Docking a UAV using a Robotic Arm and Computer Vision

J.H. (Jort) Baarsma

MSc Report

March 2015

008RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The aim of this Thesis is to research the possibility of docking an airborne UAV using a robotic arm and computer vision. In the scope of the SHERPA project on robot collaboration in Alpine rescue scenario‘s a robust and autonomous way of retrieving small scale UAV‘s is required. In a scenario where landing the UAV on a mobile ground station is not possible and landing on the ground could be harmful to the drone, docking the UAV while it is airborne using a robotic arm would be the best solution.

In this work a system is presented to accurately follow and dock an UAV near a ground vehicle using an external monocular camera mounted on a robotic arm and internal UAV sensors. The system combines the strengths of visual pose estimation and IMU motion tracking while minimizing the effect of their weaknesses. The result is a system with an accurate pose estimate and which is also robust through temporary occlusion or motion blur. A functional proof of principle system was implemented using Robotic Operating System and a KUKA LWR4+ robot arm to show the advantages provided by the system.

Contents

1 Introduction

1.1 Context

1.1.1 Alpine search and rescue

Introducing robotic platforms in a rescue system is envisioned as a promising solution for saving human lives after an avalanche accident in alpine environments. With the popularity of winter tourism, the winter recreation activities has been increased rapidly. As a consequence, the number of avalanche accidents is significantly raised. According to the statistics provided by the Club Alpino Italiano, in 2010 about 6,000 persons were rescued in alpine accidents in Italy with more than 450 fatalities and about thirty thousand rescuers involved, and with a worrying increasing trend of those numbers. In 2010 the Swiss Air Rescue alone conducted more than ten thousand missions by helicopters in Switzerland with more than 2,200 people that were recovered in the mountains [1].

To this aim, a European project named "Smart collaboration between Humans and ground-aErial Robots for imProving rescuing activities in Alpine environments (SHERPA)" has been launched.

SHERPA

The activities of SHERPA are focused on a combined aerial and ground robotic platform suitable to support human operators in accomplishing surveillance and rescuing tasks in unfriendly and often hostile environments, like the alpine rescuing scenario specifically targeted by the project.

What makes the project potentially very rich from a scientific viewpoint is the heterogeneity and the capabilities to be owned by the different actors of the SHERPA system: the "human" rescuer is the "busy genius", working in team with the ground vehicle, as the "intelligent donkey", and with the aerial platforms, i.e. the "trained wasps" and "patrolling hawks". Indeed, the research activity focuses on how the "busy genius" and the "SHERPA animals" interact and collaborate with each other, with their own features and capabilities, toward the achievement of a common goal.

Trained wasps The "Trained wasps" are small rotary-wing unmanned aerial vehicles (UAVs), equipped with small cameras and other sensors/receivers and used to support the rescuing and surveillance activity by enlarging the patrolled area with respect to the area potentially "covered" by the single rescuer, both in terms of visual information and monitoring of emergency signals. Such vehicles are technically designed to operate with a high degree of autonomy and to be supervised by the human in a natural and simple way, like they were "flying eyes" of the rescuer, helping him to comb the neighbouring area. UAVs are specifically designed to be safe and operable in the vicinity of human beings. As a consequence they have a limited operative radius.

Intelligent donkey The "intelligent donkey" is a ground rover serving as a transportation module for the "SHERPA box" which serves as a hardware station with computational and communications capabilities, a recharing station for small-scale UAV's and transport for rescuer equipment. It is technically conceived to operate with a high-degree of autonomy and long endurance, as well as to have a payload calibrated to carry relevant Hardware. It is wirelessly connected to the rescuer, able to follow his movements, and to interact in a natural way. In order to improve the autonomous capabilities of the robotic platform, a multi-functional robotic arm is also installed on the rover, which will be useful especially in relation to the deployment of the small scale UAVs (both in terms of take-off and landing). The key elements carried by the rover, constituted by the computational and communication Hardware, by the recharge station of the small scale UAVs and the equipment storage element, are mechanically conceived to be confined in the so called “SHERPA box”.

Patrolling hawks The “patrolling hawks” are Long endurance, high-altitude and high-payload aerial vehicles, with complementary features with respect to the small-scale UAVs introduced before, complete the SHERPA team. Within the team, they are used for constructing a 3D map of the rescuing area, as communication hub between the platforms in presence of critical terrain morphologies, for patrolling large areas not necessarily confined in the neighborhood of the rescuer, and, if needed, to carry the “SHERPA box” in places non accessible to the rover. They fly at a height of around 50-100 m above ground (or trees).

More information on the SHERPA project can be found at [2].

1.2 Problem statement

In a system where robots collaborate to assist rescue operations small scales multi-rotor UAV‘s need to be able to autonomously return to a ground rover where they can be recharged and stores. A multifunctional robotic arm on the ground rover can be used to dock the UAV and return it to the recharge station inside of the “SHERPA box”. The Conceived scenario‘s for docking the UAV were:

- The multirotor lands on a helipad on top of the rover and is then docked by the robotic arm.
- The multirotor lands or falls on the ground near the rover and is then docked by the robotic arm.
- The multirotor is docked while flying in close proximity to the rover.

The first scenario would be the ideal scenario whenever possible but this is not possible in all weather conditions or when the rover is angled at a slope. In the second scenario the type of surface which is chosen is important, if a flat surface without any foliage or rocks can be found this is a good alternative. However this scenario is also not robust when the surrounding area is rocky or covered with foliage where the drone could be damaged or lost when attempting to land. The last scenario would be the most challenging to realize but would best for the longevity of the drone when executed reliably, and will be the focus of this research.

In this scenario the drone can theoretically be docked in any situation where the drone can approach the rover. The key piece of information in this problem is relative position and movement of the drone with respect to the rover. The GPS signals of the rover and UAV are only up to a few meters and can only get the vehicles in close proximity of each other. From this distance the drone can be found using a camera mounted on the robotic arm and followed by estimating the pose of the drone. The sensors inside the drone can assist the estimate when the pose cannot be estimated. Following the drone is a virtual servo-ing “Camera in hand” problem and docking the drone is an extension to that problem by decreasing the follow distance until docking can be performed.

To summarize the problem statement in one sentence: Is it possible to follow and dock a multirotor UAV using a robotic arm and computer vision?

1.3 Prior work

Prior work on visual tracking of a quadcopter using an external stereo camera and inertial sensors but without the use of a robotic arm was performed by the University of Munich [3]. And Prior work on robot collaboration by having a “Camera in hand” system look at an UAV to find the relative pose has been done by the Robotics and Perception group of the University of Zurich [4]. However no prior work was found on the task of docking a multi-rotor type UAV using a robotic arm and computer vision.

2 Analysis & Design

In this chapter the system to follow and dock the drone is explained.

2.1 System introduction

The system used to follow and dock the UAV is using visual pose estimation system in combination with IMU measurements from the UAV.

To be able to dock the UAV using a robotic arm the pose estimate of the UAV needs to be precise. The best way to achieve this is to measure the pose itself by means of some sort of pose estimation. In this system the method of pose estimation is done by means of computer vision. To improve the pose estimation additional sensors can collaborate to support the weak features of the vision sensor. In this system it was chosen to use the IMU data, of the target UAV that is to be docked, as an additional sensor and fuse the measurements with a state estimator.

2.1.1 Subsystem allocation

For clarity the system has been divided into subsystem which will be discussed in detail in their respective sections. The subsystems that are envisioned are:

- Robotic Arm subsystem
- Setpoint generator subsystem
- State estimator subsystem
- Computer vision subsystem
- UAV subsystem

The connection between the system is shown in figure 2.1. The computer visions pose estimation $\boldsymbol{H}_m^c$ is combined with the acceleration $\vec{a}_i^i$, orientation$\vec{q}_i^w$ and angular velocity$\vec{\omega}_i^w$ of the UAV into the state estimator which produces an optimal state estimate of the UAV position $\overset{*}{\vec{p}}{}_i^g$ based on the known information. From this optimal estimate of the drone location a setpoint is derived which followed by the controller of the robotic arm.

2.1.2 Definitions and notation

This section define some variables and notations which hold throughout all the subsystem of the system.

In this report Vectors $\vec{v}$ will be indicated be by small letters with a harpoon and matrices $\boldsymbol{A}$ will be a capitol boldface letters. The notation for a estimate is a tilde above the symbol $\tilde{a}$ and an optimal estimate is denoted with a asterisk above the symbol $\overset{*}{a}$.

Relative poses are described by homogeneous matrices $\boldsymbol{H}_a^b \in \mathbb{R}^{4\times4}$ where the a defines the destination frame and b describes the reference frame. A alphabetic characters in superscript notation for a variable for will indicate the reference frame in which that variable is viewed. A homogeneous matrix is composed of a rotation matrix $\boldsymbol{R}_a^b$ and translation vector $\vec{t}_a^b$ and is a powerful method of describing kinematic chains.

$$\boldsymbol{H}_a^b = \begin{bmatrix} \boldsymbol{R}_a^b & \vec{t}_a^b \\ \vec{0}^T & 1 \end{bmatrix}$$

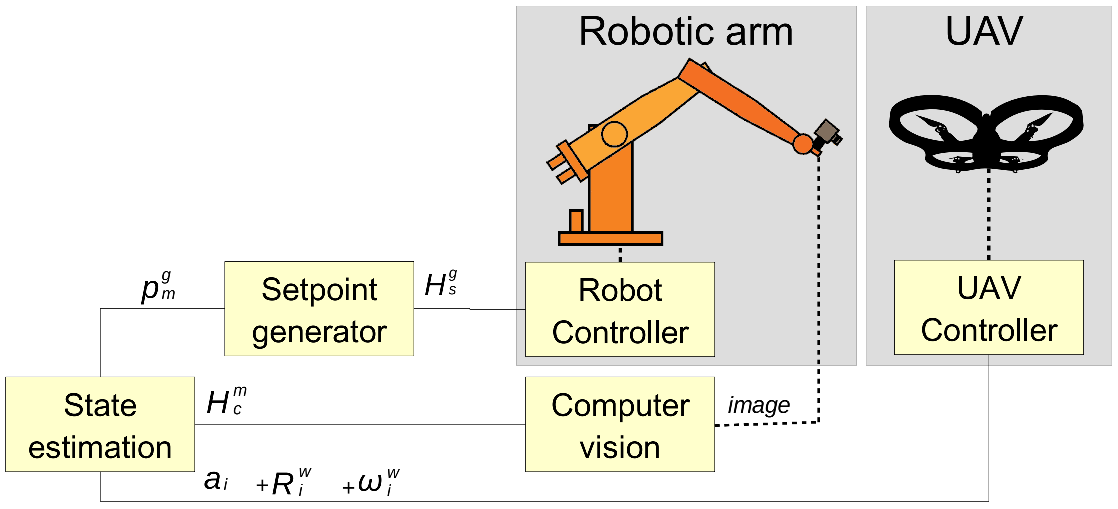


Figure 2.1: Schematic representation of the setup.

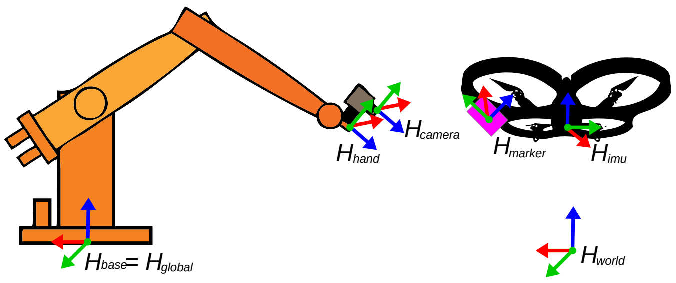


Figure 2.2: Frames in the method

The system and its subsystems have several frames which are referred to throughout the report by only their letter, these are listed in table 2.1 and illustrated in figure 2.2. The global reference frame is not alligned with the world coordinate frame due to the unknown orientation with respect to magnetic north.

Letter	Frame description
g	The base of the robotic arm
b	The base of the robotic arm
i	The IMU of the UAV.
w	Defined by magnetic north and gravitational vector
h	End-effector of the robotic arm
c	Camera frame mounted on end-effector.
m	Marker frame that is mounted on the UAV
s	Setpoint frame

Table 2.1: Table of frames

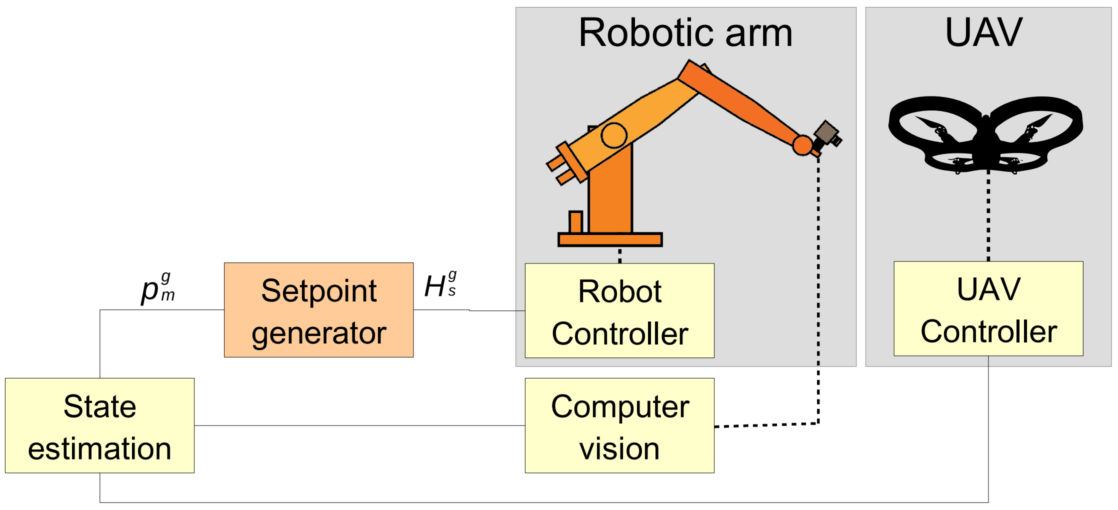


Figure 2.3: Control schema - Set-point generator.

2.2 Setpoint generator-subsystem

The set-point generator sub-system is responsible of generating set-points for end-effector of the robotic arm to follow. The generated setpoint is based on the optimal estimate of the UAV position and a strategy the generator is set to. These strategies are determined beforehand based on likely scenarios and what strategy is used is determined by the system with input from the user. There is also a safety layer in this subsystem which limits the (angular) velocity of the moving set-point and keeps the robot inside of its work envelop.

The subsystem is located in the system as shown in figure 2.3.

The input of the system is:

- The optimal estimate of the position of the marker with respect to the robotic arm base $\overset{*}{\boldsymbol{H}}{}_m^g$.

The output of the system is:

- The a desired pose set-point for the robotic arm controller $\boldsymbol{H}_h^g$.

2.2.1 Strategies

The strategies that are used are based on the distance of the drone to the work envelop of the robot and the desired behavior specified by the user. The selection of strategy and transition between strategies is done by the system itself but the user has control over what strategies are allowed. The user can force idle mode or (dis)allow any state to get the desired behavior. The strategies names are influenced by hunting behavior and most strategies focus on following the marker at a distance$\vec{d}$. The set-point generator strategies are:

- **Idle** : This is the resting position in which the system does not do anything. The set-point is an exact copy of the current pose and the strategy only changes when the user requires it.

- **Spot** : In this strategy the set-point is set to a fixed point from where the camera can see the scene. When the marker is found the strategy transitions into the "Watch" strategy.

- **Watch** : If the marker is found but too far away to follow at a fixed distance $\vec{d}_{stakl}$ it is followed with the smallest follow distance which is within the work envelop. If the marker is close enough the strategy transitions into the "Stalk" strategy.

