Cooperative Localization of Unmanned Aerial Vehicles in ROS - The Atlas Node

Paul Kremer, Jan Dentler*Somasundar Kannan†, Holger Voos‡
University of Luxembourg, SnT
6, rue Richard Coudenhove-Kalergi L-1359 Luxembourg
**Email: jan.dentler@uni.lu*
†*Email: somasundar.kannan@uni.lu*
‡*Email: holger.voos@uni.lu*

***Abstract*—This paper is presenting the implementation and experimental validation of the cooperative robot localization framework "Atlas". For ease of application, Atlas is implemented as a package for the Robot Operating System (ROS). ATLAS is based on dynamic cooperative sensor fusion which optimizes the estimated pose with respect to noise, respective variance. This paper validates the applicability of Atlas by cooperatively localizing multiple real quadrotors using cameras and fiduciary markers.**

1. Introduction

Position sensing is a key feature for many applications in robotics such as pick and place tasks, collision avoidance or navigation. Multiple technologies are available to estimate the position of an object such as the Global Positioning System (GPS) or the OptiTrac Motion Capture system. But these technologies are either unavailable in indoor environments or costly.

Today, many Unmanned Aerial Vehicles (UAV) are equipped with CCD cameras for video recording. One idea is therefore to use these cameras for position measurements using planar markers. The most commonly used marker type is the ArUco marker, a black and white binary matrix, originally developed for applications in the augmented reality domain. By detecting these markers with a camera, it is possible to determine their position and orientation (pose) within the camera frame. Besides low costs, the application of this approach in multi-robot scenarios has two further advantages as shown in Figure 1. This includes fusing multiple sensor measurements to increase the accuracy of the estimated pose. Furthermore the sequential relative localization from robot to robot allows the localization even in areas with visual obstacles. However, the dynamically changing topology of such systems combined with the fusion of sensor data results is a challenging task. This paper is addressing this issue by presenting the implementation of the Atlas node and validating its performance in a cooperative quadrotor localization scenario.

* *Supported by FNR "Fonds national de la Recherche" (Luxembourg) through AFR "Aides à la Formation-Recherche" Ph.D. grant scheme No. 9312118.*

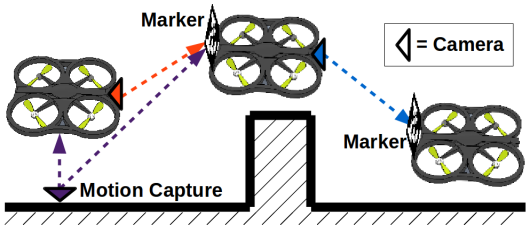

Figure 1. Cooperative Localization Scenario

2. State of the art

Similar to ArUco markers, AprilTags [1] are also fiducial marker based on a binary matrix. The main difference between the two is an improved coding scheme which significantly improves the robustness against false detection and increases the reliability of the marker's orientation detection.

A comparison [2] of the performance of AprilTags and ArUco markers of different sizes under varying light conditions with respect to the distance and viewing angle has shown that both systems are highly efficient in terms of computing time and perform well under most of the tested conditions with a small advantage for the AprilTags. Tests [2] have shown that planar markers of 3x3cm can be detected up to a distance of 1m using a fairly low resolution camera (640x480 pixels). Naturally the range in which these markers can be detected increases with the resolution of the camera as well as the size of the markers. Another important factor is the viewing angle between the camera and the marker. A viewing angle of 60° is achievable [2] even under bad light conditions. Both marker systems are well suited [2] for indoor localization systems.

However, both the ArUco and the AprilTag severely suffer from motion blur which is inevitable in the detection of moving objects like drones. The motion blur softens and smuggles the edges of the markers and they become hard or even impossible to detect. This phenomenon can also

be explained in frequency domain. As stated in [3], the sharp edges making up the planar markers can in fact be considered as high frequency regions which are attenuated by the motion blur. The solution [3] is a new type of planar marker which consists only of low mono-frequency components which are hardly affected by motion blur. As such, this new type of marker is still detectable under conditions were the detection of the ArUco marker and AprilTag fails.

