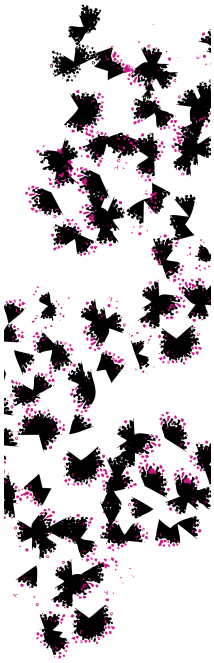
RAM ROBOTICS AND MECHATRONICS



Collaborative position control of an UAV using vision and IMU data

R.W.P. (Robin) Hoogervorst

BSc Report

Committee:
Dr.ir. F. van der Heijden
Dr.ir. M. Fumagalli
Dr. H.K. Hemmes

July 2015

017RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE.

MIRA BIOMEDICAL TECHNOLOGY AND TECHNICAL MEDICINE

CTIT

Summary

Pose estimation for UAVs is usually done using internal sensors. Often a filter is used to combine different sensors, e.g. a camera on the UAV and its IMU. However, sensors on the UAV have limitations. For instance the camera on the UAV needs to detect keypoints, which might not always be available in real-life applications.

Therefore it is investigated if the use of an external UAV can aid in the pose estimation of the drone. The camera from the external UAV will detect the pose of a marker on the main UAV. This vision measurement and the IMU from the main UAV are fused using a Kalman Filter to provide a reliable pose estimation. Using this pose estimation, control experiments are performed to validate if the UAV can be controlled using this estimate.

Results show that this is possible, first when the external UAV is at a fixed position and also when the external UAV is flying next to the main UAV. The control performance decreases during a simultaneous flight, but control is still possible. A limitation is that the UAV needs to stay in view of the camera, otherwise there is a chance of the UAV moving out of vision, causing it to drift off and crash.

Contents

1 Introduction

UAVs have a great potential for everyday use and are a popular topic for research nowadays. The potential of UAVs is big, especially when flying autonomously. They can take over certain human tasks that are dangerous or have to be done at places that are difficult to reach. Compared to ground robots, UAVs can easily and quickly manouvre and fly to great heights, making them ideal for inspection of big industrial objects.

Aeroworks is a research project that looks into "collaborative Aerial Robotic Workers". Industrial inspection and maintenance work is dangerous and costly. Its goal is to reduce these risks and costs by using collaboration of multiple UAVs. The goal is to have an autonomous group of UAVs executing infrastructure inspection and maintenance works. As an illustration, a typical kind of inspection would be investigating the blades of a windmill or a wind turbine inside a combustion chamber. For these investigations to be more useful, some kind of interaction is needed as well. As a result, an UAV needs to fly close to the object it tries to investigate.

In these situations, the UAV needs to behave autonomously. Therefore it is vital to have a good pose estimate of the UAV. Usually this is done using internal sensors, which include camera, accelerometer, gyroscope, GPS, etc. In real life applications, it may be that the information from one of these sensors cannot be used. In fact, when the UAV is close upto the windmill to perform interaction as described above, it cannot use a front mounted camera due to lack of texture on the windmill blade. External tracking options are not really an option as well in that situation. The UAV needs to operate in an environment that is unknown and is different each time. Adapting each possible area of operation to include sensors for aid of pose estimation of the UAVs is expensive and brings more complications with regard to infrastructure. This makes it useful to look at the collaboration of multiple UAVs to detect each others pose and use the fusion of their knowledge to obtain a good pose estimation of every UAV and make control of the group better.

1.1 Previous work

For pose estimation of a single UAV, research generally focusses on sensor fusion of the local sensors. [Matias Tailanian (2014)] shows a succesful fusion of GPS and IMU information using a Kalman Filter, overcoming the problems of different sample rate. [Jeroen D. Hol (2007)] shows pose estimation on a 6DOF robot, using an Extended Kalman Filter to fuse this information. [Luis Rodolfo Garcia Carrillo (2011)] fuses stereo vision odometry together with inertial measurements on a quad-rotor UAV using a Kalman Filter. They are able to control the UAV using this fused data stream. These researches show that IMU information is a good way to add fast pose estimation information. An often used method for pose estimation of UAVs is using a Simultaneous Localisation and Mapping algorithms (or SLAM, for short) algorithm. These researches also make use of visual and inertial measurements to generate a map and a pose estimation of the robot itself in that map.

[Christian Forster (2013)] uses the collaboration of multiple UAVs to combine multiple SLAM algorithms and create a more reliable pose estimation using that.

Other research focusses on tracking external objects using vision. [Markus Achtelik and Buss (2009)] makes use of an external stereo vision camera to detect an UAV with illuminated markers. This data is also fused together with inertial measurements to determine a reliable pose, but not by using a Kalman filter. A method for detecting a simple marker is shown by [Olson (2011)]. They show the performance of their detection algorithm which performs better than already existing techniques. [Elias Mueggler and Scaramuzza (2014)] uses these tags to control a robot using relative vision measurements from an overseeing UAV. The UAV detects AprilTags

markers in a environment and aids the robot in navigation, showing a practical example for use of these tags.

1.2 Project aim

While an UAV can estimate its pose using internal sensors, using the collaboration of a second UAV and its camera can aid when an internal sensor or an combination of those can not be used. For this collaboration to work, sensor fusion of external and internal sensors is needed. This work combines different imlementations for pose estimation that has already been done and implements this for use with two UAVs. It will be investigated which problems will occur when using this collaboration for control of the UAV.

In this project, the camera from a second UAV will detect a marker on the main UAV and determine a pose of that. After that, a Kalman Filter is used to fuse this information together with IMU information of the main drone. This combines the fast measurements of the IMU together with the slow measurements from the camera, in order to make the final estimate more reliable. It is assumed that the external UAV knows its full pose, which is realised by using the OptiTrack system as reference. In the final experiments, the main UAV will be position controlled using this final estimate.

2 Background information

2.1 Pose estimation

There are several methods for determining the pose of an object. Most of them are based on data from sensors, like using an IMU or a camera. Other sensors like GPS an ultrasone sensor can provide only a position (and no orientation, which is included in a pose as well). GPS can only be used outdoors and is a way to determine a position on the globe. The accuracy of this is usually in the range of meters, and the update rate is very slow. Ultrasone sensors can be used to measure a distance to the nearest object. This is usually pretty accurate and fast, but it is not known what is being measured exactly and just in one direction. For example, this can be used to determine the height of an object easily. For this project, pose estimation using an IMU and camera are used and thus will be further introduced.

2.1.1 Pose estimation with IMU

One method of estimating a pose is with use of acceleration. When the acceleration of an object is measured, its position can be determined by integrating these and adding it to an initial state (2.1). It immediately shows that there are two unknowns, x_0 and v_0 which highly influence the estimate. Usually these are set to 0, which means the measurement starts with the drone at (0,0,0) with no movement.

$$\vec{x}(t) = \vec{x}_0 + \int_0^t \vec{v}_0 + \int_0^t \int_0^t \vec{a} \tag{2.1}$$

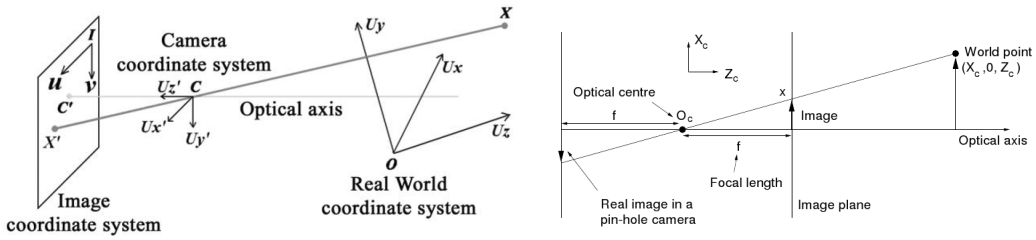
These accelerations can be measured using an Inertial Measurement Unit, or IMU for short. An IMU is a combination of accelerometer and gyroscope and measures accelerations and rotations using these. Al is always the case with sensors, the measurement is not perfect. This measurement includes a bias $\vec{b_a}$ and random noise Ω (2.2). The bias is not a fixed value, but can depend on the state of the IMU, e.g. a fast acceleration or temperature can change the bias. As a result this bias needs to be estimated and subtracted from the measurement to know the acceleration. The noise has a zero mean, which gives a 0 result when integrated. For this reason, this noise can be neglected during the integration.

$$\vec{a_{meas}} = \vec{a_{real}} + \vec{b_a} + \Omega \tag{2.2}$$

The advantage of using the IMU is that it is fast. Even cheap IMUs have the capability to provide datastreams of 200Hz, while other IMUs can provide streams at a rate of several kHz. This allows for a quick pose estimate. On the contrary, the bias and initial conditions are unknown and need to be estimated and there can be a lot of noise. This makes the pose estimate unreliable, especially when integrated over a longer period of time. The better the IMU, the smaller these deviations will be, but they will always be present nevertheless.

