrone_base_link upon take-off. This occurs at the moment the first altitude readings become available in the drone driver node and introduces a new frame which we will denote \( \hat{O} \) and is oriented toward magnetic north. We determine the transformation \( T_{O\hat{O}} \) by computing the difference of transformations \( T_{O\hat{C}} \) (published by the driver) before and after take off, where \( L \) denotes the ardrone_base_link frame. This yields the correction \( T_{O\hat{O}} = T_{O\hat{C}} L T_{O\hat{C}}^{-1} \). The introduced frames and necessary transformations are summarized in Figure 2. We can express the current body fixed frame of the drone after take off expressed in the \( \text{odom} \) frame by following the transformation tree down to \( \text{orb\_pose\_unscaled} \) and multiplying the passed transformations in succession. Doing this we arrive at:

\[
T_{\hat{O}B}(t) = T_{O\hat{O}} T_{O\hat{C}} R T_{S\hat{F}} R T_{B\hat{F}} R(t)^{-1} R^{-1}
\]

This transformation also defines the pose of the drone as measured by ORB-SLAM. Once the scale has been estimated correctly the true drone pose is defined by the frame \( \text{orb\_pose\_scaled} \) denoted by \( \mathcal{P} \) and is defined by the following transformation:

\[
T_{\hat{O}P}(t) = \Lambda T_{\hat{O}B}(t)
\]

where

\[
\Lambda = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\]

Note that this is a pure scaling matrix applied to the 3-dimensional position vector.

### III. Scale Estimation

One of the challenges that arises when using monocular-SLAM for localization and mapping is determining a length scaling factor for the generated map and robot pose. In this work, the maximum likelihood approach in [8] was used since it has been experimentally validated on the AR.Drone 2.0.

#### A. Mathematical Derivation

Let \( \mathbf{x}(t) \) and \( \mathbf{y}(t) \) denote position measurements in the same coordinate frame from an unscaled and metric sensor respectively. We would like to derive a model relating these measurements to determine an estimator for the visual scale. When comparing the raw measurements we see that they are related over a scaling factor and an unobserved offset since their coordinate frames are not necessarily collocated. We can avoid estimating this offset if we consider the time derivatives of the position measurements, \( \dot{\mathbf{y}}(t) \) and \( \dot{\mathbf{x}}(t) \) since velocity vectors should also obey the same scaling law.

In prior works [14], [8] this differentiation was approximated with finite differences between measurements.

Fig. 2. The transformation setup consists of all correction transformations specific to the AR.Drone 2.0. To pass from one frame to the next the vectors expressed in the parent frame must be multiplied by the transformation defined at the branch leading to the target frame. Note that the scaled pose is directly published in the \( \text{odom} \) frame for simplicity - all transformations on the right side are still needed in order for this transformation to be realized.

Measurements at two consecutive time-steps are given by \( \mathbf{x}(t_i), \mathbf{x}(t_{i+1}), \mathbf{y}(t_i) \) and \( \mathbf{y}(t_{i+1}) \). Then measured displacement over that time-step is

\[
\Delta \mathbf{x}_i = \mathbf{x}(t_{i+1}) - \mathbf{x}(t_i),
\]

\[
\Delta \mathbf{y}_i = \mathbf{y}(t_{i+1}) - \mathbf{y}(t_i).
\]

It is assumed that the the actual displacement is \( \Delta \mu \), which is scaled by a constant \( \lambda \), for the unscaled measurement, and corrupted by i.i.d. Gaussian noise.

\[
\Delta \mathbf{x}_i \sim N(\Delta \mathbf{x}_i | \lambda \Delta \mu_i, \sigma^2_x I_d),
\]

\[
\Delta \mathbf{y}_i \sim N(\Delta \mathbf{y}_i | \Delta \mu_i, \sigma^2_y I_d),
\]

where \( \sigma^2_x \) and \( \sigma^2_y \) are variances for additive Gaussian noise.

The covariances are assumed isotropic and scale independent. These assumptions are valid in particular for the variance of the visual sensor when \( \mathbf{x}_i \) is renormalized after initialization such that the initial point cloud has unit depth [15]. This is because the variance correlates with the average landmark depth in the first frame.