As for the application of cooperative position sensing of moving drones, these mono frequency markers [3] are in fact more suited than the monochrome markers. However, in contrast to ArUco markers and AprilTags, there is no reference implementation available for an evaluation under real conditions. Consequently, the AprilTags have been chosen as the markers of choice for this application.

An advantage of using planar markers instead of e.g. laser range finders (LRF) is that they carry a unique ID which can be used for two cases: First, if the marker's world position is known, it can be used to calculate the position of the measuring drone and, second if the marker's relative offset to the object, where the marker is attached to is known, it can be used to calculate the object's relative position to the drone. With the knowledge of both it becomes possible to make the drones fully aware of their environment and the objects which are part of it.

In case of setups with multiple cameras, markers and entities to which the markers are attached to, it becomes likely that multiple sensors (cameras) are estimating e.g. the pose of the same marker. These redundant measurements can then be used to achieve higher accuracy and reliability by fusing them. A comparison of the most common filters [4] for that task has shown that the Kalman filter can perform significantly better than simple filters like the median, threshold voters or weighted average. However, the authors also indicated that their test scenario is particularly well suited for the Kalman filter as their sensor data is perfectly normal distributed which might not be the case in real applications.

Nevertheless, the Kalman filter is widely used when it comes to sensor fusion and is considered to be an optimal linear estimator [4]. However, in this work the weighted average fusion algorithm is used as it can deal with quaternions with relative ease.

In order to share information between multiple drones it is important to have a reliable communication. The Robot Operating System (ROS) [5] offers a messaging infrastructure between different nodes. Nodes in the context of ROS represent a process (or program) running on a device. These nodes can be on the same machine or distributed (physically) among different devices. Nodes can send and receive messages on different topics. This facilitates the cooperative reception, fusion and sharing of the drone positions.

3. Problem Statement and Implementation

The following system is supposed to provide the poses of different drones in an arbitrary environment by using the sensors either provided by the drones (cameras) or the environment (GPS, optitrack, etc.). The system has to fuse the results from different sensors in order to improve the measurements with respect to noise and variance.

The implementation of the system has to be compatible with the Robot Operating System (ROS) i.e. it has to be implemented as a ROS node and make use of the ROS communication subsystem in order to receive and send data from or to other nodes.

The first step of solving the problem of cooperative sensing is to derive the abstract data structure of the scenario, which can be distinguished in sensors, markers and entities. With these it is possible to model a wide variety of scenarios.

- Sensor:

A sensor is defined by a name, a pose and the reference frame. It can be attached to either a drone or the environment (world). A sensor is supposed to provide the pose and ID of the marker(s) it detects. Also, if possible, a sensor has to provide an estimate of the standard deviation of its measurements.

- Marker:

A (fiducial) marker has a unique ID, and a certain pose. Just like the sensors it can be attached to either the environment (world) or an entity.

- Entity:

An entity is defined by a name only. It can be any object, real or fake (i.e. with physical meaning or without), with zero or more sensors and with zero or more markers attached to. It is an abstract representation of a drone or any other object, serving as container for markers and sensors.

The structural implementation of the Atlas node consists of 4 main blocks:

1) Sensor listener: Listens to sensor information (MarkerData) on the ROS messaging bus. This block also performs some filtering to reject noise.
2) Transform graph: Takes the sensor information from the sensor listener, builds a transformation graph, and evaluates the poses of the entities.
3) Graph publisher: Takes the information contained in the transform graph and publishes the transformations via the ROS *tf2* subsystem or on the message bus via a ROS topic of type *FusedPose*.
4) Config: Reads a user defined config file which is used to configure the blocks above.

The block diagram showing the interaction between these blocks is shown in (Figure 2).