2.1.2 Pose estimation with vision

Another method of estimating a pose is by using a camera. A camera can be modeled by using a pinhole camera model. Keypoints of the object to be detected are determined, of which the exact relative position is known. Then the object position in 3d world frame can be calculated using the camera intrinsics (focal length, optical centre and distortion coefficients), the pixel positions of keypoints in the 2d projection and the actual relative position of the keypoints. The object creates an image in the camera plane. When the focal length and optical center are known, the pose of these points can be calculated. The calculation is graphically shown in figure 2.1.



(a) 3d camera mapping overview **(b)** 2d overview mapping

Figure 2.1: Mapping of point in 3d space to an image in camera frame

OpenCV[1] is an excellent C++ library for image processing and has features to do these calculations. First, camera calibration is performed, to determine the camera intrinsics. When this is done, a set of object points and image points can be provided to the *solvePnp* function of OpenCV which gives a 3d pose relative to the camera.

In contrast to the use of the IMU, this pose estimation is only dependent one one measurement. This makes cumulation of errors not possible. Quality of the camera can be easily improved by using different camera and we can use multiple sources. The two big downsides are the speed and the need of keypoints. Processing of the image can take long time (50-300ms depending on the type of detection, computer and resolution) and this time increases exponentially when using a higher resolution. Regular cameras have a framerate of 60 fps or even 30. This combination creates a slow set of measurements. Secondly, the camera needs some way of detecting key points. When an external camera is used, a marker can be used to determine these points. When an on-board camera is used, keypoints are detected from the environment and the relative position and speed of the camera are usually calculated using those. But in order to use this, the environment needs to provide this. This can be a textured wall or specific objects, but it is not guaranteed that these are available.

2.2 Sensor fusion

These methods of pose estimation can be combined to utilize the advantages of each method. Combining this information is done using sensor fusion. There are a wide variety of algorithms available to combine measurements. Since these measurements include an error, fusion aims at finding the optimal value of these estimates. Examples of this include the central limit theorem, filtering based on bayesian networks and Kalman filters. The central limit theorem and bayesian networks use probability theory to calculate a most optimal mean. A Kalman filter is an algorithm to combine different noisy measurements. This filter is popular for sensor fusion and will be the one used.

The Kalman filter is a linear discrete time algorithm that provides an optimal estimation of two gaussian measurements. It consists of an prediction and an update step, which can run independently from each other. During the prediction step, a state estimation is made. The update step compares the prediction to a measurement and filters these based on covariances to create a more reliable state estimation. The basic equations are given and explained here, while the implementation of it is discussed in section 3.5.1.

Prediction step

First the prediction step. It uses the state at time k-1 ($x_{k-1|k-1}$) and an input vector (u_k) to predict a new state (eq 2.3). F_k and B_k are the state and input matrices respectively and are the core of the calculation. The covariance gets calculated with use of the state matrix. Q_k is the

[1] http://opencv.org/

process noise, which is in this case just the covariance of the input.

$$\mathbf{x}_{k|k-1} = F_k\mathbf{x}_{k-1|k-1} + B_k\mathbf{u}_k \tag{2.3}$$

$$P_{k|k-1} = F_kP_{k-1|k-1}F_k^T + Q_k \tag{2.4}$$

Update step
When a prediction is available, an update step can run to change the state based on the difference between the prediction and the measurement. H_k is the measurement matrix that converts state model to the measurement model. The update step first calculates a diff y_k based on the measurement and the state.

$$\mathbf{y}_k = \mathbf{z}_k - H_k\mathbf{x}_{k|k-1} \tag{2.5}$$

Based on the covariances, the Kalman gain factor is calculated.

$$S_k = H_kP_{k|k-1}H_k^T + R_k \tag{2.6}$$

$$K_k = P_{k|k-1}H_k^TS_k^{-1} \tag{2.7}$$

This gain is used to apply the difference proportionally to the state. At the same time, the gain is used to adapt the covariances.

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + K_k\mathbf{y}_k \tag{2.8}$$

$$P_{k|k} = (I - K_kH_k)P_{k|k-1} \tag{2.9}$$

This Kalman filter can be used on linear systems. Since the physics of an UAV is non-linear system, the Kalman filter should be extended, giving an Extended Kalman Filter. This extension essentially linearises the system for the filter by changing the F and H matrices to a function. With this in mind, equations 2.3 and 2.5 will change to:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{2.10}$$

$$\mathbf{y}_k = \mathbf{z}_k - h(\mathbf{x}_k) \tag{2.11}$$

These functions cannot be aplied to the covariances directly, thus the Jacobian of these functions is computed and substituted for the F and H matrices in the calculations with the covariance.

$$F_k = \frac{\partial f}{\partial \mathbf{x}} \qquad H_k = \frac{\partial h}{\partial \mathbf{x}} \tag{2.12}$$

3 Implementation

The final goal of this project is to control an UAV using collaboration. For this to work, an software implementation has been written that will handle the information of both UAVs. First, the hardware will be introduced. After that, the software architecture is shown generally and each part will be dicussed in greater detail. Although this project makes use of each of these elements to control the UAV, each element of this implementation could be used individually.

3.1 Hardware overview

The hardware is the base for the software to operate on. As already discussed, two UAVs will be used to control one of them, shown in figure 3.1. Drone B will provide a measured pose of the marker, which is attached to drone A. Drone A itself will publish its IMU measurements. These two will be fused to create a pose estimate for drone A.

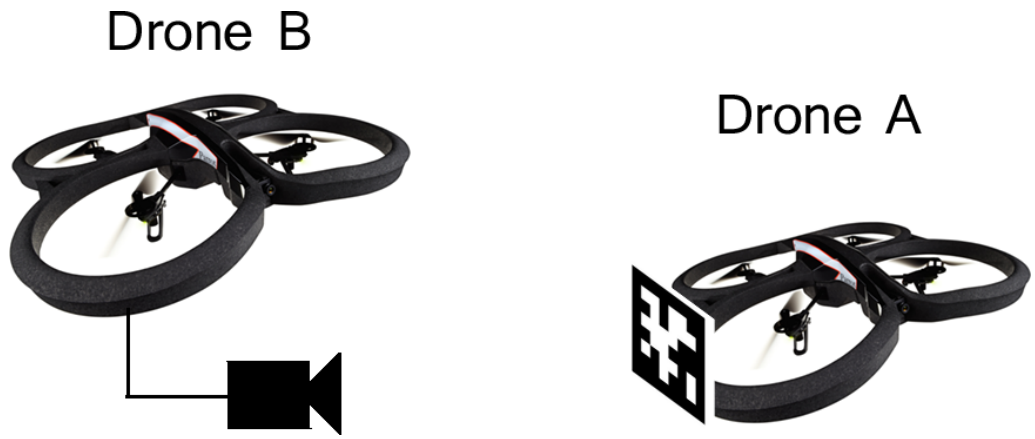


Figure 3.1: Overview of the experimental setup. Drone A has a marker on the side. Drone B is used for the camera stream, using its internal camera.

As UAV the AR.drone 2.0[1] is used, made by the company parrot (shown in the set-up figure 3.1). This drone is meant to be controlled by a tablet or smartphone and mostly used as a toy. It comes at a cheap price (about 300$), which means it has cheap hardware. The AR.Drone is a quite closed platform, not many details are available of the internals. One of the big advantages is that the parrot AR.drone has a lot of features build in, including:

- HD camera (720p) 30 fps[2]
- Wide angle lens
- Auto pilot for hovering
- 3 axis gyroscope
- 3 axis accelerometer

There is an SDK available, which makes it easy to interact with the drone for experiments, but this can only perform interactions with the basic input and output of the AR.Drone and not influence the inner control loops of the AR.Drone. [Pierr-Jean Bristeau (2011)] shows some of the internals of the first AR.Drone, which can be used to get some more insight of the internals.

[1] http://ardrone2.parrot.com/

[2] Due to limitations by the driver for ROS, only a 480p image stream can be obtained

The accuracy of the position estimate and performance of control needs to be evaluated. Besides that, it is assumend that drone B knows its pose and the pose of drone A is used to measure the results. Therefore, it is neccessary to know the ground truth pose of both UAVs. In order to do this, the OptiTrack system is used, made by the company NaturalPoint. The OptiTrack system uses a set of high-speed infrared cameras to detect reflective balls which can be mounted to an object. By using a combination of these balls and the software set from the OptiTrack, a full pose of multiple objects can be detected at a rate of 100Hz. The accuracy and speed of this system is high and therefore ideal to be used as ground truth for the measurements. When this project refers to the ground truth pose, it references to the pose estimate from the OptiTrack.

The UAV only has one marker mounted on it. This marker is mounted pointing in the y-axis of the inertial frame of the UAV. The built-in camera of drone B is used, which is pointing in the x-axis of its inertial frame. This marker will be kept into view of the camera by using the position controller for the UAVs. The yaw of each UAV is kept constant with an offset between them of about 90 degrees.

