POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

MASTER THESIS

Sensor Fusion for Mobile Manipulators

KUKA

Advisor:
prof. Michele Taragna

Candidate:
Barnaba Ubezio
Student Number: 236054

Company Supervisor
KUKA Deutschland GmbH
M.Sc Shashank Sharma
Guglielmo Van der Meer

December 2018

Embargo notice

This final project contains confidential data from KUKA AG and all associated companies according to Section 15 ff of the German Stock Corporation Act. At the request of KUKA AG and all associated companies according to Section 15 ff of the German Stock Corporation Act, this final project shall be embargoed from public use for a period of 3 years.

Augsburg, 31/10/2018 Barnaba Ubezio

For any request please contact:

M.Sc. Shashank Sharma
KUKA Deutschland GmbH
Corporate Research
Blücherstraße 144
86165 Augsburg
Tel. +49 821 797 - 3549
Fax. +49 821 797 - 41 - 3549
Shashank.Sharma@kuka.com

Summary

The end-effector tracking for a mobile manipulator is achieved through Sensor Fusion techniques, implemented with a visual-inertial sensor suite and an Extended Kalman Filter. After a study on the suitable noise model for the sensors, a simulation environment is created in order to test the improvement with respect to standard approaches. The Filter is built in such a way that its complexity remains constant and independent from the visual algorithm, with the possibility to insert additional sensors, to further improve the estimation. Precise and robust tracking in real-time is obtained in a real world hardware implementation with the 12-DOF KUKA VALERI robot. Along with the physical results, issues related to calibration, working frequency, networking and physical mounting are described.

Acknowledgements

This work is the result of several months of research, during which many people had an impact on the overall experience.
I would like to thank my supervisors **Shashank Sharma** and **Guglielmo Van der Meer**, for helping me throughout the project and for teaching me all kind of things, that I know will be very useful for my future.
My gratitude goes also to all the Corporate Research department at KUKA, to my fellow students in Augsburg and to the employees in the laboratory, from which I gained knowledge about different engineering topics every day. A special mention goes to **Kiril Safronov**, for the extra help in implementing the Hand-Eye Calibration.

Thanks to professor **Michele Taragna** for his work as Advisor and for his lectures, that set a strong foundation for this work.

Thanks to **Adriana**, for the support, the patience and the presence.
Finally, my deepest gratitude to my Family, **Laura** and **Graziella**, for the inspiration and motivation that they have always provided.

Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile manipulation is one of the new arising goals pursued by companies operating in advanced industrial environments.

The simple idea of combining mobile robots with robotic arms having 6 or more degrees of freedom (DOF) opens up a wide set of applications; the working range of the end-effector is expanded, making the robot able to reach higher targets with respect to the ground, or to perform tasks in a wider Cartesian space around the robot itself. Moreover, this type of manipulators is not ground-fixed or limited in any direction, as often it is the case with robotic arms, allowing now the completion of industrial tasks (welding, gluing, painting...) with bigger surfaces and working pieces. Clearly, every extra DOF increases the complexity of several common algorithms in robotics, as well as the hardware needed to build, sense and move the robot; this means that the number of error sources increases as well, and somewhat simple tasks can easily become troublesome.

Size and cost of these robots can vary quite a lot, depending on the application for which they are intended; *KUKA Deutschland GmbH*, the german company where this thesis project has been carried on, developed two very different products in this sense: the KUKA youBot and the KUKA KMR iiwa, both shown in Figure 1.1. For many real industrial tasks, the second one is more versatile, having a 7 DOFs arm with high-performance torque sensors at each joint, an autonomous mobile platform and a powerful embedded KUKA Sunrise Controller. The work presented in this thesis has been created for a research robot coming from the same "family" as the KMR iiwa, but with two extra DOFs: the VALERI.

(a) KUKA youBot (b) KUKA KMR iiwa

Figure 1.1: Examples of mobile manipulators.

1.1 The VALERI project

An example of environment where mobile manipulation could be well exploited, still in the perspective of human-robot collaboration, is for aerospace manufacturing. It is in this field that the VALERI Project began in November 2012, initiated by the *Fraunhofer Institute for Factory Operation and Automation (IFF)* and carried out by several companies such Airbus SE and KUKA. The goal was to build a mobile manipulator able to help workers and constructors in operations with airplanes' parts, improving the speed of the process and allowing the completion of those jobs considered too difficult, dangerous or even tedious and repetitive.

The VALERI robot shown in Figure 1.2, result of the first two years of research, is a three-part structure derived by the KMR and composed by:

1. Autonomous mobile platform equipped with four KUKA Mecanum omni-wheels for omnidirectional drive and two laser scanner sensors for collision avoidance.

2. KUKA LBR4 robotic arm with 7 DOFs and a 7 kg payload, for a weight of approximately 16 kg.

3. Extra linear and rotational axes provided by a rotating vertical lift, on which the arm is attached.

The whole system weights around 270 kg and has a total of 12 DOFs: 3 given by the platform, 2 by the lift and 7 by the arm. Since every possible task in space needs at most

Figure 1.2: The VALERI OmniRob with a representation of the increased working space.

6 DOFs, this robot implements a high redundancy, crucial to have enough dexterity and to allow the execution of multiple tasks simultaneously.

The lift unit is probably the main feature that specializes the VALERI but, as it will be clear shortly, it is also an important source of errors. It allows the arm to move up and down in a range of almost 1 m (plus the 0.83 m of the platform), and to rotate from about -150° to 120° with respect to the origin position shown in Figure 1.3 along with other dimensional data; to do these motions, it encorporates two motors with respective gearboxes. The translational axis is particularly critical from the accuracy point of view, due to its mechanical structure and components, very far from ideal: springs and gears imperfections, unmodeled frictions and aging of components impact heavily on the performance of this device. The friction issue has been recently treated in another research project here at KUKA, but will not be covered in this thesis.

The arm by itself, instead, can surely be considered more precise; the internal sensors can provide a considerably accurate measurement of the angles at each of the seven revolute joints, and the repeatability, i.e., the ability to replicate previously reached configurations, is less than 0.5 mm. The particular construction of the end-effector allows the connection

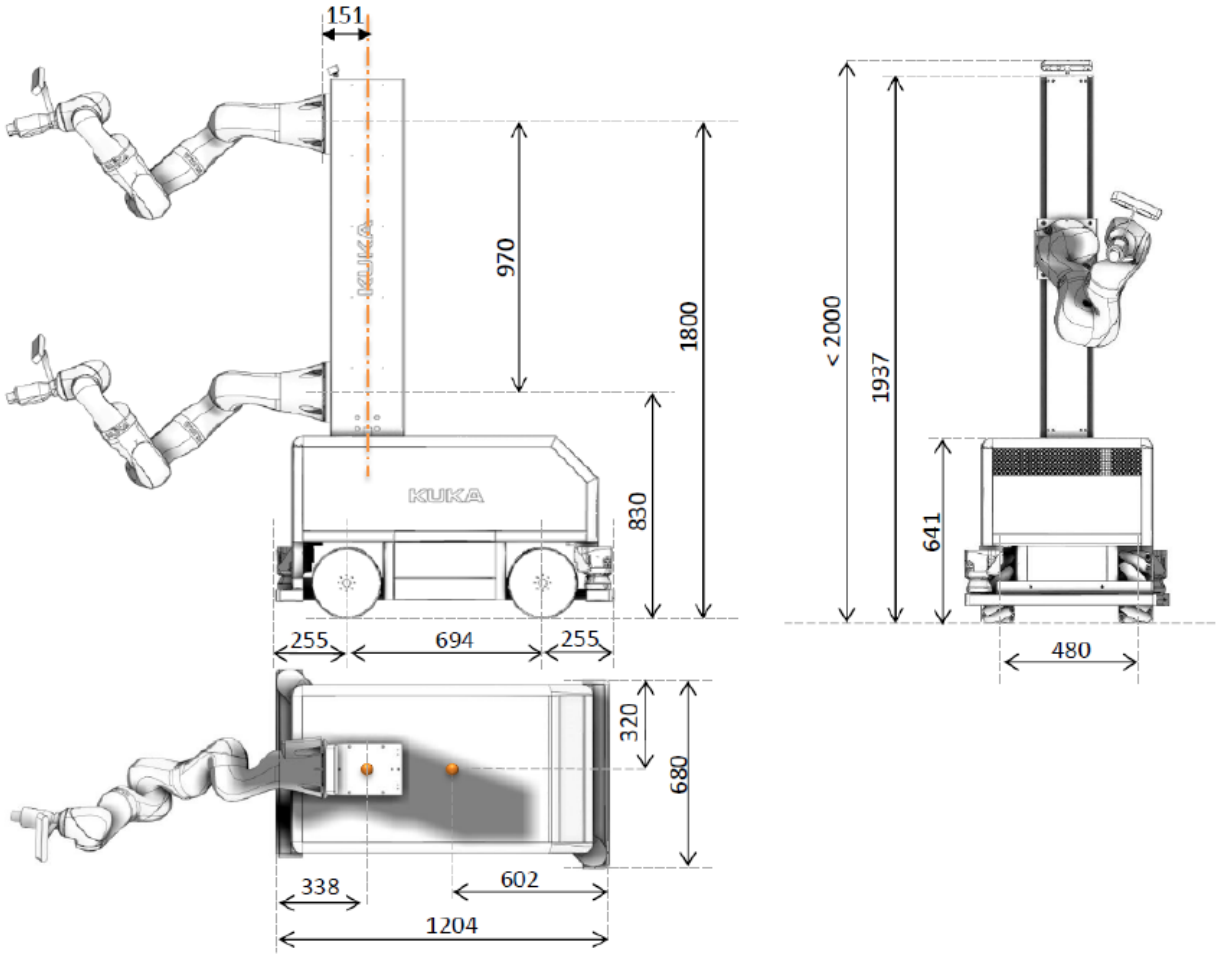


Figure 1.3: Dimensions of the entire VALERI OmniRob in mm.

with ideally any possible kind of tool, and extra electrical pins have been added to exploit internal connections embedded in the arm.

Issues related to the non-ideality of the mobile platform and to its particular wheels, as well as the improvements required in the arm end-effector/tool pose estimation, will be covered more in detail in the next section of this chapter, where the starting idea behind this thesis project will be explained.

1.2 Motivation

Since the beginning of the construction, the main tasks programmed for the VALERI have been sealant application along a groove and parts' inspection. The amount of precision required calls for the best possible localization of the end-effector of the arm, where the gluing tool is applied. In the first version, this robot could perform those tasks, with some

constraints and with not so much accuracy (drifting on several cm when following a straight line back and forth), also due to excessive unfiltered vibrations. A way to improve the *pose* (position and orientation) estimation of the robot's tool has been searched, and that is when the idea of using Sensor Fusion came. In order to understand this, it is important to start from the already mentioned concept of localization.
Localization, and the naturally following trajectory planning, are some of the most important and studied tasks in robotics. For robotic arms, it is well known that given the measured joint angles it is possible to retrieve the pose of the end-effector via the *Forward Kinematics* (FK) algorithm. The LBR4 arm mounted on the VALERI can obviously perform this computation in a quite precise way, but the whole system is still subject to vibrations and unmodeled elastic components that introduce errors. More details about the FK algorithm and its use with the VALERI can be found in Appendix A.

Now, the normal flow to be followed would be to first localize the mobile platform with odometry measurements (with respect to a fixed reference frame), then apply the necessary homogeneus transformations to get to the arm base, then apply the FK to get to the end-effector frame and finally get to the tool application point via some fixed transformation.
In this process, problems arise when considering that the base of the arm on the lift, where the FK starts, is not localized with the same precision as in the arm only: this type of pose estimation is in fact subject to different kinds of errors due for example to the not optimized mechanical components of the lift already mentioned, or to the integration of the velocity measurements at the wheels, where the odometry measurement starts. The odometry is based on rotary encoders present in the wheels, measuring their angular position; here drift errors become evident due to the omni-wheels, characterized by a large slippage of their contact point with respect to the ground.

All of these error sources lead then to the idea of operating directly at the end-effector, attaching one or more sensors and retrieving its pose without passing through all the previously described steps. Once these data are available, they can be used in a feedback control system to calculate the error between a desired reference trajectory and the actual pose of the tool. The error could then be used by a sort of Joint Mapper algorithm, based on the *Inverse Kinematics* (IK), to adjust the joint angles in order to follow more precisely the desired trajectory at every step.

The pose estimation part will be the crucial goal leading this project: precisely localize the end-effector in space, cutting as much as possible the noise coming from the sensors' measurements.
A number of considerations is mandatory here, in order to understand the importance of proper modeling and testing of the system:

1. Even if directly related to the end-effector or to the *Tool Center Point* (TCP), every sensor might still give too bad measurements, completely wrong or even missing data. All of these uncertainties must be taken into account, and robustness checks should be implemented.

2. Different sensors work at different frequencies. This means that each of them provides new values at its own rate, leading to certain periods of time where the behaviour of the transmitted data may be unpredictable or very different from sensor to sensor.

3. When dealing with real-time systems, everything is sampled with the controller frequency. Usually this is higher than the one of any of the sensors involved, meaning that errors in synchronization could arise periodically.

These issues can be treated by choosing the appropriate sensors, the appropriate noise model and the appropriate filter algorithm; in this thesis work, these choices will lead to the use of an Inertial/Visual Sensor Suite, Gauss-Markov processes and Extended Kalman Filter respectively. These three elements combine together in the concept of Sensor Fusion, i.e., the integration of all the measurements obtained at different time instants, in an unified and faster control frequency.

1.3 Contribution and structure of the thesis

In the previous section, the steps to be followed have been defined. This thesis provides a dedicated section or chapter for every step , plus a detailed research in the literature regarding all the covered topics, the description of a very useful simulation environment created ad-hoc in Matlab/Simulink and a report of results and considerations regarding the hardware implementation in the real robot, as well as the issues that can be encountered when working in similar projects.
The content of this work provides a double purpose contribution, not only as a master thesis

work but also as a set of mathematical models, codes and schemes useful for employees operating in the company. In the *Research & Development* Department of KUKA Deutschland GmbH, people working with the VALERI project itself, with mobile manipulation, sensors, navigation, high-precision tasks etc. can retrieve useful information from this manuscript. Due to company policies, no code whatsoever can be reported; nevertheless, the following contents are provided:

- Survey of the current state of the art in sensor fusion, filtering algorithm and robot navigation in general, leading to the choice of the sensors → **Chapter 2**
- Description of the sensors and measurement noise model → **Chapter 3**
- Step-by-step construction of the Extended Kalman Filter → **Chapter 4**
- Software environment, offline simulation and related issues → **Chapter 5**
- Description of the hardware setup and related issues → **Chapter 6**
- Experimental results → **Chapter 7**
- Final considerations and possible further research discussion → **Chapter 8**

In addition to this, the reader will find an explanation of the most relevant mathematical functions and algorithms used throughout the thesis in **Appendix A**.
Sensor Fusion and Kalman Filters are well documented and researched topics, so that some standard procedures and algorithms have already been defined. Nevertheless, in this thesis there are further improvements taken from external literature and added here, that will be highlighted along the way.
Even though the VALERI has been the target for this thesis work, the implemented algorithms and the obtained results are generally valid and extendable to every type of robot where sensor fusion is required.

Chapter 2

State of the Art

The research in Sensor Fusion as a general field is advancing constantly, but finds its application mostly in navigation fields. Unmanned Aerial Vehicles (UAV), Automated Guided Vehicles (AGV) or Teleoperating Space Robots are examples of applications, but perhaps the most evident is now Autonomous Driving cars, a sector where all the major car companies in the world are investing more and more.

Companies like Amazon.com Inc. or Swisslog Holding AG are instead interested in robot navigation for logistic solutions in warehouses or distribution centers; it is well known how Amazon, for example, has a fleet of autonomous mobile robots that move along designated paths to pick up and transport goods, faster and more efficiently than any human.

Autonomous navigation tasks like that are usually performed by building a map of the environment where the body will move (Figure 2.1), either a priori or in real time, measurement after measurement. A widely adopted method is the *SLAM* algorithm (Simultaneous Localization And Mapping), where a camera is used to collect a set of reference points in the space, the so called *features*, in order to build a map of the surrounding environment.

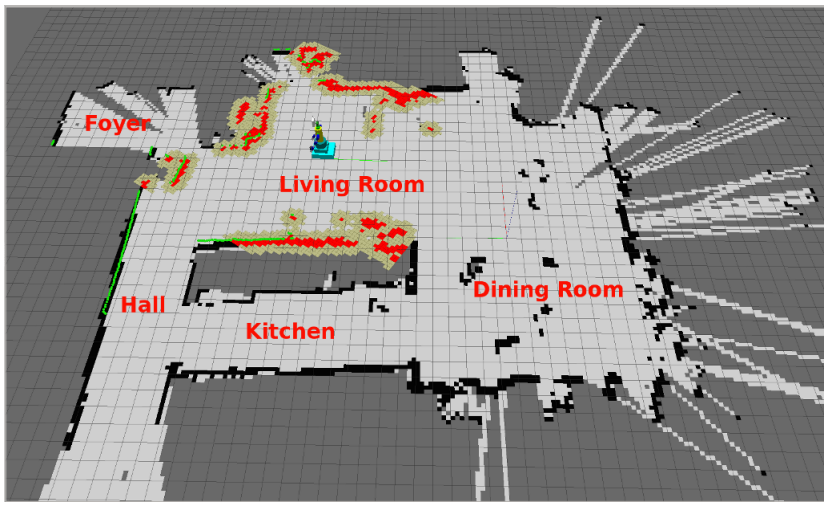


Figure 2.1: Example of an indoor map created with SLAM.

In [1] and [2] for example, the authors present a SLAM algorithm using visual and laser sensors. A disadvantage of such an approach is the computational effort required by calculators, that grows exponentially with the number of features.
Mapless approaches are also used when the robot or the vehicle does not have a pre-defined path to follow as in the case of this thesis. Nevertheless, the visual algorithms need very often difficult computations, as well as the right hardware available with precise information on how to calibrate the sensors, model their noise and retrieve a pose measurements.
Reference [3] gives a very well organized overview on SLAM, its main derived versions and Visual Odometry, i.e., the process of estimating relative motions by analyzing a sequence of camera images. Both monocular and stereo cameras, as well as other visual or distance sensors have been used as main devices in localization (e.g. [4], [5]), but often one or more other sensors are paired to improve the robot's state estimation. For example, in [6] the authors use a fish-eye camera along with an *Inertial Sensor System* (INS). Cameras' updates, while maybe more precise in measurements with respect to other devices, can in fact be very slow and may fail in providing real time information.