3.1. Sensor Data Acquisition (Sensor Listener)

The sensor data is acquired directly from the ROS messaging bus. The Atlas node subscribes to the sensor's respective topic and expects the messages to be of type MarkerData which is as defined in (Table 1). The sensor

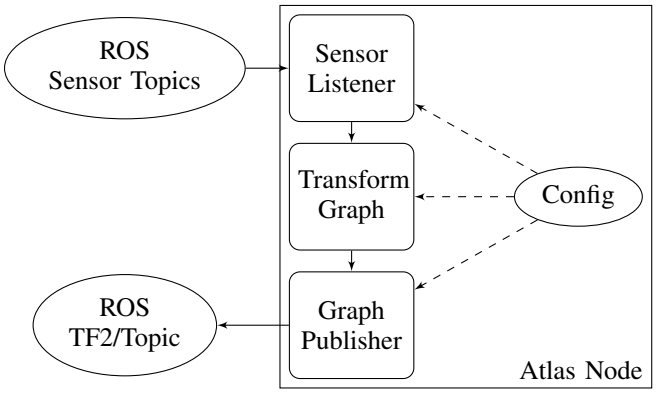

Figure 2. Atlas block diagram

TABLE 1. MARKERDATA TYPE

Field	Type	Comment
id	int32	The marker's ID
sigma	float64	The sensor's standard deviation
pos	geometry_msgs/Point	Translation in sensor space
rot	geometry_msgs/Quaternion	Rotation in sensor space

is expected to deliver the marker's pose in the sensor's frame. It should also deliver an estimate of its standard deviation (sigma) which is later used to fuse sensor readings. Next, with the help of the marker's pose with respect to its parent (either the world or another entity) and by taking into account the sensor's pose with respect to its parent, it is possible to calculate the transformation from one entity to another. As depicted in (Figure 3) the following three transformations matrices are known:

- $\mathbf{T}_1$: The entity to sensor transformation. This transformation is linked to the physical pose of the sensor to the entities baselink.
- $\mathbf{T}_2$: The sensor to marker transformation. This is the output of the sensor i.e. the marker's pose in the sensor frame. As such this is the only quantity that varies at runtime.
- $\mathbf{T}_3$: The entity to marker transformation. Again, this is linked to the marker's relative pose to the entity's baselink.
- $\mathbf{T}$: The entity to entity transformation. The entity's pose as seen by the observing entity.

The transformations $\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{T}_3$ can be used to calculate the entity to entity transformation $\mathbf{T}$ as follows:

$$\mathbf{T} = \mathbf{T}_1 \cdot \mathbf{T}_2 \cdot \mathbf{T}_3^{-1} \tag{1}$$

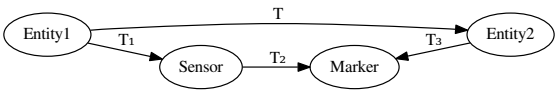

Figure 3. Sensor data transformation

3.2. Transform Graph

A graph consists of an arbitrary amount of vertices and edges. The edges form the transition between the vertices. They can either be directed or undirected (bidirectional). Both edges and vertices can have an arbitrary number of properties.

The main function of the graph is to calculate transformations over an arbitrary amount of entities (vertices) e.g. if DroneA sees DroneB which sees DroneC, the graph will be able to calculate the transformation from DroneA to DroneC by passing though DroneB.

In practice the Atlas node uses the graph's vertices as entities and the edges as sensor data. The transformation contained within the sensor data is invertible which makes edges traversable in both directions hence resulting in a bidirectional graph.

3.2.1. Building the graph. Atlas expects the user to specify the entities present in the world on startup (normally via config file) which are then represented as vertices in the graph. Hence, the vertices present in the graph are static, contrary to the edges which are created and destroyed on the fly whenever new sensor data becomes available.

- Vertices (Entities):

The vertices, representing entities, are statically specified in the config and are known to Atlas on startup. The *world* is also considered to be an entity. As such, the graph of an empty world always contains a vertex called *world*.

The data saved within a vertex is shown in (Table 2).

- Edges (Sensor information):

The edges represent sensor information. They are always created in both directions by using the transformation matrix and its inverse. Which means that for every sensor information there are two edges created in the graph.

The *stamp* field keeps track of the edge's age. Whenever the edge is updated with new sensor information its age gets set to zero. If it exceeds a certain value (typically $250ms$), it is removed. This process makes sure that no old information is used to calculate the pose of the entities. On the other hand, if a sensor does not provide information for a few frames, its last information might still be valuable as the pose did not change much during that time, and hence gets used for pose calculations.