Control of the drones is done using an implementation of a PD controller, created internally at RaM. This allows us to control both UAVs on a setpoint that is provided. This implementation is made by Cees Trouwborst and is called the *ram_ba_package*. Details of his implementation can be found at (https://www.ce.utwente.nl/aigaion/publications/show/2312). For position estimation experiments, the OptiTrack pose is used as input to keep the drones at a position. For control experiments, this same controller is used, but instead of using the OptiTrack pose as source for the controller, the filtered pose estimate is fed into it.

3.2 Software architecture

The software implementation is built on top of ROS. ROS is an open-source framework for robot software. ROS consists of packages, which contain nodes. These nodes can create topics, which allow for dataflow between them. Each topic can subscribe to a topic to retreive the data from there and can publish information to an own topic. Due to these nodes and packages, ROS is highly modular and allows for easy reuse of different packages.

The code is run on an external computer which communicates with all elements. No image processing is done on the UAV itself. The computer is using WiFi to communicate with the UAVs and OptiTrack information is received through a wired network. This communication is done using already existing packages. *Mocap_optitrack* is used for the OptiTrack interface and the *autonomy_ardrone* for the AR.Drone. Those packages provide the information that is needed into ROS topics, to be used by other nodes. The WiFi communication with multiple AR.Drones is also a feature of the position controller made by Cees Trouwborst. The combination of these packages allows for easy communication between all elements.

The code exists of one ROS package with two seperate nodes, called *ram_tracker_buddy*. One node handles the vision detection while the second node handles the position estimation using the filter. The code is split up into three main sections, graphically shown in figure 3.2:

1. Camera pose estimator
2. Detector
3. Fusion Filter

The *Camera pose estimator* is the first ROS node. It subscribes to an image stream and the pose of drone B given by the OptiTrack. It will publish a 3d pose of the detected marker in the global frame. This node makes use of the *Detector*, which makes an easy interface to detect the pose of a certain marker within an image view. The third and most important component is the *Fusion*

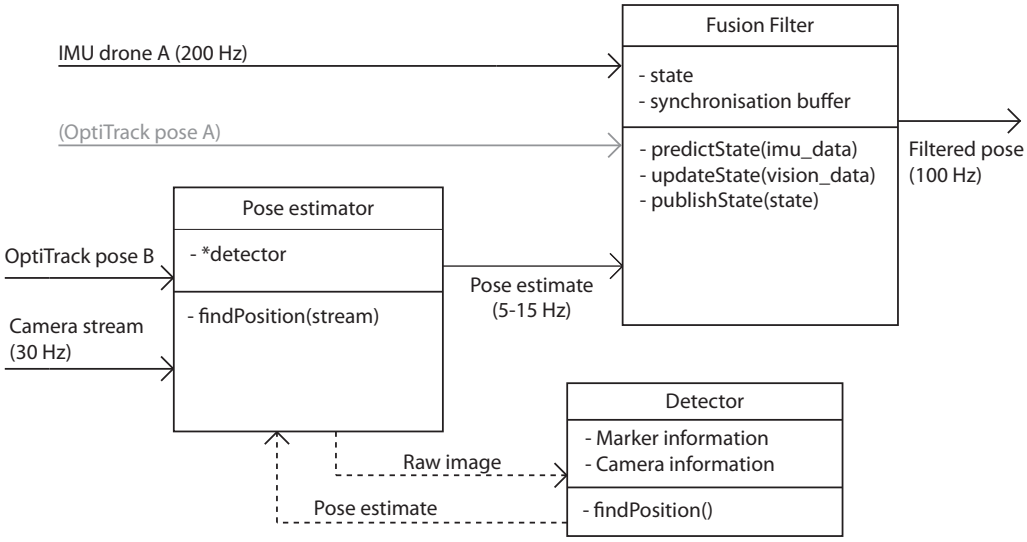


Figure 3.2: A (simplified) overview of the codebase. Solid lines are ROS topics, while dashed lines represent relations between the classes. The OptiTrack pose of drone A is gray since it is used for calculations with the orientation, but this should not be neccessary to use.

Filter. This is the second ROS node, houses Kalman filter and produces the filtered position estimate.

3.3 Vision pose estimation

The vision esimation is handled by the Camera pose estimator. This node is the bridge between the camera stream, the marker detection and the 3d pose of the marker.

Inputs: Camera stream from the drone, ground truth pose of drone B
Outputs: Marker pose in global frame

It uses a ROS camera stream as input. For each image that is received, it will convert this image to OpenCV format and call a detector to get the pose for that image. The pose that is received from the detector is transformed into world frame and published. The image stream is 30 fps and thus 30Hz. The processing time is too long to be able to provide measurements at 30Hz as well. With that in mind, images that are received during processing are discarded and the first image received after processing has completed is used for the next measurement. This is done automaticcaly by ROS, since it discards messages received if the process is still busy. The rate of pose estimation from vision is about 5-15 Hz, depending on the environment.

When receiving an pose of drone B, it stores the position of drone B in a buffer together with a timestamp. When doing transformations to get the marker pose in global frame, it will use the pose of the drone at the time the image was received. This pose needs to be looked up in the buffer.

3.3.1 Transformations

The camera pose estimator transforms the measured position of the marker into the global frame. The total transformation loop is graphically defined in figure 3.3.

To begin with there is the transformation of the marker in camera frame. This defines the orientation and position of the marker, relative to the camera. Due to initialisation of the drones, the marker also has a fixed rotation relative to the camera. Which is defined as R_{m_fixed}

$$H_{MA}^{CB} = \begin{bmatrix} R_{MA}^{CB} \cdot R_{m\_fixed} & P_{MA}^{CB} \\ 0 & 1 \end{bmatrix} \tag{3.1}$$

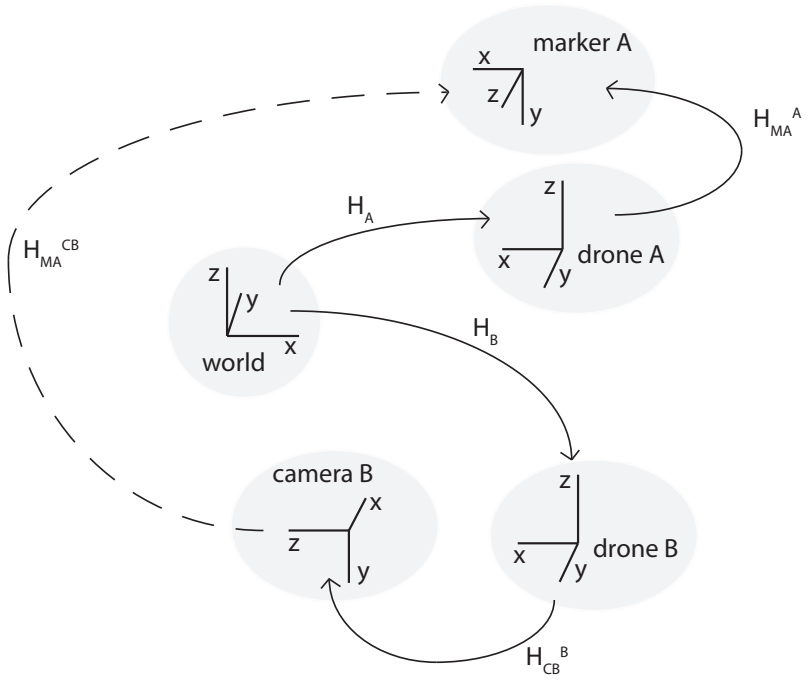


Figure 3.3: Overview of transformations

$$R_{m\_fixed} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \tag{3.2}$$

Here R_{MA}^{CB} and P_{MA}^{CB} are the position and orientation relative to the camera, which will be measured by the detector.

Secondly, there is the transformation of the camera with relation to the drone. This is a fixed rotation, including a translation of 21 cm on the x-axis. The same could be defined for A, but since that camera is not used, it is not shown here. The transformation is defined as:

$$H_{CB}^{B} = \begin{bmatrix} R_{CB}^{B} & P_{CB}^{B} \\ 0 & 1 \end{bmatrix} \tag{3.3}$$

$$R_{CB}^{B} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \qquad P_{CB}^{B} = \begin{bmatrix} 0.21 \\ 0 \\ 0 \end{bmatrix} \tag{3.4}$$

H_A and H_B are the position and orientations measured from ground truth[3]. The zero orientations for the drones are defined with an 180deg rotation around z compared to the ground truth, due to definition used by the position controller. Both drones are (obviously) able to rotate. This gives us the following transformation matrix.

$$H_B = \begin{bmatrix} R_B \cdot R_{180z} & P_B \\ 0 & 1 \end{bmatrix} \tag{3.5}$$

Where R_B and P_B are the rotation matrix and position optained from the OptiTrack for drone B, respectively. The same can be written for drone A, but this is practically only used for verification and not in this node.