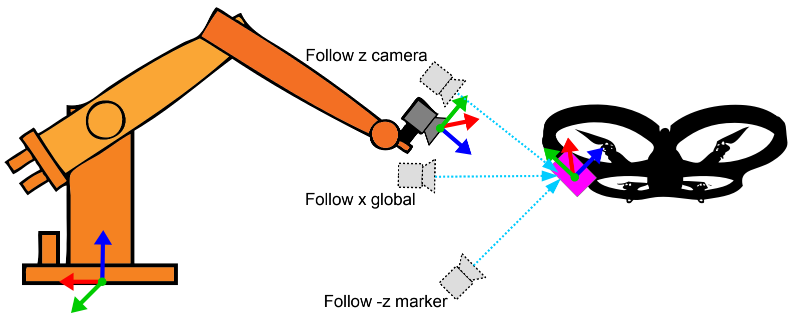


Figure 2.4: Follow distance illustration

- **Stalk** : In this strategy the marker pose is followed at a fixed distance $\vec{d}_{stalk}$. If the set-point goes outside of the work envelop the the strategy transitions back into the “Watch” strategy. If the marker can also be followed at a close distance $\vec{d}_{prey}$ the strategy transitions into the “Prey” strategy.
- **Prey** : In this strategy the marker is followed at a close distance of $\vec{d}_{prey}$. The distance $\vec{d}_{prey}$ is the closest distance to which the vision system can reliably estimate the pose of the marker and is used as a buildup to docking the drone. Keeping the marker in view of the camera when the drone is moving at this distance can be challanging. If the setpoint goes outside of the work envelop the the strategy transitions back into the “Stalk” strategy. If the marker is so close that it can be docked the strategy transitions into the “Catch” strategy.
- **Catch** : In this strategy the last known marker pose is saved and the controller quickly approaches this marker disregarding new measurement. This is because the vision measurement is assumed to be not reliable at distances closes than $20cm$.

When the pose is lost for long enough the strategy transitions back into the “Spot” strategy for any of the strategies in which the pose estimate is used.

2.2.2 Follow distance

The reference frame of the follow distance $\vec{d}$ was deliberately kept ambiguous in reference frame in the discussing on generator strategy. The system can either follow the marker a fixed distance in the frame of the marker $\vec{d}^m$, in the camera frame $\vec{d}^c$ or in the global frame $\vec{d}^g$ as illustrated in figure 2.4. All three methods center the marker on the image plane but have different results as end-effector pose. Following of the drone with respect to the minus Z axis of the marker is the most logical method of following but has the disadvantage of amplifying any rotational errors due to the fact they are multiplied by the follow distance $\vec{d}$. Following with respect to the marker will be implemented for docking because this allows for the best alignment but assuming that the drone is more or less parallel to the ground during the following and docking the method of following in the global frame was preferred for all other scenarios.

2.2.3 Path generator

If the set-point and current robot end-effector are far away from each other the robot cannot follow and dangerous scenarios can occur. For this reason a path planner is used that calculates intermediate set-points between the current pose and the desired pose with a limited velocity for every step. This velocity limiting is achieved by means of a logarithmic map and applying a limit on the resulting twist. The path and twist is updated every time there is a new set-point. The desired pose for the end-effector $\boldsymbol{H}_s^g$ can be written as a combination of the current end-effector pose and the set-point pose in the end-effector frame.

$$\boldsymbol{H}_s^g = \boldsymbol{H}_h^g \boldsymbol{H}_s^h$$

The exponential map is used which describes the resulting pose when a constant twist is applied for t time. The set-point is used as the end-point and the end-effector is subject to a constant twist in body fixed frame.

$$\boldsymbol{H}_s^g = \boldsymbol{H}_h^g(t) = \boldsymbol{H}_h^g(0)e^{\tilde{\boldsymbol{T}}_h^{h,g}t}$$

In this equation the time dependance $\boldsymbol{H}_h^g(t)$ is shown for clarity, all other mentions of $\boldsymbol{H}_h^g$ will be at time instance 0. This reworked to find the twist of the end-effector with respect to the base in its body fixed frame.

$$\tilde{\boldsymbol{T}}_h^{h,g} = \frac{1}{t}\log\left(\boldsymbol{H}_g^h \boldsymbol{H}_s^g\right) = K_p \log\left(\boldsymbol{H}_s^h\right)$$

In which $\tilde{\boldsymbol{T}}$ is a skew symmetric matrix of the Twist vector $\vec{T}$ and the logarithm is a matrix logarithm. What eqation 2.2.3 represents is the required constant twist to get to pose $\boldsymbol{H}_s^g$ in t time. Because the requirement is to get to the setpoint as fast as possible with a limited twist a the factor $\frac{1}{t}$ is replaced by a gain factor K_p and a limiter is put on the Twist.

$$\text{if } ||\vec{T}|| > T_{max} \text{ then}: \; \vec{T} = \left\{ \frac{\vec{T} \cdot T_{max}}{||\vec{T}||} \right\}$$

The limiter used scales down the twist when the norm of the twist vector $\vec{T}$ becomes higher than a set value. Because the twist is only scaled the traveled path will be identical to the path that was not limited.

To calculate the intermediate pose on the path $\boldsymbol{H}_p^g$ the matrix exponential is used.

$$\boldsymbol{H}_p^g = \boldsymbol{H}_h^g e^{\tilde{\boldsymbol{T}}}$$

The robot controller is sent this intermediate pose to protect the robot from large jumps in set-point due to errors.

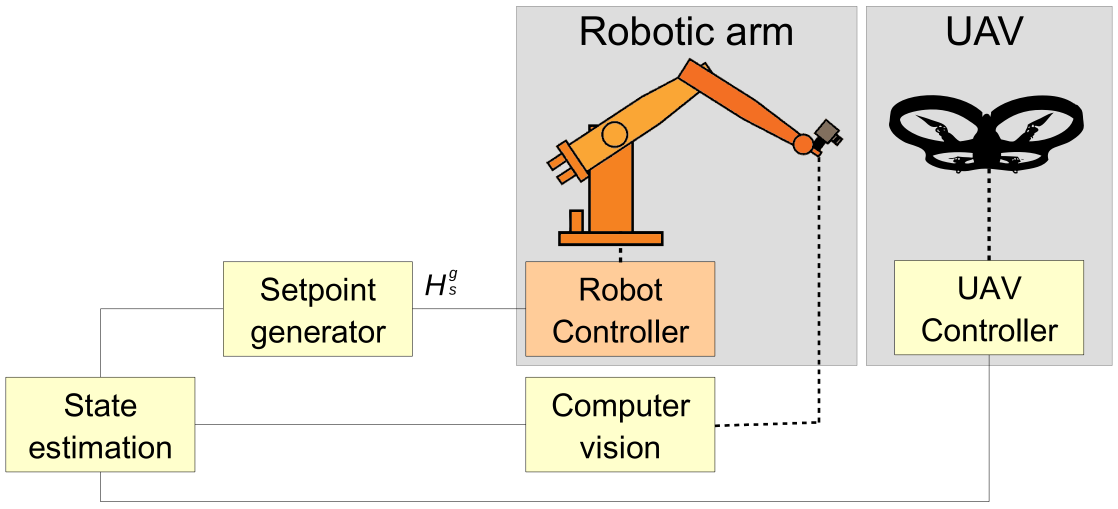


Figure 2.5: Control schema - Robotic arm.

2.3 Robotic arm-subsystem

The Robotic arm-subsystem is responsible for controlling the robot towards the desired pose setpoint. This subsystem is required for the communication to the hardware of the robot arm, the robot itself will have the low-level motor controllers.

The subsystem is located in the system as shown in figure 2.5.

The input of the system is:

- The a desired pose set-point for the robotic arm controller $\boldsymbol{H}_h^g$.

The output of the system is:

- A connection to the robotic arm.

2.3.1 Kinematic chain

The robot arm can be modeled as a serial kinematic chain. A kinematic chain refers to an assembly of rigid bodies connected by joints that is the mathematical model for a mechanical system. As in the familiar use of the word chain, the rigid bodies, or links, are constrained by their connections to other links. An example is the simple open chain formed by links connected in series, like the usual chain, which is the kinematic model for a typical robot manipulator.

As illustrated in figure 2.6 a human arm can also be modeled as a kinematic chain in a similar way a robotic arm can. The Robotic arm needs a minimum of 6 joints to be have the full 6degree's of freedom (DOF) to move in space. Mathematically such a robot could reach every position and orientation near the robot however due to joint limitations and collision a 6DOF can not reach a lot of positions certain orientations especially near its base. With more then 6 joints a robot can reach more positions and orientations which would otherwise be limited by joint limitations and collisions. The same applies for humans which has 7 Degrees of freedom in total to reach almost any position and orientation near the human body with a few exceptions such as between the shoulder blades. The extra degrees of freedom however do give a non-unique solution for one end-effector pose which results in a degree of freedom in the configuration of the arm. This can again be explained using the human body by fixing all 7DOF in the hand and still being able to move your elbow.

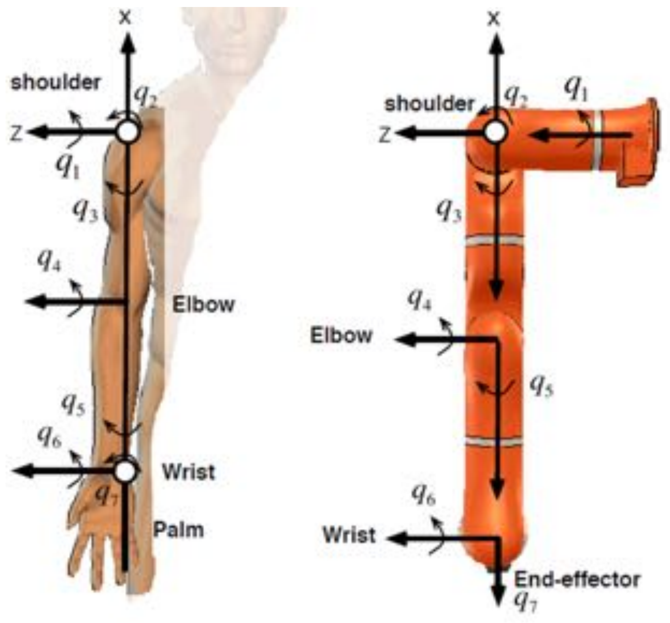


Figure 2.6: Kinematic chain

Behavior

The robot follows a set-point $\boldsymbol{H}_s^g$ which is near the current robot pose $\boldsymbol{H}_h^g$. The robot is assumed as a commercial of the shelf product and there will be no analysis on the behavior of the arm will. In the hardware implementation the specifications of the robotic arm are further discussed

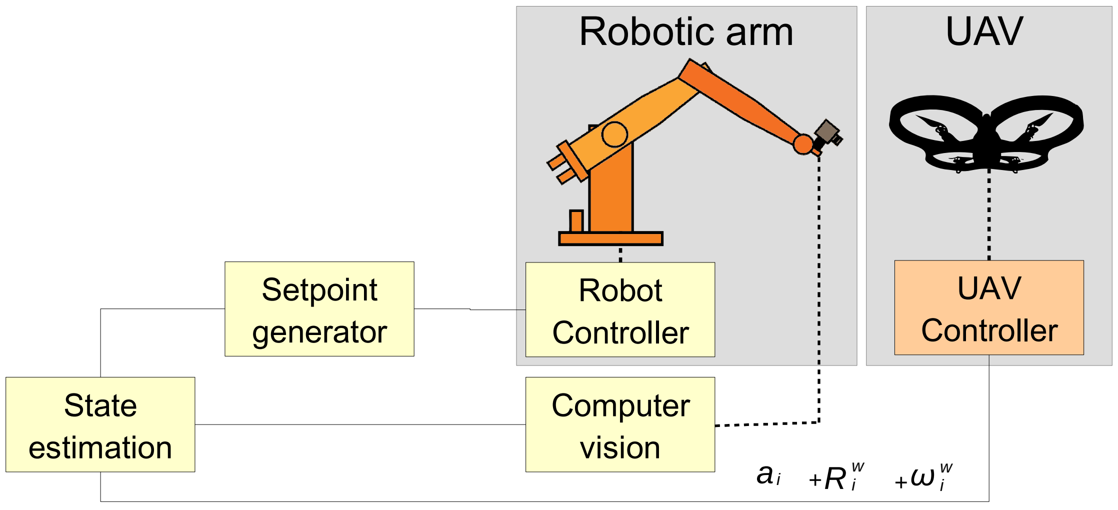


Figure 2.7: Control schema - UAV.

2.4 UAV-subsystem

The UAV subsystem is responsible making the drone fly and sending IMU measurements to the system. The drone is manually controlled by an operator

- Feature extraction
- Pose estimation

The subsystem is located in the system as shown in figure 2.7.

The input of the system is a network connection coming from the drone.

The output of the system is:

- The acceleration measurement from the UAV $\vec{a}_{imu}$, in body fixed coordinates.
- The orientation measurement from the UAV $\boldsymbol{R}_{imu}^{world}$

2.4.1 Quadcopter

A quadcopter drone is a type of unmanned aerial vehicle using four rotors for its actuation. A simplified quadcopter is shown in Figure 2.8.The four rotors can be seen as inputs and are used to move the quadcopter in six degrees of freedom (three translations and three rotations). The degrees of freedom that can be controlled in a quadcopter are the euler angles roll, pitch, yaw and the movement in body fixed z direction. Because the body of the drone can be rotated with respect to the fixed world the movement in body fixed z direction the drone can also translate in global x and y direction.

The control is done using rotor pairs on opposing sides of the quadcopter that spin in opposite direction. The propellers are made much that all rotors generate a lift upwards but the clockwise rotating rotors generate a negative moment $-M_z$ and the counter-clockwise rotors a positive moment M_z. These moments counteract when both pairs of rotors are controlled to generate the equal lift but the Yaw orientation can be controlled by changing the ratio of thrust contribution by each rotor pair. The pitch and roll orientation are controlled by having one rotor in a pair generate more thrust and the other less, this way the resulting thrust is off-center and will result in a moment with which pitch and roll can be controlled. The movement in body fixed z direction is controlled by the total thrust of all rotors. Using these methods four degrees of freedom of the drone can be independently controlled, the other two degrees of freedom

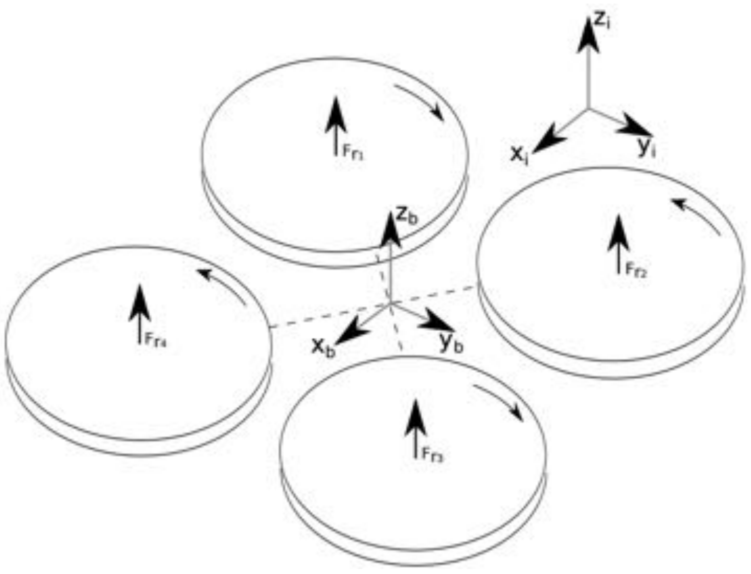


Figure 2.8: Quadcopter basics

(body fixed x and y) have almost no stiffness or damping due to being airborne and are thus susceptible to disturbance.