A reasonable goal for the VALERI would be the independece of the localization from its particular movements, or from the environment settings, without the excessive computational effort that would make impossible the real time control.
Here the main idea behind Sensor Fusion comes into play: use a fast enough sensor to get frequent pose estimates, and "fuse" some other slower sensor(s) to add periodic information updates. With this in mind, in the next sections considerations leading to the choice of the main sensor suite and the appropriate filter algorithm are reported.

2.1 Choice of the sensors

With a structure like the one of the VALERI, with enough space and mounting surfaces, many choices are possible for what concerns the sensors that will monitor the tool movement. One first common denominator in the consulted papers is the use of GPS sensors. For example in [7] the pair INS/GPS is exploited to build a robust localization system for autonomous land vehicles. In [8] the authors propose a method to estimate orientation and velocity for an electric vehicle, dealing with the slow GPS updates (1-10 Hz versus a 1 kHz controller) by incorporating quite a big number of dynamic sensors. While fault detection of the GPS

signal has been investigated thoroughly (e.g. in [9] and [10]), the small precision in the measurements is not feasible for the particular application researched. Furthermore, it cannot be completely assured that VALERI's operations will always be performed in an open environment, where the GPS system could properly receive signals.
Starting from the GPS/INS core in [7], one can observe how an inertial sensor, such as an **Inertial Measurement Unit (IMU)**, could be generally paired to another pose estimator, such the GPS, to compensate for its drift. Divergence for long periods of measurement time is, in fact, an infamous characteristic of IMUs. For different reasons though, like the normally small cost, small size, high working frequency and independence from illumination and visual occlusion, inertial sensors can be abundantly found in sensor fusion applications and literature from all over the world ([11], [12], [13] and [14]). Given this popularity of IMU sensors for localization, the Corporate Research department at KUKA decided to proceed in their project with the purchase of a high performance IMU from the american company **Honeywell**, specialized in aerospace applications.
This is now the first sensor in the suite; its specifications, description and modeling will be covered in the next chapters.

For what concerns the other sensor, the correction one, a particular decision path was followed. Since the goal is to track the tool position at the end of a robotic arm, an estimate of its pose can be obtained in the first place by the FK algorithm, as already mentioned. Taking into account the considerations made before, the situation seems already perfect: a direct pose is provided, without the need to buy any other external device, and its updates could actually come at a frequency higher than the one of the IMU! This is possible thanks to the very fast internal torque sensors. Actually, if only the robotic arm is used, then there would be *theoretically* no need to use any IMU or other sensor whatsoever.
Experimental results with the VALERI showed how, in reality, the first part of the kinematic chain (from the omni-wheels to the arm's base) suffers from error accumulation due to the mechanical structure of the robot. For example, physical offsets between some plates, unmodeled in the algorithms, lead to errors in position of several cm. The LBR4 arm itself is also outdated now in KUKA, due to the lower precision with respect to other models.
To exploit the full potential of such a mobile manipulator, it has been decided to add the other most used sensors in localization: cameras.
In order to avoid the same problems faced before, when talking about SLAM and Visual

Odometry, it has been decided to look for a visual sensor that could directy give pose information. This idea lead to the choice of an **Optitrack** Motion Capture system, comprised of 12 cameras, tracking a certain number of markers placed on the rigid body of interest. The whole system is of the type used also for Human Motion Capture, shown in Figure 2.2, in movies or games industries. With respect to normal stereo or monocular cameras, the Optitrack system does not work with images or SLAM, but provides a direct pose of a rigid body created via reflective markers. As for the IMU, its description is reported in the next chapters.

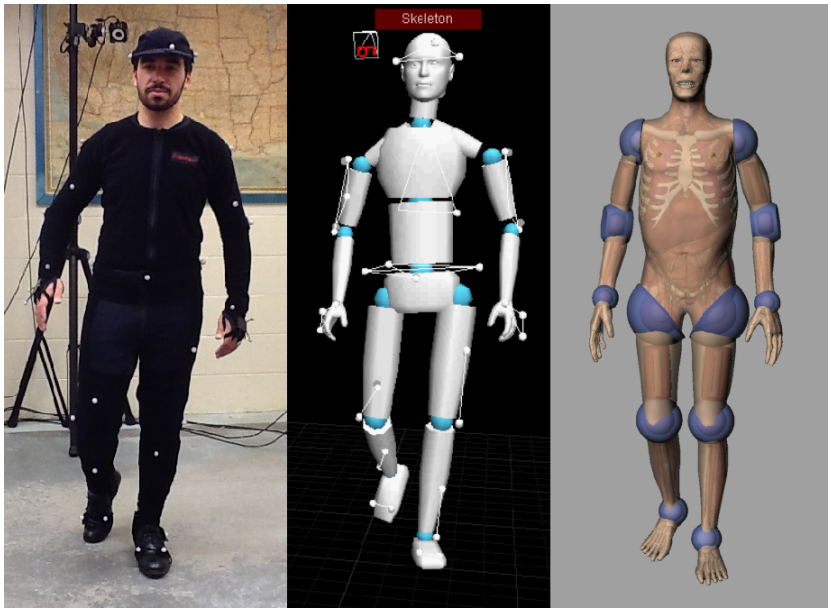


Figure 2.2: Human Motion Capture. The subject is filled with reflective markers from which a skeleton is created and then modeled.

The sensor suite is now complete with actually three measurement sources: an IMU, the Optitrack system and the combination of torque sensors and encoders in the robot. How and when to use them, to perform sensor fusion, will be the topic of Chapter 3 and 4.

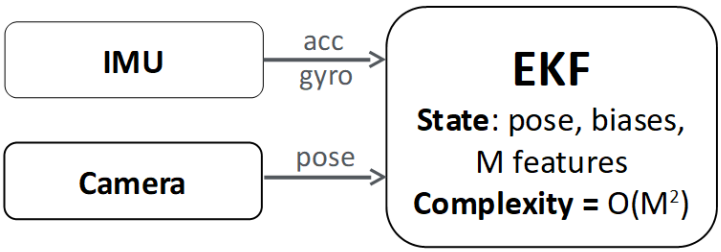
2.2 Choice of the filter algorithm

For what concerns the algorithm used to fuse the measurements and to account for uncertainties and frequency differences, the choice is almost entirely directed towards the *Kalman Filter*. Its capability of estimating a set of variables, the so called *state*, accounting for the errors at the inputs, is the perfect tool for sensor fusion applications.
The only distinction to be done is in the particular version needed. A simple Linear Kalman

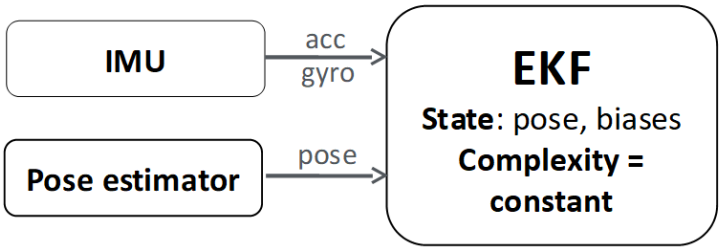
Filter, the standard one, is not applicable for most of the real world systems, often nonlinear. An example of nonlinearity is when the tracking of orientation is made using quaternions, as in the case of the majority of high-precision applications.

The **Extended Kalman Filter** (EKF) is currently the most used alternative to overcome this problem, thanks to a Taylor series linearization around the current state. EKF convergence to the right solution can become too slow when the system is highly nonlinear. An alternative to this is the *Unscented Kalman Filter* (UKF) [15], which provides a better modeling of the nonlinearity at the expense of a higher computational cost. Other less known versions of the Kalman Filter can be found in [3].

The need to save computational resources for other simultaneous operations in the VALERI, as well as the higher availability of literature, lead to the choice of using the EKF.



(a) State of the art version



(b) Proposed solution in this project

Figure 2.3: Comparison between the way the EKF is being used with visual sensor and SLAM algorithm, as in most applications, and the way it is used here.

Examples on its creation and use in visual-inertial sensor suites can be found in [16], [17], [18] and [19]. These papers provided a very good basics for the whole thesis, given their modeling and filtering approach very similar to what has been done here, despite their precision results, still too low (in the order of cm in position) for the goal fixed for the VALERI. Following these references, a constant complexity filter can be implemented as shown in Figure 2.3. In fact, in [17], while still using a sensor pair made with an IMU and a monocular

camera working with the SLAM algorithm, the authors can estimate a MAV's pose independently on the number of features collected.

The Optitrack system used in this thesis does not rely on such difficult or computationally expensive methods to provide a pose. Nevertheless, it has been decided to follow the same approach, in order to open the possibility to add other sensors if needed. In general, with this framework it will be possible to choose which and how many sensors to use, with or without SLAM.

Chapter 3

Sensors Modeling

Chapter 2 showed how an IMU can be used as the main device in a sensor suite, given its high update frequency. The specific IMU sensor used in this project is a **Honeywell HG4930 MEMS IMU**, a mid-high price device highly performant in terms of speed and noise; it has been mounted on the flange of the robot, so that from its measurements it is possible to retrieve its own pose and, via some fixed homogeneous transformation, the one of the end-effector or the one of the TCP. Some specifications, data and setup considerations are reported in Chapter 6.
A first suite with two sensors will be analyzed. Every device, but particularly the IMU, will have to be modeled, describing the way in which the measurement data are treated and providing a mathematical model of the measurement noise.
The following sections serve to this purpose, the first including also a brief explanation on how an IMU works. For the second sensor instead, it will be considered as a simple pose estimator, since both the FK and the Cameras will provide a direct pose in 6D.

3.1 Inertial Measurement Unit

A simple graphic explanation of the working principle of an IMU is given in Figure 3.1, taken from [12]. This device provides measurements of its own angular velocity and linear acceleration in three dimensions. To do this, most of the IMU packages are composed by a tri-axial gyroscope and a tri-axial accelerometer, while some devices include also a tri-axial magnetometer. Since the available sensor does not include this last device, its explanation and modeling will not be covered.
In usual applications, one can attach the IMU on the rigid body to be tracked. Illustrated in Figure 3.1 is the so called dead-reckoning process, i.e., the estimation of position and

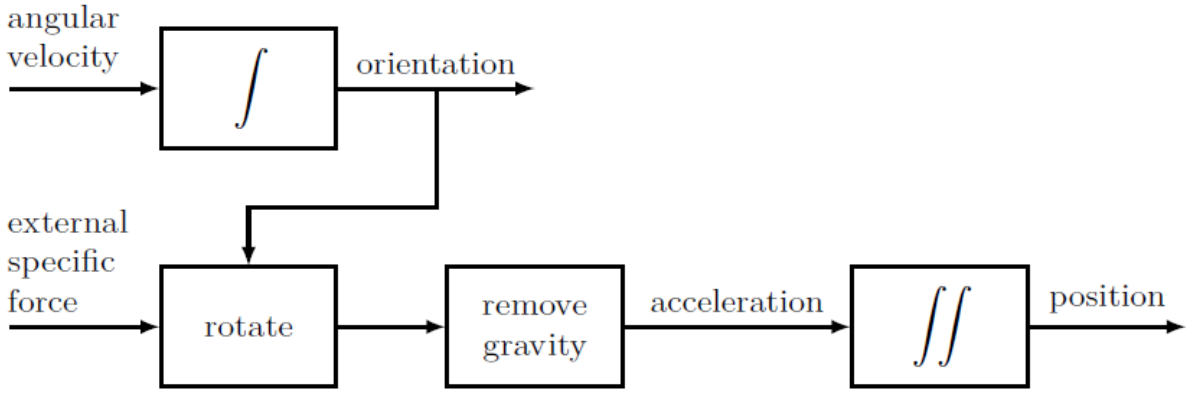


Figure 3.1: Schematic of the dead-reckoning process. Note how the acceleration comes from a force measurement.

orientation of a body in motion, step after step, using the previously reached configuration and the current inertial sensor's measurements.
Acceleration is integrated once to get the velocity, and twice to get the position. This is a first relevant source of numerical errors; in discrete time, the normal way to get the position at the k-th time step would be:

$$v_k = v_{k-1} + \boldsymbol{a_k}\Delta T$$

$$p_k = p_{k-1} + v_k \Delta T$$

where $\boldsymbol{a_k}$ is the measured acceleration at time step k.
In this thesis it has been used a more powerful algorithm, not present in any of the paper consulted, the *Verlet* integration:

$$v_k = v_{k-1} + \frac{1}{2}(a_k + a_{k-1})\Delta T \tag{3.1}$$

$$p_k = p_{k-1} + v_k \Delta T + \frac{1}{2} a_k \Delta T^2 \tag{3.2}$$

The double integration divergency is one of the major problems that make the accelerometer a "bad" sensor that, for this reason, is almost never used alone.

In modern IMUs the tri-axial sensors are MEMS (Micro Electro Mechanical Systems) devices. A MEMS gyroscope exploits the Coriolis Effect, graphically explained in Figure, 3.2, to measure angular rates. When a mass m is moving in direction v and angular rotation velocity Ω is applied, then the mass will experience a force in the direction of the yellow arrow, as a result of the Coriolis force; this force causes a displacement, which is proportional to Ω. Integrating the angular velocity, the angle of rotation of the body on the relative axis

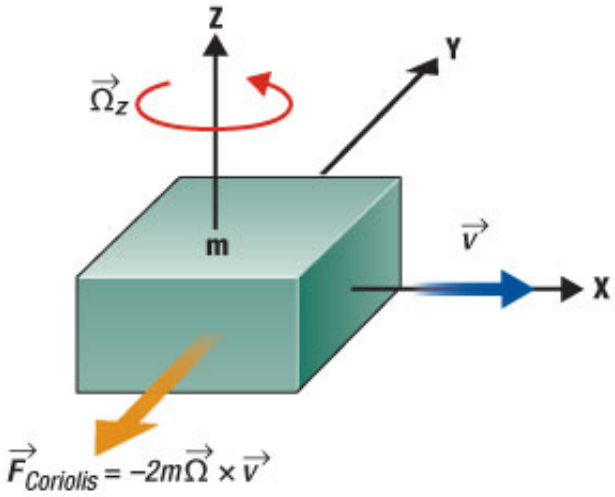


Figure 3.2: The Coriolis effect.

can be obtained. Three uniaxial gyroscopes perpendicular to each other will form a tri-axial gyroscope, able to measure the rotation of a body in the 3-dimensional space.
For what concerns the MEMS accelerometer, Figure 3.1 shows how it is actually based on a force measurement, acting on a spring-mass mechanism, providing the acceleration indirectly. Once again, three uni-axial accelerometers are combined to get 3D measures.
An important step consists in removing the gravity from the IMU reading. To understand this, one can imagine an IMU in free falling: since a MEMS accelerometer has a spring-mass inside, and the displacement of the mass is proportional to the force acting on it, the IMU will read $a_z = 0$ due to the zero displacement of the mass, but the IMU is certainly changing its vertical position since it is falling! The correction is made by subtracting the vertical gravity component, after the acceleration measurement is reported in the world frame.

IMUs are often chosen due to their small size and fairly high sample rate, but they suffer from some sources of error. One is the already mentioned double integration problem for the accelerometer; this error is also present with the single integration in the gyroscope and while the accelerometer could help in the correction, it cannot measure the *yaw* angle, i.e., the angle around the vertical axis, where the gravity vector lies.
Another important problem is a time-varying bias error, present alongside the measurement noise. With the term *bias* it is intended the offset between a real value and a sensor output, while the measurement noise is an additive White-Gaussian process. The complete noise modeling for the IMU will be explained later in the relative section; for now, it is sufficient

to say that all of these uncertainties lead to the necessity of at least another sensor to periodically correct the pose estimation.

3.2 Pose Estimator

Considering, for now, only the IMU tracking, to perform the sensor fusion it is needed at least another measurement of the sensor pose with respect to the world. Exploiting the fact that a robot is being used, one can perform experiments choosing the most appropriate pose estimate between the FK output and the Camera one. For example, a fusion between IMU and Cameras can be performed and then compared to the FK, or viceversa, IMU and FK can be fused to compare the output with the high precision Optitrack cameras setup. This last case could actually be not so useful, since the maximum frequency of the used Cameras is 240 Hz (with respect to a 500 Hz IMU and a 1 kHz FK).
A normal, low cost camera has actually a much smaller update rate; when used in the EKF, the Cameras will be slowed down to a range of 30-100 Hz, in order to expand the applicability of the proposed solution.

Whatever the sensor used, the following model is applied to the pose measurement:

$$p_m = p + n_p \tag{3.3}$$

$$q_m = q + n_q \tag{3.4}$$

where the terms are

- p_m = measured value in position. [m]
- n_p = noise term relative to the position measurement, expressed in the same units.
- q_m = measured value in orientation, expressed in quaternions.
- n_q = noise term relative to the orientation measurement.

These straight-forward equations simply tell that the measured values, in position and orientation, are equal to the true values plus an additive *White Gaussian Noise* (WGN) term with zero mean and variance σ^2, whose standard representation is

$$n \sim N(0, \sigma^2)$$

The WGN addition is a model needed for the construction of the Kalman Filter, but clearly does not necessarily match the reality.
As it will be shown in Chapter 4, the noise term must not be actually added in the code, since the measurement arrives already corrupted. The only needed parameter is the variance of n_p and n_q, but being this an ad-hoc model for sensor fusion, these values can not be easily found in datasheets or other documentation. Actually, this is an advantage of this representation: the variance can be chosen and modified for every application, becoming a very important tuning parameter, that assigns a weight to the correction, according to the sensor's uncertainty.
The orientation term is shown in (3.4) for the first time and it is expressed in the form of a quaternion. This choice is very common in navigation and tracking, since the 4-elements quaternion is a more powerful way of expressing rotations with respect to Euler or RPY angles, suffering from the gimbal lock singularity problem. Only quaternions and rotation matrices will be used in computations throughout the whole thesis, bearing in mind that rotation matrices need nine elements instead of four, increasing numerical effort and leaving quaternions as the best representation. The only problem quaternions have, is their non-intuitive visualization, so for a final comparison between values it will be useful to transform everything into RPY angles expressed in degrees. For more information on quaternion's definition and mathematical properties, see Appendix A.

3.3 IMU bias and noise modeling

When treating the measurement errors of an IMU, one can observe the deterministic and the stochastic ones. The first type, which includes for example constant biases or axis misalignments, is normally removed from the raw measurement by calibration, while the second should be modeled with mathematical processes. In this thesis the following general model has been applied:

$$x_{meas} = x + b_{to} + b_{ir} + n \tag{3.5}$$

where the terms are

- x_{meas} = measured quantity, i.e., measured linear acceleration a_m or measured angular velocity ω_m. $[m/s^2]$, $[rad/s]$
- x = true value of the relative measured quantity.

- b_{to} = *Turn-On Bias* or *Bias Repeatability*, i.e., the initial bias at powerup, different every time. A higher value of this parameter means a longer period to converge to a good estimate of the bias after start-up.