[3]The OptiTrack driver for ROS implements its own rotation, which is a rotation of 90 deg ccw around x. These rotations are based on the output of the driver and thus are different from the global frame shown in OptiTrack software

The final position of the marker in global frame is then easily determined by combining these transformations (3.6).

$$H_{MA} = H_B \cdot H_{CB}^{B} \cdot H_{MA}^{CB} \tag{3.6}$$

3.4 Marker detection

The marker detection is done using the detector. This detector exists of multiple classes:

- DetectorFactory
- AbstractDetector
- AprilDetector
- DotsDetector

The DetectorFactory generates a detector, based on an identifier string that is provided. Each detector is an implementation for detection of a different type of marker. Accordingy, each detector must incorporate the AbstractDetector, allowing for the same interface to be used throughout when swapping these. The AbstractDetector has an interface for an actual class to use and some helper functions for debugging. In the ros package for this project, two types of markers are supported: AprilTags and a marker created internally at RaM, by Jort Baarsma. For the latter, the DotsDetector is implemented.

3.4.1 AprilDetector

The final implementation used the AprilDetctor for detection of the marker. This detector is an implementation that uses the ApilTag library to detect images. For detection of the drone, an AprilTag[4] marker has been used (figure 3.4).

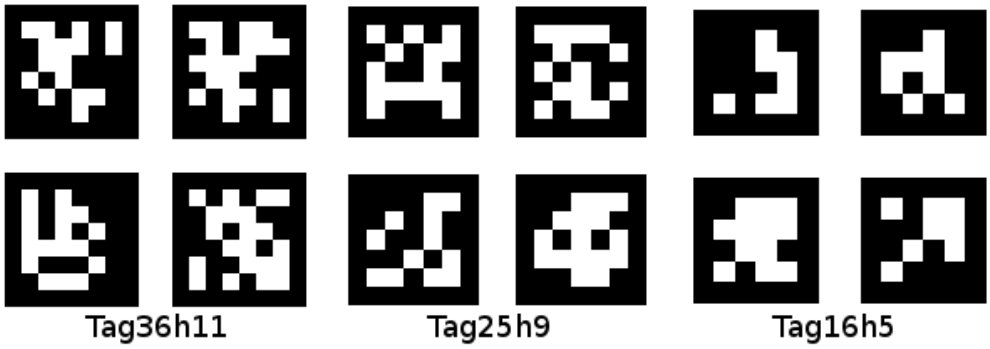


Figure 3.4: Samples of different sets of AprilTags

The detector makes use of the 36h11 set, since that is the recommended one. A detection library has been developed by the University of Michigan in C, with examples of use within ROS. The big advantage of these tags is that they can be easily printed and mounted.

Furthermore, the detector can detect a set of different tags, each with a different id. This makes it possible to use different tags for detection, to identify different UAVs, different sides of an UAV, or other objects for the UAV to interact with.

The default library uses the solvePnp function from OpenCV to calculate the 3d pose from the detected points of the marker in the image and the known properties of the image (e.g. its size). The library does not pass distortion coefficients to the solvePnp function. However, those are necessary when using the camera from the AR.Drone, because there is quite some lens

[4] http://april.eecs.umich.edu/wiki/index.php/AprilTags

distortion. Therefore, this function is reimplemented in the detector and calls the OpenCV function itself with the distortion coefficients.

3.4.2 DotsDetector

The DotsDetector is the detector for the marker made internally at RaM. This marker uses a set of dots and HSV filtering to detect the pixel position of the marker in the image. After that, it also calls the solvePnp function from OpenCV to determine the 3d pose.

The advantage of this marker is that it is quicker than the AprilTag, since it can filter quickly on color, but this also has its downside. Since this marker relied on filtering based on color, the color balance needs to stay the same or needs to be calibrated. During tests in the SmartXP lab, it seemed that the white balance is different from the lab at RaM. Under those circumstances, it was not able to detect the marker reliably. Therefore the choice was made to use the AprilTag marker for the experiments.

3.5 Sensor fusion

The most important part of the implementation is the sensor fusion. This is done using the fusion filter node, which produces the final pose estimate. Figure 3.5 shows the general overview of all interactions. This node subscribes to the calculated marker position from the camera pose estimator, the OptiTrack pose of drone A and the IMU topic from drone A. From the OptiTrack pose, only the orientation is used for the calculations and the position is neglected. It would also be possible to omit this and use an estimation of the orientation. Therefore, the line is dashed in the figure overview (3.2) and is italic in the input list.

Inputs: Marker pose in global frame, IMU data, *ground truth pose of drone A*
Outputs: Pose estimate

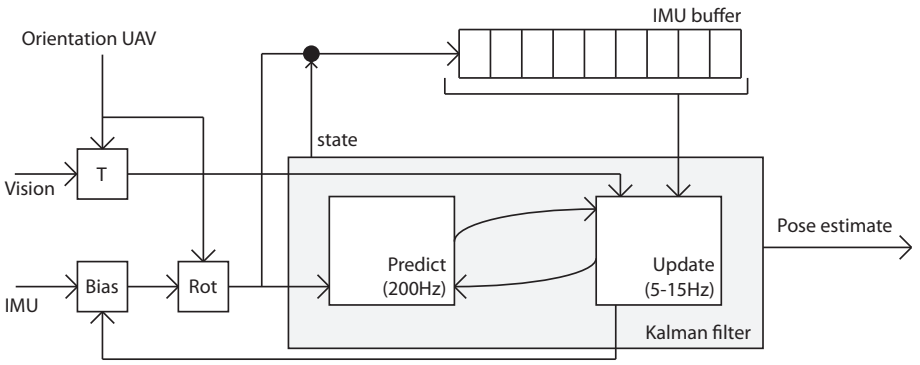


Figure 3.5: Overview of the implementation for the fusion filter. The vision and IMU measurements get feeded into the Kalman Filter, including some preprocessing.

The IMU measurement gets preprocessed by removing its bias and performing a rotation using the orientation of the UAV. A prediction step runs using this data and this data is put into the buffer together with the state at that time as well. The vision measurement gets transformed using the orientation of the UAV. The filter gets the marker position. Hence it needs to calculate the actual position, based on the orientation from the drone (transformation H_A^{MA} in figure 3.3, defined in section 3.5.2).

Using this measurement and the IMU buffer, it performs an update step for the Kalman filter, in a way described in section 3.5.3. The final result of this Kalman filter gets published as ROS topic. For publishing, a fixed rate has been chosen to have it run independently of the Kalman filter. ROS has an timer function, so it is easy to call a publish function at a fixed interval. It has been chosen to implement this at an interval of 0.01 seconds and thus 100Hz.

3.5.1 Kalman filter setup

While earlier it is mentioned that this system should make use of an Extended Kalman filter, it was chosen in the earlier stages to first opt with a standard Kalman filter. The standard filter is easier to implement and this quicker implementation allowed for further experimentation on other problems. Due to time contraints in the end, this has not been improved by creating an implementation of an Extended Kalman filter.

The state of the Kalman filter consists of 6 variables, which is the position and velocity in three dimensions in world frame.

$$\vec{s} = \begin{bmatrix} x, y, z, \dot{x}, \dot{y}, \dot{z} \end{bmatrix} \tag{3.7}$$

Orientation has been left out of this state. Firstly, getting a reliable orientation is more difficult than the position, since the IMU does not give a reliable yaw and the orientation from the marker is slow. Secondly, pulling the orientation calculations out of the Kalman filter decreased the size of the state a lot and simplified the calculations and therefore the implementation of the Kalman filter.

By omitting this orientation, the state matrix only uses identity and a Δt. Since the rotation of the IMU measurment is done before providing it to the Kalman filter, input matrix only uses a Δt to add the acceleration to the velocity. The state matrix $\vec{F}$ and input matrix $\vec{B}$ then become:

$$\vec{F} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \vec{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \tag{3.8}$$

Since the transformations of marker pose to drone are made before putting theem into this filter, the H matrix is a 6x6 identity matrix. The Q_k and R_k matrices in the equations denote the noise of the system. This is just the covariance of each measument on the diagonal of the matrix and the rest 0. Since the covariances of the measurments are not known really well, the choice was made to be able to dynamically adapt these values. In the final experiments, a value of 0.05 for the position covariance, 9.6 for the twist variance and 0.06 for the imu covariance has been used. The twist variance has been chosen higher, since sometimes it would provide non-accurate values. This way, those values would not influnce the final estimate too much, while it is sill able to filter the IMU drift as well.

A velocity measurement is necessary to compensate for the drift in IMU. If this would not be done, the velocity would never be compensated using a fixed measurement, which would increase the error in the final estimate. The velocity for the update step is estimated using two measurements from vision and dividing them by Δt, simply:

$$\vec{v_t} = \frac{x_t - x_{t-\Delta t}}{\Delta t} \tag{3.9}$$

These matrices and state are put into the functions stated in section 2.2 to get the final equations for the filtered position.

3.5.2 Transformations

In addition to the transformations in section 3.3.1, the filter transforms the marker position to the drone position, before entering it into the Kalman filter. When using an Extended Kalman

filter, it is advised to put these transformations into the h function, which transforms the measurement to the Kalman state. This way, the errors in the rotation are incorporated as well, which is not the case with this implementation.