Because the drone is chosen to be an off-the-shelf product and not being controlled by the system no additional analysis information on the drone is required.

2.4.2 IMU

An Inertial Measurement Unit is a an electronic device that measurement an aircrafts velocity, orientation and accelerations and typically used to maneuver UAVs and other aircrafts. It works using a combinations of a three axis accelerometer, gyroscope and magnetometer and combines these signals to make an estimation for the orientation. Body fixed accelerations are generally modeled as noise to these systems and these systems work best in an inertial frame with constant or zero velocity.

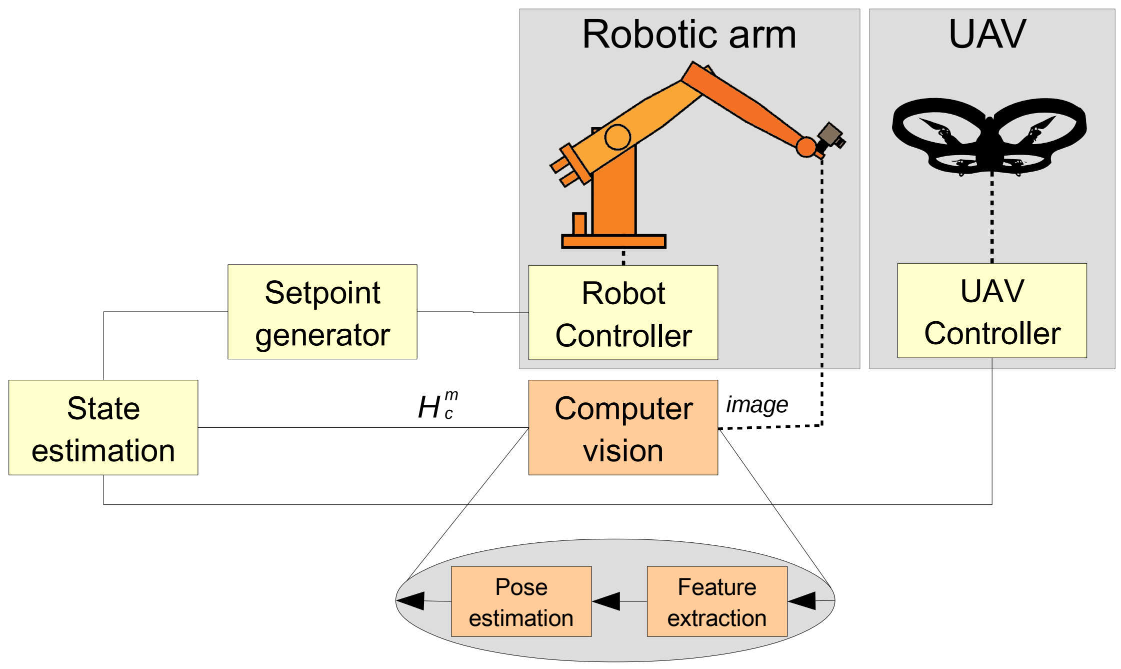


Figure 2.9: Control schema - Computer vision.

2.5 Computer vision-subsystem

The computer vision subsystem is responsible interpreting coming from the end-effector mounted camera. Specific features are extracted from the images based on color and used as points for pose estimation. The computer vision subsystem itself comprises of these two parts: the *feature extraction* and *pose estimation*.

The subsystem is located in the system as shown in figure 2.9.

The input of the system is:

- a digital video stream of image coming from the camera.

The out of the system is:

- A relative pose between the camera and the marker $\boldsymbol{H}_{camera}^{marker}$.

2.5.1 Feature extraction

2.5.2 Marker design

The marker has been designed with two clear goals in mind: A clear vibrant color that is distinguishable in most scenes and non-planar constellation of image points to avoid pose ambiguity. The first requirement is met by using Magenta, a color that is not very common but vibrant when printed on a sticker due to the use of CMYK ink in printers. The second is accomplished by folding the sticker over a bend plate that has an inside angle of 155 degrees. The marker can be seen in figure 2.12.

Color representation

To extract the dot features the images will be processed on the basis of color, this process is also known as chroma keying. In computer graphics the red, green and blue (RGB) additive primary colors are commonly used to represent graphics however this format does not define a clear relationship between perceived color and the channel values R G and B. An example of the unclear relation can be seen by comparing a saturated orange color with an unsaturated orange color which show an a seemly arbitrary change in R G and B values as shown in figure 2.10.

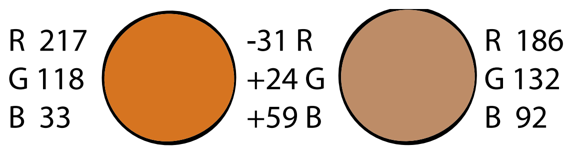


Figure 2.10: Unintuitive color representation in RGB.

The hue saturation value (HSV) representation was chosen as representation for the image which is more suitable for filtering out a desired color from the image. The HSV representation remaps the RGB colorspace by tilting the cube representation of RGB on its side with the white value [255,255,255] point to the top and black [0,0,0] to the bottom. The colors red, green, blue, cyan, magenta and yellow are now projected onto a plane and expanded into a cylindrical space. The angle of this cylindrical representation is the hue which represents the color, the radial component is the saturation and the height is the value. The process of going from RGB to HSV is illustrated in figure 2.11.

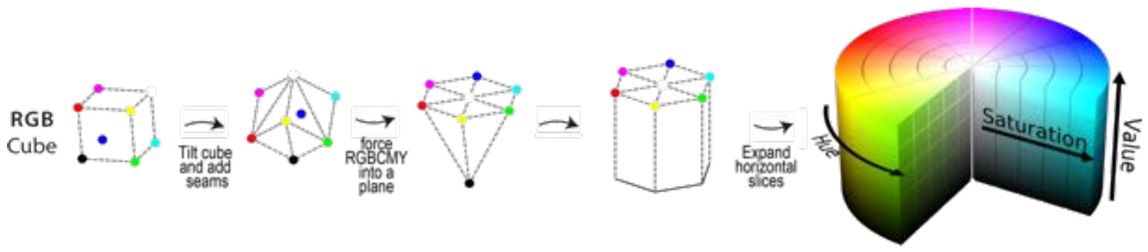


Figure 2.11: RGB to HSV

Filters

The marker that was designed has a distinct magenta color that is vibrant due to the printing process using CMYK ink. In figure 2.13a an example image from the camera can be seen. Using a threshold operation in the HSV color-space all the magenta regions are selected. As can be seen in figure 2.13b the image contains a large number of small magenta regions due to noise. The image is improved by using a morphological erode operation followed by a morphological dilate operation, this combination of morphological operations is called "opening". The resulting image shown in figure 2.13c only shows large magenta regions.

Every magenta region remaining in the image is cut out of the original image into a new image that will be individually processed. For every magenta cutout region a HSV threshold filter is is applied now selecting a yellow color. The resulting image is again enhanced by a morphological "opening" operation and every resulting region is evaluated for it's : roundness, height/width ratio and area. Every region that passes the evaluation is labeled as a "dot feature" and if exactly eight dot features are found the feature extraction is considered successful and the image processing is stopped. The other magenta regions of interest are not processed if a eight "dot feature" object has been found. If multiple markers are present in the scene, only one will be found.

The centerpoints of the dot features are sorted and saved as image points $\vec{p}_i^I$ to be used in the pose estimation.

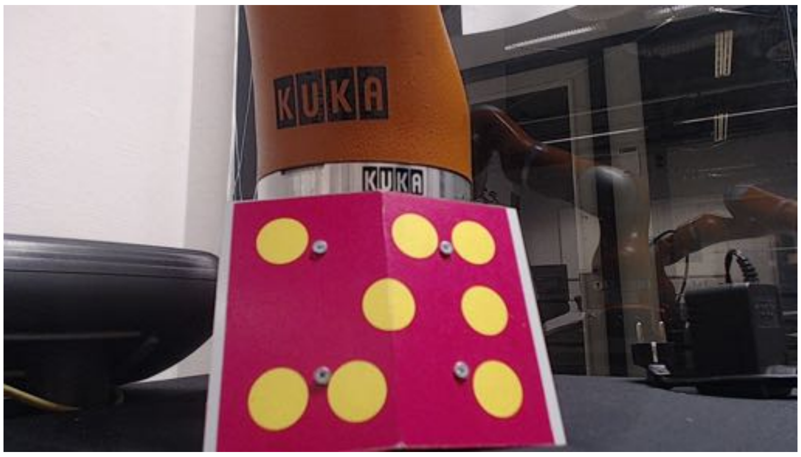
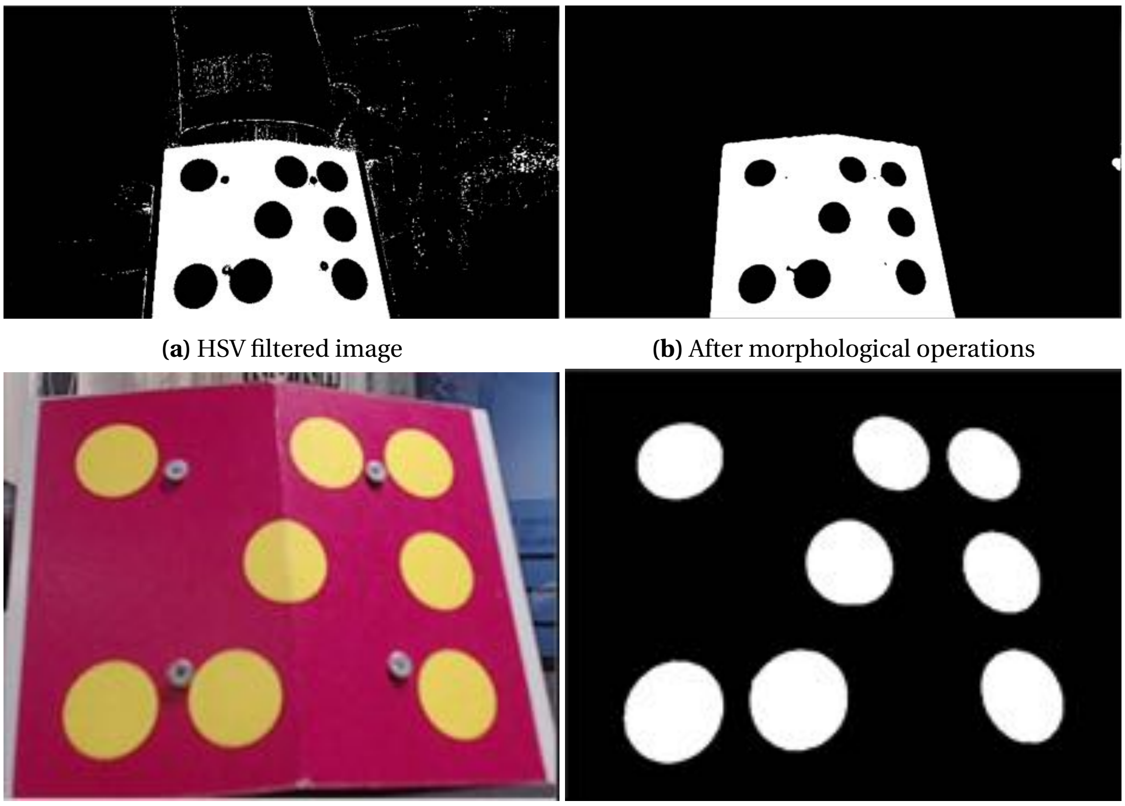


Figure 2.12: Original image.



(a) HSV filtered image **(b)** After morphological operations

(c) Cut out magenta region of interest **(d)** HSV filtering and morphological operations

Figure 2.13: Stages of feature extraction

2.5.3 Pose estimation

If the feature extraction was successful n points are found. To calculate the relative pose of the marker with respect to the camera the n point 2D to 3D point correspondence is calculated ,which projects the three dimensional points onto the image plane. The pose estimation start with modeling the 2D to 3D projection which is done using a pinhole camera model.

Pinhole camera

The pinhole camera model was chosen because we are only interested in the centers or the ellipses for pose estimation which are unaffected by the point spread function of the lens. With this basic model three dimensional points can be mapped to a two dimensional image plane by means of matrix multiplication. In the pinhole camera model the pinhole is the origin of the frame ,the image plane is located focal distance f behind the origin and the Z axis is the axis perpendicular to the image plane through the origin. In the pinhole camera model the points on the image plane and in the three dimensional space are described in homogeneous coordinates, making them scale invariant by having one extra dimension. The notation for the image plane points $\vec{\psi} = \{u, v, 1\}^T$ and three dimensional Cartesian points $\vec{p} = \{x, y, z, 1\}^T$ is different to avoid confusion.

The point $\vec{p}$ will be mapped to location $\vec{\psi}$ on the image plane which is the intersection point between the line, coming from point $\vec{p}$ through the origin, and the image plane. The image on the image plane is inverted due to the fact that the projection lines cross at the pinhole. The model is shown in figure 2.14.

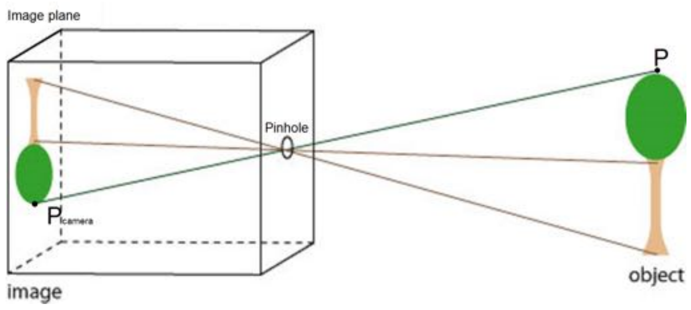


Figure 2.14: Pinhole camera model

From this model the following two projective equations can be derived:

$$u^* = \frac{fx}{z} \qquad v^* = \frac{fy}{z} \tag{2.1}$$

The asterisk in equation 2.1 is to indicate that the value for u and v is in meters and not pixels. The image sensor has a sensitivity of m_x and m_y (pixels/meter) on the sensor surface in X and Y respectively. The origin of the image is also shifted by c_u and c_v (in pixels) to be at the edge of the sensor, which ensure that the range of image indexes is positive.

$$u = m_x \frac{fx}{z} + c_u \qquad v = m_y \frac{fy}{z} + c_v \tag{2.2}$$

These equations for the projection can be rewritten into matrix representation using homogenous coordinates:

$$s\vec{\psi} = s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} m_u f & 0 & m_x c_u & 0 \\ 0 & m_v f & m_y c_v & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \left[\boldsymbol{K}|\vec{0}\right] \vec{p} \tag{2.3}$$

The $\mathbf{K} \in \mathbb{R}^{3\times3}$ matrix is called the intrinsic camera matrix and described how 3D points are mapped to a 2D plane when the camera is located at the origin and aligned with the reference frame. This matrix is also sometimes called the “Camera Calibration Matrix” because it is a which is constant when not changing the capture resolution or focus distance. The intrinsic camera matrix is assumed to be known for the camera used.