- b_{ir} = *In-Run Bias* or *Bias Instability*, i.e., the effective variation of the bias during normal operation of the device.

- n = random white noise term, called *Velocity Random Walk (VRW)* for the accelerometer and *Angular Random Walk (ARW)* for the gyroscope respectively.

The measured value provided by the sensor is here computed as the true one, corrupted by three terms; the stronger the corruption, the worse the estimation, therefore the filter will have to provide a way to deal with too wrong measurements by assigning a proper "weight" to each of them. This weight will be a measure of the uncertainty based on the noise model parameters of the three terms above.
Upon direct suggestions from the manufacturer, it has been decided to build a first-order Gauss-Markov process for the propagation of the In-Run Bias, while keeping the Turn-On Bias as a constant offset. The discrete-time version of the process is as follows ([20], [21]):

$$b_{ir}(k+1) = a_d b_{ir}(k) + \eta(k) \tag{3.6}$$

$$a_d = e^{-\frac{\Delta T}{T_c}} \tag{3.7}$$

where ΔT is the sampling time, T_c is the constant correlation time and $\eta(k)$ is a discrete-time white noise with variance σ_η^2. It can be proven that:

$$\sigma_\eta^2 = \sigma_b^2(1 - e^{-2\Delta T/T_c}) \tag{3.8}$$

where σ_b^2 is provided in the datasheet for both accelerometer and gyroscope, and it is assumed equal for all three axes. It will be written, from now on, as σ_{ir}^ω or σ_{ir}^a.
The set of parameters needed for the Extended Kalman Filter and extracted from the datasheet of the available Honeywell HG4930 MEMS IMU is reported in Table 3.1.

Every value has to be reported in standard measurement units. In order to be consistent, from now on the computations are done considering standard deviations instead of variances.

MEMS Device	Noise	Value	Units
Gyroscope	b_{to}^{ω}	7	°/hr
Gyroscope	σ_{ir}^{ω}	0.25	°/hr
Gyroscope	ARW	0.04	$°/\sqrt{hr}$
Accelerometer	b_{to}^{a}	1.7	mg
Accelerometer	σ_{ir}^{a}	0.025	mg
Accelerometer	VRW	0.03	$m/s/\sqrt{hr}$

Table 3.1: Datasheet parameters for the Honeywell IMU. The Turn-On terms are a constant offset, while the other values are expressed in terms of standard deviation.

For the biases, the equations are[1]:

$$\boldsymbol{b_{to}^{\omega}} = 7 \times \pi/(180 \times 3600) = 3.393 \times 10^{-5} rad/s \tag{3.9}$$

$$\boldsymbol{\sigma_{ir}^{\omega}} = 0.25 \times \pi/(180 \times 3600) = 1.212 \times 10^{-6} rad/s \tag{3.10}$$

$$\boldsymbol{b_{to}^{a}} = 1.7 \times 9.806 \times 10^{-3} = 16 \times 10^{-3} m/s^2 \tag{3.11}$$

$$\boldsymbol{\sigma_{ir}^{a}} = 0.025 \times 9.806 \times 10^{-3} = 2.451 \times 10^{-4} m/s^2 \tag{3.12}$$

For what concerns the additive white noise with respect to ARW and VRW, as well as for η_ω and η_a, it is instead necessary to relate them also to the sampling and correlation times. Since the IMU provides acceleration and angular rate at around 500 Hz (2 ms), and its correlation time is 300 s by datasheet, the resulting values are:

$$\boldsymbol{\sigma_{n\omega}} = 0.04 \times \pi/(180 \times \sqrt{3600})/\sqrt{\Delta T} = 2.601 \times 10^{-4} rad/s \tag{3.13}$$

$$\boldsymbol{\sigma_{\eta\omega}} = \sigma_{ir}^{\omega} \times \sqrt{1 - e^{-2\times(1/Tc)\times\Delta T}} = 4.425 \times 10^{-9} rad/s \tag{3.14}$$

$$\boldsymbol{\sigma_{na}} = 0.03/\sqrt{3600}/\sqrt{\Delta T} = 11 \times 10^{-3} m/s^2 \tag{3.15}$$

$$\boldsymbol{\sigma_{\eta a}} = \sigma_{ir}^{a} \times \sqrt{1 - e^{-2\times(1/Tc)\times\Delta T}} = 8.951 \times 10^{-7} m/s^2 \tag{3.16}$$

Every time a $\boldsymbol{\sigma}$ is computed, it can be used to generate its correspondent gaussian noise vector, with the Matlab expression $»\sigma * randn(3,1)$, where once again the vector components are considered equal for the three axes. In this way, one can obtain η_a, η_ω, n_a, n_ω, b_{ir}^{a} and b_{ir}^{ω} for further use in the filter.

The order of magnitude of all the IMU noises is such that their contribution will be far smaller than the one of the integration error. Honeywell claims that this particular IMU can actually sense the effect of the Earth's rotation. These results are possible only with high performant, thus expensive, sensors; one can expect a Turn-On bias of few degrees per

[1]Bold symbols are just for better visualization, but do not necessarily imply that the quantity is a vector.

second, instead of few degrees per *hour*, in normal low cost devices.
This concludes the noise modeling of the sensor suite. The important equation to remember is (3.6), that expresses the propagation phase of the In-Run Bias Instability. In the next chapter it will be shown how to use this equation in the filter prediction, and how to account for the noise variances when building the correction phase.

Chapter 4

Filter Framework

All the work done until now finds its application in this chapter, that includes several of the main points of the whole project and has the function of merging everything together from the mathematical and conceptual point of view.
In order to present things efficiently and understand the algorithms, it is useful to remember once again the main goal of the project, i.e., the tracking of the tool mounted on the flange of the arm. This resorts to its pose estimation in discrete time, done by using the measurements of the IMU and whatever pose estimator one decides to add. From now on, given the considerations done in Chapter 3, the pose estimator will be the Camera setup.
When talking about pose estimation, it is important to set the *Reference Frame* with respect to which the estimates are reported. Since pose is just the name to indicate orientation and position, a reference frame provides these information in the sense that its orientation is given by three ortogonal axes, while its position is just the position of the axes' origin.
The following section is needed in order to present the reference frames involved and their mutual relations.

4.1 Hand-Eye Calibration

A needed anticipation of the hardware setup, described later in Chapter 6, is shown as a simplified version in Figure 4.1. Here the four main reference frames that will have a role in the filter algorithm are reported:

1. The World Frame **W** (fixed): when the robot is initialized, its pose in the 2D plane (X, Y and Θ, not Z since the robot cannot fly) is set to zero. After that, a reference frame is created in the center of the platform, with the Z axis pointing up. This becomes the point where every following measurement has to be related to, and obviously the point

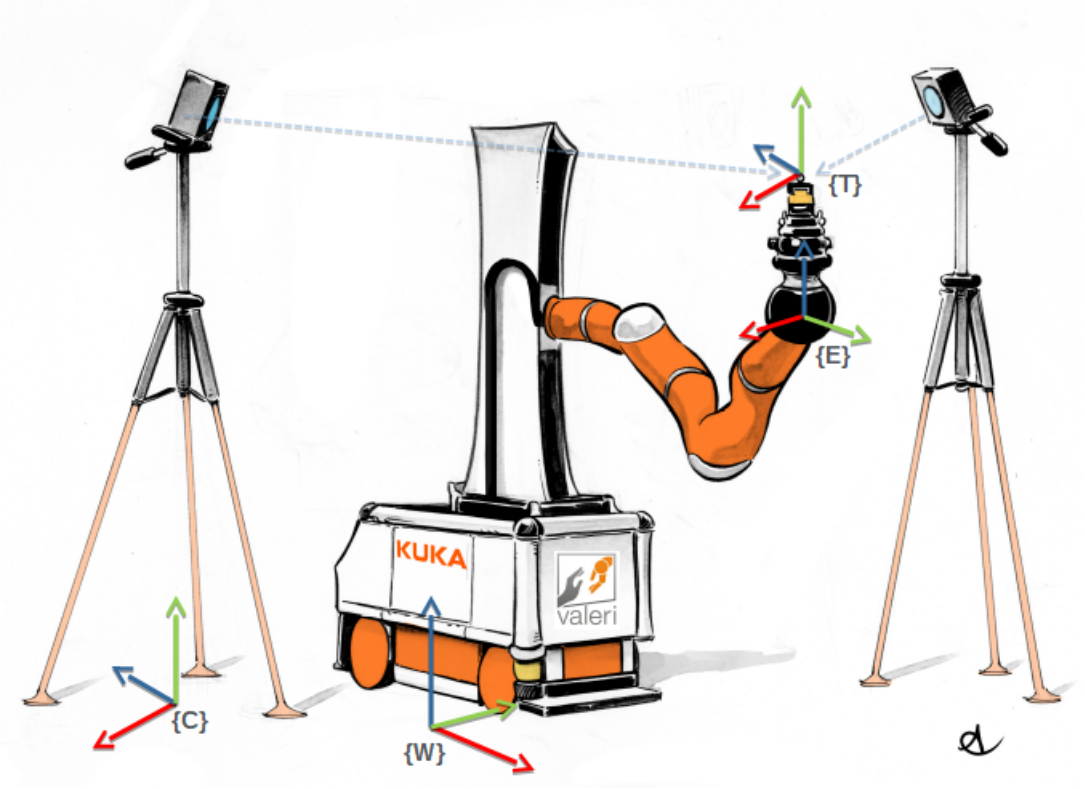


Figure 4.1: Description of the reference frames involved. Note the color convention that assigns red to the X axis, green to the Y axis and blue to the Z axis.

with respect to which the IMU pose needs to be estimated.

2. The End Effector Frame **E** (moving): the FK algorithm provides by default the pose of the center of the black ball, cover of the last joint. Additional offsets can be added in order to get the flange's pose, but in this part this computation is not important.

3. The Tool Frame **T** (moving, but fixed with respect to E): a pivot marker is placed on top of the flange and tracked by the cameras.

4. The Camera Frame **C** (fixed): Cameras will track the tool providing its pose with respect to this reference frame created during calibration.

An IMU frame also exists, but its usage will be introduced only later in this Chapter.
One of the first steps in Sensor Fusion is to get all the transformations between frames, in order to easily report data from one point to another. The pose of a frame B with respect to another frame A can be indicated by a 4x4 homogeneous transformation matrix written as T_B^A. This type of matrices has the very important property of composition thanks to which, given an intermediate frame C, one can obtain:

$$T_B^A = T_C^A T_B^C$$

This result is useful in order to obtain the needed frame in different ways, according to the available information. Here, the final goal is the pose of the tool with respect to the World; it can be expressed by the matrix T_T^W, which is not directly provided, but has to be computed in the way just explained:

$$T_T^W = T_E^W T_T^E$$

More importantly, the same matrix should be related to the visual measurements, for its usage in the filter:

$$T_T^W = T_C^W T_T^C$$

The last two equations make use of all the four transformation matrices that can be identified with this setup, further simplified in Figure 4.2. In the beginning, only T_E^W and T_T^C are known, from the FK and the cameras measurement respectively.

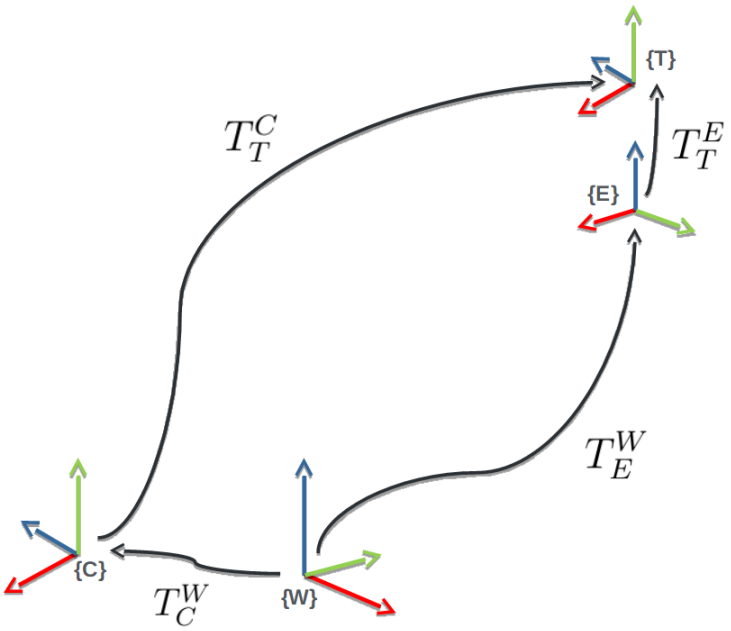


Figure 4.2: Transformation matrices of interest.

The simultaneous computation of the other two is the job of the *Hand-Eye Calibration*; this algorithm has got its name by common applications in the robotics community, where a camera ("eye") is mounted on the gripper ("hand") of a robot. When using monocular or stereo cameras, the algorithm is performed using a calibration pattern, like a black and white chess texture printed on a surface. This operation can be performed also by moving the robot in a certain number of fixed locations, and then providing T_E^W and T_T^C as input to the calibration algorithm. Since a program to perform the Hand-Eye Calibration was already made for another project in the KUKA laboratory, it has been reutilized as a black

box without further researches or modifications. The reader will find more information about the particular procedure used, and about the topic in general, in [26].
Finally, the transformation matrix T_C^W is computed, and every possible relation between frames can be calculated and used as needed. In particular, the two versions of the matrix T_T^W will be provided as input to the EKF, to be used for the initial state and the update/correction phase.

4.2 Linear and Extended Kalman Filter

Having now all the sensors' noise data, as well as the calibration parameters, the next step is building the filter.
As a general rule, the Kalman Filter requires a state space representation of a system, so the needed quantities are collected inside the so called state vector. Then, the operations are divided in two phases: **propagation** and **update**. These two parts are also called prediction and correction, and the base for all the calculations is the predictor-corrector form of the Linear Kalman Filter. Basic equations are then derived by a state-space representation of the system, evolving from the initial estimates according to the scheme reported in Figure 4.3. All the equations will be explained in due time, but the working principle is simple:

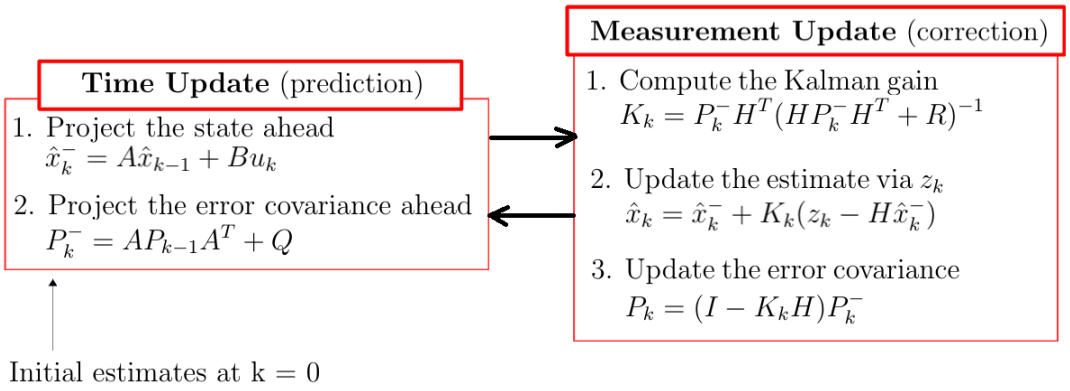


Figure 4.3: Working principle of the general Kalman Filter. The two phases of prediction and correction contain specific equations coming from a state space model and Bayesian probability theory.

keep propagating the state variables and their uncertainty according to their differential equations until a measurement of the estimates arrives. At this point, compute the weight to be assigned to those measurements according to their reliability and correct the current estimates with the new information, before getting back to the propagation phase.

For this application, the state is initially composed by position and orientation. Remembering the considerations made in the first section, everything needs to be estimated with respect to the World Frame. To improve estimation accuracy, also the velocity is inserted, along with turn-on and in-run biases, leading to the 22-elements state vector X:

$$x = \begin{Bmatrix} p^T & v^T & q^T & b_{to}^{\omega T} & b_{ir}^{\omega T} & b_{to}^{aT} & b_{ir}^{aT} \end{Bmatrix} \tag{4.1}$$

where actually $p = p_i^w$, $v = v_i^w$ and $q = q_i^w$, being i the IMU frame, now coming into play[1]. In order to simplify the calculations, using the direct measurements from the IMU, everything is in fact computed in the World-IMU coordinates, remembering that once retrieved the pose of the IMU (p_i^w and q_i^w), the extension to the tool frame is trivial.
Note that all the quantities involved are actually vectors (either 3x1 or 4x1 for the quaternion) since they represent a 3D quantity.
Turn-On biases have also been inserted, but only their update phase is taken into account, mantaining them constant during propagation. Refer to Chapter 3, section 3, for the explanation of the bias and noise values that will be used from now on.

The state propagation is constructed starting from differential equations; there is one differential equation for every element of the state, and it will be assumed to be the same for every axis, leading to:

$$\dot{p} = v \tag{4.2}$$

$$\dot{v} = C_q^T(a_m - b_{to}^a - b_{ir}^a - n_a) - g \tag{4.3}$$

$$\dot{q} = \frac{1}{2}\Omega(w_m - b_{to}^\omega - b_{ir}^\omega - n_\omega)q \tag{4.4}$$

$$\dot{b}_{to}^\omega = 0 \tag{4.5}$$

$$\dot{b}_{ir}^\omega = \eta_\omega \tag{4.6}$$

$$\dot{b}_{to}^a = 0 \tag{4.7}$$

$$\dot{b}_{ir}^a = \eta_a \tag{4.8}$$

where C_q is the rotation matrix relative to the quaternion q, g is the gravity vector in the World Frame and $\Omega(\omega)$ is the quaternion multiplication matrix, as defined in [19] and reported in Appendix A.

[1]The superscript/subscript notation, here reported with lowercase letters for differentiation reasons, has the same meaning as for transformation matrices T_B^A.