The marker transformation is a fixed rotation, with a translation of 24 cm in the negative y direction with relation to the UAV itself. Mathematically defined as:

$$H_{MA}^{A} = \begin{bmatrix} R_{MA}^{A} & P_{MA}^{A} \\ 0 & 1 \end{bmatrix} \tag{3.10}$$

$$R_{MA}^{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad P_{MA}^{A} = \begin{bmatrix} 0 \\ -0.24 \\ 0 \end{bmatrix} \tag{3.11}$$

Since we measure the marker and want to know the drone coordinates, the inverse of this transformation is used, which is defined as:

$$H_{A}^{MA} = \begin{bmatrix} (R_{MA}^{A})^{T} & -(R_{MA}^{A})^{T} \cdot P_{MA}^{A} \\ 0 & 1 \end{bmatrix} \tag{3.12}$$

The IMU is assumed to be in the same frame as the UAV itself. The ROS driver for the ardrone includes a rotation before publishing its IMU data, which ensures the IMU is in the same frame as the drone. The exact position relative to the center of the drone cannot be determined due to the closed internals of the parrot and is therefore assumed to be in the centre of the drone frame. Therefore, the only transformation for it to convert to global frame is the inverse of drone A.

3.5.3 Synchronisation

Another problem that the Fusion Filter takes care of is the synchronisation of the measurements. Image processing can take relatively long. During quick movements, the IMU will give a fast estimate, while the vision position estimate is delayed. Dependent on the computer used and the method of detection, this can take up to 300ms[5]. During final test, the delay was about 100-200ms depending on the situation and background. This delayed vision measurement can counteract the position prediction made by the IMU. It is better to use the vision measurement at the moment the image was taken. [Pornsarayouth and Wongsaisuwan (2009)] shows two different methods of synchronisation of vision in a Kalman filter. The classical method of compensating for vision delay is used and has been implemented in the following way (visualised in figure 3.6).

- When the filter recieves an IMU callback. A prediction is applied to the state of the Kalman filter using this information (an I in the figure). At the same time, the measurement is stored into a buffer, together with the state at that moment. At the current moment, the best guess of the estimate is using a long run of IMU information.
- When the filter receives a vision position estimate, it will look in the buffer for the last state before the timestamp of this estimate. It will delete everything from the buffer before this timestamp and set the state (backtrack) to before this last data point. (b -> c in figure 3.6)
- Then, the measurement from the vision is applied to this backtracked state. (d)
- The IMU data that is still in the buffer, which is thus the data between the time of the vision and now, is applied (instantaneously) to the new state which includes the vision

[5]This is in some cases seen during the development of this implementation

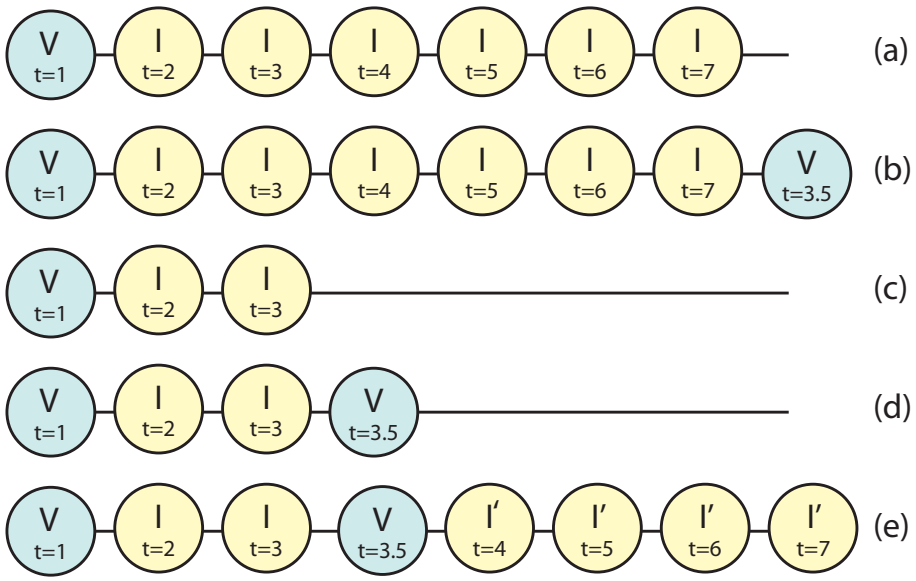


Figure 3.6: Synchronisation process. The timeline is the history of vision and IMU measurements coming in, including a timestamp. Current time, t=7. a) Current state; b) A new vision measurement comes in at t=7 with timestamp t=3.5; c) Backtrack to latest state before the new vision timestamp; d) Apply the vision measurement; e) Apply the IMU measurements to the new state.

measurement. It will re-perform the Kalman Filter calculations for each measurements. After this, we have a new estimate (e), using the vision from less long ago. (a) is the best estimate before the measurement, while (e) is the best estimate including the vision measurement. So those two estimates are the only ones being published.

Using this, the camera measurement is applied at the correct time, which does not counteract the IMU information. Downside of this implementation is that the position estimate relies more on the IMU, since there is always about 200ms of data from the IMU incorporated in the position that is published. The longer the delay of the image is, the higher the variance of the pose estimation will be, since more prediction steps are performed based on the IMU information.

3.5.4 Bias compensation

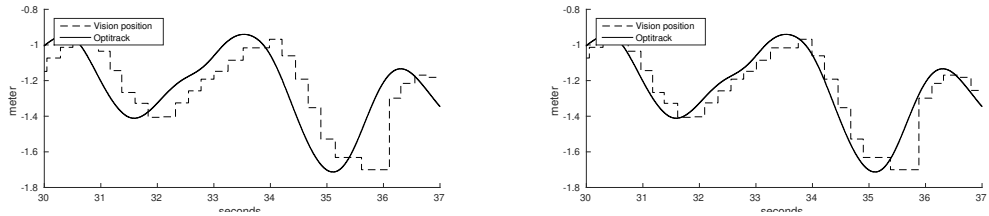
As mentioned in section 2.1.1, the IMU measurement contains a bias. This is estimated by comparing the state velocity to the velocity estimated using vision. Based on this difference, the bias can be calulated. Since both measurements are not perfect and the bias does not change really quickly this difference is applied to the estimated bias proportionally using a Δt and a α factor. This α factor can be configured dynamically and is usually set around 0.2. If the IMU bias has much less influence, it is advised to lower this value.

4 Results

First, the accuracy of the pose estimation will be discussed. After that, this pose estimation is fed into a position controller. Both these experiments are performed while keeping the drone with the camera fixed, and while flying with both UAVs. This is done to elimate the movement of the camera and show the performance difference between these two.

4.1 Vision synchronisation

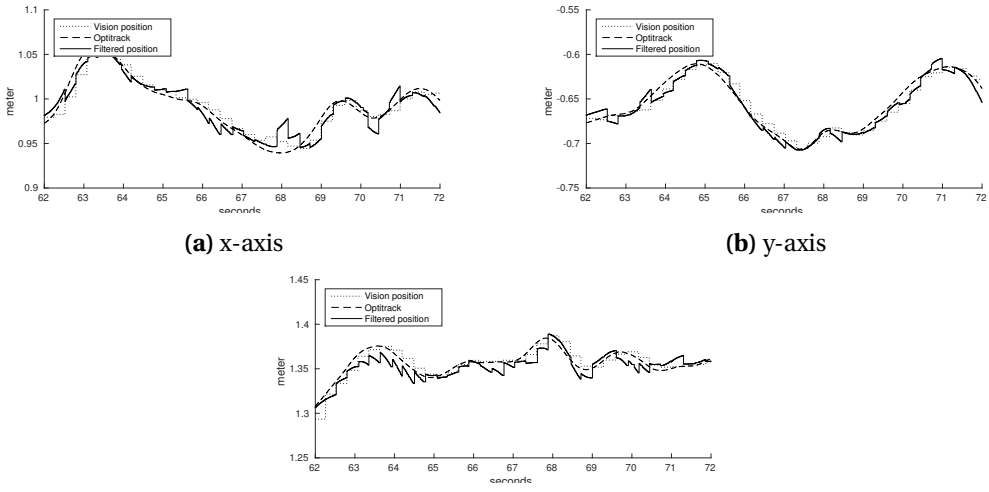
In section 3.5.3 the synchronisation mechanism was explained. Figure 4.1 shows the practical result of that. Without synchronisation, the position as shown in (a) would be applied. This is the best estimate that would be possible without the aid of the IMU. In the filter, the timestamped postition is used, as is plotted at (b). It can be seen that (b) resolves the delay that is present at (a) almost completely



(a) Published time **(b)** Used time (timestamp)

Figure 4.1: Vision measurements compared to ground truth. (a) shows the time of publishing of the measurement, while (b) shows the timestamps that are used for applying the measurement.