Camera movement

In the pinhole camera equation 2.3 it is presumed that the coordinates of the marker points $\vec{p}$ are known in camera coordinates. In the problem of pose estimation this is no longer the case because the camera is transformed by an unknown rotation $\boldsymbol{R}_C^M$ and translation $\vec{t}_M^C$ with respect to the marker frame. The individual marker points are now defined as $\{\vec{p}_1^M \cdots \vec{p}_n^M\}$ in the marker reference frame, and the resulting image points as $\{\vec{\psi}_1 \cdots \vec{\psi}_n\}$. The projection model now becomes :

$$s\vec{\psi}_i = \boldsymbol{K}\left[\boldsymbol{R}_C^M \;\; \vec{t}_C^M\right]\vec{p}_i^M = \boldsymbol{K}\boldsymbol{T}_C^M\vec{p}_i^M = \boldsymbol{P}\vec{p}_i^M \tag{2.4}$$

In this equation $\boldsymbol{P} \in \mathbb{R}^{3\times4}$ is defined as the projection matrix which can be decomposed into the intrinsic camera matrix $\boldsymbol{K}$ which (explained in 2.5.3) and the extrinsic camera matrix $\boldsymbol{T}_C^M \in \mathbb{R}^{3\times4}$. The notation for the extrinsic camera matrix is equal to the homogeneous transform matrix $\boldsymbol{T}_C^M$ without the last row, making the matrix scale dependent.

In case of pose estimation the image points $\vec{\psi}_i$, the marker points in the marker frame $\vec{p}_i^M$ and the intrinsic camera matrix $\boldsymbol{K}$ are known and equation 2.4 needs to be solved for the relative pose $\boldsymbol{T}_C^M$.

DLT : Pose estimation

In the Direct Linear Transform method the 12 parameters of the projection matrix $\boldsymbol{P}$ are estimated up to a scalar value.

$$s\vec{\psi}_i = \boldsymbol{P}\vec{p}_i = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{34} \\ P_{21} & P_{22} & P_{23} & P_{34} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \vec{p}_i$$

Writing out this matrix into three equations:

$$\begin{aligned} P_{11}x_i + P_{12}y_i + P_{13}z_i + P_{14} &= s(u_i) \\ P_{21}x_i + P_{22}y_i + P_{23}z_i + P_{24} &= s(v_i) \\ P_{31}x_i + P_{32}y_i + P_{33}z_i + P_{34} &= s(1) \end{aligned}$$

The third equation can be eliminated to solve the homogeneous scaling factor s, yielding the following equations per marker point.

$$\begin{aligned} P_{11}x_i + P_{12}y_i + P_{13}z_i + P_{14} - (P_{31}x_i + P_{32}y_i + P_{33}z_i + P_{34})u_i = 0 \\ P_{21}x_i + P_{22}y_i + P_{23}z_i + P_{24} - (P_{31}x_i + P_{32}y_i + P_{33}z_i + P_{34})v_i = 0 \end{aligned}$$

Combining the equations for all n points and placing them into a matrix where the columns represent projection matrix parameters results in [5]:

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1z_1 & -u_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1z_1 & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 & 0 & 0 & 0 & 0 & -u_nx_n & -u_ny_n & -u_nz_n & -u_n \\ 0 & 0 & 0 & 0 & x_n & y_n & z_n & 1 & -v_nx_n & -v_ny_n & -v_nz_n & -v_n \end{bmatrix} * \begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \\ P_{14} \\ P_{21} \\ P_{22} \\ P_{23} \\ P_{24} \\ P_{31} \\ P_{32} \\ P_{33} \\ P_{34} \end{bmatrix} = \vec{0}$$

With at least six points correspondence a estimate for the projection matrix $\tilde{\boldsymbol{P}}$ can be found with singular value decomposition as the Eigen vector corresponding to the lowest singular value. The inverse of the known intrinsic camera matrix $\boldsymbol{K}$ is used to find the extrinsic camera matrix $\tilde{\boldsymbol{H}}_C^M$ up to a scalar value.

$$s\tilde{\boldsymbol{T}}_C^M = \boldsymbol{K}^{-1}\tilde{\boldsymbol{P}}$$

Because it is known that the determinant of the rotation matrix should equal one the scaling factor s can be found by:

$$s = \frac{1}{\det(\boldsymbol{R})}$$

The DLT method gives a good initial estimate of the pose but is susceptible to noise [5] which is likely to be present in the signal. The DLT method is used to get a rough estimation of the pose but it was chosen to use an iterative parameters optimizer to improve the results.

Iterative parameter optimization

Parameter optimization is achieved by minimizing a non-linear least squares error. The error that is to be optimized is the re-projection error which is the difference between the image point and the projected object point in the image plane. For the method that is used the parameters $\vec{\theta}$ to be optimized are : the components of the translational vector $\vec{t}_C^M$ and the angles derived from the rotation matrix $\boldsymbol{R}_C^M$.

$$\vec{\theta} = \begin{bmatrix} t_x & t_y & t_z & r_x & r_y & r_z \end{bmatrix}^T$$

The re-projection error $\vec{e}(\vec{\theta}) \in \mathbb{R}^{2n}$ and cost function $g(\vec{\theta}) \in \mathbb{R}$ to be optimized are defined as:

$$\vec{e}(\vec{\theta}) = \sum_{i=1}^{n} \left(\vec{\psi}_i - \boldsymbol{K}\tilde{\boldsymbol{T}}_C^M(\vec{\theta})\vec{p}_i\right)$$

$$g(\vec{\theta}) = \frac{1}{2}\left\|\vec{e}(\vec{\theta})\right\|^2 = \frac{1}{2}\vec{e}(\vec{\theta})^T\vec{e}(\vec{\theta})$$

$$s\vec{P}_i = \boldsymbol{K}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\vec{p}_i$$

Because the first estimate of the parameters are close to the minimum the cost function can be expanded into a second order Taylor series to get:

$$g(\vec{\theta}_0 + \vec{\Delta}_\theta) = g(\vec{\theta}_0) + \nabla g(\vec{\theta}_0)\vec{\Delta}_\theta + \frac{1}{2}\vec{\Delta}_\theta^T \mathscr{H}_{g(\vec{\theta}_0)}\vec{\Delta}_\theta$$

In which the gradient $\nabla g(\vec{\theta}_0) \in \mathbb{R}^{2n}$ denotes the first derivative of $g(\vec{\theta}_0)$ and the Hessian matrix $\mathscr{H}_{g(\vec{\theta}_0)} \in \mathbb{R}^{2n\times 2n}$ the second derivative. The cost function can be minimized by finding the derivative with respect to Δ_θ and setting it to zero, which will leave the following expression:

$$\mathscr{H}_{g(\vec{\theta}_0)}\vec{\Delta}_\theta = -\nabla g(\vec{\theta}_0) \tag{2.5}$$

By evaluating the gradient with respect to the re-projection error $\vec{e}(\vec{\theta})$ it is found that:

$$\nabla g(\vec{\theta}_0) = \frac{\partial \vec{e}(\vec{\theta}_0)}{\partial \vec{\theta}}\vec{e}(\vec{\theta}_0) = J(\vec{\theta}_0)^T\vec{e}(\vec{\theta}_0)$$

In which the Jacobian $J(\vec{\theta}_0) \in \mathbb{R}^{6\times 2n}$ is a combination of all the image Jacobians of the marker points for the parameters $\vec{\theta}_0$. The parameter change $\vec{\Delta}_\theta$ as function of the Hessian, Jacobian and error function can now be formulated as:

$$\vec{\Delta}_\theta = -\mathscr{H}^{-1}_{g(\vec{\theta}_0)} J(\vec{\theta}_0)^T\vec{e}(\vec{\theta}_0) \tag{2.6}$$

This expression is the basis of four non-linear optimizers which differ in the way they handle the Hessian:

- **Netwon method** : The hessian is calculated with the second derivative of the error.

  $$\mathscr{H}_{g(\vec{\theta}_0)} = \frac{\partial J(\vec{\theta}_0)^T}{\partial \vec{\theta}}\vec{e}(\vec{\theta}_0) + J(\vec{\theta}_0)^T J(\vec{\theta}_0)$$

  The Newton optimizer gives a good approximation particularly near the minimum but is slow to converge when not starting near the minimum, due to the fact that calculating the Hessian with a second derivative every iterative step is computationally heavy it is not suitable for real-time applications.

- **Guass-Netwon method** : The Hessian is approximated by assuming the error locally linear thus eliminating the second derivative.

  $$\mathscr{H}_{g(\vec{\theta}_0)} = J(\vec{\theta}_0)^T J(\vec{\theta}_0)$$

  A Gauss-Newton optimizer is very similar to the Newton optimizer in its performance but does not have the computational load by approximating the Hessian.

- **Gradient descent method** : The Hessian is quote "approximated (somewhat arbitrarily) by the scalar matrix $\lambda \boldsymbol{I}$." [6]

  $$\mathscr{H}_{g(\vec{\theta}_0)} = \lambda \boldsymbol{I}$$

  The Gradient Descent optimizer updates towards the most rapid local decrease in error with a factor λ, it is characterized by a rapid movement towards the minimum but a slow convergence near the minimum due to overshoot. By itself it is not a good minimization technique but can be used to approach the minimum or get out of false local minimum.

- **Levenberg-Marquardt** : This method combines the Gauss-Netwon and gradient descent method to approximate the Hessian.

  $$\mathscr{H}_{g(\vec{\theta}_0)} = \left(J(\vec{\theta}_0)^T J(\vec{\theta}_0) + \lambda \boldsymbol{I} \right)$$

  This extension to the Gauss-Newton method interpolates between Gauss-Newton and gradient descend. This addition makes Gauss-Newton more robust meaning it can start far off the correct minimum and still find it. But if the initial guess is good than it can actually be a slower way to find the correct pose.

The optimization method that was chosen for this system was the Levenberg-Marquardt method which changes the parameters $\vec{\Delta}_\theta$ every iteration by:

$$\vec{\Delta}_\theta = \left(J(\vec{\theta}_0)^T J(\vec{\theta}_0) + \lambda I \right)^{-1} J(\vec{\theta}_0)^T \vec{e}(\vec{\theta}_0)$$

The covariance of the pose estimate can also be calculated using the Jacobian and the pixel noise which is assumed Gaussian with equal variance for all image points σ^2_{pose}. [[6]]

$$\Sigma_{pose} = \sigma^2_{pose} \left(\boldsymbol{J}^T \boldsymbol{J} \right)^+$$

Where + denotes the pseudo inverse operation.

The solver uses a local approximation for the error function and Jacobian which has the disadvantage that it can only find the local minimum. This is especially relevant when using a coplanar marker which can leads to pose ambiguity and multiple local minima close to each other [7]. Whether the correct pose is returned by the optimizer depends on the initial rough estimate of the pose.

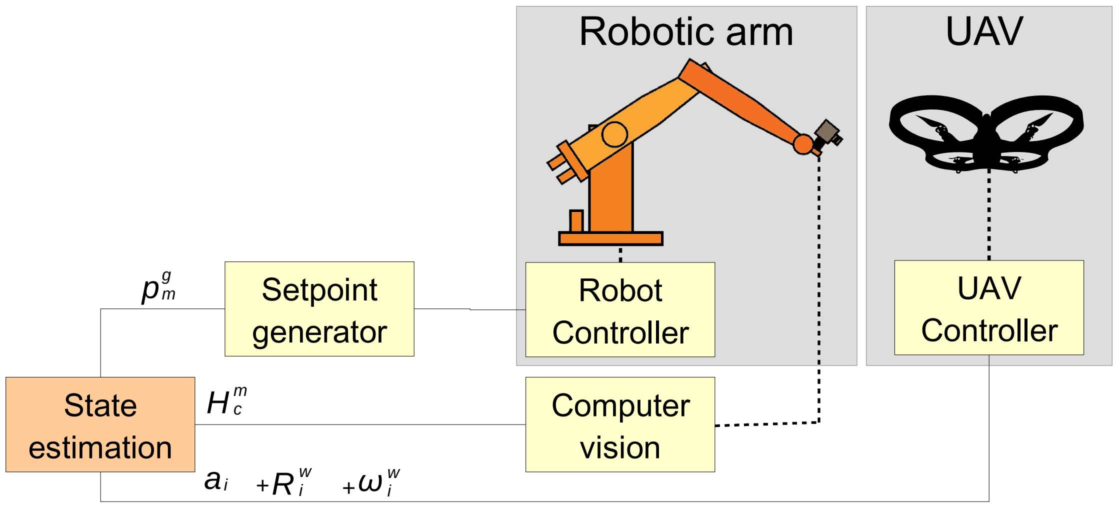


Figure 2.15: Control schema - State estimator.

2.6 State estimator-subsystem

The State estimator subsystem is responsible for combining the information from the different sensors and producing a optimal state estimate based on the system dynamics, the sensor signals and the uncertainty information provided. The different sensors all act in their own respective frame of reference so before the signals can be compared they have to transformed to the global frame of reference which is defined to be the frame of the robotic arm base.

The input of the system is:

- The estimated relative pose of the camera with respect to the marker $\overset{*}{\boldsymbol{H}}{}_c^m$.
- The current position of the end-effector of the robotic arm $\boldsymbol{H}_h^g$. (not in diagram)
- The acceleration measurement from the UAV $\vec{a}_i$, in body fixed coordinates.
- The orientation measurement from the UAV $\boldsymbol{R}_i^w$

The output of the system is:

- The optimal estimate of the position of the marker with respect to the robotic arm base $\overset{*}{\boldsymbol{H}}{}_m^g$.

2.6.1 Transformations

The pose estimation estimates the relative pose between the camera and the marker $\boldsymbol{H}_c^m$, this pose should be be expressed with respect to the global frame. The relative pose between the end-effector and camera aperature $\boldsymbol{H}_c^h$ is assumed a known and the end-effector pose with respect to the robot base $\boldsymbol{H}_h^g$ is measured by the robotic arm itself, making the equation:

$$\boldsymbol{H}_m^g = \boldsymbol{H}_h^g \boldsymbol{H}_c^h \boldsymbol{H}_m^c$$

The UAV is assumed to be a rigid body with the IMU located at the center of mass and center of rotation. The marker is assumed rigidly connected to the body and located a distance $\vec{p}_m^{\,i}$ from the center. Because of the rigid body assumption it can be stated that $\vec{a}i^g = \vec{a}m^g$

The IMU measurements consists of a orientation $\boldsymbol{R}_i^w$ and a linear acceleration vector $\vec{a}_i$, in body fixed coordinates. The measured acceleration needs to be in the same reference frame as the vision measurement however world frame is not clearly defined in the global frame of the robotic arm base. The world frame is defined as the magnetic north as X-axis and the gravitational direction as minus Z-axis. With the robotic arm mounted in upright position the Z-axis of the global frame corresponds to the Z-axis of the world frame but global frame has no reference with respect to magnetic north. For this reason the yaw rotation of the UAV is removed and the remaining rotation is applied to the accelerometer data. The relative yaw between the global frame and marker is available from the vision measurements. The gravity rejection for the IMU still works because the gravity is not dependent on the magnetic north yaw information.

2.6.2 Kalman filter

Notation

In this analysis of the mathematics behind Kalman filters the notation will slightly change. The subscript x_k or x_{k-1} will denote the discrete time instance k and $k-1$ respectively. Stochastic Gaussian noise variables will be described with $\vec{\eta}_{x_k}$ which indicates that is is the noise acting on x at time instance k. The covariance matrices of this noise process will be written as $\boldsymbol{\Sigma}_k^{\eta_x}$, not to be confused with the uncertainty of the state $\boldsymbol{\Sigma}_k^X$. With this notation the stochastic noise processes and covariance matrices are not in the same alphabet as the system matrices and system vectors for convenience. The notation for a estimate is a tilde above the symbol and the optimal estimate is denoted with a asterisk above the symbol. Lastly $\tilde{\vec{x}}_{k|k-1}$ is the notation used for the estimate of $\vec{x}$ at discrete timestep k given the estimate $\tilde{\vec{x}}_{k-1}$ at discrete time $k-1$.