Given a dataset of measurements \( \mathcal{X}' = \{ \Delta \mathbf{x}_i \}_{i=1}^N \) and \( \mathcal{Y} = \{ \Delta \mathbf{y}_i \}_{i=1}^N \) the negative log likelihood function has the following form:

\[
\mathcal{L}(\mathcal{X}', \mathcal{Y}) \propto \sum_{i=1}^N \frac{||\Delta \mathbf{x}_i - \lambda \Delta \mu_i||^2}{2\sigma^2_x} + \frac{||\Delta \mathbf{y}_i - \Delta \mu_i||^2}{2\sigma^2_y}
\]

Minimizing yields a globally optimal solution for \( \lambda \):

\[
\hat{\lambda}_{ML} = \frac{s_{xx} - s_{yy} + \text{sgn}(s_{xy}) \sqrt{(s_{yy} - s_{xx})^2 + 4s_{xy}^2}}{2\sigma_x s_{xy}}
\]

where \( s_{xx} = \sigma^2_x \sum_{i=1}^N \Delta \mathbf{x}_i^T \Delta \mathbf{x}_i, s_{yy} = \sigma^2_y \sum_{i=1}^N \Delta \mathbf{y}_i^T \Delta \mathbf{y}_i \) and \( s_{xy} = \sigma_x \sigma_y \sum_{i=1}^N \Delta \mathbf{x}_i^T \Delta \mathbf{y}_i \).
B. Application to AR.Drone 2.0

The above approach is generic and must be adapted to specific sensor readings from the drone. Possible candidates for position measurements include acceleration from the IMU, horizontal velocity estimates from optical flow in the bottom facing camera and height information from the ultrasonic sensor.

We observed that the horizontal motion (in the plane of the drone’s body fixed frame) estimated by the navdata channel was unreliable due to drift in the estimate and occasional data corruption. On the other hand, the altitude component was the most reliable measurement for scale estimation. This observation is also consistent with what is reported in other works [16], [8] where ultrasonic altitude measurements yielded good results and fast convergence.

In the following discussion $z_{\text{slam}}(t)$ and $z_{\text{nav}}(t)$ will denote quad-copter’s vertical pose coordinate as estimated by ORB-SLAM and the navdata channel respectively. We will try to find a model which describes the distribution of local velocity estimates. Note that for small time intervals we can make the following approximation

$$\dot{z}_{\text{slam}}(t_i) \approx \frac{\Delta z_{\text{slam},i}}{\Delta t},$$

$$\dot{z}_{\text{nav}}(t_i) \approx \frac{\Delta z_{\text{nav},i}}{\Delta t}. \tag{7}$$

Using the stochastic model equations (4) for small displacements we arrive at

$$\dot{z}_{\text{slam}}(t_i) \sim \mathcal{N}(\dot{z}_{\text{slam}}(t_i)|\mu(t_i), \sigma^2_{\text{slam}}),$$

$$\dot{z}_{\text{nav}}(t_i) \sim \mathcal{N}(\dot{z}_{\text{nav}}(t_i)|\mu(t_i), \sigma^2_{\text{nav}}), \tag{8}$$

where $\dot{\mu}(t_i) := \frac{\Delta \mu(t_i)}{\Delta t}$ is the true local velocity and $\sigma^2_{\text{nav}} := \frac{\Delta \sigma_{\text{nav}}^2}{\Delta t^2}$ and $\sigma^2_{\text{slam}} := \frac{\Delta \sigma_{\text{slam}}^2}{\Delta t^2}$ are the variances of both velocity measurements. This illustrates that we can use the same estimator as in (6) to estimate the scale by simply replacing small displacement segments by local derivatives.

C. Processing asynchronous measurements

Note that equations (8) imply that $\Delta z_{\text{slam},i}$ and $\Delta z_{\text{nav},i}$ are measured at the same time $t_i$.

However, the navdata channel publishes a pose estimate at 200 Hz while the SLAM process publishes a pose estimate at roughly 10 - 20 Hz depending on the computation time of ORB-SLAM for the incoming frames. This means that care must be taken to synchronize the measurements.