The matrix C_q is obtained as the 3x3 top left submatrix extracted from T_I^W, where

$$T_I^W = T_E^W T_I^E \tag{4.9}$$

The matrix T_I^E is constructed from a rotation and a translation, measured with laboratory instruments. Its computation is also reported in Appendix A.
What happens in the first three equations is the following:

1. IMU's gyroscope measures an angular rate, from which bias and noise are removed according to the noise model developed in Chapter 3. (eq. 4.4)

2. The angular rate is used to get the rotation C_q. This rotation is used to report also the accelerometer's measurement into the World Frame, compensating also for the gravity. (eq. 4.3)

3. The new acceleration is integrated to obtain velocity (eq 4.2)

For the system constructed with the state vector (4.1), it is not possible to resort to the standard, linear, Kalman Filter. With respect to equation (1) inside the Time Update block of Figure 4.3, that describes a linear relation for the previous state and the control input u, it can be seen from equations (4.2) to (4.4) how here the state is actually propagated in the form

$$x_k = f(x_{k-1}, u_k) + g(w_k)$$

where f and g are differentiable functions and w is the so called process noise. For this application, the IMU measurements constitute the control input u, while its biases and WGNs are modeled inside w.
The EKF linearizes the functions f and g around the current estimate, in order to compute Jacobian matrices relatable to the covariance matrix of the process noise. Normally, the same goes for the output equation, which for the EKF framework is

$$z_k = h(x_k) + v_k$$

but, as it will be explained later, the part related to h will be trivial having a direct pose measurement available.
The standard approach would now consist in the computation of the Jacobians. In this project however, following [17], [18] and [19], the Jacobians will be computed with respect to the *Error State Kalman Filter* (ESKF).

4.3 Error State KF Implementation

In this formulation, a distinction is made between true, expected (or nominal) and error state values, with the true ones expressed by a suitable composition of the other two. Instead of doing every calculation with the expected state, as in the normal EKF, the filter algorithm is constructed around a small-signal error state. When a measurement from the cameras comes, the error state is updated and its posterior estimate is injected into the expected state, before being reset to zero.
The advantages of such a formulation are listed in the following:

- The dimension of the state is reduced, since the error quaternion is represented with three elements instead of four. This means less numerical computation time, less memory used, smaller matrices and faster runs.

- Being made of small values, the error state operates close to the origin, improving the validity of the linearization. Second order products are negligible in the computations of the Jacobians, which is then simplified and more robust.

- The errors have a slower dynamics with respect to the expected state, i.e., they change less quickly. This means that the correction phase can actually be performed at a much lower rate than the prediction one. This is crucial for the Sensor Fusion concept.

The error state is computed first taking the expectations, i.e., the values without noise, of the state variable and the IMU control inputs. So, defining:

$$\hat{a} = (a_m - \hat{b_{to}^a} - \hat{b_{ir}^a}) \tag{4.10}$$

$$\hat{\omega} = (\omega_m - \hat{b_{to}^\omega} - \hat{b_{ir}^\omega}) \tag{4.11}$$

it can be then written:

$$\dot{\hat{p}} = \hat{v} \tag{4.12}$$

$$\dot{\hat{v}} = C_{\hat{q}}^T \hat{a} \tag{4.13}$$

$$\dot{\hat{q}} = \frac{1}{2}\Omega(\hat{\omega})\hat{q} \tag{4.14}$$

while all of the bias derivatives' expectations are equal to zero.
After that, the error is defined as the algebraic difference, i.e., $\tilde{x} = x - \hat{x}$, for all quantities apart from the quaternion, since its error is instead defined as a small rotation between the

frames indicated by true and expected value:

$$\delta q \coloneqq q \otimes \hat{q}^{-1} \approx [\frac{1}{2}\delta\theta^T \quad 1]^T$$

A 21-elements error state can then be defined:

$$\tilde{x} = \left\{\Delta p^T \quad \Delta v^T \quad \delta\theta^T \quad \Delta b_{to}^{\omega T} \quad \Delta b_{ir}^{\omega T} \quad \Delta b_{to}^{aT} \quad \Delta b_{ir}^{aT}\right\}$$

with its new differential equations, from which the matrices will be constructed:

$$\dot{\Delta p} = \Delta v \tag{4.15}$$

$$\dot{\Delta v} = -C_{\hat{q}}^T \lfloor \hat{a_x} \rfloor \delta\theta - C_{\hat{q}}^T \Delta b_{to}^a - C_{\hat{q}}^T \Delta b_{ir}^a - C_{\hat{q}}^T n_a \tag{4.16}$$

$$\dot{\delta\theta} = -\lfloor \hat{\omega_x} \rfloor \delta\theta - \Delta b_{to}^{\omega} - \Delta b_{ir}^{\omega} - n_{\omega} \tag{4.17}$$

$$\Delta \dot{b}_{to}^{\omega} = 0 \tag{4.18}$$

$$\Delta \dot{b}_{ir}^{\omega} = \eta_{\omega} \tag{4.19}$$

$$\Delta \dot{b}_{to}^{a} = 0 \tag{4.20}$$

$$\Delta \dot{b}_{ir}^{a} = \eta_a \tag{4.21}$$

where $\lfloor Y_x \rfloor$ is the 3x3 skew-symmetric matrix relative to Y.

In the following, the creation of the discrete time matrices for propagation and update phases is reported. Apart from the quaternion error, there is no particular difference to the normal Kalman Filter procedure, that will be again summarized later.

4.3.1 State Propagation

The discrete time version of equations (4.12) to (4.14) for the expected state, is then written in order to transform the derivatives in a difference between current and previous step value, in a sort of backward integration framework. This involves the Verlet algorithm reported in Chapter 3 for the acceleration, and the First Order Quaternion integration reported in Appendix A.
In the real time online estimation, the order of the operations has to be reversed, in order to mantain control on the current and previous values.

$$\hat{b_{to}^{\omega}}(k) = \hat{b_{to}^{\omega}}(k-1) \tag{4.22}$$

$$\hat{b_{ir}^{\omega}}(k) = a_d \hat{b_{ir}^{\omega}}(k-1) + \eta_{\omega}(k) \tag{4.23}$$

$$\hat{b_{to}^{a}}(k) = \hat{b_{to}^{a}}(k-1) \tag{4.24}$$

$$\hat{b_{ir}^{a}}(k) = a_d \hat{b_{ir}^{a}}(k-1) + \eta_{a}(k) \tag{4.25}$$

$$\hat{q}(k) = \hat{q}(k-1) \otimes \hat{\omega}\Delta T \tag{4.26}$$

$$\hat{v}(k) = \hat{v}(k-1) + \frac{1}{2} C_q(\hat{a}(k) + \hat{a}(k-1))\Delta T \tag{4.27}$$

$$\hat{p}(k) = \hat{p}(k-1) + \hat{v}(k)\Delta T + \frac{1}{2} C_q \hat{a} \Delta T^2 \tag{4.28}$$

where $\hat{a}$ and $\hat{\omega}$ are computed according to (4.10) and (4.11). Here it is included also the first order Gauss-Markov process for the In-Run bias propagation, explained in Chapter 3.

Next, the error state equations (4.15) to (4.21) can be summarized into the **linearized, continuos-time** error state equation:

$$\dot{\tilde{x}} = F_c \tilde{x} + G_c n$$

where the subscript c indicates the continuos time framework, and n is the noise vector

$$n = [n_a^T, \quad b_{to}^{aT}, \quad b_{ir}^{aT}, \quad n_{\omega}^T, \quad b_{to}^{\omega T}, \quad b_{ir}^{\omega T}]^T.$$

Considering then the IMU sampling time $\Delta T = 2$ ms, the needed discrete time matrices can be computed, having:

$$F_d = exp(F_c \Delta T) = \boldsymbol{I_d} + F_c \Delta T + \frac{1}{2!} F_c^2 \Delta T^2 + ...$$

for the error state discrete transition Jacobian, and

$$Q_d = \int_0^{\Delta T} F_d(\tau) G_c Q_c G_c^T F_d(\tau)^T d\tau$$

as the process noise covariance matrix in discrete time, depending on F_d, G_c and Q_c, i.e., the continuous system noise covariance matrix:

$$Q_c = diag(\sigma_{na}^2, 0, \sigma_{\eta a}^2, \sigma_{n\omega}^2, 0, \sigma_{\eta\omega}^2).$$

In order to compute the last integral, the Van Loan procedure [23] has been applied, and the result can be approssimated to:

$$Q_d \approx \frac{1}{2}[F_d G_c Q_c G_c^T + G_c Q_c G_c^T F_d^T]\Delta T$$

4.3.2 Measurement Model

For what concerns the correction phase, things are easier: The pose is in fact directly provided to the algorithm, after the manipulation of the visual data, without any dependancies from the state variables, apart from a constant coefficient for the quaternion error. With the usual EKF notation, the Jacobian matrix related to the measurement model $\tilde{\boldsymbol{z}} = \boldsymbol{H}\tilde{\boldsymbol{x}}$, can be in this case simply written as:

$$H = \begin{bmatrix} \boldsymbol{I_3} & 0_3 & 0_3 & 0_{3x12} \\ 0_3 & 0_3 & 0.5\boldsymbol{I_3} & 0_{3x12} \end{bmatrix}$$

The measurement error vector $\tilde{z}$ is actually divided in position and orientation measurement errors, which are computed in different ways due to the quaternion error definition.

$$\tilde{z_p} = p_m - \hat{p} \tag{4.29}$$

$$\tilde{z_q} = q_m \otimes \hat{q}^{-1} \tag{4.30}$$

where p_m and q_m are the position and orientation measurements extracted from the T_I^W matrix, here computed as :

$$T_I^W = T_C^W T_T^C T_I^T .$$

where the matrix T_I^T comes from the Hand-Eye Calibration, if the procedure is computed with $T_I^W = T_E^W T_I^E$ (4.9), instead of T_E^W, as input parameter. Note that this change introduces errors in the calibration, due to the low precision in calculating T_I^E (Appendix A).

4.3.3 Final procedure

Referring once again to the alternation of time update (propagation) and measurement update (correction), this section serves the purpose to order the whole procedure and provide a better understanding of the EKF principles to the reader.
During propagation, the expected state is updated according to equations (4.22) to (4.28). The expected Covariance matrix $\mathbf{P}$, i.e., the belief on the precision of the current estimates in the error state, is also updated according to the EKF model. All things considered, the error state has the following behavior:

$$\tilde{x} \sim N(\hat{\tilde{x}}, P)$$

More precisely, during this phase the following operations are performed:

1. Compute (4.22) to (4.28), paying close attention to the quaternion integration(4.26).

2. Switch to the error state and compute F_c, G_c, F_d and Q_d at the current time step.

3. Compute the Error State Covariance Matrix P according to Kalman Filter theory:

$$P(k|k-1) = F_d(k)P(k-1|k-1)F_d^T(k) + Q_d(k)$$

   where $P_{0|0}$ is a matrix tunable according to the belief on the initial conditions. If one is sure that the starting point is the true one, then $P_{0|0}$ will have almost zero values, of course not completely null to avoid singularity problems.
   The values inside this matrix will become bigger at every step, since every new prediction increases the uncertainty.

The measurement update phase instead, performed according to the pose estimator frequency, consists of the following operations:

1. Compute the residuals $\tilde{z} = z - \hat{z}$ which is actually divided in position and orientation as in (4.29) and (4.30)

2. Compute the **Kalman Gain K** $= PH^T(HPH^T + R)^{-1}$ where R is the measurement noise matrix containing on its diagonal the variances of the visual sensor

3. Compute the correction $\hat{\tilde{x}} = K\tilde{z}$, to be injected in the nominal expected state

4. Compute the update state covariance matrix as:

$$P(k|k) = (\boldsymbol{I_d} - KH)P(k|k-1)(\boldsymbol{I_d} - KH)^T + KRK^T$$

   This is the Joseph form of the P matrix update, a formulation considered numerically more stable.

This concludes the implementation of the filter framework. Several changes were made throughout the project, mainly to adapt the equations to the effectively available sensors, and to find a common ground regarding the frames to be considered.
At this point, an estimate of the pose of the IMU with respect to the World can be extracted from the nominal state vector in output to the EKF and compared with, for example, the Forward Kinematics. Before showing the results in this sense, the following Chapters will describe the work done from the software and the hardware side and the problems encountered.

Chapter 5

Software and Simulation Environment

Since the beginning of the project, the importance of offline simulation has been clear. It has been decided to spend considerable time in the creation of a versatile and reliable environment in Matlab and Simulink, to account for the following considerations:

- Physical sensors were not available at the beginning of the thesis period.
- Being this a project inside a research group, it is fundamental to set a starting point for all the people (employees or even students) that will have to work on the topic next.
- Debug is necessary at every step, from the single function to the complete sensor fusion framework. Block schemes and offline software tools help considerably in bug fixing.
- Block schemes in Simulink can easily be set to discretize operations and generate object files usable in the Real-Time Operating System (RTOS) of the robot.
- In the hardware implementation, data have to be measured, elaborated, collected and finally compared. Matlab and Simulink offer mathematical functions and graphical instruments to easy handle these tasks.

More than one simulation program has been created, mostly due to the first point above: it is obvious that, if IMU and Cameras are not available, they have to be somehow simulated[1]. This has been done in the beginning, using randomly generated 5th-order polynomials to create a sort of ground truth, from which simulated measurements have been created with the addition of white noise. In particular, the first polynomial has been derived twice to

[1]This is not valid for the FK, since it is computed internally in the robot and can always be used.

get a "true" simulated position trajectory, as well as its velocity and acceleration. A similar approach has been used to create a "true" angular rate, derived once to get an orientation evolution. Adding noise to the these polynomials simulates the IMU measurements, while adding noise to position and orientation simulates the cameras.
This approach has been used to construct and test the provisional version of the filter, although without all the parts related to gravity, reference frames and lack of data.
While being this a simplified situation, it is actually the approach to follow when dealing with the temporary absence of real devices. Furthermore, it permits to simulate any kind of sensor for which the noise parameters can be obtained or reasonably guessed, leading to a faster development for the actual solution.
Once the sensors finally came and their readings were succesfully performed, the new version of the offline simulation has been written, adding the missing parameters.
Successive updates were made, keeping in mind the concept of versatility. After a lot of step-by-step modifications, and considering also the networking part, the final software environment performs the following operations:

1. Open a socket for the IMU communication via TCP/IP protocol and start listening for acceleration and angular rate data.

2. Open a socket for the cameras communication via UDP and start listening for pose data.

3. Use cameras' pose data and FK pose data to perform the Hand-Eye Calibration.

4. Use all of the sensors data, together with the calibration results, as input to the EKF module created via Simulink

5. Run everything on the *VxWorks* RTOS and save results' data in log files

6. Put log data in Matlab to check visually the performance of the filter.

Clearly, the last passage is not necessary for the real application, but it is always useful for debugging and searching for improvements.
Points 1 and 2 were made inside programs written in C++ . From the Matlab code of the filter, instead, an automatic code generation has been performed in order to transfer it inside **Sunrise** and perform steps 3 and 4. Sunrise is a software environment, property of KUKA, based on Eclipse IDE and Java language. It is almost mandatory to get everything running here, because it functions as an interface between Windows and *VxWorks*, the Real Time

Operating System used in point 5 to control the VALERI (and several other robot models in KUKA). From Sunrise it is possible to communicate also with automatically generated object files from Simulink schemes, as in the case of the implemented EKF. Finally, another Simulink model, almost identical to the final version, has been written for offline testing (point 6).
In parallel to all of the operations listed above, one more thing is always running in the background: the software for the cameras system.

5.1 Motive Software

When purchasing a sensor, it is usually common to get also some documentation or even codes to implement its data transmission, so that the user can tailor the communication according to his/her needs. Depending on the manufacturer, the type and also the cost of the device, the provided material can vary significantly.
For the IMU sensor, a C++ program to read all the transmitted data was given in the beginning by Honeywell; later, this program has been modified in order to send only the values of acceleration and angular rate, with the possibility to access these data in Sunrise.
A very similar approach has been adopted for the cameras, where once again an already provided C++ code was modified to get everything in Sunrise.
The difference with respect to the IMU is the presence of a very complete and user-friendly software from Optitrack, called **Motive**. With its latest release, it is possible to visualize and record the trajectory made by the tracked body, plot data and specify the transmission parameters. This has been very useful for debugging purposes, since the *User Datagram Protocol* (UDP) communication inside the whole network was not guaranteed to be free of delay or missing bytes.
As already introduced in Chapter 2, the Optitrack cameras are able to see only reflective markers, thus the system is actually a Motion Capture one. By strategically placing a certain number of markers, one can construct a virtual rigid body inside the software, and track its movements in several ways. Figure 5.1 shows an example of the main window displayed during experiments.

After a rigid body is identified, the software assigns a frame to it and send its pose data, referred to what will be called, from now on, the Camera Frame, also visible in Figure 5.1 in the lower left corner; Motive provides the position in meters and the orientation expressed in

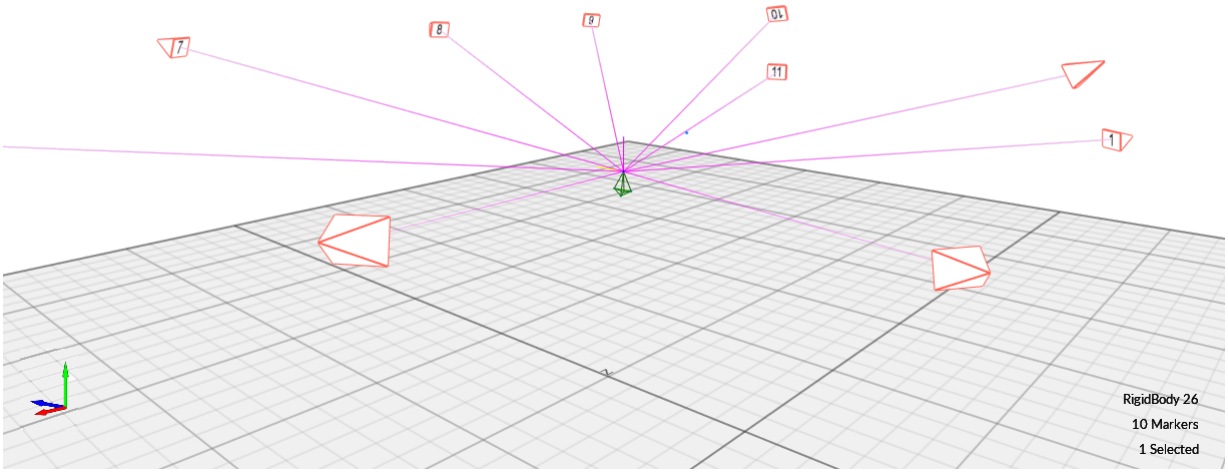


Figure 5.1: Example of Motive layout used in one of the experiments (3D Perspective View)

quaternions, to confirm once again the usage of this representation in professional applications. Furthermore, it is possible to change the orientation of the Camera Frame in the data transmission, to avoid possible confusion and error in the transformations. In fact, it is common practice in navigation to have the Y axis pointing up (default Motive configuration), while in other robotics applications, such as industrial ones, the vertical axis is Z. At the creation of the rigid body, its frame's origin is automatically placed in its geometrical center, but it can be shifted to match one of the markers. In both cases, the axis will be placed in the same directions of the Camera Frame ones, and can be resetted to this configuration at any time, if needed.