3.2.2. Evaluating the graph. Evaluating the graph requires some care due to the following problems:

1) Dependencies: Naturally the pose of an entity depends on the pose of its observers (recursively).
2) Cyclic dependencies: Sometimes the pose of an entity depends on the pose of the observer and vice versa.
3) Quality: The quality of the measurements decreases with the distance from the world as the error/uncertainty increases with every transformation step.

TABLE 2. DATA STRUCTURES OF THE GRAPH

Type	Field	Description
Vertex	Name	The entity's name
	Pose	The entity's pose in world space
	Level	The distance in the graph from *world*
	Evaluated	Flag that tells if the entity has been evaluated
Edge	Key	A unique key describing the origin and target of the information
	Transform	The transformation assigned to the edge (directed)
	Stamp	Time of the last activity
	Sigma	Standard deviation of the sensor

To tackle these problems, Atlas evaluates the graph in two phases:

- Broad Phase:

The broad phase performs a breadth-first search of the graph structure starting from the *world* vertex. The algorithm works as follows:

1) Create a stack (or queue) starting with the starting vertex
2) Take the topmost vertex from the stack and add its unvisited children to the bottom of the stack
   a) Set the vertex's *level* to its parents *level+1*
   b) Mark the vertex as visited
3) Go to 2) until all vertices are visited

The starting vertex is always the *world*. This makes sure only vertices connected to the world are visited while those who are in no way linked to the world or any of its children are not. As shown in (Figure 4) the vertices D and E are not visited which means that their pose cannot be determined in the world space.

After the broad phase, all the vertices' transformation steps with respect to the *world* vertex are known. Using this information will be sufficient to tackle the problems of the graph evaluation.

- Pose Evaluation Phase:

Now follows the actual evaluation of the pose using the level information from the broad phase. The order in which the nodes are visited is the same as during the broad phase. The pose $\mathbf{P}$ can easily be calculated by fusing their parents' poses multiplied by their respective transformations relative to each other:

$$\mathbf{P}_{child} = \text{Fuse}\left(\mathbf{P}_{Parent1} \cdot \mathbf{T}_1, \mathbf{P}_{Parent2} \cdot \mathbf{T}_2, ...\right) \qquad (2)$$

The children's parents eligible to be used during fusion need to have a direct edge to the child and to be exactly one *level* lower in the stack. This avoids cyclic dependencies by not considering vertices at the same level and also avoids fusing data with inferior quality by ignoring vertices with a higher *level* i.e. greater uncertainty.

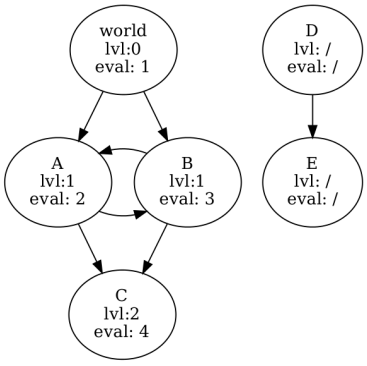

Figure 4. Transform graph after broad phase. Node A and B form a cyclic dependency. Vertex D, E are not connected to the world vertex and hence not evaluated. The field *eval* denotes the order in which the vertices are evaluated. The field *level* denotes their distance from the world.

4. Filtering and fusion

The Atlas package is using an exponential moving average (EMA) to filter sensor data and a weighted mean filter to fuse it. The exponential moving average (EMA) also called single exponential smoothing [6] is an infinite response filter which can be formulated as

$$Y_t = Z_t \cdot \alpha + Y_{t-1} \cdot (1 - \alpha) \qquad (3)$$

Z_t being the observation at time t, Y_t the output of the filter at time t and Y_{t-1} being the previous filter output. The constant α can be seen as the memory of the filter. With α close to 1, the filter will forget old values within a few iterations. With $\alpha = 1$ the filter will simply pass through the measurement value Z_t. With α close to zero, the filter will keep old values in mind much longer (Figure 5). Contrary to the weighted moving average, the exponential moving average does not keep track of all values within its observation window. In fact, only the last value Y_{t-1} has to be kept in memory, making it very fast and lightweight thus ideal for real time applications.