Kalman filter

To fuse the sensor information a state estimator is used to estimate the most likely position of the drone using information from the pose estimation and drone IMU. Defining $\boldsymbol{A}$ as the system matrix and $\boldsymbol{B}$ as the input matrix; $\vec{x}$ as the state variable; $\vec{\eta}_{x_k}$ as the system noise and $\vec{\eta}_{u_k} \in \mathbb{R}^{9\times 1}$ as the input noise. This will give us the following state equations:

$$\vec{x}_k = \boldsymbol{A}\vec{x}_{k-1} + \boldsymbol{B}(\vec{u}_k + \vec{\eta}_{u_k}) + \vec{\eta}_{x_k}$$

Where $\vec{x}_k$ is the true state at sampling instant k. $\vec{\eta}_{u_k}$ and $\vec{\eta}_{x_k}$ are assumed to be multivariate normal distributed processes with zero mean and covariance $\boldsymbol{\Sigma}_k^{\eta_u}$ and $\boldsymbol{\Sigma}_k^{\eta_x}$ respectively.

Defining $\boldsymbol{C} \in \mathbb{R}^{3\times 9}$ as the observed matrix; $\vec{y} \in \mathbb{R}^{3\times 1}$ as the observed output and $\vec{\eta}_{y_k} \in \mathbb{R}^{3\times 1}$ as the measurement noise the following measurement equations is found:

$$\vec{y}_k = \boldsymbol{C}\vec{x}_{k-1} + \vec{\eta}_{y_k}$$

Where $\vec{y}_k$ is the true system output at sampling instant k. $\vec{\eta}_{y_k}$ is assumed to be a multivariate normal distribution with zero mean and covariance $\boldsymbol{\Sigma}_k^{\eta_u}$ and $\boldsymbol{\Sigma}_k^{\eta_x}$ respectively

The algorithm used is the Kalman Algorithm uses a prediction and correction or update step.

Prediction phase In the prediction phase the previous state estimate is used to estimate the value of the state at the current timestep. Later in the correction phase measurements from a sensor are used refine the prediction and find a more accurate state estimate.

The prediction step is represented by the following equation:

$$\tilde{\vec{x}}_{k|k-1} = \boldsymbol{A}\tilde{\vec{x}}_{k-1} + \vec{B}\vec{u}_k$$

For the predicted state estimate, the uncertainty and thus covariance of the prediction $\boldsymbol{\Sigma}_{k|k-1}^x$ also evolves :

$$\boldsymbol{\Sigma}_{k|k-1}^X = \boldsymbol{A}\boldsymbol{\Sigma}_{k-1}^X\boldsymbol{A}^T + \boldsymbol{B}\boldsymbol{\Sigma}_k^{\eta_x}\boldsymbol{B}^T$$

Prediction step
(1) Predict state
$\tilde{\vec{x}}_{k\|k-1} = \boldsymbol{A}\tilde{\vec{x}}_{k-1} + \boldsymbol{B}\tilde{\vec{u}}_{k-1}$
(2) Predict error
$\boldsymbol{\Sigma}^X_k = \boldsymbol{A}\boldsymbol{\Sigma}^X_{k-1}\boldsymbol{A}^T + \boldsymbol{B}\boldsymbol{\Sigma}^{\eta_u}_{k-1}\boldsymbol{B}^T + \boldsymbol{\Sigma}^{\eta_x}_{k-1}$

Correction step
(1) Predict system output
$\tilde{\vec{y}}_{k\|k-1} = \boldsymbol{C}\tilde{\vec{x}}_{k-1}$
(2) Compute the Kalman gain [8]
$\boldsymbol{K}_k = \boldsymbol{\Sigma}^X_k\boldsymbol{C}^T\left(\boldsymbol{C}\boldsymbol{\Sigma}^X_k\boldsymbol{C}^T + \boldsymbol{\Sigma}^{\eta_y}_{x_k}\right)^{-1}$
(3) Update state (Optimal state estimate)
$\overset{*}{\vec{x}}_k = \tilde{\vec{x}}_k + \boldsymbol{K}_k\left(\vec{y}_k - \tilde{\vec{y}}_{k\|k-1}\right)$
(4) Update error
$\overset{*}{\boldsymbol{\Sigma}}^X_k = \left(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}\right)\boldsymbol{\Sigma}^X_k$

Table 2.2: Kalman filter overview

Correction As soon as a new measurement comes in from the sensor the measurement residuals are calculated using latest prediction and new measurement:

$$\vec{s}_k = \vec{y}_k - \boldsymbol{C}\vec{x}_{k|k-1}$$

$$\boldsymbol{\Sigma}^S_k = \boldsymbol{C}\boldsymbol{\Sigma}^X_{k|k-1}\boldsymbol{C}^T + \boldsymbol{\Sigma}^{\eta_y}_k$$

The optimal kalman gain now calculated by [8]

$$\boldsymbol{K}_k = \boldsymbol{\Sigma}^X_{k|k-1}\boldsymbol{C}^T_k\left(\boldsymbol{\Sigma}^S_k\right)^{-1}$$

The optimal updated state estimate now is calculated using the Kalman gain $\boldsymbol{K}_k$ and the measurement residuals $\vec{s}_k$.

$$\overset{*}{\vec{x}}_k = \tilde{\vec{x}}_{k|k-1} + \boldsymbol{K}_k\vec{s}_k$$

And uncertainty of the optimal state estimate is update accordingly.

$$\overset{*}{\boldsymbol{\Sigma}}^X_k = \boldsymbol{\Sigma}^X_{k|k-1} - \boldsymbol{K}_k\boldsymbol{\Sigma}^S_k\boldsymbol{K}^T_k$$

An overview of the Kalman filter is given in figure 2.2.

2.6.3 Extended Kalman filter

Because rotations are involved the system cannot be directly written as regular kalman filter because the system behavior is time variant and nonlinear. The different sensors signals are defined in different frames and the there is uncertainty in the relation between these frames. A good solution would be to use an extended kalman filter that linearizes the system every timestep to be able to use the normal equation for a kalman filter.

The state equation is now governed by the non-linear stochastic difference equation:

$$\vec{x}_k = f\left(x_{k-1}, u_{k-1}, \eta_{x_{k-1}}\right)$$

Just as with the regular kalman filter new state can be estimated using the posteriori state estimate $\vec{k}_{k-1}$ and the assumption that the mean of the noise is zero.

$$\tilde{\vec{x}}_k = f\left(x_{\tilde{k}-1}, u_{k-1}, 0\right)$$

Prediction step
(1) Predict state
$\tilde{\vec{x}}_{k\|k-1} = \boldsymbol{A}_k \tilde{\vec{x}}_{k-1}$
(2) Predict error
$\boldsymbol{\Sigma}^X_k = \boldsymbol{A}_k \boldsymbol{\Sigma}^X_{k-1} \boldsymbol{A}^T_k + \boldsymbol{W}_k \boldsymbol{\Sigma}^{\eta_x}_{k-1} \boldsymbol{W}^T_k$

Correction step
(1) Predict system output
$\tilde{\vec{y}}_{k\|k-1} = \boldsymbol{C}_k \tilde{\vec{x}}_{k-1}$
(2) Compute the Kalman gain [8]
$\boldsymbol{K}_k = \boldsymbol{\Sigma}^X_k \boldsymbol{C}^T_k \left(\boldsymbol{C}_k \boldsymbol{\Sigma}^X_k \boldsymbol{C}^T_k + \boldsymbol{V}_k \boldsymbol{\Sigma}^{\eta_y}_{x_k} \boldsymbol{V}^T_k \right)^{-1}$
(3) Update state (Optimal state estimate)
$\overset{*}{\vec{x}}_k = \tilde{\vec{x}}_k + \boldsymbol{K}_k \left(\vec{y}_k - \tilde{\vec{y}}_{k\|k-1} \right)$
(4) Update error
$\overset{*}{\boldsymbol{\Sigma}}^X_k = \left(\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{C}_k \right) \boldsymbol{\Sigma}^X_k$

Table 2.3: Extended Kalman filter overview

This equation now needs to be linearized

$$\boldsymbol{A}_{k-1} = \frac{\partial f\left(\tilde{\vec{x}}_{k-1}, \vec{u}_{k-1}, 0\right)}{\partial \vec{x}}$$

$$\boldsymbol{W}_{k-1} = \frac{\partial f\left(\tilde{\vec{x}}_{k-1}, \vec{u}_{k-1}, 0\right)}{\partial \vec{\eta}_{x_{k-1}}}$$

These same steps can be done for the output equation of the system.

$$\vec{y}_k = h\left(x_{k-1}, u_{k-1}, \eta_{x_{k-1}}\right)$$

$$\tilde{\vec{y}}_k = h\left(x_{\tilde{k}-1}, u_{k-1}, 0\right)$$

$$\boldsymbol{C}_{k-1} = \frac{\partial h\left(\tilde{\vec{x}}_{k-1}, \vec{u}_{k-1}, 0\right)}{\partial \vec{x}}$$

$$\boldsymbol{V}_{k-1} = \frac{\partial h\left(\tilde{\vec{x}}_{k-1}, \vec{u}_{k-1}, 0\right)}{\partial \vec{\eta}_{x_{k-1}}}$$

Combining these equations with the normal deviration of the Kalman filter reasults in the following predict and update steps.

It is important to note that a fundamental flaw of the EKF is that the distributions (or densities in the continuous case) of the various random variables are no longer normal after undergoing their respective nonlinear transformations. The EKF is simply an ad hoc state estimator that only approximates the optimality of Bayes rule by linearization [8].

2.6.4 Extended Kalman filter design

For the designed Extended Kalman filter the choice was made not to estimate the rotation in the filter. This was done under the assumption that either the UAV is only making very small rotations which is the case when trying to hoover in place while waiting to be docked.

As first six state variables $\vec{x}$ the position of the drone IMU $\vec{p}$ and its velocity $\dot{\vec{p}}$ in the reference frame of the robot base are used. The last three state variables the biases of the accelerometer $\vec{b}_a$ in its own reference frame are used, which are also estimated by the state estimator. The inputs of the system are the body fixed accelerations $\vec{a}_{imu}$ and the gravity $\vec{g}$.

Care should be taken into reading the following formulas because the subscript notation has a different meaning for the position, velocity and rotations. The first subscript denotes the frames and the outer subscript is the discrete time instance.

$$\vec{x} = \left\{\vec{p}^T \dot{\vec{p}}^T, \vec{b}_i^T, \right\}$$

$$\vec{u} = \left\{a_x^{imu} - b_x^{imu}, a_y^{imu} - b_y^{imu}, a_z^{imu} - b_z^{imu}, 0, 0, g\right\}$$

The Kalman matrices $\boldsymbol{A}_k$, $\boldsymbol{B}_k$, $\boldsymbol{C}$ matrices are filled in the following way. The $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ matrices are time dependent due to the rotations and $\boldsymbol{I}_n$ denotes an $\mathbb{R}^{n\times n}$ identity matrix and $\mathbf{0}_n$ a zero matrix with dimensions $n \times n$.

$$\mathbf{A}_k = \begin{bmatrix} \boldsymbol{I}_3 & T\boldsymbol{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \boldsymbol{I}_3 & T\left[\boldsymbol{R}_i^g\right]_k^{-1} \\ \mathbf{0}_3 & \mathbf{0}_3 & \boldsymbol{I}_3 \end{bmatrix} \qquad \mathbf{B}_k = \begin{bmatrix} \mathbf{0}_3 & \mathbf{0}_3 \\ T\left[\boldsymbol{R}_i^g\right]_k & \boldsymbol{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} \boldsymbol{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix}$$

3 Implementation and Realisation

3.1 Software Implementation : ROS

For implementation of the software the Robotic Operating System framework has been chosen. This is a software framework developed by the Open Source Robotics Foundation [9] and helps developer create complex and robust robotic systems with the help of tools, libraries and an underlying communications framework. The version of the framework used for this Project is the 7th distribution: Hydro Medusa and the operating system that the system is running on is Ubuntu 12.04 LTS.Although the framework runs quite low level the system has no guarantees for hard-realtime application because the underlying Linux distribution is not a real-time operating system. The system runs under the highest priority in Linux and can be considered soft-realtime. The system is a Intel I7 2600K with 16 Gb of DDR3 ram.

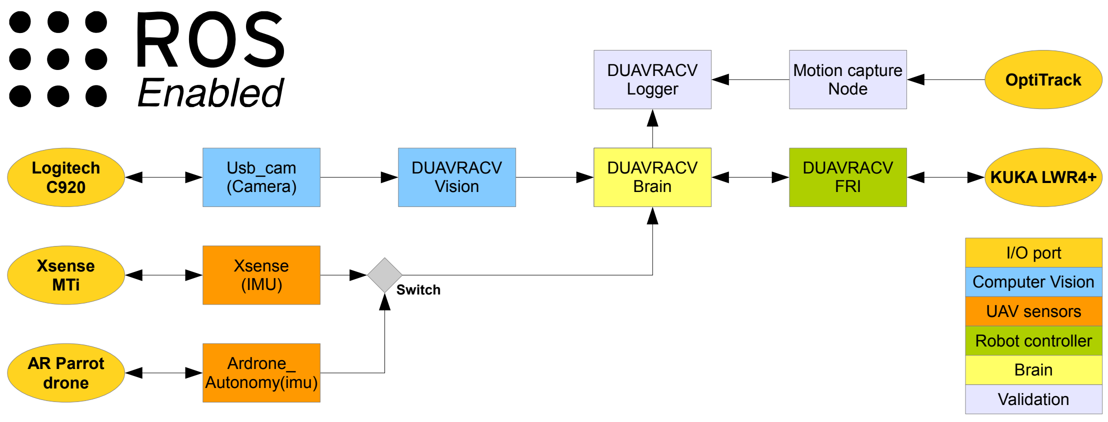


Figure 3.1: ROS node diagram

The software implementation diagram is shown in figure 3.1. The abbreviation DUAVRACV is used as an affix for the nodes created for this project and stands for "Docking a Unmanned Arial Vehicle using a Robotic Arm and Computer Vision". The nodes correspond to the subsystem as discussed in the analysis and their relation can be seen in the colored legend. The subsystems for the state estimator and set-point generator are merged into the node called "'Brain". The reason for this is that the State Estimator and Set-point generator both needs many signals from the other nodes which would introduce involve a large number of additional topics and extra delays which can be avoided by merging. As can be seen from the diagram there are two possible choices of IMU in the implementation of the system but only one will be used simultaneously. More information about these IMUs and the reasoning behind using a drone substitute is discussed in the hardware realization.

All "DUAVRACV" nodes created for this project will be discussed in their respective paragraphs and the other nodes will be in short explained here.

- **USB cam** : This node takes camera parameters from a launch file and then starts streaming the images to a topic. This is a library from the ROS Wiki and more information can be found at `http://wiki.ros.org/usb_cam`
- **Xsense** : This node is used to connect to a Xsense IMU which which sends out IMU messages on a topic. This is a library from the ROS Wiki and more information can be found at `http://wiki.ros.org/xsens_driver`

- **Xsense** : This node is used to connect to a Xsense IMU which which sends out IMU messages on a topic. This is a library from the ROS Wiki and more information can be found at http://wiki.ros.org/xsens_driver
- **Ardone autonomy** : This node is used to connect a Parrot AR.Drone over wifi and it posts IMU messages to a topic. The nodes also sends a lot more information and the drone can even be controlled using this node, however these features are not used for this system. This is a library from the ROS Wiki and more information can be found at http://wiki.ros.org/ardrone_autonomy
- **Motion capture node** : This node is used to connect to the OptiTrack motion capture system that is used for validating the measurements and experiments, it posts pose messages for every "'Trackable' registered in the OptiTrack system. This is a library from the ROS Wiki and more information can be found at http://wiki.ros.org/mocap_optitrack

Information transfer and threading Nodes in the ROS framework run on separate threads which is useful for code that can block a thread by either requiring input/output (IO) connections or have calculations with high computational requirement or delays. The threading in ROS can still starve a node of CPU time if the total computational load is too high. Threading is the reason is why IO connections are done in separate nodes in ROS system. The "DUAVRACV Vision" program has a high computational load and runs at a low frequency and can therefor not be included into the main "Brain" node, while the State estimator and set-point generator are computational light and run at much higher frequencies. The Topic/ subscriber message system implemented by ROS handles the data transfer between nodes(threads) while ensuring thread safety [10]. The ensured thread safety and multi-threading framework one of the key features of ROS for inexperienced C++ developers.