In Chapter 6 it will be mentioned again how to operate on Motive in order to calibrate the hardware for the tool tracking of the robot. The next section, instead, explains what are the issues related to the simulation world with respect to what will be the real situation.

5.2 Simulation Issues

In the Matlab code for the first filter, the one without the actual sensors, the results were positive but not similar enough to the reality. For example, it can be seen in Figure 5.2 how the simulated measurements (black circled dots) are distributed around the ground truth in a random way, but always keeping the blue line in the middle, so that the filter progressively narrows the estimates' probability range. At the time it was not anticipated that fixed offsets, due to mechanical structures or transformation errors, could shift every measurement, in a way that the correction phase in the filter could actually bring the output further from

the truth.

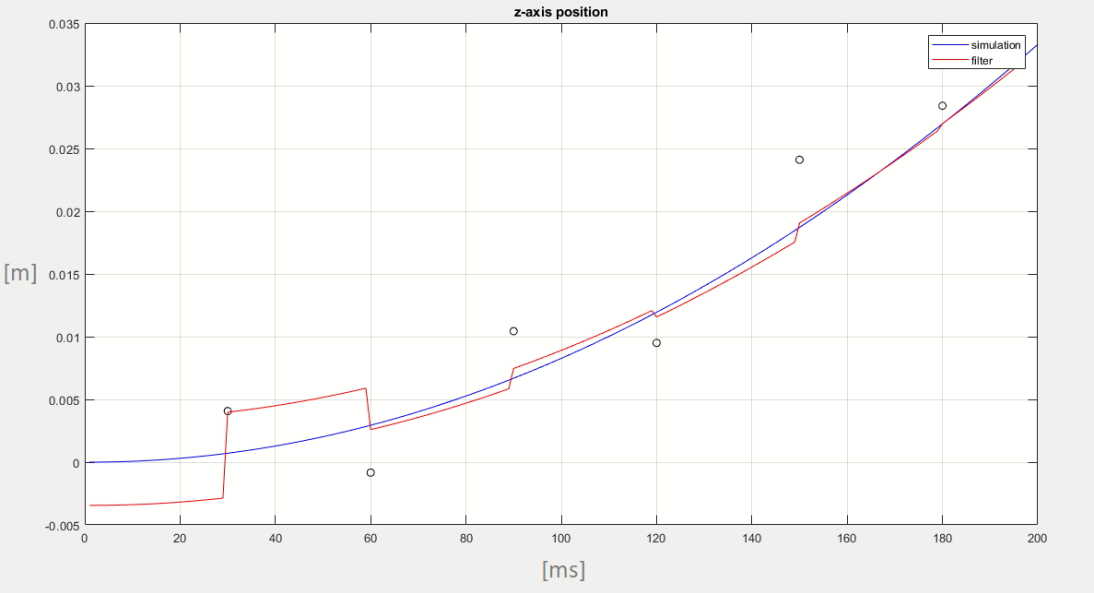


Figure 5.2: Ground truth (blue) vs estimated output (red) in the first version of the EKF. Black dots are the simulated measurements used for correction.

Another important issue is the one concerning the ground truth. This concept can exist only in the simulation environment, where it is actually created from mathematical functions. For the real application it will compared an estimate, computed thanks to filtered sensors data, with another estimate still coming from sensors data. Deciding which values to trust more can become very difficult, since it depends on noises, movements, used algorithms, calibration errors etc.

The Hand-Eye Calibration process itself, not considered in any of the Matlab/Simulink programs, is an example of relation between corrupted measures, all needed together in order to find the missing offsets and use them in other computations.

A way to test effectively the improvement brought by the filter, with respect to the FK alone, would be to perform a task similar to the sealant application one, so a sort of line following. This requires a closed-loop control system based on the error computed from the filter output and the reference, i.e. the line trajectory. Due to timing constraints, this demo could not be prepared and analyzed in this project, so the final comparison of the real results will still miss a piece with respect to the simulation. An obstacle to this experiment is also the particular way that Sunrise uses to bind modules and object files, that created some problems when writing the application for the motion of the whole VALERI.

Finally, nor Motive or Simulink can account for missing samples and synchronization issues.

The arrival of a measurement can be changed and made also variable, but if is there a transmission/receiving issue, then no update will be actually skipped or performed wrongly from the software simulation side.
In any case, a proper simulation environment should be created and documented, to ease the work of future researchers and set up an extendable software where to implement possible solution to these kind of problems. More on the future work, to improve every side of the project, will be written in Chapter 8.

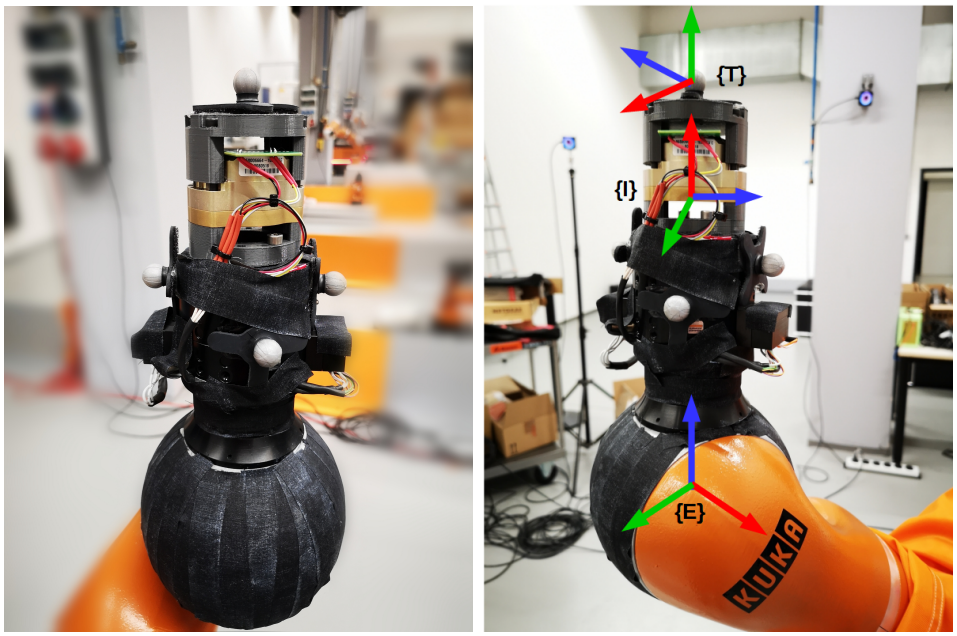
Chapter 6

Hardware Implementation

In this chapter, the whole assembled set up is described, before presenting to the reader the most important results obtained. All the consideration made in the previous chapters will find here their application, and perhaps some concepts will be better clarified.
Most of the issues related to the physical implementation are also reported, explaining where the sources of error can be found. The dedicated section is particularly important in order to better understand the final results; possible solutions to some of the encountered problems are then reported in the next chapter.

6.1 Setup description

The main hardware components needed in the experiments are surely the VALERI robot, the Optitrack system and the Inertial Sensor.
It has been already explained how, ideally, the IMU should be used to track directly the end-effector or the tool of the robot. For this reason, it has been mounted on the flange of the robot with an ad-hoc 3D printed plastic support, designed with two parts that hold the sensor in between, and provide a flat surface on top of it. On this flat surface, several possible tools or grippers can be attached; for all the experiments, it has been decided to not mount any tool, but to exploit the flatness of the support to attach one of the markers. Figure 6.1 shows the whole flange, where the golden IMU is clearly visible and surrounded by the light-grey round markers. Also, all the three reference frames that were still not shown together in Chapter 4, for the end-effector E, the IMU I and the TCP marker T respectively, are here properly placed and visualized.
An electronic board, connected to the IMU, allows to provide power to the sensor using an internal wiring system inside the arm. This is very useful to reduce the amount of floating



(a) (b)

Figure 6.1: Flange of the LBR4, with the IMU and some markers. The top marker, considered as a possible TCP, will be the one tracked by the cameras system.

cables around the robot, that could interfere with many movements.

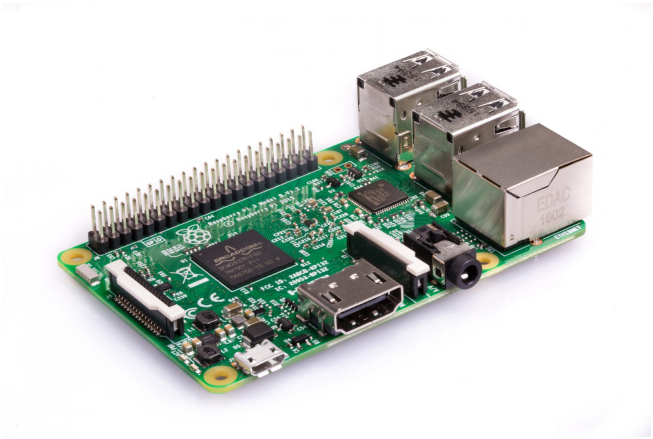


Figure 6.2: Raspberry Pi board. This device has been provided by Honeywell in order to perform a Serial-to-Ethernet conversion.

Serial data are then sent by the IMU to a Raspberry Pi board (6.2), connected to the VALERI computer with a fixed IP address. This is mandatory in order to have everything running on the same network.

Dark, opaque tape is placed all over the last joint's spherical cover, as well as around the 3D printed support and the connection pins, to avoid any confusion by the cameras. It is very

common, in fact, that shiny parts could be seen as a marker, maybe not even fixed; if, at some point, a fake marker gets in the line between a camera and a true marker, jitters and failures to keep the right visual tracking can be observed.

Before utilizing the cameras, the whole visual system has to be first calibrated[1]. This is done by creating the setup for the cameras, here assembled with several tripods and some wall supports, and with the aid of the calibration tools, shown in Figure 6.3. To perform

(a) Calibration Wand (b) Ground Plane

Figure 6.3: Calibration tools, provided with the whole Optitrack system. Special markers are attached to both of them, so that the software recognizes the wand and the plane during the calibration phase.

the calibration, the captured area has to be cleared from any object, VALERI included, as everything is a possible source of reflection. Then the wand (6.3 a) has to be waved in the air, covering all the possible working area, so that each of them sees the wand and collects a sufficient number of samples. The Motive software is then able to find the relative position of the cameras with respect to one another. Then, the ground plane (6.3 b) is placed somewhere on the floor and, after adjusting its vertical offset, it is recognized by the software as the ground level to which refer every measurement.

A larger view of the total set up is then shown in Figure 6.4. It can be seen how the robot is surrounded by the Optitrack cameras from a certain number of directions, so that the biggest possible number of devices simultaneously track the body of interest in a wide area. In this case, the body of interest is created in the Motive software, by selecting the five markers placed around the IMU, and setting as pivot point the top marker attached onto

[1]This process should not be confused with the Hand-Eye Calibration one, that is related to the whole setup. In this case, only the cameras have to be recognized and placed in space.

the flat support. Markers have to be placed so that the body created in such a way is as rigid as possible, so no change in their relative position shall occurr.
When the rigid body is created, a frame is associated to its pivot point. In the beginning, the axes are oriented in the same way as the Camera frame, which coincides with the ground plane (short axis = X, long axis = Z, Y axis coming up).

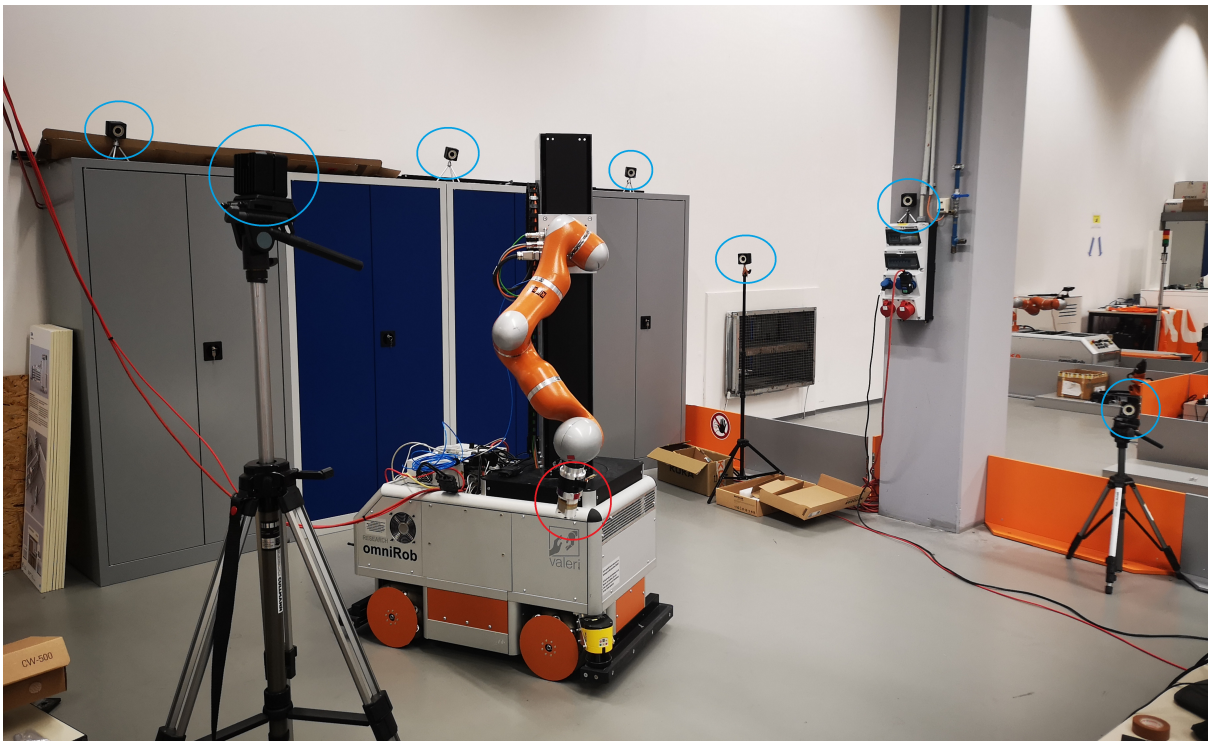


Figure 6.4: The VALERI in the KUKA laboratories. Blue circles identify the cameras, the red circle identifies the IMU.

Next, the Hand-Eye Calibration is performed, following the procedure described in Chapter 4. When the final transformation T_C^W is retrieved, the system can run on Sunrise and VxWorks and the motion for the VALERI can be programmed and executed.

6.1.1 Networking

A little has to be told about the networking setup of the Optitrack system, since its proper configuration took some time.
All the cameras are connected to a **NETGEAR GS728TPP** Ethernet switch, that provides the so called *Power over Ethernet* (PoE) feature on the 24 available ports. Devices connected to the switch via special ethernet cables can receive alimentation without the need of any other wire.
Despite some indication from the manufacturer, for this project it has been possible to run

the data transmission in a non-isolated network, since the NETGEAR was also connected to the VALERI and consequently to the IMU. This was possible by setting a static IP and Subnet Mask in the same network of the one used by the robot and by all the computers connected.
For the actual reading of the data from Motive, Optitrack provided different examples of C++ programs, but almost everyone of them made use of an internal server, inapplicable to this particular application. In the end, a program used to depack a whole buffer of streamed data, outside the internal server, has been isolated and modified in order to send only the pose of the rigid body via *User Datagram Protocol* (UDP).
A key aspect of Sensor Fusion is how to manage the different sensors, with their own working frequency and communication protocols. In this project, everything successfully runs at the controller frequency of 1 kHz, same for the FK, with the IMU updates coming at 500 Hz and the cameras always running at 240 Hz, but scaled according to the experiments. A perfect synchronization between all sensors is not possible with the current setup, but the important thing is to have real-timing as much as possible, in the sense that every measurement needs to come at a programmed and known time step. Nevertheless, in every Sensor Fusion application there will be issues related to the networking part. The following section describes all the sources of errors encountered in the real world implementation, with respect to the more ideal world of simulation.

6.2 Hardware Issues

Having just explained the importance of networking, this will be the first topic about which to report some considerations. Lack of documentation regarding the C++ program used to read data from Motive outside the proprietary server, suggests that the applied changes may actually be not optimized. While data are correctly read at the expected frequency, some sporadic observed failures in the values could perhaps be avoided with a better monitored transmission. Examples of these failures will be presented in the next chapter along with the experimental results.
For what concerns the IMU-RaspberryPi pair, the introduction of the board lowers the transmission frequency from 600 Hz to 500 Hz. It certainly remains a very fast working frequency, but it is a performance reduction that should be taken into account when dealing with high precision requirements.

The introduction of the Raspberry Pi board was necessary to overcome the lack of buffer size of another Serial-to-Ethernet conversion device. A lot of time was spent in troubleshooting the IMU, that in the beginning was providing meaningful data only during 5% of its working time. The reader interested in communicating with a high-performance IMU should be aware of the problems related to the receiving buffer size. Thanks to this work, everyone in the KUKA Corporate Research department can now use the programs written ad-hoc to utilize the sensor.

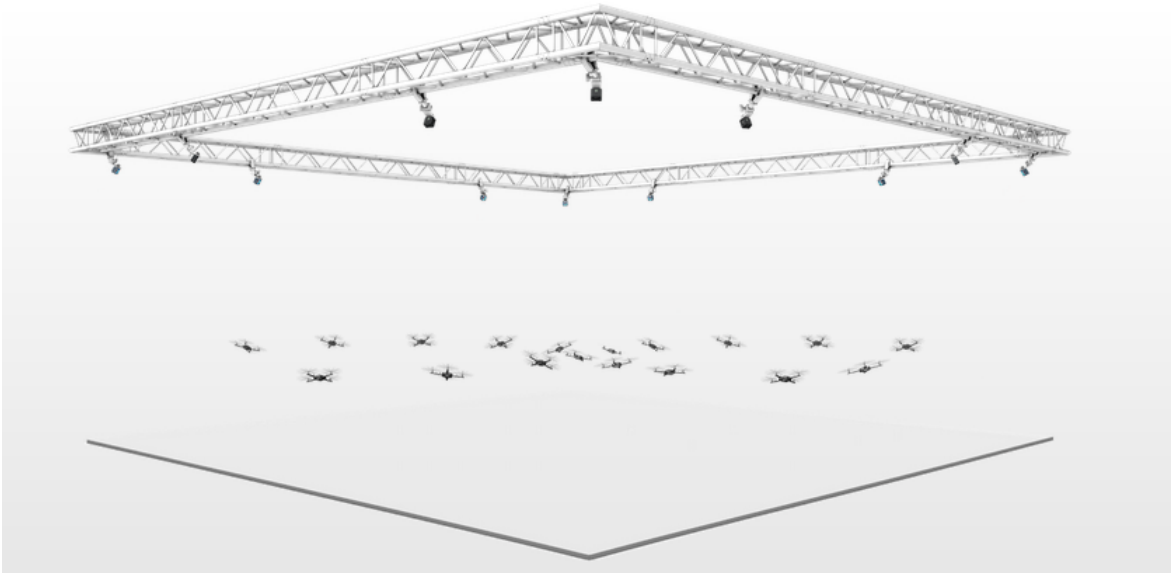


Figure 6.5: Wall or ceiling truss system, as suggested by the Optitrack documentation.