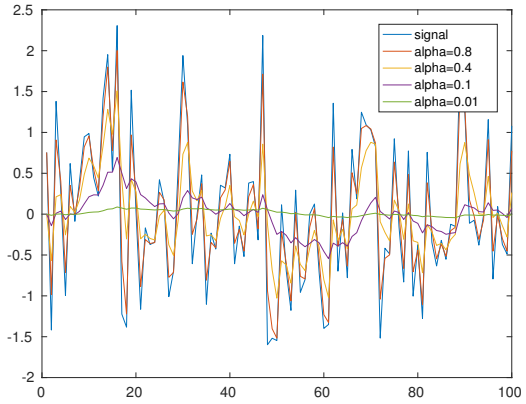

Figure 5. Influence of α on the behavior of the EMA filter. The filter gets slower with decreasing α.

The weighted mean is used to fuse sensor data from multiple sources, resulting in a single more reliable and qualitatively better measurement. It is one of the simplest [4] fusing methods and well suited for real time applications. The weighted mean is calculated as follows:

$$\vec{y} = \frac{\sum w_i \cdot \vec{x}_i}{\sum w_i} \qquad \begin{array}{rcl} \vec{y} & = & \text{output} \\ \vec{x}_i & = & i\text{th sensor data} \\ \vec{w}_i & = & i\text{th weight} \end{array} \tag{4}$$

While this is straight forward to calculate for scalars and vectors things become more difficult when talking about quaternions. The main problem here is that the quaternion q and $-q$ describe the same rotation [7], but using the formula above they would cancel each other out. To address this issue [7] describes a different method to calculate the weighted average of quaternions by

$$Q = \begin{bmatrix} w_1 \cdot q_1 & w_2 \cdot q_2 & \dots & w_n \cdot q_n \end{bmatrix} \tag{5}$$

Then the weighed average of the quaternions is given by calculating the normalized eigenvector corresponding to the largest eigenvalue of $Q \cdot Q^T$.

The quality of the fused result depends largely on how well the weighting factors w_i can be chosen [4]. Atlas calculates the weighting factors w_i by looking at the standard deviation σ_i of all the sensor readings in question:

$$\sigma_{min} = \min(\sigma_i) \tag{6}$$

$$w_i = \frac{\sigma_{min}}{\sigma_i} \tag{7}$$

The sensor readings with a lower standard deviation will thus be weighted higher, hence having a greater influence on the result. This is important as it allows high quality sensors to outvote their lower quality counterparts.

5. Validation

The node has been developed using the test driven development approach. This means that for each (important) part of the node there exists a unit test making sure it works as intended.

5.1. Test 1: Self localization test

Moreover, a field test was conducted in order to test the self localization functionality of the Atlas node. In the SnT flight arena a total 13 AprilTags (size: $18cm$) were placed on the floor. The markers' pose was captured using the OptiTrack motion capture system and subsequently translated into a YAML config file with the markers attached to the *world* entity of the Atlas node. A Parrot AR.Drone 2 was used as testing drone to locate itself using the markers on the floor by fusing one or more detected marker poses together. The AR.Drone was defined as another entity in the YAML config file with the front camera being defined as a sensor. Due to the weak resolution (320p) of the bottom camera, the front camera (720p) had to be used by pointing it to the bottom. The camera's pose relative to the base-link of the drone had been taken into account by specifying the corresponding sensor transformation (entity to sensor transformation). Finally, the drone's pose as detected by the Atlas node and by the OptiTrack system was recorded while piloting the drone arbitrarily through the flight arena.

The results of the test are shown in Figure 6. The position of the drone as seen by the OptiTrack system and as seen by the drone (Atlas node) is plotted over time. Moreover, the total number of measurements which is equivalent to the number of markers detected at any time is shown in Figure 6. Following observations can be made:

- The drone was oscillating heavily (related to its controller's parameters)
- The detection rate of the markers was quite poor due to the oscillating movements of the drone and the resulting motion blur
- The position as given by the Atlas node corresponds to the reference position of the OptiTrack system
- The number of measurements is varying greatly, often dropping to zero. Again a result from motion blur.