4.2 Pose estimation



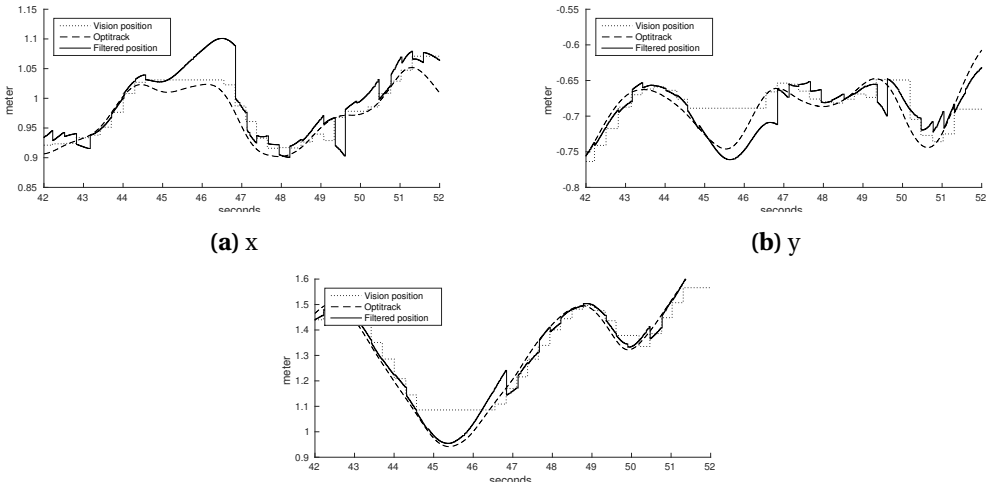
(a) x-axis **(b)** y-axis

(c) z-axis

Figure 4.2: Position estimate using a fixed camera, while having good vision.

In this first setup, the UAV with the camera stays in a fixed position, eliminating the influence of movement from the camera. Figure 4.2 shows both the filtered position estimation and the estimation using vision only in this setup. The vision estimate in this plot is compensated for the processing time, so plotted at the timestamp of the image and not the time of receiving the image, as explained in section 3.5.3 about synchronisation.

The IMU should be able to keep a reliable pose estimation in between the vision measurements. With this in mind, it is useful to look into moments when vision is not available for a couple of seconds. Figure 4.3 shows the response when the vision estimate is lost for a couple of seconds. Due to a large movement on the z-axis, the UAV moved out of view of the camera. It shows that the IMU does allow to have a good pose estimation on the y and z-axis. On the x-axis, it can be seen that the estimation floats off about 10cm in 3 seconds. This is probably due to the bias estimation in this direction. During other tests, it showed that it was vital to keep estimating the bias. Unfortunately, this bias estimation was not always correct and it was often the case that the estimation would float off when vision was lost.



(a) x **(b)** y

(c) z

Figure 4.3: Position estimate using a fixed camera, while losing vision sometime. Vision is lost from t=45-47 (straight green line from vision)

Knowing that the pose estimation with a fixed camera does work well, second experiments are performed during a simultaneous flight. In figure 4.4 the position estimate during a simultaneous flight is shown. It can be seen that the z-position is more unreliable compared to the results with a fixed camera. Compared to figure 4.2 the results in x and y are comparable, which shows that the position can still be accurately calculated when the camara moves.

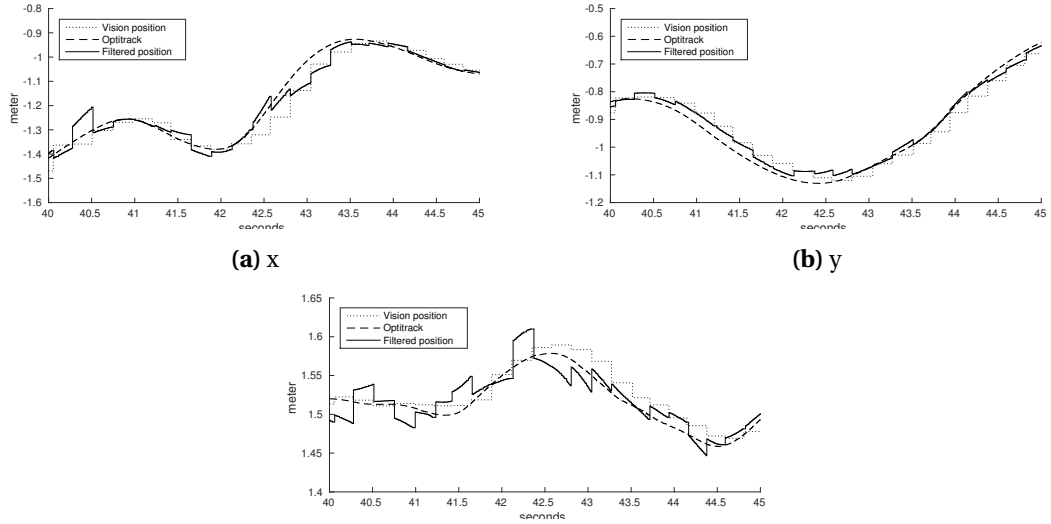
In almost all plots, there are peaks present that seem to come from the update step with the vision measurement, while the vision measurement is fine. Actually, this is a bug in the calculation of the marker to the drone frame in the code. This will be discussed in greater detail during the discussion.

4.3 Performance of control

Using this pose estimate, experiments are perfomed to validate the performance of control using this filtered estimate. During the next experiments, the position estimate is fed into the controller. This is done in three different situations

- Using a fixed camera, giving setpoints in the figure of a square, with 4 seconds at each position.
- When both drones are flying, giving a fixed setpoint
- When both drones are flying, giving the same square as with the fixed camera.

When both UAVs were flying, it was difficult to perform the experiment with the square figure pattern. Movement of both UAVs made it difficult to keep the marker UAV into the field of view



(a) x

(b) y

(c) z

Figure 4.4: Position estimate using two flying drones. Both drones position controlled at a fixed position.

and control is more difficult as well. In order to validate the performance of control better, the response for a fixed setpoint has been given as well. All control performances are compared to the performance of control when using the OptiTrack as source of pose for the controller. This is done by taking off using the OptiTrack as input and fly a certain amount of time. After that, the pose input is changed in-flight and the same flight patterns as before that are performed.

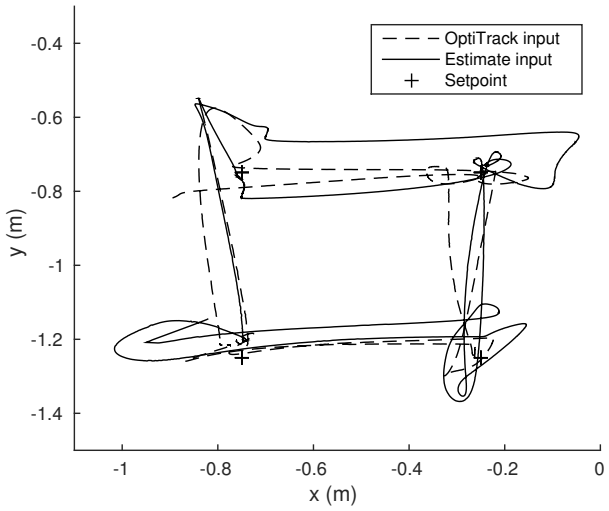
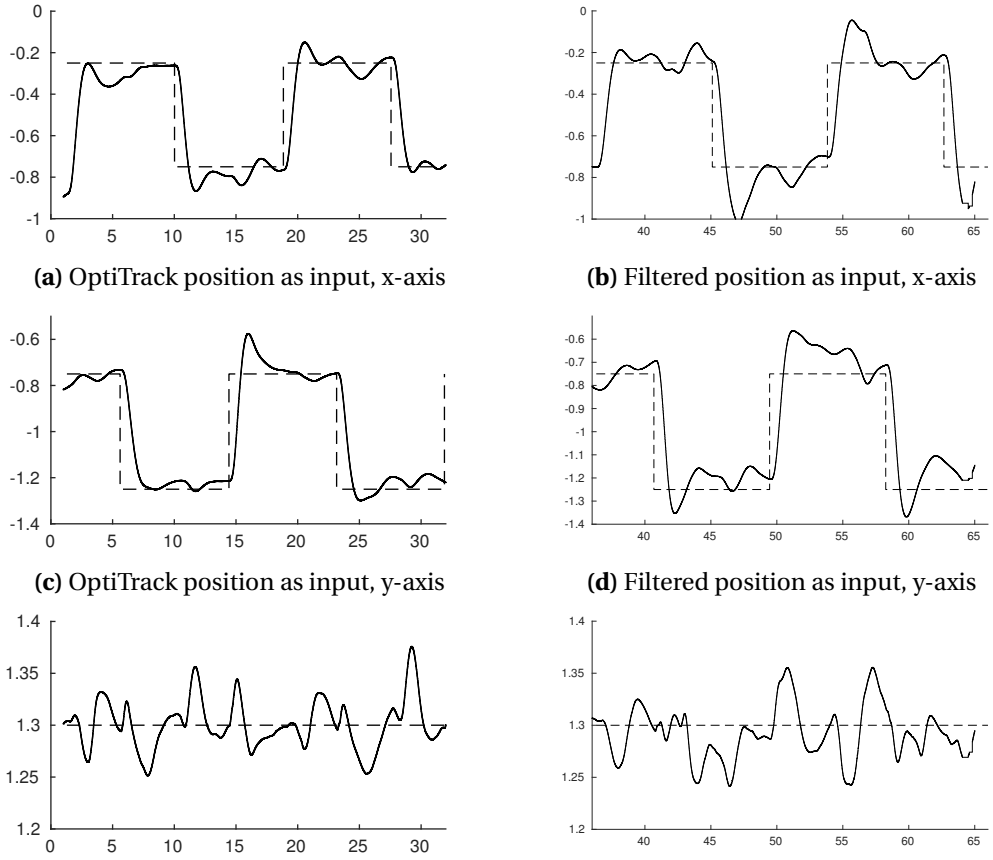


Figure 4.5: The position response from the UAV using the position controller when the camera UAV is in a fixed position. Compares the response between using the OptiTrack pose as input and the filtered pose estimate as input.