More important sources of error can be found in the Hand-Eye Calibration process. Recalling how the calibration works, one can identify several points in the setup where the needed data are not really well provided:

1. Using tripods to hold the cameras is not the best solution, especially when using smaller, more fragile ones. In a laboratory where people constantly walk over the working area, it is not uncommon to inadvertedly touch and move a little the sensors. The Motive software clearly cannot know if a camera has been moved by some millimeters, but keeps providing data in the same way. Performing constant calibration could help in recovering from this error, but this process could become long and tedious.
   It has been discussed, for future uses, to build a truss system like the one shown in Figure 6.5, even though in this way it will be difficult to move the cameras if actually

needed.

2. Ambient lights, LED illumination and shiny objects are more or less present all inside the working area, but only a small amount of them can be removed. The mobile platform and the arm too, are full of shiny, reflective metal surfaces. In order to avoid possible alignments between true and fake markers, created when a camera sees two light sources at a very small distance, the VALERI arm has been always covered with a flag during its motions. Nevertheless, failures on the visual data have been observed in every experiment. If the wrong data is sampled and sent to the calibration algorithm, the result is corrupted.

3. It has been said how two transformations should be provided to the calibration algorithm: T_T^C and T_I^W. It has also been described how to get T_I^W from the FK data, thanks to the relation:

$$T_I^W = T_E^W T_I^E$$

The only way to get T_I^E, the transformation between the center of the ball in Figure 6.1 and the IMU, is to measure it. After putting the robot in a known and well visible configuration, like a vertical one with the linear axis down, it is possible to measure a distance of about 20.85 cm. For mounting reasons, the rotational offset is not null, but it has been measured to be around 5° with respect to the X axis of the IMU (Z axis of the robot). Clearly this measurement, performed with simple instruments and a small number of tests, cannot be considered precise enough, and its error propagates from the calibration phase to the transformation of the IMU measures.

4. Throughout the kinematic chain of the VALERI, from its base to the TCP, there are numerous unmodeled physical offsets. One example can be seen back in Figure 6.4, where the black panel, on top of which the linear axis is mounted, is not actually present in any of the data used by the computer for the internal algorithms, FK included. This gives around 7-8 cm of offset, fixed but not easily quantifiable. It could be sometimes considered as a vertical translation, not changing the performance of the FK or the calibration, but this statement cannot be confirmed for every situation.
The center of the platform itself is also computed very roughly, as well as the silver surface that connects the arm to the axis, where another unmodeled offset of 1-2 cm is present due to assembling reasons.

5. Even considering its huge weight, it is not impossible that the VALERI platform moves inadvertently, even a little, especially if the brakes are released for some applications. This will have an impact in the final estimation, unless a new calibration is performed every time.

Having considered all these error sources, it is no surprise that the Hand-Eye Calibration process returned the searched transformation T_C^W with an error around 5 mm, pretty big for the goal imagined in this application. Since every measurement needs to be related to the same W frame, as explained in Chapter 4, this error will affect all the measurements coming from the camera, and will become an important error contribution in the overall result.
The IMU measurements will also be affected by the above mentioned mounting offsets and misalignments.
With all of this in mind, the next Chapter proceeds by showing the most relevant results obtained in different experiments, with the aid of the plots obtained with the Matlab/Simulink offline environment described in Chapter 5.

Chapter 7

Experimental Results

After the setup was in place, calibrated and ready to use, several experiments have been performed. In order to do meaningful tests, to account also for changes with respect to a simple line following task, it has been decided to move the VALERI arm with hand-guidance: an operator can grab the end-effector and move it randomly in space, even with fast direction changes and high-frequency elastic movements. While doing this operation, particular attention has to be paid towards the cameras, whose field of view should not be obstructed by the operator.

During the motion, IMU and cameras record data for a programmed time and save them in log files, later read and interpreted by Matlab. The EKF correction update can be set to active when the value of a measurement is changed with respect to the previous one; this works because if a sensor does not have a new measurement coming, the controller will always give as output the previous one.

In the following, some graphical results will be shown and commented. In all the experiments, the final position and orientation reported are the ones of the IMU with respect to the World, so $\boldsymbol{p}_i^w$ and $\boldsymbol{q}_i^w$. Quaternions will be converted in RPY angles in degrees for better visualization.

A remark: the final comparisons are always done by considering the Forward Kinematic as the reference to be matched. This is a strong consideration, and involves a series of problems related to the concept of ground truth. For this reason, it makes little sense to talk about the performance of the filter with respect to the FK in terms of difference in the values. For a better evaluation of the results, the reader should bear in mind that one of the goals was to avoid almost completely the use of the FK in precision tasks; in this sense, the first thing to be checked should be the stability of the filter and the *similarity* in behavior of the estimated outputs. If non negligible differences are present between the FK and the filter,

some checks on the reliability of **both** estimates should be performed before deciding who has to be trusted.

Nevertheless, offsets and big differences are surely indeces of problems in at least one of the sensors or inside the robot, and should be clearly highlighted for better modelling and possible intervention on the hardware.

7.1 Experiment 1: Updates at 30 Hz

For a normal hand-guided motion, without sudden changes in the trajectory, one can already see the offset problem of the visual measurements with respect to the FK ones in Figure 7.1, where the position in three axes is compared.

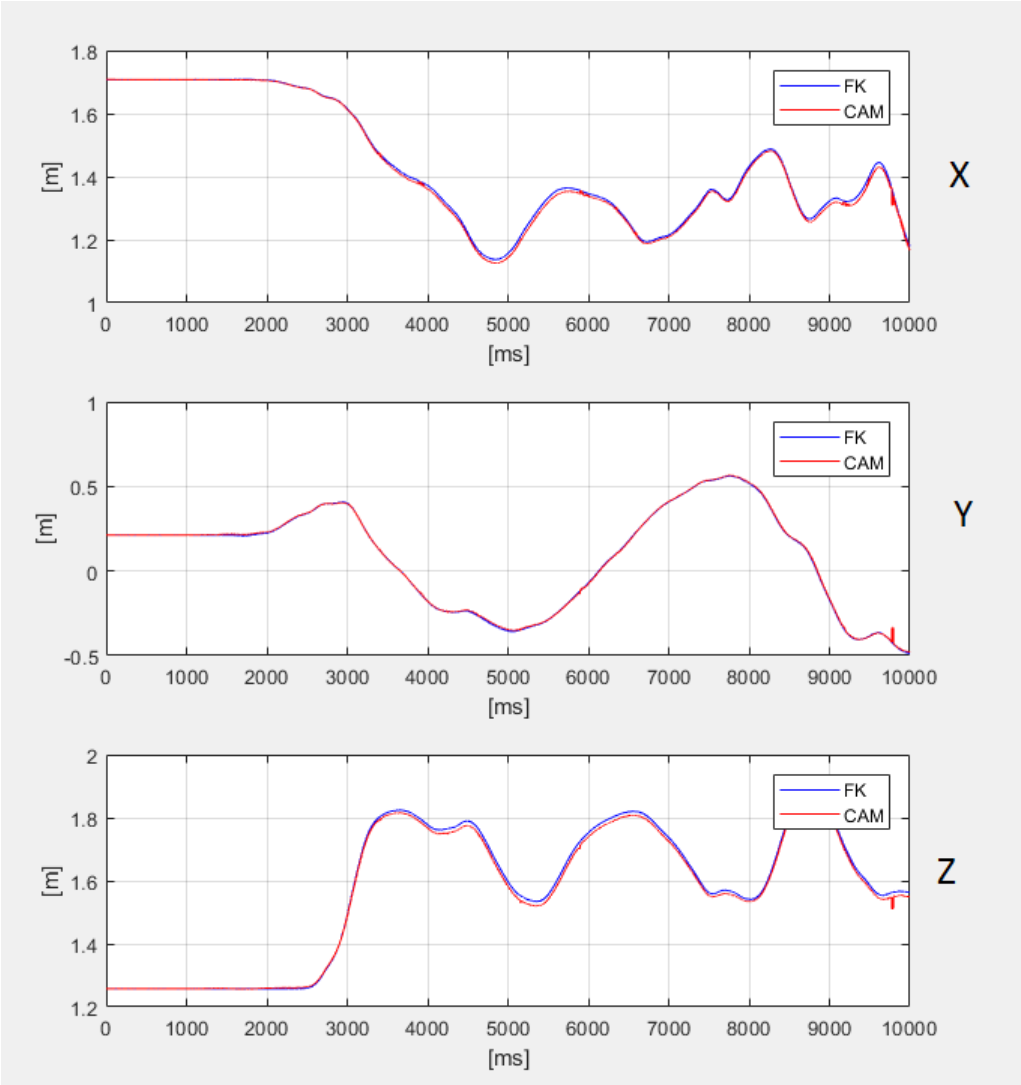


Figure 7.1: FK (blue) vs Camera-only measurement (red, at 240 Hz).

A zoomed version of the X axis in Figure 7.2 shows how the offset is not actually constant

and therefore cannot be simply removed by subtraction. Moreover, visual failures and jitters can be identified in several points. While not present in the same way for all the three axes, the order of magnitude of this offset is too big for high-precision applications, and a way to reduce it has to be researched.

Having such a considerable offset will make the filter not so good with respect to the FK, since the cameras measurements are normally trusted a lot. Increasing the amount of uncertainty of the updates is possible, but a bit dangerous, since the filter starts to propagate its state some time before the offset shows up, and during that time the cameras and the FK are a lot closer. Increasing the uncertainty here will lead to drift caused by the IMU data not corrected enough. Figure 7.3 shows a result in position after the filtering. Performances in orientation are similar.

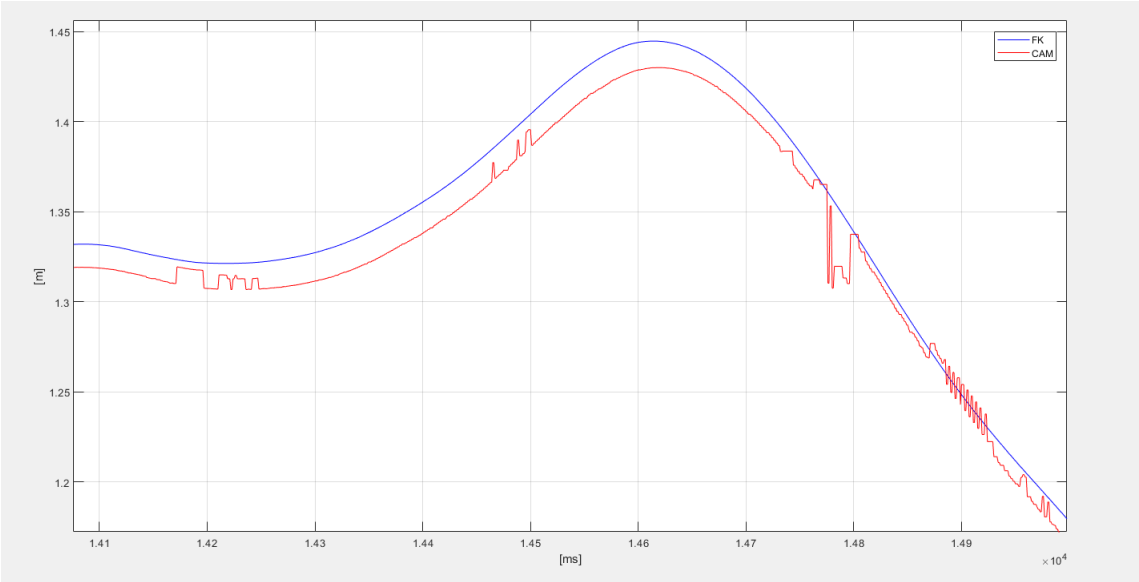


Figure 7.2: FK vs Camera, zoom in of the final samples, X axis.

7.1.1 Filtering and correction of visual data

Even though the offset is always present, it is still possible to evaluate the performance of the EKF in other aspects. A very important one is surely the filtering side, crucial in order to ignore sensor failures. An example of such failures has been already shown in Figure 7.2. When applying the EKF updates, it has been set an additional robustness control, that avoids the correction if the difference between a measurement and the current expectation is too high. To prevent the accumulation of this effect caused by the IMU drift, it is not possible to skip the correction phase more than once in a row. In this way, if it happens

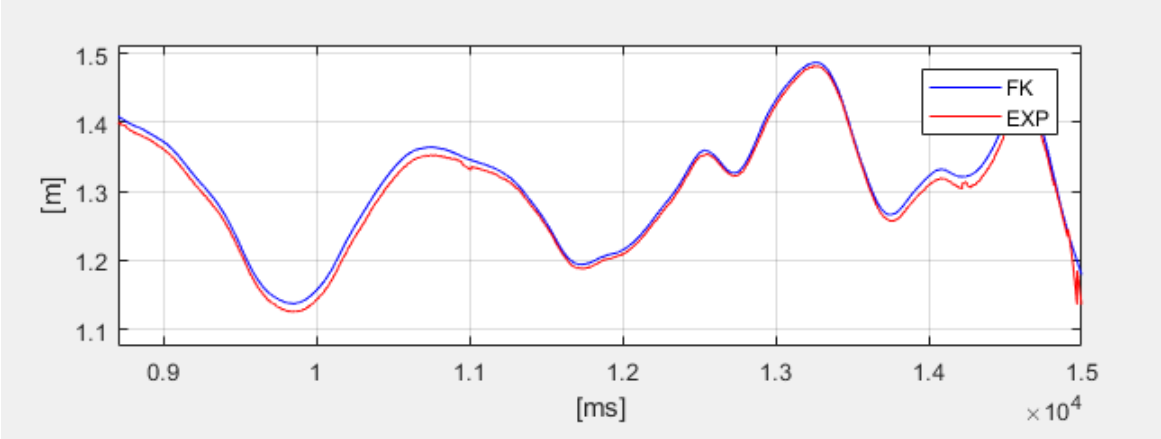


Figure 7.3: FK vs EKF estimates in position, X axis.

that a correction phase is due exactly when there is a failure in the measurement data, the algorithm ignores it and keeps propagating its state; after that, if the same thing happens also for the next correction, it will not be possible to skip it again.
Combining this ad-hoc written feature, with the concept of filtering intrinsecally present in the EKF, it is possible to have a more stable and continuos estimation. Figure 7.4, 7.5 and 7.6 show the performance, in this sense, of the EKF output with respect to the visual measurements in position and orientation.

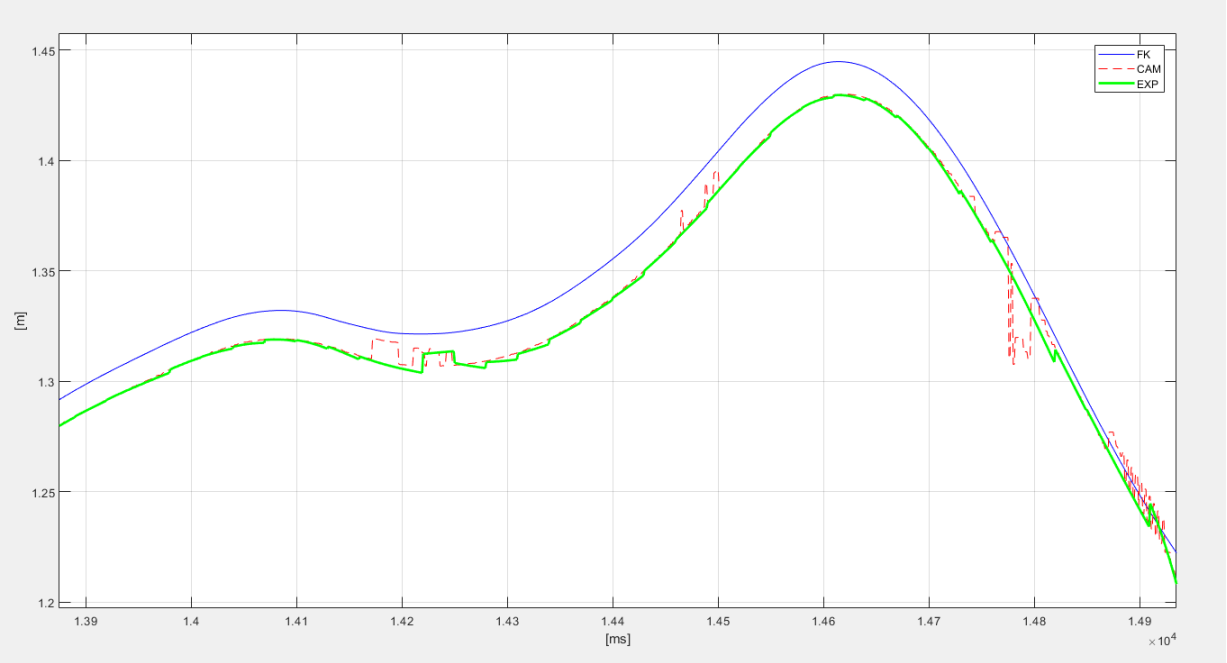


Figure 7.4: In green the EKF output, filtering out all the failures in the visual data.