3.1.1 Brain node

The Brain node is the center of the software system where all the information from the different come together.

The node is build up by a class “Brain” with four member classes that handle different signals. A simplified diagram of the communication in this node is shown in figure 3.2.

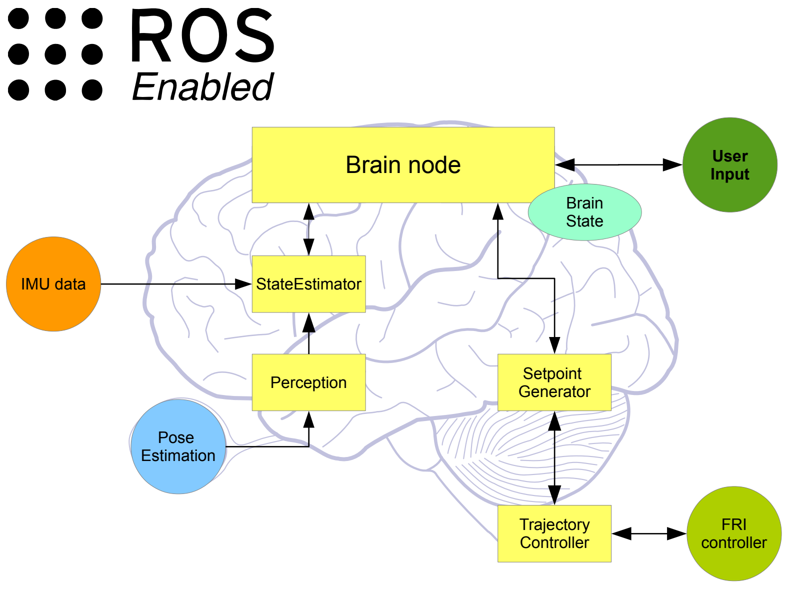


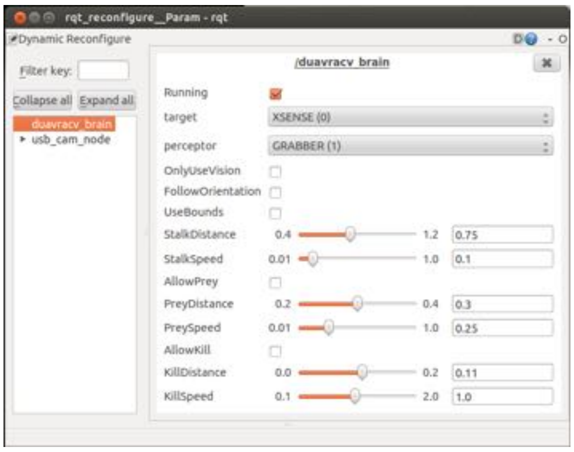
Figure 3.2: Brain node diagram

The main brain class connects all the classes together and handles the communication with ROS with the subscriptions and publishers. The arrows in the diagram are pointing the class objects because the messages coming from the other nodes are immediately send to the respective class object and not processed in the main class. The user can influence the behavior of the controller by means of the “RQT reconfigure GUI” which is a graphical window that allows for parameters of a node to be changed during run time. With this tool the user can (dis)allow behavioral states and change the controller gain or follow distances without restart the node.

Brain State The brain state corresponds to the chosen setpoint generating strategy as discused 2.2.1. Eventhough it is only one variable it is a very important variable in the behavior of the node. To determine the next brain state the brain class takes advise from the other classes which have a better view on current robot pose, distances to the bounds and the and pose estimation quality. This is done by polling the State Estimator and Setpoint generator every cycle to see what state they would advise for the next iteration cycle. If both advise to go to the next state this is done unless the user input disallows that state from being used.

Perception This object interprets the relative poses messages $\boldsymbol{H}_c^m$ of the pose estimator, transforms them to the global frame $\boldsymbol{H}_m^g$ and determines if there are no improbable shifts in position. The object also keeps track on the success-rate of the last hundred pose estimations which is available for the State Estimator. Because the camera has a delay between the image on the sensor and the time it is processed and in the “Brain” node, not the latest end-effector $\boldsymbol{H}_c^m$ pose is used but a older pose that is saved.

State Estimator As described in the Analysis this object estimates the state of the UAV position, velocity and the accelerometer biases. The IMU data is interpreted by the State estimator



/duavracv_brain

Running	✓			
target	XSENSE (0)			
perceptor	GRABBER (1)			
OnlyUseVision	☐			
FollowOrientation	☐			
UseBounds	☐			
StalkDistance	0.4		1.2	0.75
StalkSpeed	0.01		1.0	0.1
AllowPrey	☐			
PreyDistance	0.2		0.4	0.3
PreySpeed	0.01		1.0	0.25
AllowKill	☐			
KillDistance	0.0		0.2	0.11
KillSpeed	0.1		2.0	1.0

Figure 3.3: Dynamic reconfigure GUI

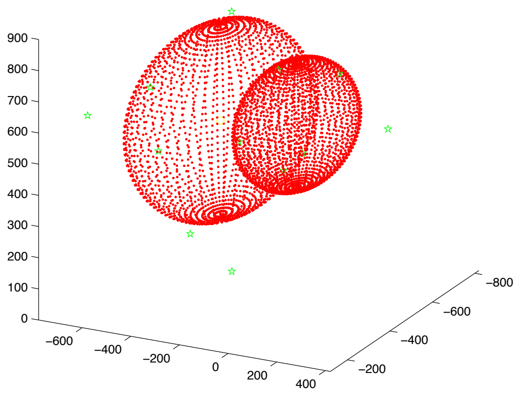


Figure 3.4: Reachable subspace approximation

itself, and tranformed into the correct reference frame. The state estimator keeps track of the success-rate of the perception class to determine if the state estimation is still reliable because it is assumed unreliable if there more than 90% estimations where unsuccessful. In this case the pose has been lost for at least 60 frames which corresponds to 3 seconds with a frame rate of 20 hz, the moment that the influence of the acceleration bias becomes significant. When this occurs the state machine is disables and re-initialized when at least 20% of the pose estimates were successful.

Setpoint generator Like described in the analysis the set-point generator tries generate a set-point according to the know UAV position and behavioral state. The reachable space is defined as a sum of spherical volumes with a center point $\vec{p}_i$ and a radius r_i. The process of checking bounds is done by checking the comparing the distance to all center-points and comparing them to the radii provided. This three dimensional approach does not cover the end-effector orientation but is adequate in preventing the robot to go out of bounds in a dangerous way. In figure 3.4 the subspace is illustrated by the red dots, the green stars are known robot limits.

Path controller The trajectory controller generates a twist that is limited to a maximum value like discussed in the analysis for path generation.

3.1.2 Robot controller : FRI node

The FRI node controls the robot using the Fast Research Interface (FRI) to communicate with the Kuka Robot Controller (KRC). The FRI node is created around the C++ FRI Library adapted in such a way that it is able to run in the ROS environment and its role is to be a bridge between the KRC and other nodes in the ROS framework. For safety reasons there are rules and procedures in the KRC to go through before being allowed to be controlled using FRI and if this is not done correctly the robot is drives are disengaged, stopping the program running on the KRC. The FRI nodes aims be a smooth interface in which the end user only gives the desired (joint) pose, stiffness, etc and the nodes handle the procedure acquiring control over the KRC.

The FRI node can handle all three controller strategies provided by KUKA but the report only handles the Cartesian Impedance control mode. The overall FRI connection and setup procedures are identical only the contents of the packages is different.

The FRI libraries handles the UDP connection and packages and is used within the FRI node. It should be noted that the FRI library is thread locking and should therefor always run on a separate thread.

FRI Statemachines

The FRI node uses a state-machine in the ROS node and also a state machine running on the KRC to handle the interface connection. The state machine on the KRC always copies the state of the ROS state machine which sends its state as an integer inside the return package to the KRC. The states of the ROS node are: IDLE, INIT, MON, PRECMD, ENDCMN, CMD, KILL. Transitions between states are shown in the diagram in figure 3.5.

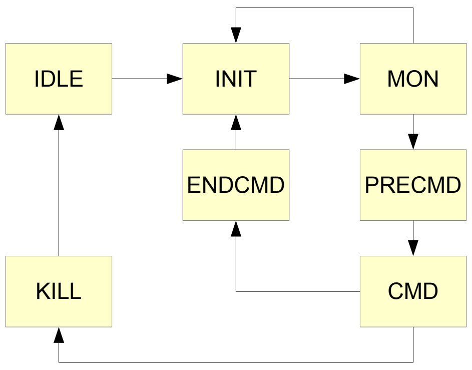


Figure 3.5: FRI statemachine diagram

The IDLE state is used when no connection is desired.

The INIT initialize the ROS node according to a set of booleans provides which tell it what topics to subscribe and publish to and what commandflags to set in return FRI packages. The node is starts listening to UDP packages coming to the desired ip adress and port. After the connection is established it immidiatly transistions into MON state.

The MON state monitors all the variables coming from the KRC and publishes them on topics. When the different type of control is required the state transitions into INIT mode and when the robot needs to be controlled the state transitions into PRECMD mode.

The PRECMD state is to signal the KRC the node wants to go into commandmode to control the robot. The KRC statemachine starts the command mode when getting the integer value of this state. The return message is prepared for the mode transition by making the commanded pose exactly match the measured pose. If all the KRC checks are good, the KRC changes to command mode and when this information has reached the ROS statemachine it follows to CMD mode. Monitoring and publishing the KRC variables is done in this mode.

The CMD state controls the robot with the values coming from the topic. Monitoring and publishing the KRC variables is done in this mode. When the different type of control is required the state transitions into ENDCCMD mode.

The ENDCMD state is used when monitor mode is required or a change in type of control. The KRC statemachine stops the command mode when getting the integer value of this state and the ROS statemachine transitions into INIT state when the KRC mode information is received.

The KILL state is to get back into IDLE state. The KRC statemachine stops the command mode and consequently closes the FRI connection when getting the integer value of this state.

3.1.3 Data acquisition : DUAVRACV Logger

The logger node is responsible from capturing the desired signals for experiments. It can capture both information from topics or from tranforms that have been broadcasted over the ROS network. The loggers runs at a frequency of $500Hz$ and saves the data to a raw coma sepperated file (.csv). The latest version of the logger logs the following measurements:

- Current pose of the end-effector of the robotic arm.
- The optimal state estimate for the marker pose.
- The pose estimation according to the computer vision.
- The setpoint generated by the setpoint generator (not the intermediate point).
- The OptiTrack validation measurement of the IMU.

3.2 Hardware realisation

3.2.1 Robotic arm

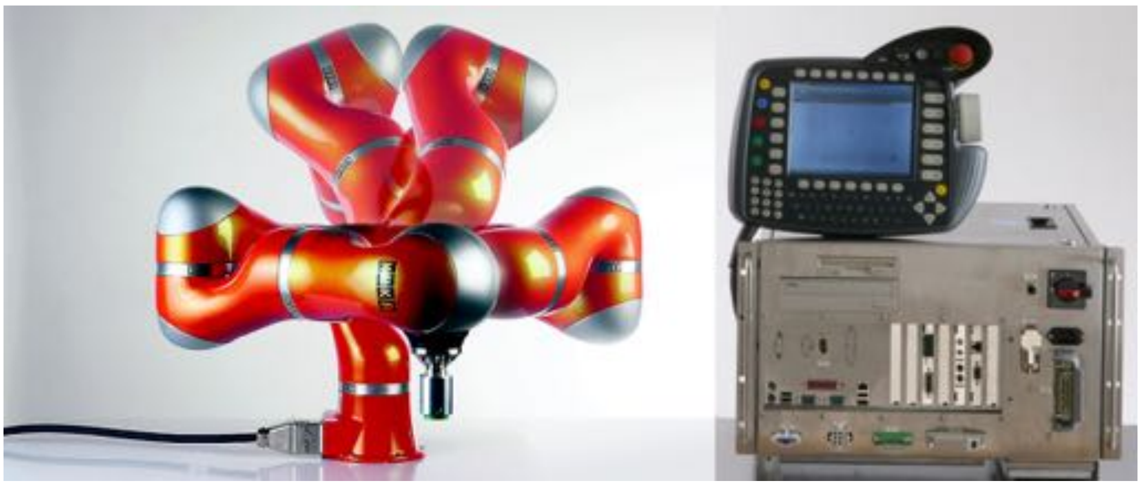


Figure 3.6: (left) KUKA LWR4+, (right) KRC controller

The robotic arm used for this project is a KUKA LWR4+ robot. This robot is a collaboration between DLR and KUKA to make a lightweight robot that has the capability of reacting compliantly to outside influences. Merging the best of both DLR and KUKA worlds to obtain the DLR Light Weight Robot (LWR) with the look and feel of an industrial robot.

The novel designs of this robot include the introduction of position sensors on both motor and output side of the joints, torque sensors on the output side of the joints and motor current measurement. Because of this amount of feedback data a compliant controller can be implemented which detect external torque which are not coming from the motors or gravity and react to them in a compliant way. The following text is from a KUKA LWR brochure.

“The aluminium housing of the LWR4+ accommodates the motors, gear units, brakes and sensors, as well as the necessary control and power electronics for 7 axes. The result is a powerful, streamlined robot with a payload capacity of 7kg, a compact footprint and minimized disruptive contours. Thanks to its integrated sensors, the LWR4+ is extremely sensitive. This makes it ideally suited to handle and assembly tasks. Its low weight of only 16kg makes the robot energy efficient and above all portable. The LWR4+ can thus be easily integrated into exsisting systems- either as a fixed installation, or mounted on a moving platform to enable it to perform different tasks at various locations. The LWR4+ is a pilot product for use in reasearch and industrial advance development.”

KRC

The Kuka Robot Controller (KRC) is the system by KUKA to control the LWR4+ robot. It uses a simple programming language to make industrial applications which are mainly fixed programs with occasional input. But the KRC also has a research interface with which the LWR4+ can be directly controlled, this interface is called the “Fast Research Interface” or FRI for short. This FRI interface will be the interface through which all the communication will be done to the robot in this project.

Fast Research Interface

First some terms and variables within the KRC are defined that are required to to understand the workings of the FRI connection.

The KRC knows two types of FRI modes.

- **FRI Monitor mode**: Cyclical communication with transfer of robot data to a remote host, the remote host exchanges data with the KRC but cannot directly the robot.
- **FRI Command mode**: Cyclical communication with transfer of robot data to a remote host, the remote host can control the robot in this mode.

To control the mode four commands are available to the KRC.

- **FRI Open** : Opens up the FRI connection by sending out the first UDP package to the remote host.
- **FRI Start** : Attempt to go into command mode but several checks need to be performed.
- **FRI Stop** : Stops the command mode and returns to monitor mode
- **FRI Close** : Closes the FRI connection from monitor mode.