To handle these asynchronous measurements, a linear time invariant (LTI) system is used to pre-filter the data and synchronize the measurements. Each LTI system is updated with time-input pairs $(t_i, z_i)$. The state of the LTI system is then forward integrated up to time $t_i$ from the last update $t_{i-1}$ with the input defined by a first order hold from $z_{i-1}$ to $z_i$ on the time interval $[t_{i-1}, t_i]$,

$$z(t) = z_{i-1} + \frac{(z_i - z_{i-1})}{(t_i - t_{i-1})} \cdot (t - t_{i-1}), \quad t \in [t_{i-1}, t_i] \tag{9}$$

An LTI system $G(s)$ is defined by its frequency domain transfer function

$$G(s) = \frac{n_0 + n_1 \cdot s + n_2 \cdot s^2 + \ldots + n_{k-1} \cdot s^{k-1}}{d_0 + d_1 \cdot s + d_2 \cdot s^2 + \ldots + d_{k} \cdot s^{k}}, \tag{10}$$

and the input-output mapping from the measurement signal $z(t)$ to the filtered signal $\hat{z}(t)$ is determined by integrating a space-state realization of (10) given by

$$\hat{z}(t) = c^T x(t), \quad \dot{x}(t) = Ax(t) + bz(t). \tag{11}$$

The input $z(t)$ to (11) is determined from measurement data according to (9). The state space realization used in our experiments is given by

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \ddots & 1 \\ -\frac{d_k}{d_0} & -\frac{d_{k-1}}{d_0} & \ldots & \frac{d_1}{d_0} \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}, \quad c = \begin{pmatrix} n_0 \\ n_1 \\ \vdots \\ n_{k-1} \end{pmatrix}. \tag{12}$$

Filtering the measurement data $z_{\text{slam}}(t)$ and $z_{\text{nav}}(t)$ with this LTI system removes some of the noise present in the measurements and allows for the output $\hat{z}_{\text{slam}}(t)$ and $\hat{z}_{\text{nav}}(t)$ to be sampled at regular time intervals to synchronize the data.

The transfer function used to filter the altitude data is the following

$$G(s) = \frac{s}{(s + 60\pi)^2}. \tag{13}$$

This transfer function has a zero at the origin which is responsible for differentiating the signal and two poles at $s = -60\pi$ (corner frequency at 30 Hz) which apply some smoothing for high frequency noise. This is because one issue with pure differentiation is that it amplifies noise at higher frequencies. The amplitude response of the pre-filter relative to a pure differentiator is shown in Figure 3. Figure
shows the unfiltered data signals from the altitude sensor and ORB-SLAM. Note that the signals do not have the same origin and have different scales. For about 30 seconds, the drone is flying up and down followed by level flight. It can be seen that the signals are highly correlated during the up and down maneuver. However, purely horizontal motion leads to poor signal-to-noise ratios and poorly correlated signals. Moreover, the visual estimate varies discontinuously as loops-closures and relocalizations are detected. This behavior violates the assumption of Gaussian noise meaning that other procedures must be used to filter out certain data.

D. Further Modifications

The calculations in Section III-A are based on the assumption that the noise distributions in [4] are Gaussian. However, Figure 4 shows that this assumption is not valid and motivates including some additional elements to the scale estimation. This includes an initial size filtering step followed by sign filtering and finally a 1-point random sample consensus (RANSAC) on the remaining data points.

1) Size Filtering: The first step in filtering is to discard individual signals based on their absolute magnitude. Limited vertical motion generates low signal-to-noise-ratio measurements with little information for estimating the scale. On the other hand, very large signals that are likely caused by large jumps violate the Gaussian noise assumption and are thus not suitable for the proposed scale estimator. In the case of visual signals, we discard data points based on an upper and lower threshold, whereas for ultrasonic signals we only use a lower threshold as the ultrasonic sensor is not prone to significant discontinuities.

2) Sign Filtering: Another indicator of signal-to-noise ratio measurements is when visual and inertial measurements are anti-correlated and have opposite sign. This would indicate a negative scale associated with that data pair which is not physically possible. When the two sensor measurements have different signs, that pair of data points is discarded.