It should be noted that the Optitrack system is, despite of these failures, a precise and

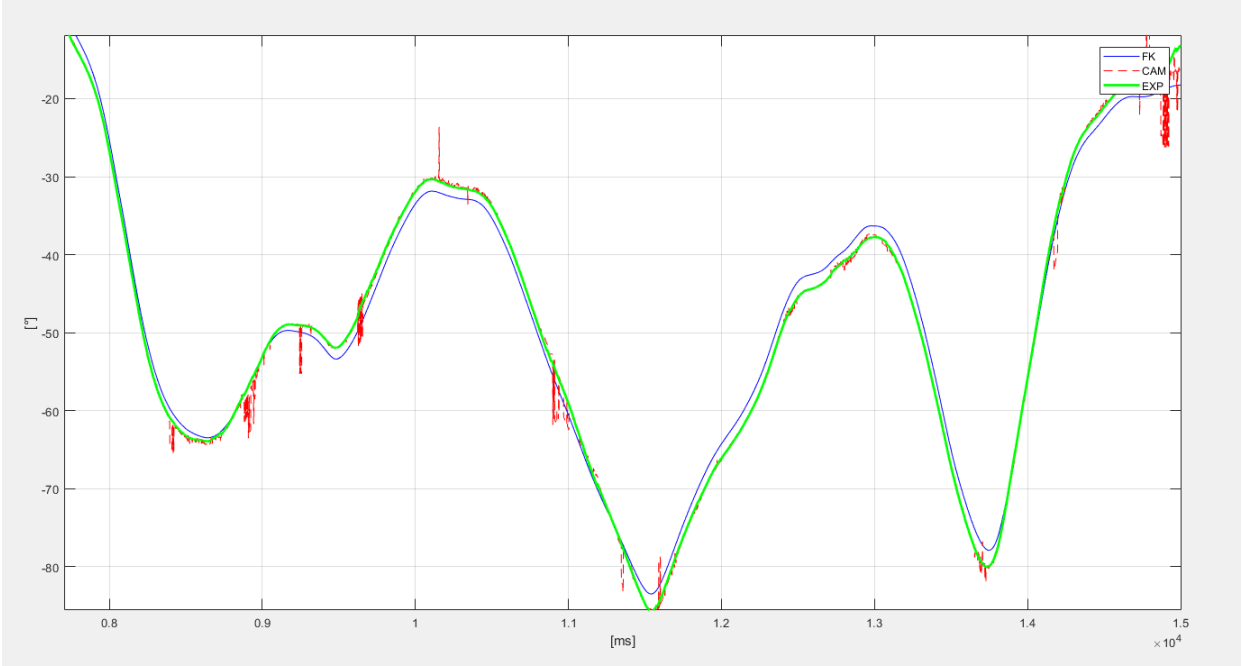


Figure 7.5: Failure correction in orientation, here presented for the yaw (Z) angle.

reliable measurement source. With a motion that does not involve the operator in the working area, as well as an environment completely cleared of reflective objects, it is unlikely to see the same amount of errors. Nevertheless, it cannot be always guaranteed that these conditions apply in industrial applications, not in the research laboratory and not in a working cell. Therefore, robustness controls like the one presented here should be always investigated.

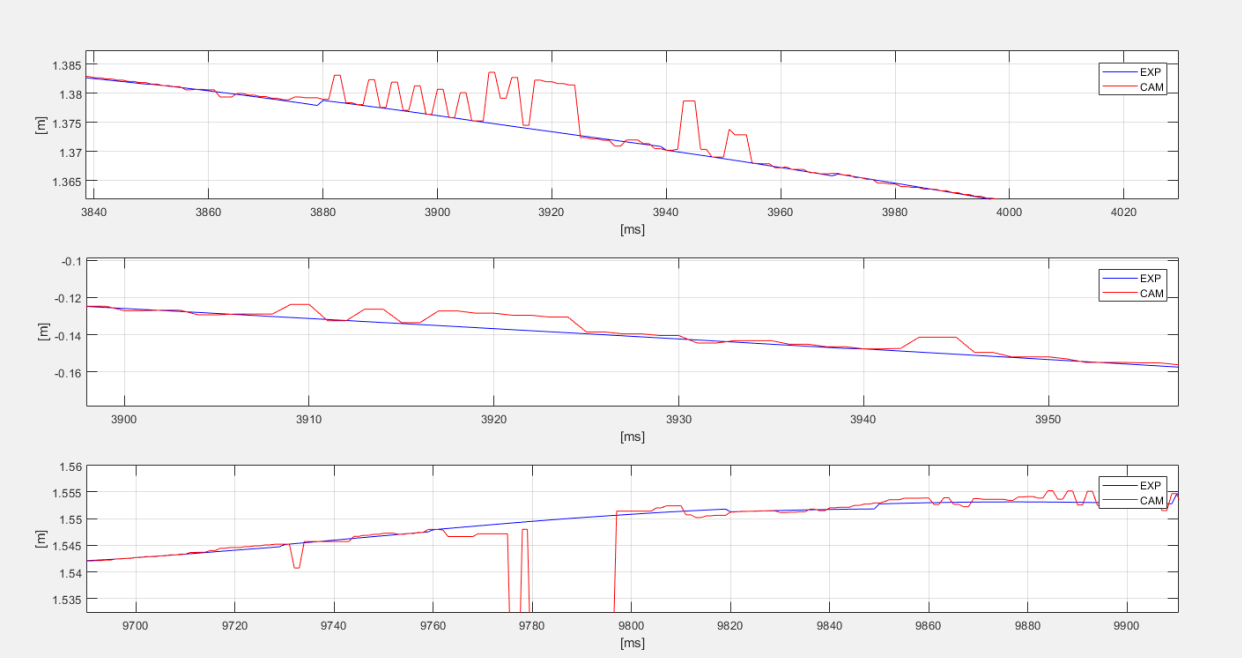


Figure 7.6: Other examples of failure correction in position all three axes. This time the comparison is between EKF estimates and Camera measurements

7.2 Experiment 2: Updates at 100 Hz

In this experiment there was no significant improvement on the overall result. This was actually expected, since a more frequent correction phase only means that the camera measurements are taken into account more times. Clearly, if the camera measurements are constantly translated with respect to the FK, a more frequent update has just the effect of deviating the IMU estimates more frequently, resulting in more or less the same situation as before.
Once again, it should be remembered how terms like "improvement" could be misleading, since there are not particularly strong reasons suggesting that estimates closer to the FK are better than estimates closer to the cameras.
Updates with higher frequencies have not been taken into account because they could reduce the performance in terms of failure corrections. In a final application, where the calibration and offset problems could be solved, a faster update could actually improve the estimation if correctly combined with the right tuning of the uncertainty parameters.

7.3 Experiment 3: High frequency movements

Exploiting the hand-guidance movement, a third experiment with a fast changing position has been carried out, mostly to test the limits of the sensors.
It has been found that even a high-performance IMU, as the one available, cannot track sudden changes and high acceleration integrations in a nice way. To retrieve an acceptable result, the correction phase should be set at the maximum available frequency of 240 Hz for the cameras. Here, the performance can be considered bad independently on the ground truth problem, since it is sure that the real motion is not correctly represented by the red trajectory in Figure 7.7. Here it can be seen a sort of sawtooth waveform due to the high sensed acceleration, that brings a quickly divergent position, periodically corrected by the camera measurements.

For what concerns orientation, the single integration of the gyroscope does not diverge as fast as with the accelerometer, so the estimates in orientation do not suffer from the same sawtooth phenomenon, apart from the pitch angle, as it is shown in Figure 7.8. There are still interesting effects going on, such as what it seems to be a time delay for yaw and roll. Since this is present also in the cameras' measurements, it is fair to assume that these delays

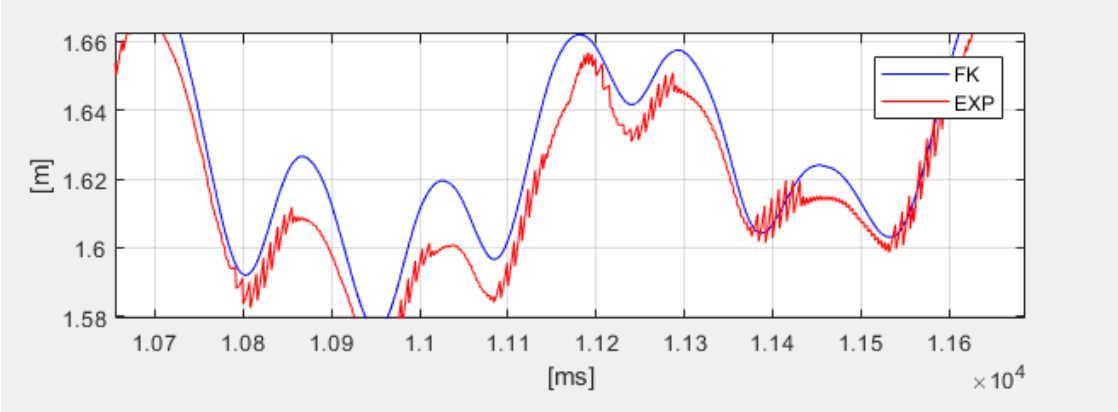


Figure 7.7: FK vs EKF output in position, axis Z, for a fast guided motion.

come from frequency limits in the communication. This should not be a problem for the majority of the applications, since high precision tasks can usually be performed without sudden changes in trajectories.

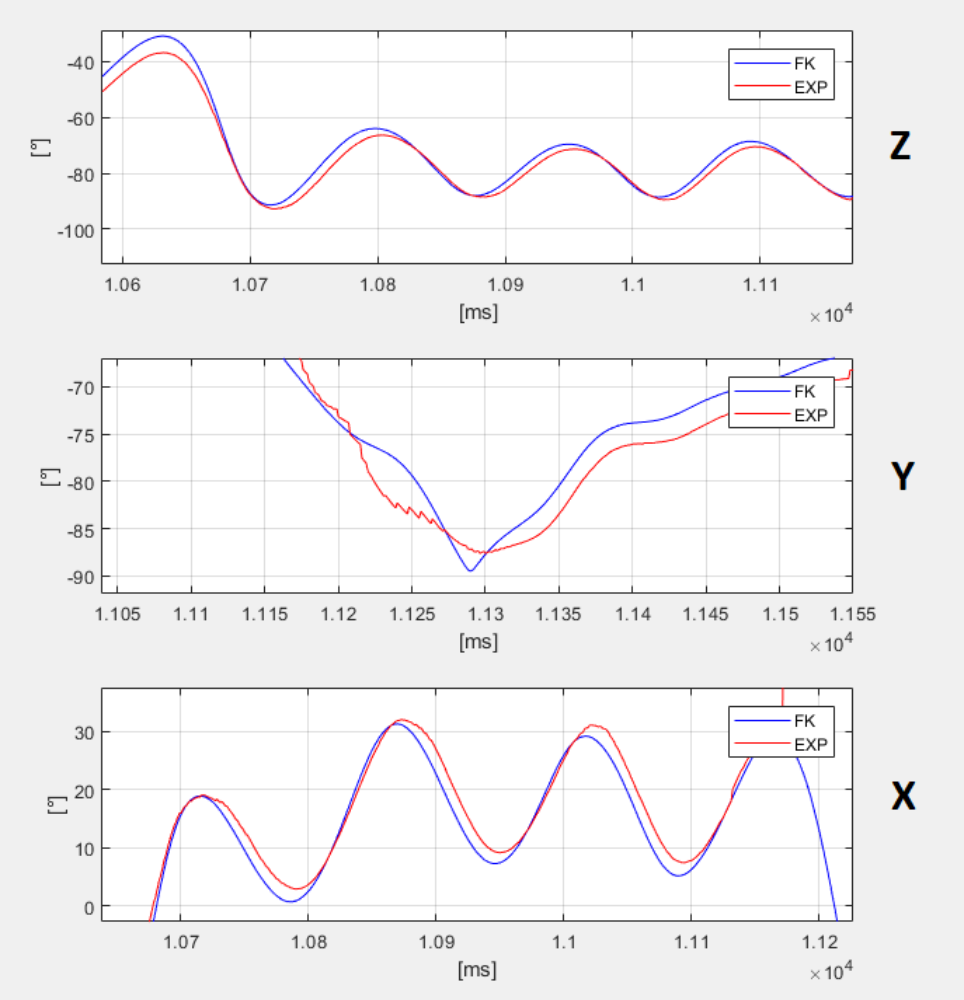


Figure 7.8: FK vs EKF output in orientation, expressed as Yaw (Z), Pitch(Y) and Roll(X) angles

Chapter 8

Conclusions and future research

Starting from the formulation of a pose estimation problem, this thesis described the whole process that lead to the implementation of a Visual Inertial Sensor Fusion in an Extended Kalman Filter framework, applied to the tool tracking of a mobile manipulator.
The project, carried out in the Corporate Research department of KUKA Deutschland GmbH, started with this work and will continue in the future with the contribution that the thesis provided.
The first results show the achievement of a robust and precise tracking algorithm that can work without the constant use of the Forward Kinematic, although requiring fast updates for rapidly changing trajectories.
Furthermore, a significant set of error sources has been identified, opening the door for improvement not only for this project, but for the whole VALERI setup.
The main problem still present is the lack of an application that could give a meaningful reference, to effectively quantify the improvement brought by the sensor fusion. Such an application has to be implemented accordingly to KUKA's internal software settings and libraries, and it can be summarized with the following needed features:

- An a-priori reference trajectory, created mathematically, to be followed first by the FK alone, then by the filtered estimates.
- A closed loop control system, involving a Joint Mapper algorithm to compute the Inverse Kinematics according to the error between reference and estimated values.
- Easily programmable motion instructions for the robot, involving repetitive movements with platform, linear axis and arm.

The cancellation of the drift problem presented in Chapter 1 should be verified and documented. According to the results obtained with this experiment, one can decide how

much time and resources to invest in further improvements. Thanks to the work done here, and in particular to the considerations in Chapter 4, 5 and 6, it is possible to list several suggestions on the future research:

1. From the transformation matrices of the frames involved in the Hand-Eye calibration, it is possible to extract position and orientation. These quantities can be inserted in the state vector of the filter for online estimation, instead of being treated as constants. Robustness could not be always guaranteed anymore, but the order of magnitude in the correction of such parameters could give an idea of the performances.

2. The Honeywell IMU available in the laboratory should be calibrated again, for measurement improvement. In order to do this, it would be needed to disassemble the sensor from the robot, and to leave it for several hours (at least 8, according to Honeywell) in a clear, flat and vibration-free surface, to collect data. Then, new bias and noise parameters could be obtained via several tools. One of them is the so called *Allan Variance*, where a deviation plot is drawn and used to differentiate the single noise contributions (bias instability, ARW etc.) and retrieve their coefficients [22].

3. According to the manufacturer, the Honeywell IMU is so precise that it can sense Earth rotation. This contribution, that eventually should to be added in the equations related to the quaternion in the EKF state, can obviously lead to drifts if ignored in high precision applications. For a brief explanation on how to insert the angular rate ω_e, see [18].

4. All the issues related to misalignments and offsets in the hardware implementation can be corrected in several ways. One can for example change the support mounted on the arm and substitute it with one having no rotational offset when attaching the IMU. CAD models of the whole flange system, from the frame E to the frame T, should be provided in order to retrieve precise measurements of the transformation matrices. Other physical offsets in the VALERI should be properly measured and inserted in the FK algorithm.

5. The numerical integration algorithm for the linear acceleration is here the main source of divergence for the IMU estimates. A better algorithm can be researched, always bearing in mind the need of online and real time estimation.

6. The whole camera setup could be improved, both in the physical mounting, with Truss systems and not unstable tripods, and in the reflection management. Although it should not be needed at this stage, the Optitrack system is actually expandable with 24 cameras instead of the current 12.

Some words should be spent on the topic of cost, since a constant goal in engineering is always the achievement of good results with low cost resources. Both the IMU and the Optitrack cameras are professional, high performant systems, and their cost limits their usage outside big and medium companies. Nevertheless, the field of application where a robot like the VALERI should work, i.e., aerospace manufacturing, justifies the need of a certain type of sensors and, while cheaper solutions should always be researched, this is not a priority with respect to other future operations like the ones listed above.

Coming back to the final consideration about this thesis, the research and development steps of the work dig into the three main fields that are normally involved in this kind of projects, i.e.,

1. The mathematical, system modeling part.
2. The software part, divided into implementation of the mathematical model and setup for the company environment.
3. The hardware part, involving the VALERI robot itself, plus camera calibration and inter-sensor communication.

In all of these parts, it has been created the environment that will be needed by future researchers, students and employees working on the project; everything is there and ready to use. Information about the code written in Matlab, Java and C++, as well as about the modelling and utilization of the available sensors with the VALERI, are now present in KUKA.

Appendix A

Appendix - Useful mathematical elements

Here, the main algorithms used throughout the thesis are presented. The following sections contain only parts of the respective topics, just enough to set an essential but solid mathematical background. For more information, the reader is suggested to check [25], whose fundamental contribution to the robotic world is universally recognized.

A.1 Homogeneous transformations

Many times, during the implementation of the EKF, it has been necessary to transform position and orientation into transformation matrices and viceversa. It is certainly easier, in fact, to have a unified element for faster and more robust calculations.
Referring to Figure A.1, the position of the point P in the frame 0 is expressed by:

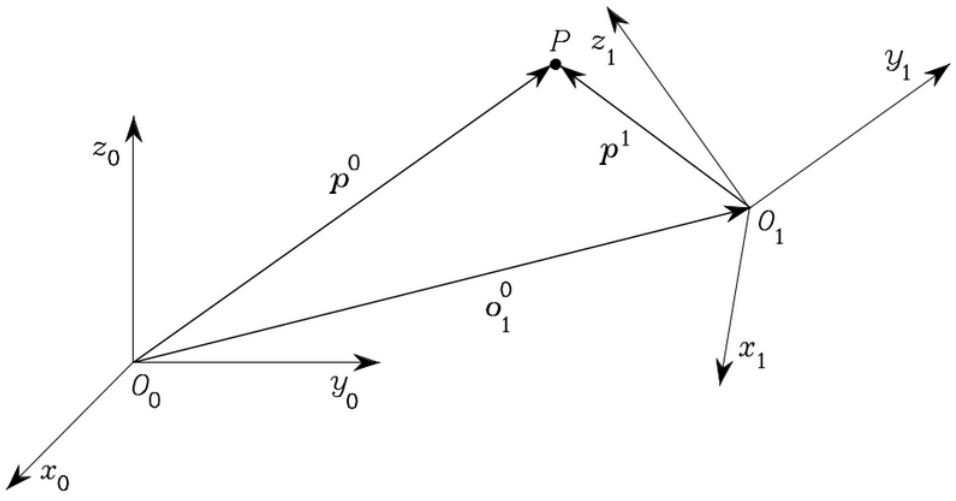


Figure A.1: Two link planar structure with two revolute joints.

$$p^0 = o_1^0 + R_1^0 p^1 \tag{A.1}$$

where the contribution of a translational and a rotational part is present. When dealing with multiple frames, thus multiple transformations, it is actually better to combine the equation (A.1) into a more compact form:

$$\tilde{p}^0 = T_1^0 \tilde{p}^1 \tag{A.2}$$

where the vector p is extended into its homogeneous representation

$$\tilde{p} = \begin{bmatrix} p \\ 1 \end{bmatrix}$$

and T_1^0 is composed in the following way:

$$T_1^0 = \begin{bmatrix} R_1^0 & p_1^0 \\ 0^T & 1 \end{bmatrix}$$

The advantage of this representation can be seen also when writing equations (A.1) and (A.2) for the inverse relation:

$$p^1 = -R_0^1 o_1^0 + R_0^1 p^0 \tag{A.3}$$

$$\tilde{p}^1 = T_0^1 \tilde{p}^0 = (T_1^0)^{-1} \tilde{p}^0 \tag{A.4}$$

Here it is shown how one can compute the inverse transformation with the same elements as the direct one, in a compact form able to handle every possible particular case.