The relation between these command events and the FRI modes can be written into a state chart as can be seen in figure 3.7. The

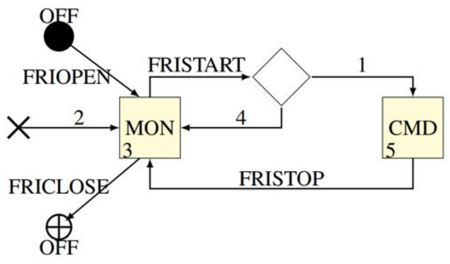


Figure 3.7: FRI modes transitions chart

The KRC has three different types of strategies to be controlled in while FRI is active.

- **Joint position control (code 10)**: In this mode the joints are controlled in joint space and the robot is acting as stiff as possible.
- **Cartesian impedance control (code 20)**: In this mode the KRC models a six dimensional spring between the setpoint and the end-effector resulting in a force-torque which accelerates the robot. This is the only cartesian mode because no cartesian position mode is available. With a high stiffness this mode approximates a position control but it has no internal velocity of torque limiting meaning that it will get into torque limit if the desired setpoint is too far away.
- **Joint impedance control (code 30)**: In this mode the joints are controlled by means of a 1 dimensional virtual spring in every joint

The Fast Research Interface is an interface that uses a standard ethernet connection with UDP packages for its communication. Because UDP is a connectionless protocol in which no guarantees are given to the if the packages are received or in what order they are received it is best to have the network over which the FRI is run as small as possible. When FRI is opened the KRC sends a UDP package is send to a “FRI-host” with a static IP address and port number which needs to reply with a UDP handshake. The connection always has to be established by the KRC and the frequency of the connection is determined when opening the FRI connection and can only be changed by restarting the connection. The frequency of the connection is between from 10 Hz to 1000Hz.

When the connection has been established UDP packages are periodically sent from the KRC unit to the remote host which replies to as fast as possible with UDP message. When the reply is not received before the next package is sent out it is considered lost and the quality of the connection is determined by the succes rate of the UPD package transfer. These packages from the KRC contain a complete set of robot control and status data. A reply message contains input data for the applied controllers.

When a FRI connection has been established the connection is considered to be in the “Monitor state” and the “FRI-hosts” exchanges information from the KRC but cannot control the robot. A important set of variables to set are the “command flags” which determine the content of the package that is send back to the KRC. When the robot needs to be controlled directly the KRC calls the command “FRI_START” which trigger safety checks which are followed by “Command mode” or a fallback to monitor mode. These requirements of the safety checks are: The connection quality needs to be at least “’good” quality, the commanded pose needs to identical to the measured pose, the drives of the robot must be engaged and several more requirements within the KRC that have to do with the estimation of position and the torques due gravity compensation. While in command mode the command-flags must be constant, the connection quality should stay at least good and no large discontinuities in commanded pose should occur. When one of these do occur the robot drives are immediately disengaged and the program on the KRC needs to be restarted. The command-flags are a set of booleans that define which variables the FRI wants to control, these should be set before command-mode is engaged and can only change by going back to monitor mode. For instance sending a stiffness to the robot while you don‘t have the commandflag for stiffness set or not sending a valid stiff while the commandflag isn‘t set will result in the robot given an error and disengaging the drivers.

3.2.2 UAV

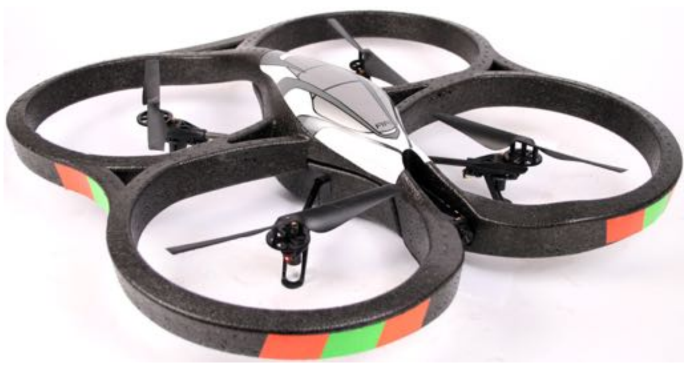


Figure 3.8: Parrot AR.Drone 2.0

The UAV used for this project is a Parrot AR.Drone. The Parrot AR.Drone is a wifi controlled flying quadcopter helicopter built by the French company Parrot. The drone is designed to be controlled by mobile or tablet operating systems such as the supported iOS or Android but also has open source API which can be used to interface the drone to other operating systems.

Figure 3.9: Xsense MTi IMU

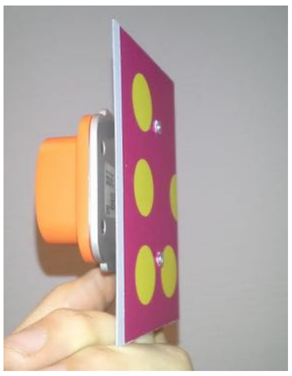
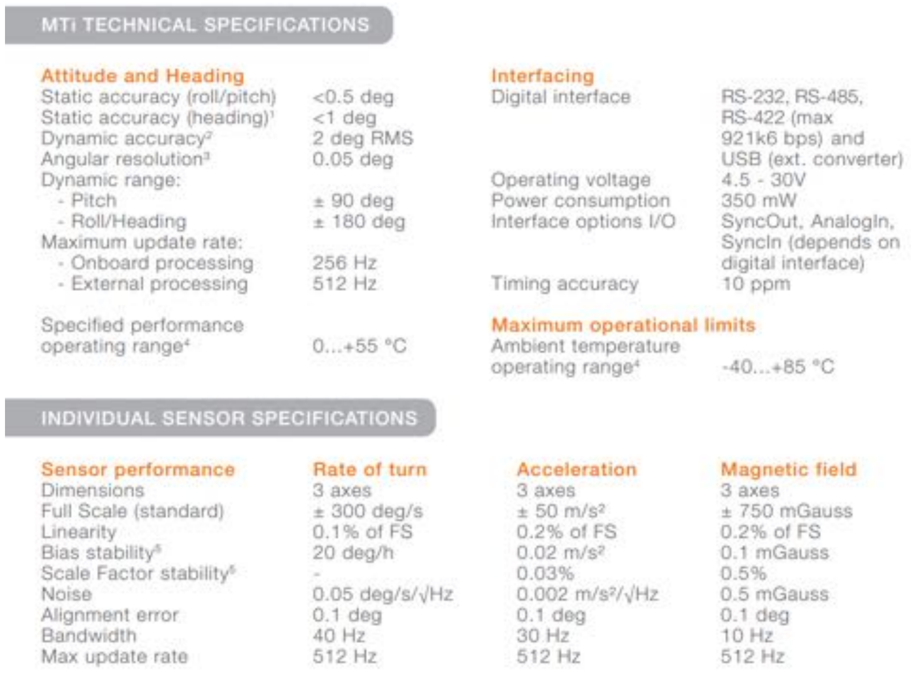


Figure 3.10: Marker plate with IMU backplate

3.2.3 Drone substitute : Xsens IMU

For experimentation a drone substitute was used. This drone substitute consisted of a marker plate with an separate IMU mounted on the back of the marker. This system provides all the required signals for the system and is much lighter and more manageable to maneuver by hand. The IMU used is a Xsense MTi-28A53G35 connected by USB to the system. The Xsense IMU is a sophisticated IMU which uses internal sensor fusion for the determination of its orientation and the compensation of accelerometer bias and gyroscope drift.

The technical specifications of the IMU according to the datasheet are shown in figure 3.11.


MTi TECHNICAL SPECIFICATIONS

Attitude and Heading		Interfacing	
Static accuracy (roll/pitch)	<0.5 deg	Digital interface	RS-232, RS-485, RS-422 (max 921k6 bps) and USB (ext. converter)
Static accuracy (heading)[1]	<1 deg		
Dynamic accuracy[2]	2 deg RMS		
Angular resolution[3]	0.05 deg		
Dynamic range:		Operating voltage	4.5 - 30V
- Pitch	± 90 deg	Power consumption	350 mW
- Roll/Heading	± 180 deg	Interface options I/O	SyncOut, AnalogIn, SyncIn (depends on digital interface)
Maximum update rate:			
- Onboard processing	256 Hz		
- External processing	512 Hz	Timing accuracy	10 ppm
Specified performance operating range[4]	0...+55 °C	**Maximum operational limits**	
		Ambient temperature operating range[4]	-40...+85 °C

INDIVIDUAL SENSOR SPECIFICATIONS

Sensor performance	Rate of turn	Acceleration	Magnetic field
Dimensions	3 axes	3 axes	3 axes
Full Scale (standard)	± 300 deg/s	± 50 m/s²	± 750 mGauss
Linearity	0.1% of FS	0.2% of FS	0.2% of FS
Bias stability[5]	20 deg/h	0.02 m/s²	0.1 mGauss
Scale Factor stability[5]	-	0.03%	0.5%
Noise	0.05 deg/s/√Hz	0.002 m/s²/√Hz	0.5 mGauss
Alignment error	0.1 deg	0.1 deg	0.1 deg
Bandwidth	40 Hz	30 Hz	10 Hz
Max update rate	512 Hz	512 Hz	512 Hz

Figure 3.11: MTi-28A53G35 Technical specification

Figure 3.12: Logitech C920 Webcam

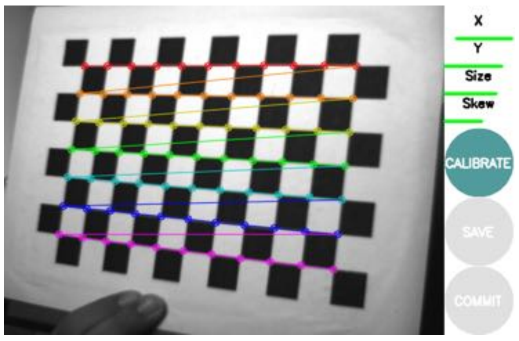


Figure 3.13: ROS camera calibration using checkerboard

3.2.4 Camera

The camera used in the realization is a Logitech HD Pro C920 webcam. This is a commercial computer product that is readily available and cheap compared to industrial cameras. The can capture a wide range of resolutions up to $1080p$ at $15fps$ and lower resolutions on $30fps$. The camera has a auto-focus option but this is always disabled in this project because it requires a camera calibration for every focus distance.

The camera is used to capture 720p at 30fps video and is calibrated using the built-in calibration tool in ROS with a checkerboard pattern.

The downsides to such a camera is that it uses a rolling shutter which means that it does not acquire all the image points at the same time but rather "scans" the image sensor top to bottom for data acquisition.

Magnetic grabber

A magnetic grabber camera mount has modeled in Solidworks and made using a 3D printer. The result can be seen in figure 3.14.

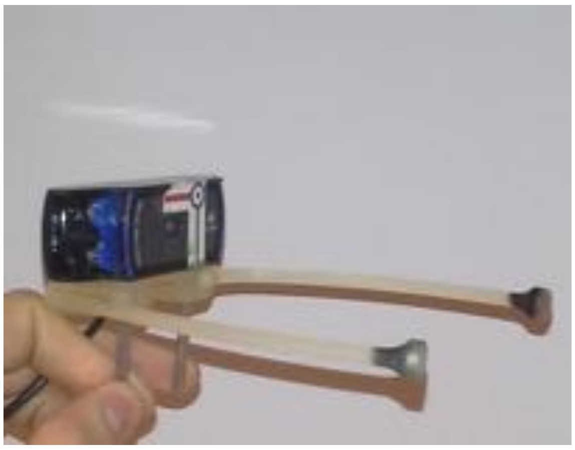


Figure 3.14: Camera mount and magnetic grabber

4 Results

4.1 Measurements

Experiments were conducted using the system described in the realization and implementation part of this report. The acquired data files of the system are coma separated files (.csv) which are processed in Matlab. Both the Parrot AR.Drone and Xsense drone substitute worked with the system but the system was more consistent and reliable using the Xsense drone substitute. For this reason quantitative experiments were performed using the Xsense drone substitute but a qualitative evaluation of the performance of the Parrot AR.Drone will be discussed.

4.1.1 Validation

The validation of the experiments will be done using a OptiTrack motion capture setup as mentioned in the software implementation section. This setup has 11 infrared camera's which use high power infrared LEDs to flood the room with infrared light. The system then detects infrared reflector markers that are attached to the rigid bodies which the user wants to track. A set of a minimum of three markers can be used to make a "trackable" within the tracking tool which is an object that has its relative pose with respect to the OptiTrack groundplane tracked at $100Hz$. The data of the OptriTrack system is send over a standard TCP/IP connection to the computer running the ROS system which translates it into a topic which can be captured by the for the logger. The base of the robotic arm is also captured by the Optitrack system because it is the reference frame for most of the frames in the system. The calibration of the OptiTrack system has a uncertainty of $0.4mm$ per marker.

4.1.2 Reproducibility

The shown measurements are performed by hand on a platform parallel to the ground. The patterns were made as consistent as possible but because of the lack of linear guides and fixed actuation there will be some variance between the measurements. The measured OptiTrack poses are used as reference to compare the performances of the system but also to compare the similarity of the marker movement.

For project continuation it should be mentioned that measuring with two KUKA LWR4+ robots, one holding the camera the other the Marker-IMU combination, was conducted but failed. The vibrations caused by the harmonic driver inside of the LWR4+ were so severe that the accelerations coming from the IMU were useless for measuring displacement.

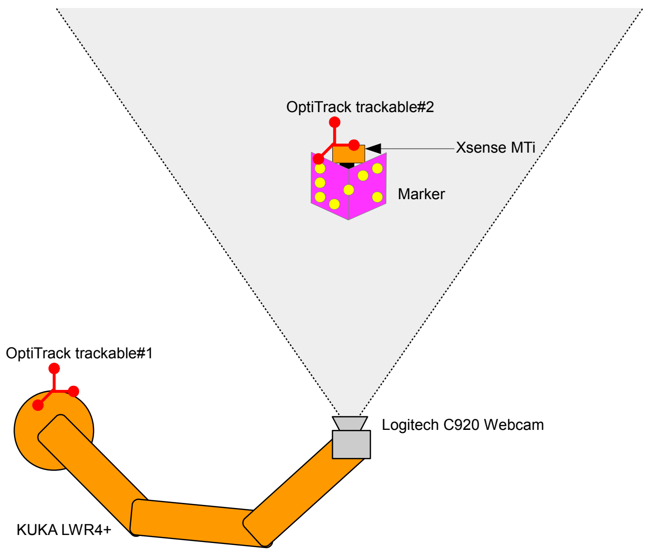


Figure 4.1: Experimental setup

4.2 Experiments with Xsense IMU drone substitute

4.2.1 General behavior impressions

The system with and without IMU assistance is able to keep track of the marker in normal circumstances. However pose estimation failure for short periods can leak to jagged movement, high acceleration and motion blur which itself also causes pose estimation failure.

With IMU assistance the system is also able able to follow the marker with temporary failure of the pose estimations due to occlusion, motion blur and response better to rapid acceleration. The estimation in case of (partial) occlusion is only good for short periods of time (1-2 seconds). After a few (3-5) seconds the exponential drift of the position due to error in acceleration measurements becomes significant and the system becomes unreliable. The reason for the drifting behavior is the drift in the bias and the error in orientation which is not included in the state estimator. The system is be configured to stop state estimation after a fixed number frames for which no correct pose estimation could be estimated to stop the robot drifting out of bounds. The system can stably follow the marker up a distance of ?? meters and down to a distance of 20cm.

The general impression is that the system works better with the assistance of IMU data by improving the response to rapid acceleration, motion blur, lighting imperfections and (partial) occlusion. The following experiments where conducted to examine the results of the system.