Figure 4.5 and 4.6 show the position response of the UAV when given setpoints in a square figure. Comparing the response when the OptiTrack input is used with the filtered position used as input, it can be seen that the response when using the OptiTrack is better. The response when using the filtered position is not much worse. The overshoot is a bit higher, but after that it corrects well and the controller keeps the UAV at the right position. In the z-axis the difference seems to be non-existent. The difference in control can be seen in these plots, but looking at it visually, it was really hard to see the difference.



(a) OptiTrack position as input, x-axis

(b) Filtered position as input, x-axis

(c) OptiTrack position as input, y-axis

(d) Filtered position as input, y-axis

(e) OptiTrack position as input, z-axis

(f) Filtered position as input, z-axis

Figure 4.6: Axis plot of position (given by the OptiTrack) against setpoint. The camera drone is in a fixed position and a square control input is given. The plots on the left are the response of the controller when using the OpiTrack pose as input, while the righthand side uses the filtered position as input.

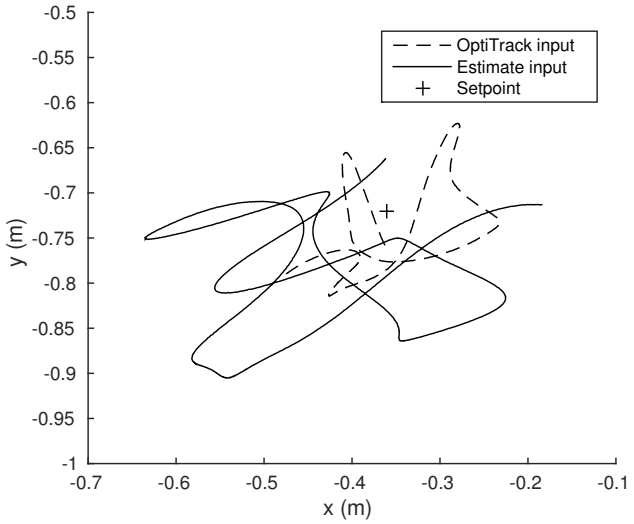
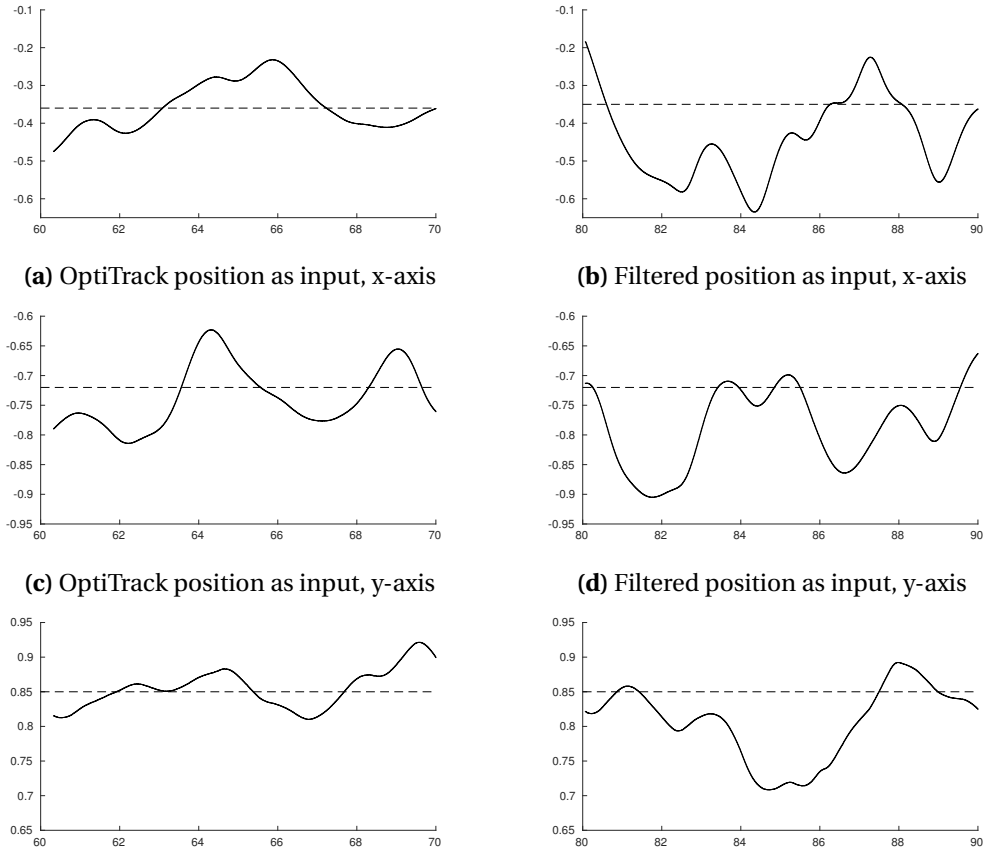


Figure 4.7: The position response from the UAV using the position controller during a synchronous flight using a fixed setpoint.

The response of control when using a fixed setpoint and two flying UAVs is shown in figures 4.7 and 4.8 . What can be seen is that in this situation the control using the filtered position is noticably worse, compared to the OptiTrack as input.



(a) OptiTrack position as input, x-axis

(b) Filtered position as input, x-axis

(c) OptiTrack position as input, y-axis

(d) Filtered position as input, y-axis

(e) OptiTrack position as input, z-axis

(f) Filtered position as input, z-axis

Figure 4.8: Axis plot of position (given by the OptiTrack) against setpoint. Both drones are flying and input is a single setpoint. The plots on the left are the response of the controller when using the OpiTrack pose as input, while the righthand side uses the filtered position as input.

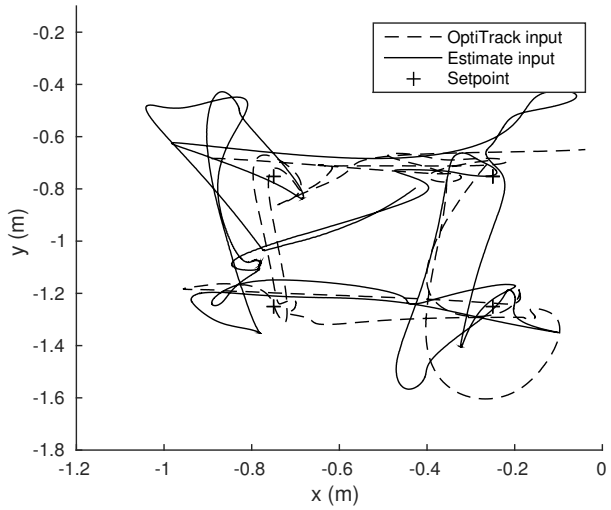
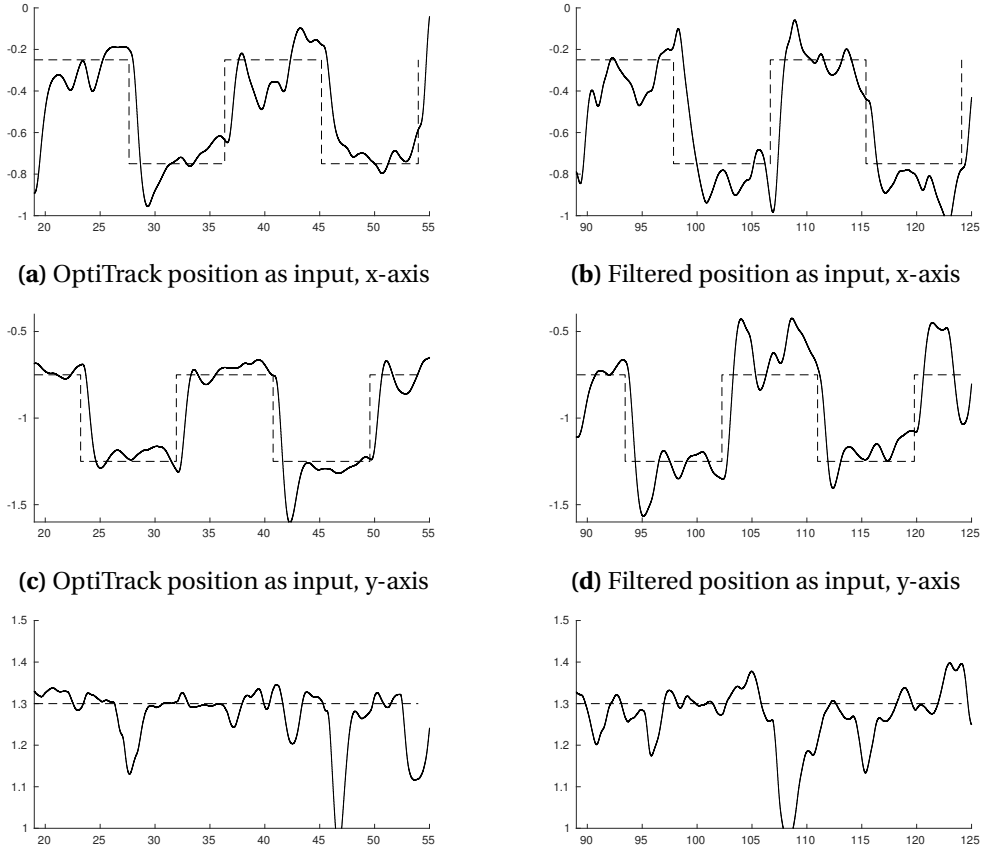


Figure 4.9: The position response from the UAV using the position controller during a synchronous flight and a square figure setpoint pattern. Compares the response between using the OptiTrack pose as input and the filtered pose estimate as input.