3) 1-point-RANSAC: Despite initial filtering of the data there are still samples that corrupt the signal and lead to issues with convergence. To identify inliers and perform robust estimation of the final scale a 1-point-RANSAC algorithm was used. This method was already used in [7] and described in detail in [17]. In particular [17] have derived an efficient $O(n)$ algorithm to remove outliers. A brief overview will be given below.

Let $\mathcal{Z}_{\text{slam}} := \{\hat{z}_{\text{slam}}(t_i)\}_{i=1}^N$ and $\mathcal{Z}_{\text{nav}} := \{\hat{z}_{\text{nav}}(t_i)\}_{i=1}^N$ denote the data collected from the LTI system. Further, let $\Lambda := \{\lambda_i\}_{i=1}^N$ denote the set of scale point estimates computed from a single data pair $(\hat{z}_{\text{slam}}(t_i), \hat{z}_{\text{nav}}(t_i))$. For a single data pair, equation (6) reduces to

$$
\lambda_i = \begin{cases} 
\frac{\hat{z}_{\text{slam}}(t_i)}{\hat{z}_{\text{nav}}(t_i)}, & \text{if } \hat{z}_{\text{nav}}(t_i)\hat{z}_{\text{slam}}(t_i) \geq 0 \\
\frac{\sigma^2_{\text{slam}} \hat{z}_{\text{nav}}(t_i)}{\sigma^2_{\text{nav}} \hat{z}_{\text{slam}}(t_i)}, & \text{if } \hat{z}_{\text{nav}}(t_i)\hat{z}_{\text{slam}}(t_i) < 0
\end{cases}
$$

In the further analysis we considered the logarithms base 10 of the scales and their distribution. This is because we observed that the density follows a symmetric distribution which looks like a bimodal Gaussian mixture as can be seen in Figure 6. Note that this is an approximation since it does not follow from the assumed distribution over the data. However, making this assumption helped gain insight into the statistics of the process.

Denote the set $L := \{\log \lambda_i\}_{i=1}^N$.

Figure 6 illustrates the distribution of log point scale estimates for a test flight with the drone. Inliers lie within a band around the median of the distribution. Let $\delta l$ denote the half-width of this band around the median. The RANSAC inliers of the distribution can be described as follows:

$$
\lambda^* = \text{median}(\Lambda), \\
L^* = \{l \in L | l^* - \delta l < l < l^* + \delta l\}.
$$

Taking the exponential of the data, we arrive back at the final inlier set representation,

$$
\lambda^* = \text{median}(\Lambda), \\
\Lambda^* = \{\lambda \in \Lambda | \lambda^* \alpha^{-1} < \lambda < \lambda^* \alpha\}.
$$

The final scale estimate is calculated according to (6) with the data associated with the inlier set $\Lambda^*$.

Experimental testing of this scale estimator is described in the next section.

IV. EXPERIMENTAL EVALUATION

To determine the quality of the scale estimator proposed in Section III 8 test flights were performed. The visual scale was estimated on-line with data samples being generated at 15 Hz. Each flight started with three up and down flying maneuvers similar to the ones described in Section III-C for the scale estimate to converge. This procedure is also suggested in [8] to speed up convergence. Once the scale estimate converged, the drone was controlled to fly 4 meters forward from the starting position followed by another 4
meters. At each step the visual pose was compared to the true displacement and used to determine the true scale $\lambda^*$. This method of measuring the distance is accurate to about 10 cm which corresponds to an accuracy of about 1.25% in scale.

Figure 5 illustrates the experimental setup and a graphical representation of the scene in RVIZ. The true dimensions and schematic of the setup are in the top right inset. The drone’s pose is represented by its body fixed coordinate frame and is surrounded by recently seen landmarks. Its trajectory and initialization procedure is shown in purple. The view from the drone’s front-facing camera is shown in the bottom left together with tracked features by ORB-SLAM.