4.2.2 Movement in Y with occlusion

In this experiment the marker is moved in Y direction from point A to B and back is performed with the marker being occluded in between the points as illustrated in figure 4.2. The distance between the points is roughly 25cm and the velocity was kept as constant as possible. The experiment has been repeated with and without the assistance of the IMU data and state estimation.

Still images from camera during the measurements without IMU assistance are shown in figure 4.3. In this set of figures a shows the marker in position A, b the marker occluded by the object, c almost at point B and in d the image is blurred due to the movement of the camera to get to the desired position as quickly as possible.

The results of the experiments are shown in figure 4.4 with the results of going from A to B on the left and the results of going back on the right. The shown measurements from A to B and

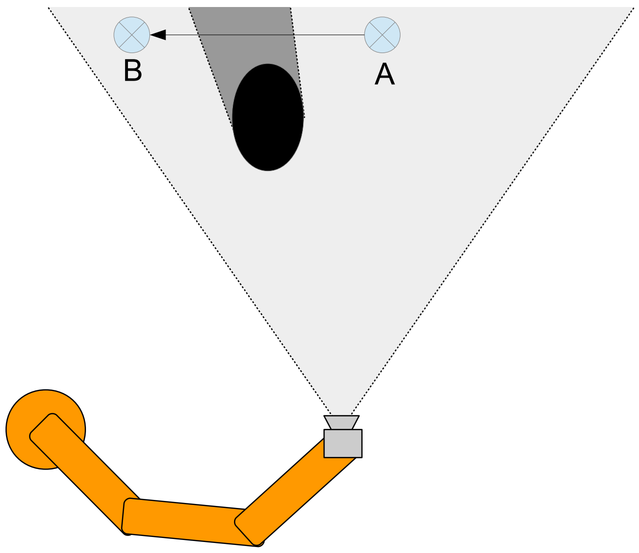


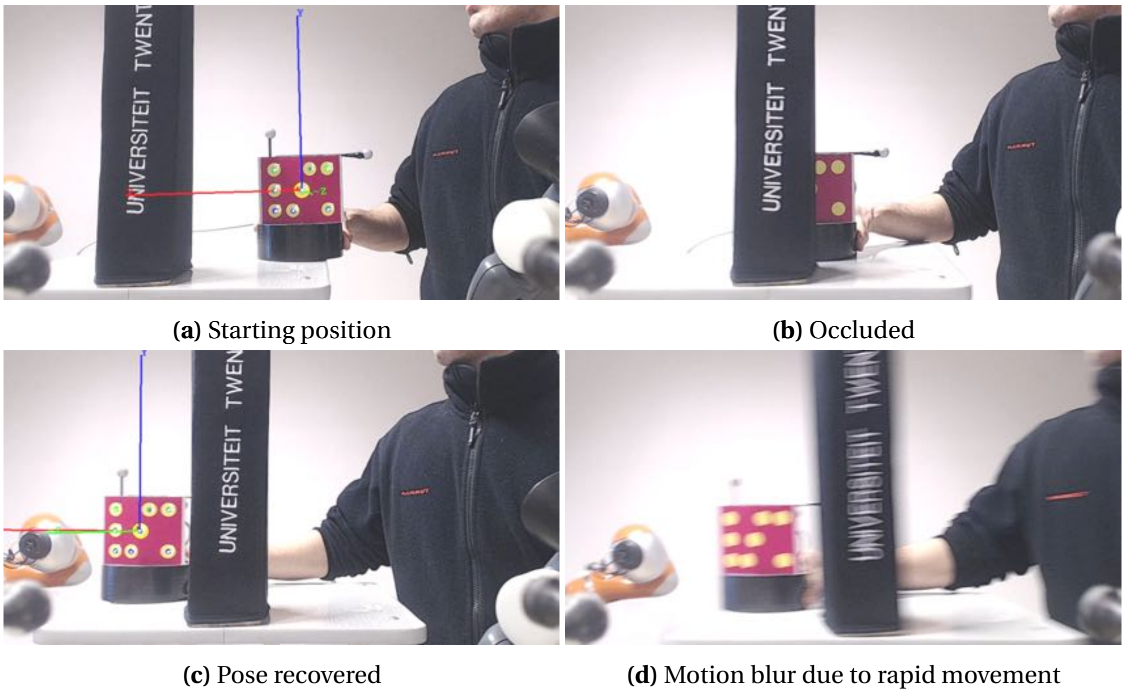
Figure 4.2: Measurement in Y direction using only pose estimation

back are recorded in one session but have been split to align the graphs such that the movement of the marker starts at the same time. The X and Y axis have also been aligned to make it easier to compare the results.

From the graphs going from A to B it can be seen that the camera delay of 150ms means that the pose estimation is slow to react to the movement of the marker while the state estimator responds due to the significant acceleration given by the IMU. Due to the low covariance of the camera the observed acceleration is damped but is still significant enough for the state estimate to increase in Y direction. At roughly 0.9 seconds the marker is occluded by the object which makes new pose estimation impossible and the system without IMU assistance stop at the last known pose set point. The measurement with IMU assistance continues the increase of Y because the IMU information is still updating. At roughly 1.7 seconds the pose estimation is recovery in the measurement with IMU assistance and adjusts the set-point to better match the pose estimate. The camera delay is still causing the set-point to lag behind the true position of the marker but the marker is normally followed now from this point until the settled Y position of the marker. The system without IMU assistance only finds the marker at 2 seconds and quickly tries to move towards the new set point which is more than 20cm away at this point. Immediately after the new set-point the pose estimation fails 6-8 times due to high velocity between 2.1 and 2.3 after which normal visual serving is resumed. On the movement back from B to A the same behavior is seen.

When not using IMU measurement it can be seen that when the pose estimate is not available the robot stops moving and rapidly accelerates when the pose estimation is restored if the desired position is a distance from the position it lost it . The results that do use the IMU measurements can follow the marker even when temporarily occluded. The performance of the system is similar when going in positive or negative Y direction.

To further examen the results figure 4.5a and 4.5b show the velocity profiles of the robotic arm and the relative position between the robotic arm and the marker. The velocity and acceleration of the robotic arm are limited and the redetetection of the marker leads to a large joint torques and should be avoided. The performance of the camera is also impaired by velocity and acceleration which is commonly known as motion blur which is the change of the scene during the exposure period. The absolute maximum speed, relative speed and accelerations during several of these measurements are shown in table 4.1.Due to the settings the rise and fall time differ insignificantly between the measumrents with and without IMU sensor fusion.



(a) Starting position **(b)** Occluded

(c) Pose recovered **(d)** Motion blur due to rapid movement

Figure 4.3: Experiment with only pose estimation

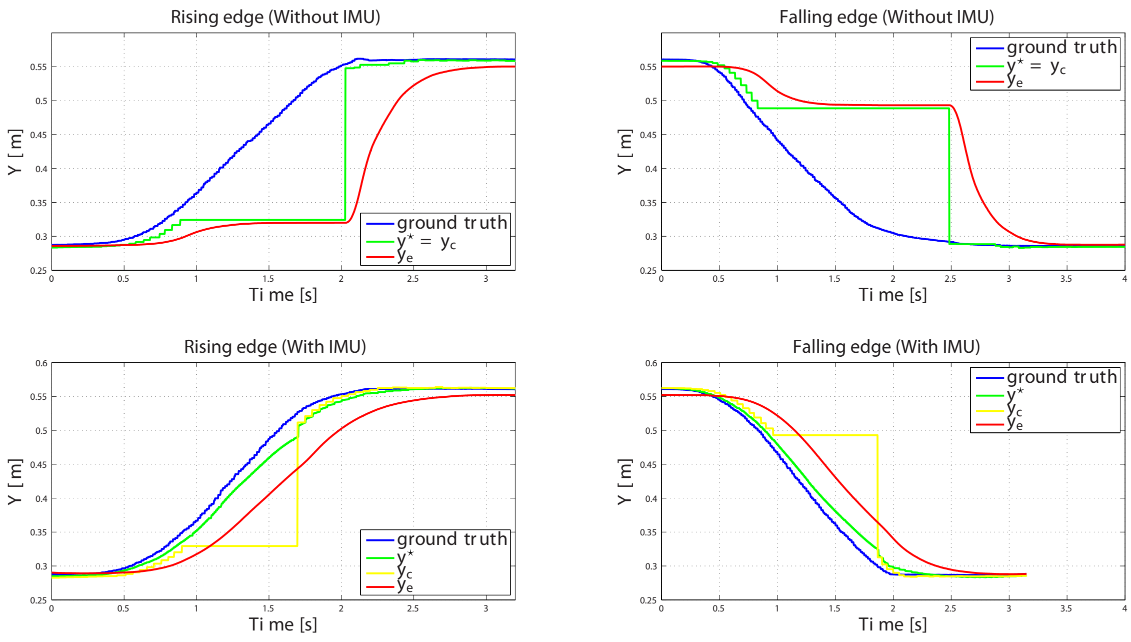
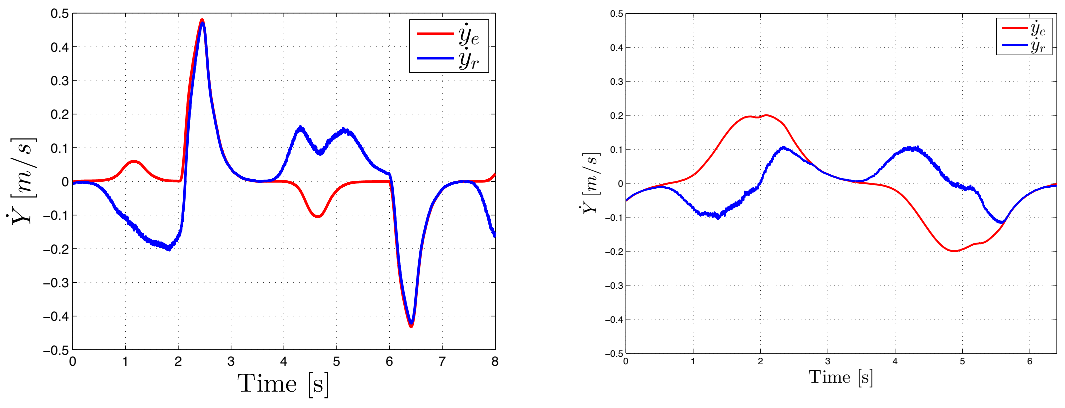


Figure 4.4: Measurement in Y direction

	Without IMU	With IMU
Maximum velocity	0.4934	0.2068
Maximum relative velocity	0.4711	0.1175
Maximum acceleration	1.2334	0.5170

Table 4.1: Table



(a) Without IMU assistance

(b) With IMU assistance

Figure 4.5: End-effector and relative camera-marker velocity

4.3 Experiments with Parrot AR.Drone

The measurement setup is shown in figure figure 4.1 using the Parrot AR.Drone as UAV.

4.3.1 Computational load

While running all the nodes required for the DUAVRACV system and communication to the AR.Drone it was noticed that the load on the computer system was very high. The FRI-connection frequently missed packages which caused the KRC to disengage the driver of the robot and the need to restart the experiment. The problem vanished while running at a FRI frequency of 200Hz instead of the regular 500Hz. The FRI node is always running at the highest priority in Linux but the node seems to be CPU starved for long enough in this configuration to not be able to respond to the FRI message within the required 2ms. This is an inherent problem with non-realtime operating systems that cannot guarantee CPU time or connection to the IO ports for crucial real-time programs.

A clear solution to this problem is to use a dedicated computer to run either the FRI-node or AR.Drone node

4.3.2 Qualitative results

The performance of the AR.Drone while having a good vision estimate is subjectively identical to the performance with the Xsense drone substitute. The performance while temporarily occluded seems not to be as good as with the Xsense drone substitute, the response to the acceleration seems slow and the total displacement while occluded is lower than the distanced traveled. This could be due to lag in the accelerometer or some fault in the calibration of the AR.Drone accelerometers. The node that is connected to the AR.Drone also sends all camera signals over the same WiFi connections as the IMU measurements which could be the cause of delay.

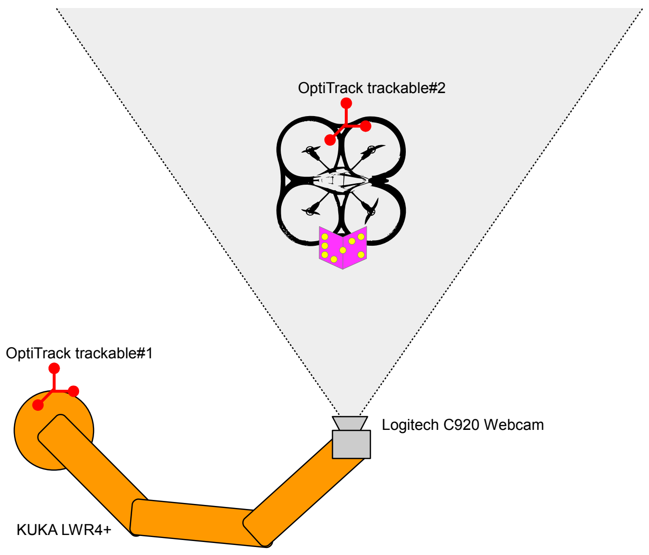


Figure 4.6: Experimental setup

5 Conclusions & recommendations

5.1 Conclusions

The proposed system works and can follow a drone-substitute within the reach of the robotic arm. The system works with and without the assistance of the IMU sensor but the robustness to rapid movement or occlusion is improved with the assistance of the IMU sensor. In experiments with occlusion the system with IMU assistance the robotic arm had a significantly lower peak velocity and acceleration due to the fact that the robot did not make a large jump in setpoint. Uncertainty in the orientation and acceleration bias of the IMU cause the estimated pose of the system to drift without visual pose estimation which makes the estimate unreliable after approximately 3 seconds.

Docking a marker was only achieved if the marker was stationary because the procedure is susceptible to motion due to the small margin for error in the magnetic docking mechanism.

The system was successfully implemented in ROS, which provided a solid framework for development and multi-threaded programming. Concerns regarding real-time performance were raised with the FRI interface failing to respond within $2ms$ in combination with the AR.Drone controller node.

5.2 Recommendations

For continuation of this project the following recommendations are given.

- Redesigning the extended kalman filter to include the orientation and angular velocity might increase the stability of the system.
- Use different drone to test the system, which might work better than the AR.Drone.
- Increase the size of the area that can be docked using a magnetic grabber to allow for more fault tolerance then 1 centimeter.
- Use a horizontal or ceiling mounted robot arm setup to be able to grab te top of the drone.

Bibliography

[1] M. A. Rahman, "Sherpa: An air-ground wireless network for communicating human and robots to improve the rescuing activities in alpine environments," in *Ad-hoc Networks and Wireless*, pp. 290–302, Springer, 2014.

[2] SHERPA, "Sherpa project website," 2015.

[3] M. Achtelik, T. Zhang, K. Kuhnlenz, and M. Buss, "Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors," in *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, pp. 2863–2869, IEEE, 2009.

[4] K. Schwabe, "A monocular pose estimation system based on infrared leds," Master's thesis, Zurich, 2013.

[5] T. Petersen, "A comparison of 2d-3d pose estimation methods," *Master's thesis, Aalborg University-Institute for Media Technology Computer Vision and Graphics, Lautrupvang*, vol. 15, p. 2750, 9999.

[6] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[7] G. Schweighofer and A. Pinz, "Robust pose estimation from a planar target," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, pp. 2024–2030, Dec 2006.

[8] G. Welch and G. Bishop, "An introduction to the kalman filter," 2006.

[9] O. S. R. Foundation, "Robotic operating system website," 2015.

[10] Boost, "Thread safety guarantees," 2015.