5.2. Test 2: Entity localization test

In this test, two Parrot AR.Drone 2 are supposed to localize another entity (in this case a marker with a size of $12cm$) using their front cameras while moving around the marker and always keeping it in sight. The actual position of the marker has been measured using the OptiTrack system and is at:

$$\vec{p}_0 = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -4 \\ -52 \\ 9 \end{pmatrix} mm$$

The results of the experiment are visualized in (Figure 7) which yields to following observations:

- The drones are able to localize other entities (Figure 7). Although only with a precision of about $30cm$. The results in (Figure 6) are much closer to their actual value which might be related to the marker size and viewing angle.
- The detection rate was high as can be seen from the number of measurements. Most of the time at least one drone was able to detect the marker. As opposed to test 1, the drones were moving much slower, hence reducing motion blur.

6. Conclusion and future work

The results of test 1 have shown that the drone can locate itself using the markers placed in the environment and that the position corresponds to the actual position as measured by the OptiTrack system. The result show furthermore, that motion blur is in fact a practical problem as it prevents the detection of the markers during (fast) movements. Fortunately, this is a hardware problem and can be mitigated by reducing the exposure time of camera

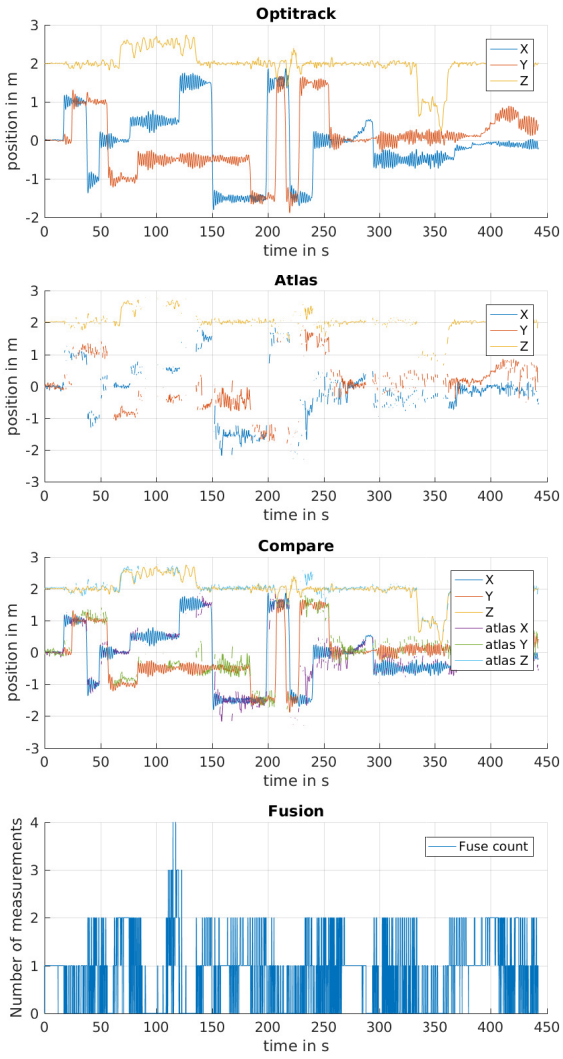

Figure 6. Results of test 1: (From top to bottom) Data from the OptiTrack system which serve as reference, data from the Atlas node (fused position), comparison of the results from the OptiTrack and Atlas, number of measurements for a given position