A. Signal Filtering

Figure 6 shows a histogram of 500 scale samples collected over one flight. Table I shows the median scale and the scale estimator computed for the entire set after each step of data filtering. Note that the true scale in this case is $\lambda = 0.1$. It can be seen that the pure raw data yields a very poor estimate of the scale at 0.0659. Additionally the median is located at 0.2261 which is also far off from the true scale. The discrepancy between median and estimator is a consequence of the bimodal distribution of the raw data. To improve the estimator we must focus on the strong maximum occurring at $\log \lambda = -1.3$. Filtering by signal size removes excessively large and small scale samples which are colored blue. We see a small improvement in the scale estimate of 0.0659 and the median of 0.1755 due to size filtering. Scale samples with anti-correlated data points are shown in yellow and removed during sign filtering. Note that these samples make up a significant number and contribute to the secondary peak seen in the distribution. The sign filtering step significantly improves the scale estimate to 0.1062 and the median to 0.0976. The closeness of these two values indicates that the remaining green inliers exhibit a unimodal distribution which is visible in the Figure. Finally 1-point-RANSAC manages to catch all inliers in green but we see little difference in the scale. This is because we chose large boundaries which add to the robustness of the estimator.

B. Scale Accuracy

To determine the accuracy of the scale estimate 18 ORB-SLAM runs were performed on 6 bags and for each procedure the true scale $\lambda^*$ was determined as described. Two flights were neglected because they exhibited significant motion blur during initialization which almost always yielded inaccurate scale estimation. To understand exactly why these flights have bad properties further research is needed.

To compare scales derived from individual flights a normalized scale was used $\lambda = \frac{\lambda^*}{\lambda}$ which should converge to 1 for all data sets. Figure 7 shows the time evolution of the average normalized scale. The green line is the mean of the normalized scales while the shaded area represents one standard deviation of distance from the mean. As can be seen the average scale converges after about 15 seconds with a steadily decreasing standard deviation which bottoms out at 15%. The best scales were recorded after around 20 seconds but then continued to drift until after 40 seconds there is about a 7.44% bias. However, compared to a variance of 15% this bias is not significant. Other works report a convergence rate of about 7 seconds and a standard deviation of 2.75%

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale sample filtering steps ($\lambda^* = 0.1$)</td>
</tr>
<tr>
<td>Raw</td>
</tr>
<tr>
<td>median</td>
</tr>
<tr>
<td>estimator</td>
</tr>
</tbody>
</table>
TABLE II

<table>
<thead>
<tr>
<th>Convergence speed (s)</th>
<th>Bias</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>this work</td>
<td>15 15%</td>
<td>15%</td>
</tr>
<tr>
<td>[15]</td>
<td>1 2.75%</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Fig. 7. Normalized scale $\hat{\lambda} = \frac{\lambda}{\lambda_0}$ averaged over 18 ORB-SLAM runs (green), accuracies and the associated errors over time. After 40 seconds the average error is 7.44% with a standard deviation of 15%.

[15] A comparison is detailed in Table II. This could be due to lower sampling frequencies of about 1 Hz compared to our sampling frequency of 13 Hz. This significantly increases the signal to noise ratio which leads to lower variance. To adapt our system to lower sampling frequencies however, the assumption of small time steps must be sacrificed which makes the approximations ?? and ?? inaccurate. Further research is needed to improve the estimator. It is also possible that properties of ORB-SLAM affect the noise distribution. The filtering processes in this work must be further optimized to achieve the best scale estimation.

Some flights were disregarded as they exhibited too rapid movement during initialization. This lead to bad tracking when ORB-SLAM was running due to motion blur which lead to bad height estimates. We have observed that scale estimates showed significant errors and were prone to drifting in these cases.

V. CONCLUSION

In this work, we described a ROS package for scale corrected monocular-SLAM on the AR.Drone 2.0. This package makes use of the ORB-SLAM library, a state of the art SLAM library. This package allows researchers to get an experimental testbed for motion planning and control algorithms up and running quickly.

We were able to build upon earlier research on monocular-SLAM scale estimation for the AR.Drone 2.0 and integrate these techniques with the ORB-SLAM library.

The next steps include incorporating a real-time motion planning algorithm to generate collision free motions through the environment, and a feedback controller to provide the control inputs to execute the planned motions.

VI. ACKNOWLEDGEMENT

We would like to thank the IDSC at ETH Zürich for funding this research.

REFERENCES