Figure 4.9 and 4.10 show the performance of the controller when using a square input and two flying UAVs. A known issue of the controller is that it behaves worse when there are two drones



(a) OptiTrack position as input, x-axis

(b) Filtered position as input, x-axis

(c) OptiTrack position as input, y-axis

(d) Filtered position as input, y-axis

(e) OptiTrack position as input, z-axis

(f) Filtered position as input, z-axis

Figure 4.10: Axis plot of position (given by the OptiTrack) against setpoint. Both drones are flying and input is a square figure. The plots on the left are the response of the controller when using the OpiTrack pose as input, while the righthand side uses the filtered position as input.

flying, due to influence of each other. Therefore, it is expected that the controller performs worse during this setup, in comparison with a single flying UAV. This experiment confirms that by showing that the controller performance using the OptiTrack is worse with two drones flying, instead of one. The second issue with the controller during a simultaneous flight is that an UAV can sometime perform an unexpected movement in the z direction. This happens more often when both UAVS are flying close to each other. In the z-axis plot of figure 4.10 drops of about 30cm can be seen during both control experiments.

Taking that into consideration, the performance difference between the OptiTrack as input and the filtered estimate as input needs to be investigated. What can be seen from these plots is that the controller again performs worse when it uses the filtered position. The deviations become much larger now, but still the UAV is controlled within a reasonable error. The controller needs to correct more, which leads to bigger deviations of the pose compared to the setpoint.

5 Discussion

While these results are good enough for control, it is far from perfect. First of all, the Kalman filter could be improved to include more variables. Some calculations in this implemenation are not optimal, which could improve the state estimation as well. Since wrong estimations can propagate for a couple of seconds, improving this will vastly improve the overal result. At the same time the controller is not optimal, which makes it diffcult to see exact performances using the filtered state for control.

In the first place, when looking at the results for the position estimation. Some peaks can be seen after an update step of the Kalman filter. This does get corrected after the next measurement. However, the vision measurement itself is fine since the marker position calculation is working nicely. After these results were made, an error in the code was found. The transformation of the marker to the drone frame in the fusion filter uses the orientation from the OptiTrack. Unfortunately, no pose buffer was implemented, which causes this calculation to use a pose of 300ms later than should be actually the case. This can cause deviations of a couple of centimeters. Since these rotations are not big, overall it does not have a very big impact. It would be useful to implement a pose buffer in this implementation to overcome this problem.

Although this may be true, the best would be to take the orientation into the state. The orientation from the OptiTrack is now passed through in the filtered estimate, but ideally the OptiTrack should not be used for this at all. Due to time considerations, implementing this was not possible anymore. Taking the orientation into the state immediately solves the problem of buffering the pose for the calculation. This orientation for the Kalman filter can be implemented by using the pitch and roll rotations from the IMU and filtering this with the yaw, pitch and roll from the marker. The yaw from the IMU in this implementation with the AR.Drone cannot be used as it tries to fuse that with the magnetometer information internally. However, the magnetometer is unreliable indoors, providing wrong yaw information. This problem can be solved by using another platform for doing these experiments.

The orientation is not the only state variable that can be added to the Kalman Filter. The current implementation uses a custom function to adapt the IMU bias. This is simply done by looking at the difference between the vision measurement and the state. This bias can also be included into the Kalman state. This makes the calculation dependend on the various covariances in the system. Using that, no custom α value needed anymore and the bias will be estimated in a more optimal way.

Finally, the controller that has been used is not perfect. Especially during a simultaneous flight of two drones, sometimes strange manouvres are performed. This can include flying up and then down again, or a quick movement in the x-y plane. This effects gets worse when the UAVs are flying close together. For this reason, a larger distance between the UAVs (about 2-3m) is necessary but this also decreases the performance of the marker detection.

6 Conclusion

This report shows the results of collaborative pose estimation using two UAVs. All things considered, UAV control using the collaboration is possible. However, performance of control differs in different situations and it is vital to not lose the vision estimate for too long.

The position estimate using the combination of vision and IMU is quite reliable. There was not a big difference in error of the position estimation using a fixed camera or when both drones are flying. The movement from the second UAV does not significantly influence the error in the pose estimation. Using the vision information with a timestamp, it is possible to eliminate the delay of image processing in the final estimate, while still keeping a reliable pose estimation.

This pose estimation has been proven to be good enough for position control of the UAV. Compared to using the ground truth as source for the contoller, it performed a worse, but the quality of control was good. Square pattern experiments have been performed. During the flight of a single UAV, the square pattern control is good. In the plots, it can be seen that control performance is worse when the filtered position is used as input. Then again, to the eye the UAV control seems to be the same.

During a simultaneous flight, issues concerning the controller can be seen. The UAVs have influence on each other, degrading the performance of control. Deviations between control get bigger using this as input. When both UAVs are moving during such a flight, it is difficult to keep the marker in view of the camera. When the marker can not be detected, the position estimate might drift off due to IMU bias. When using movement commands, it is vital to keep the UAV in view or it will usually not come back and drift off.

6.1 Recommendations

As discussed earlier, the pose estimate can probably be improved more by improving the implementation. Improving this will improve the performance of control as well. Besides that, the implementation will probably work better when better hardware is used. Using a higher resolution for the camera will increase the accuracy of the vision estimate, especially at larger distances. Using a better IMU might decrease the effects of the bias and give a better orientation estimate.

For further research, the robustness of this method can be investigated. The assumption here is made that the marker is always in straight view of the camera. In real-life applications this is usually not the case, since the UAVs can manouvre a lot. Therefore, performances can be investigated while using multiple markers on the UAV and using multiple UAVs to detect these. This will give more vision measurements and should increase the accuracy of the pose estimation. Another option would be to totally remove the marker and use another method of detection of the UAV.

Another point of research would be to investigate the performance when the exact position of the camera UAV is not known. When its pose is an estimate, errors of that estimate should propegate through the vision measurement. These extra errors will decrease the accuracy of the pose estimation. Inherent to this will be to perform experiments in outdoor situations where the OptiTrack information is not available.

The goal of this project is to provide a reliable pose estimation using collaboration that can be used when an UAV can not use its vision. Further research can focus on the performance of this implementation in such kind of situations. For instance, interaction with a wall could be performed using this implementation and using regular SLAM on the UAV to evaluate performance differences.

Bibliography

Christian Forster, Simon Lynen, L. K. D. S. (2013), Collaborative Monocular SLAM with Multiple Micro Aerial Vehicles, *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*.

Elias Mueggler, Matthias Faessler, F. F. and D. Scaramuzza (2014), Aerial-guided Navigation of a Ground Robot among Movable Obstacles, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Jeroen D. Hol, Thomas B. Schon, H. L. P. J. S. F. G. (2007), Robust real-time tracking by fusing measurements from inertial and vision sensors.

Luis Rodolfo Garcia Carrillo, Alejandro Enrique Dzul Lopez, R. L. C. P. (2011), Combining Stereo Vision and Inertial Navigation System for a Quad-Rotor UAV, *Journal of Intelligent & Robotic Systems*.

Markus Achtelik, Tianguang Zhang, K. K. and M. Buss (2009), Visual Tracking and Control of a Quadcopter Using a Stereo Camera System and Inertial Sensors, *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*.

Matias Tailanian, Santiago Paternain, R. R. R. C. (2014), Design and implementation of sensor data fusion for an autonomous quadrotor.

Olson, E. (2011), AprilTag: A robust and flexible visual fiducial system, *Robotics and Automation (ICRA), 2011 IEEE International Conference on*.

Pierr-Jean Bristeau, Francois Callou, D. V. N. P. (2011), The Navigation and Control technology inside the AR.Drone micro UAV.

Pornsarayouth, S. and M. Wongsaisuwan (2009), Sensor Fusion of Delay and Non-delay Signal using Kalman Filter with Moving Covariance, *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*.