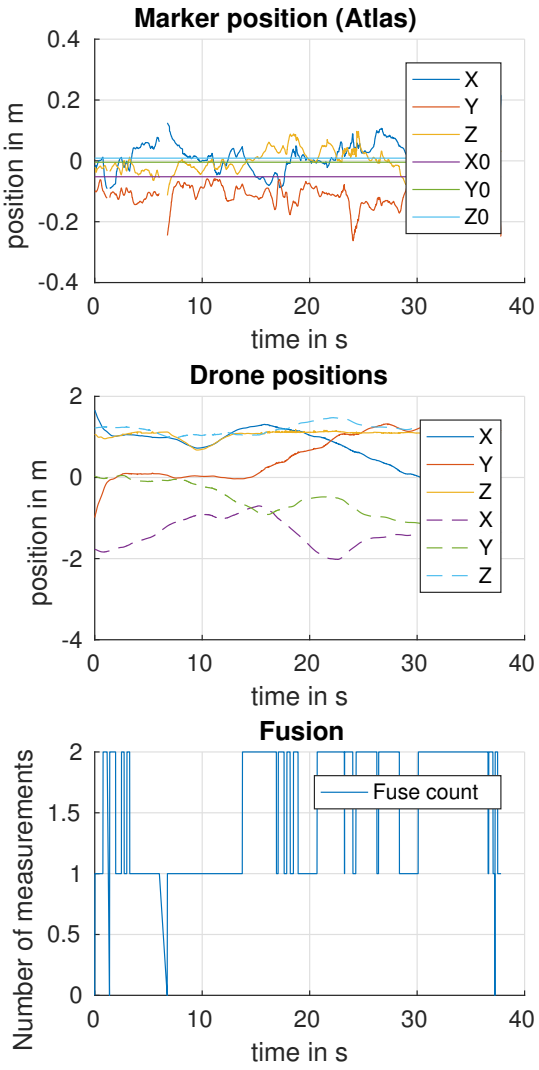

Figure 7. Results of test 2: (From top to bottom) The position as measured by Atlas and also the actual position (X0,Y0,Z0) as measured by OptiTrack, the position of the two drones, the amount of measurements for the given entity.

e.g. by increasing the brightness of the environment, using global shutter cameras or by fusing a different type of marker. The results of test 2 have shown that the drone(s) can localize other drones although with rather weak precision. Nevertheless, the Atlas node works as expected and the inaccuracy is related to the hardware of the drone (i.e. the camera) and related to the small marker size. Hence, the experimental results validate the cooperative localization capability of the presented framework.

Future development will focus on the extension of framework features, as for example it is useful for some applications to have the relative pose of two entities. Currently this is only possible if the pose of the entities is known in world space. This is due to how the graph is evaluated using a breadth-first algorithm starting from *world* (Figure 4). Furthermore, the current Atlas implementation favors the most direct path when evaluating the transformation graph. Although it is logical that the shortest path yields the best results, this might not be the case if the sensor quality is differing significantly. A possible solution could be to use the standard deviation information of the graph as *distance* parameter and consequently using it to find the most promising transformation chain using a path finding algorithm like Dijkstra. The final focus of future work is the implementation of an Extended Kalman Filter (EKF) fusion filter [4] instead of the primitive fusion algorithm (weighted average). EKF has the advantage to be applicable to the non-linearity of quaternions. In addition the filter inherent system model leads to higher accuracy and can extrapolate the pose in small periods of missing sensor data.

References

[1] E. Olson, "AprilTag: A robust and flexible visual fiducial system," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3400–3407, 2011.

[2] G. C. La Delfa, S. Monteleone, V. Catania, J. F. De Paz, and J. Bajo, "Performance analysis of visualmarkers for indoor navigation systems," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 8, pp. 730–740, aug 2016. [Online]. Available: http://link.springer.com/10.1631/FITEE.1500324

[3] M. Toyoura, H. Aruga, M. Turk, and X. Mao, "Mono-spectrum marker: An AR marker robust to image blur and defocus," *Visual Computer*, vol. 30, no. 9, pp. 1035–1044, 2014.

[4] S. Blank and F. Tobias, "A Case Study towards Evaluation of Redundant Multi-Sensor Data Fusion," *Robotics*, 2010.

[5] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, jul 2009. [Online]. Available: http://pub1.willowgarage.com/{∼}konolige/cs225B/docs/quigley-icra2009-ros.pdfhttp://www.ncbi.nlm.nih.gov/pubmed/8844323http://arxiv.org/abs/1106.4561http://dx.doi.org/10.1613/jair.1129

[6] P. Kalekar, "Time series forecasting using Holt-Winters exponential smoothing," *Kanwal Rekhi School of Information Technology*, no. 04329008, pp. 1–13, 2004. [Online]. Available: http://www.it.iitb.ac.in/{∼}praj/acads/seminar/04329008{_}ExponentialSmoothing.pdf

[7] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman, "Quaternion Averaging," *NASA Goddard Space Flight Center*, pp. 1–10, 2007.