A.1.1 Example of computation: T_I^E

A matrix T_B^A can be constructed knowing the position and the orientation of one frame with respect to another. It is the case presented in this thesis, for the measurement transformations performed in the filter, to represent, for example, the acceleration of the IMU or the pose of the tool in the World Frame.

An example is when the matrix T_I^E, from the end-effector to the IMU frame, had to be computed in Chapter 4: having measured a translational offset of 20.85 cm along the Z axis of the end-effector frame, plus the combination of a coordinate change and a rotation of $\theta = 5°$, one can obtain:

$$T_I^E = \begin{bmatrix} R_I^E & p_I^E \\ 0^T & 1 \end{bmatrix} \tag{A.5}$$

where $p_I^E = \begin{bmatrix} 0 \\ 0 \\ 0.2085 \end{bmatrix}$ and R_I^E is obtained as:

$$R_I^E = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & -s\theta \\ 0 & s\theta & c\theta \end{bmatrix} \tag{A.6}$$

The last equation is composed by two rotation terms. The first is a trivial coordinate swap, computed column by column, that places for example the IMU X axis along the end-effector Z axis; the second is the rotation of θ along the IMU X axis.

This example showed how a typical transformation matrix construction is performed. Actually, there is one more intermediate step that consists in the computation of a rotation matrix R from a quaternion measurement. This will be explained in section 3 along with the other quaternion related equations.

A.2 Forward Kinematics

Constant references to this fundamental algorithm in robotics are present in every chapter. The purpose of the FK is to determine the end-effector pose as a function of the joint variables, of a robotic arm manipulator. This will result in the homogeneous transformation matrix from the base to the end-effector frame $\boldsymbol{T}_e^b(\boldsymbol{q})$, where in this case $\boldsymbol{q}$ is a vector containing the joint angles of rotation, and should not be confused with a quaternion.
The FK can be computed geometrically according to the type of structure of the robot. A trivial example is the two arm planar manipulator shown in Figure A.2. Here, one can exploit trigonometrical rules to easily obtain:

$$\boldsymbol{T}_e^b(\boldsymbol{q}) = \begin{bmatrix} 0 & s_{12} & c_{12} & a_1c_1 + a_2c_{12} \\ 0 & -c_{12} & s_{12} & a_1s_1 + a_2s_{12} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where s_x and c_x indicate the sine and cosine of the angle θ_x, s_{12} and c_{12} indicate sine and cosine of the angle $(\theta_1 + \theta_2)$ and a_x is the length of the link x.
The matrix above can be computed visually column by column, or as a result of two intermediate transformations passing through the middle joint.

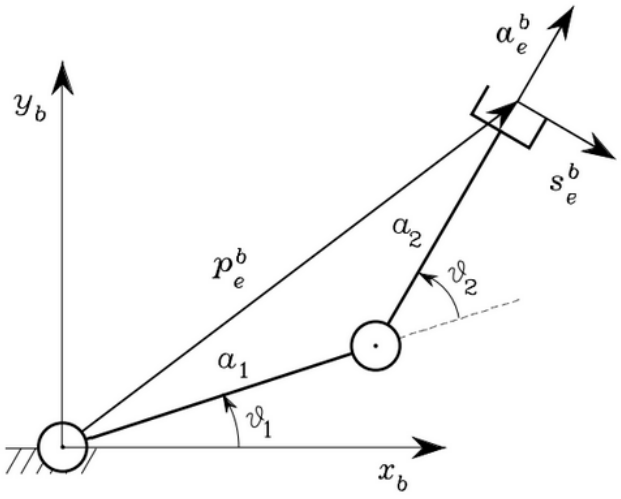


Figure A.2: Two link planar structure with two revolute joints.

Its general structure can be summarized in the following way:

$$\boldsymbol{T}_e^b(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{n}_e^b(\boldsymbol{q}) & \boldsymbol{s}_e^b(\boldsymbol{q}) & \boldsymbol{a}_e^b(\boldsymbol{q}) & \boldsymbol{p}_e^b(\boldsymbol{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where **n**, **s** and **a** are versors conventionally chosen with respect to a gripping tool. Refering to Figure A.2, **a** is directed towards the object to be grasped, **s** is orthogonal to **a** along the gripping plane, and **n** is orthogonal to the other two, so that the total frame is right-handed. It can finally be noted how the last column represents the position $\boldsymbol{p}_e^b(\boldsymbol{q})$), while the submatrix

$$\boldsymbol{R}_e^b(q) = \begin{bmatrix} 0 & s_{12} & c_{12} \\ 0 & -c_{12} & s_{12} \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{n}_e^b(\boldsymbol{q}) & \boldsymbol{s}_e^b(\boldsymbol{q}) & \boldsymbol{a}_e^b(\boldsymbol{q}) \end{bmatrix}$$

represents a rotation of the TCP with respect to the base frame.

Along the various chapters, the term Forward Kinematics has been often used to indicate the transformation along the whole kinematic chain of the VALERI, which has 9 revolute joints and 3 prismatic joints, for a total of 12 variables. For complex structures like the VALERI, it would be too difficult to use an immediate and intuitive approach as the one used for the two link planar robot.

In robotics, a general method of representing poses and reference frames, virtually placed along joints, is the so called *Denavit-Hartenberg convention* (DH). With two constant and two variable parameters assigned to each joint and representing translational and rotational offsets, the relations between intermediate frames are defined with homogeneous transformations related to these parameters. The final pose is obtained multiplying these matrices

throughout the whole chain.

For typical manipulation structures, as 6 or 7 DOF arms, the computation of the DH parameters has of course already been set in the literature. The VALERI robot makes no exception [28], [29], and its kinematic chain and DH parameters are reported in Figure A.3 and Table A.1 respectively.

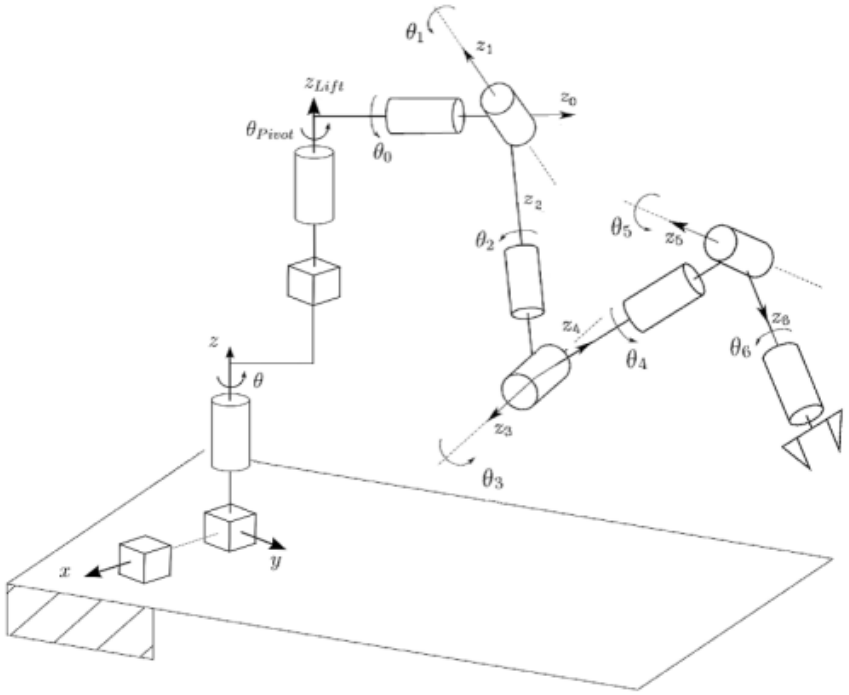


Figure A.3: Kinematic chain of the KUKA VALERI OmniRob. Note the convention that places cilinders for revolute joints, and cubes for prismatic ones.

Joint	a [m]	α [rad]	d [m]	θ [rad]
1	0	-$\pi/2$	q1	0
2	0	$\pi/2$	q2	$\pi/2$
3	0.264	0.04	0.805	q3
4	0.151	1.7	0	q4
5	0	$\pi/2$	q5	$\pi/2$
6	0	-$\pi/2$	0.31	q6
7	0	$\pi/2$	0	q7
8	0	$\pi/2$	0.4	q8
9	0	-$\pi/2$	0	q9
10	0	-$\pi/2$	0.39	q10
11	0	$\pi/2$	0	q11
12	0	0	0	q12

Table A.1: DH parameters for the VALERI. The variable parameters are written as q1, q2... qn, where n is the number of the joint.

A.3 Quaternion algebra

The advantages obtained using quaternions instead of Euler angles have been listed in Chapter 3. This section describes the structure of such a representation, as well as the main algorithm used in the EKF.

A unit quaternion is defined with four parameters representing a rotation of an angle θ along an axis $\boldsymbol{r}$, as $q = (\boldsymbol{\epsilon}, \eta)$, where $\boldsymbol{\epsilon} = sin(\theta/2)\boldsymbol{r}$ and $\eta = cos(\theta/2)$. Among these four parameter, η is the scalar part, while $\boldsymbol{\epsilon}$ is the vectorial part. The main feature of the unit quaternion is the following constraint:

$$\epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 + \eta^2 = 1 \tag{A.7}$$

so that a rotation of $-\theta$ along $-\boldsymbol{r}$ has the same quaternion as a rotation of θ along $\boldsymbol{r}$.

Listing all the characteristics and properties normally explained in quaternion's literature would need a whole long chapter. Here, only the used operations are reported, without any proof. All the following parts can be found more in detail in [18] and [19].

A.3.1 Quaternion to rotation matrix and viceversa

These results come from the axis-angle representation and are here extended to the complex domain of quaternions. The rotation matrix corresponding to q is

$$R(\boldsymbol{\epsilon}, \eta) = \begin{bmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x\epsilon_y - \eta\epsilon_z) & 2(\epsilon_x\epsilon_z + \eta\epsilon_y) \\ 2(\epsilon_x\epsilon_y + \eta\epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y\epsilon_z - \eta\epsilon_x) \\ 2(\epsilon_x\epsilon_z - \eta\epsilon_y) & 2(\epsilon_y\epsilon_z + \eta\epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{bmatrix}$$

that can also be written in a more compact form as:

$$R(\boldsymbol{\epsilon}, \eta) = \boldsymbol{I}_3 - 2\eta\lfloor\boldsymbol{\epsilon}\rfloor + 2\lfloor\boldsymbol{\epsilon}\rfloor^2 \tag{A.8}$$

The inverse operation can be performed in several ways. The one adopted consists in the computation of the rotation matrix's trace, i.e., the sum of its diagonal elements:

$$T = R(1,1) + R(2,2) + R(3,3)$$

Then, one of the quaternion parameters is related to this value, while the other three follows accordingly. For example

$$\epsilon_z = \sqrt{1 + 2R(3,3) - T}/2 \tag{A.9}$$

$$\epsilon_x = (R(1,3) + R(3,1))/4\epsilon_z \tag{A.10}$$

$$\epsilon_y = (R(2,3) + R(3,2))/4\epsilon_z \tag{A.11}$$

$$\eta = (R(1,2) - R(2,1))/4\epsilon_z \tag{A.12}$$

A.3.2 First Order integration

Assuming a linear evolution of the angular rate ω during the integration interval ΔT, the final first order integration formula is:

$$\boldsymbol{q}(t+1) = (exp(\frac{1}{2}\Omega(\bar{\omega})\Delta T) + \frac{1}{48}(\Omega_1 - \Omega_2)\Delta T^2)\boldsymbol{q}(t) \tag{A.13}$$

$$\Omega_1 = \Omega(\omega(t+1))\Omega(\omega(t)) \tag{A.14}$$

$$\Omega_2 = \Omega(\omega(t))\Omega(\omega(t+1)) \tag{A.15}$$

where $\bar{\omega} = \frac{\omega(t+1)+\omega(t)}{2}$ is the average angular velocity and

$$\Omega(\omega) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}$$

is the 4x4 version of a skew-symmetric matrix.

Using this algorithm requires saving the gyroscope measure ω of the previous step too, increasing precision with respect to the zeroth order integration that assumes a constant angular rate.

A.3.3 Product and Inverse

If a quaternion is represented as

$$q = q1\boldsymbol{i} + q2\boldsymbol{j} + q3\boldsymbol{k} + q4$$

then the product between two quaternions $\mathbf{q}$ and $\mathbf{p}$ can be expressed as

$$q \otimes p = (q1\boldsymbol{i} + q2\boldsymbol{j} + q3\boldsymbol{k} + q4)(p1\boldsymbol{i} + p2\boldsymbol{j} + p3\boldsymbol{k} + p4)$$

Alternatively, for matrix computations, one can exploit the Ω matrix introduced in the previous subsection, obtaining

$$q \otimes p = \Omega(q) \begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \end{bmatrix} = \begin{bmatrix} q4 & q3 & -q2 & q1 \\ -q3 & q4 & q1 & q2 \\ q2 & -q1 & q4 & q3 \\ -q1 & -q2 & -q3 & q4 \end{bmatrix} \begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \end{bmatrix} \tag{A.16}$$

The inverse quaternion, used for example in the product computed to obtain the difference between a measurement and an estimate, is defined as $q^{-1} = q(-\boldsymbol{\epsilon}, \eta)$. Then, one can easily compute $\tilde{z}_q = q_m \otimes \hat{q}^{-1}$ in Chapter 4.

Bibliography

Sensor Fusion and EKF

[1] A. Garulli, A. Giannitrapani, A. Rossi and A. Vicino, *Mobile robot SLAM for line-based environment representaion*, Dipartimento di Ingegneria dell'Informazione, Università di Siena, Siena, 2006.

[2] O. Wulf, K.O. Arras, H.I. Christensen and B. Wagner, *2D mapping of cluttered indoor environments by means of 3D perception*, in: Proceedings of the 2004 IEEE International Conference on Robotics and Automation, New Orleans, LA, April 2004.

[3] K. Yousif, A. Bab-Hadiashar and R. Hoseinnezhad, *An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics*, Springer Science+Business Media, Singapore, 2015.

[4] D. Cole and P. Newman, *Using laser range date for 3D slam in outdoor environments.*, in: IEEE International Conference on Robotics and Automation, Orlando, FL, 2006.

[5] A. Davison, *Real-time simultaneous localisation and mapping with a single camera.*, in: Ninth IEEE International Conference on Computer Vision, 2003.

[6] G. Nuetzi, S. Weiss, D. Scaramuzza and R. Siegwart, *Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM.*, ETH Autonomous Systems Laboratory, 8092 Zurich, Switzerland.

[7] F. Caron, E. Duflos, D. Pomorski and P. Vanheeghe, *GPS/IMU Data Fusion using Multisensor Kalman Filtering: Introduction of Contextual Aspects*, Ecole Centrale de Lille Cité Scientifique and Université Lille I, Villeneuve d'Ascq Cedex, France, 2004.

[8] B.M. Nguyen, Y. Wang, H. Fujimoto and Y. Hori, *Advanced Multi-rate Kalman Filter for Double Layer State Estimator of Electric Vehicle Based on Single Antenna GPS and Dynamic Sensors*, Department of Advanced Energy, University of Tokyo, Japan.

[9] J. Stephen, *Development of a multisensor GNSS based vehicle navigation system*, PhD Thesis, Department of Geomatics Engineering, Univerity of Calgary, 2000.

[10] D. McNeil Mayhew, *Multi-rate sensor fusion for GPS navigation using Kalman filtering*, PhD Thesis, Department of Electrical Engineering, Virginia Polytechnic Institute and State University, 1999.

[11] A.T. Erdem and A.O. Ercan, *Fusing Inertial Sensor Data in an Extended Kalman*

Filter for 3D Camera Tracking, IEEE Transaction on Image Processing, VOL.24, NO.2, February 2015.

[12] M. Kok, J.D. Hol and T.B. Schoen, *Using Inertial Sensors for Position and Orientation Estimation*, Department of Engineering, University of Cambridge, 2017.

[13] M.B. Alatise and G.P. Hancke, *Pose Estimation of a Mobile robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter*, University of Pretoria, Pretoria 0028, South Africa and City University of Hong Kong, Hong Kong, China, September 2017.

[14] E. Kiriy and M. Buehler, *Three-state Extended Kalman Filter for Mobile Robot Localization*, 2002.

[15] S.J. julier and J.K. Uhlmann , *New extension of the Kalman filter to nonlinear systems*, in: AeroSense'97. International Society for Optics and Photonics, 1997.

[16] J. Hol, T. Schoen, H. Luinge, P. Slycke and F. Gustafsson , *Robust real-time tracking by fusing measurements from inertial and vision sensors*, Journal of real-time Image Processing, Department of Electrical Engineering, Linkopings universitet, September 2007.

[17] S.M. Weiss, *Vision Based Navigation for Micro Helicopters* PhD Thesis, ETH Zurich, Chapter 3, 2012.

[18] J. Solà, *Quaternion kinematics for the error-state Kalman filter*, Cornell University Library, October 12, 2017.

[19] N. Trawny and S.I. Roumeliotis, *Indirect Kalman filter for 3D attitude estimation*, University of Minnesota, Department of Computer Science and Engineering, Tech. Rep. 2005-002, 2005.

Noise Modeling

[20] Y. Zhao, M. Horemuz and L.E. Sjoeberg, *Stochastic Modeling and Analysis of IMU Sensor Errors*, Division of Geodesy and Geoinformatics, Royal Institute of Technology (KTH), Stockholm, Sweden, 2011.

[21] P. Pektov and T.Slavov, *Stochastic Modeling of MEMS Inertial Sensors*, CYBERNETICS AND INFORMATION TECHNOLOGIES, Volume 10, No 2, Department of Automatics, Technical University of Sofia, Sofia, 2010.

[22] N. El-Sheimy, H. Hou and X.Niu, *Analyis and Modeling of Inerial Sensors Using Allan Variance*, IEEE Transaction on Instrumentation and Measurement, Volume 57, No. 1, January 2008.

[23] C.F. Van Loan, *Computing Integrals involving the matrix exponential*, IEEE Transactions on Automatic Control, June 1978.

[24] J. Bernstein, *An Overview of MEMS Inertial Sensing Technology*, Sensors Weekly, February 1, 2003.

Robotics

[25] B. Siciliano, L. Sciavicco, L. Villani and G. Orioli, *Robotics - Modelling, Planning and Control*, Springer-Verlag London, Advanced Textbooks in Control and Signal Processing, 2009.

[26] H. Zhuang and Y.C. Shiu, *A noise tolerant algorithm for wrist-mounted robotic sensor calibration with or without sensor orientation measurement*, in: Proceedings of the 1992 lEEE/RSJ International Conference on Intelligent Robots and Systems, volume 2, pages 1095–1100, 1992.

[27] P. Corke, *Robotics, Vision and Control - Fundamental Algorithms In MATLAB*, Springer International Publishing, 2017.

[28] H. Wei, *Operational Space Control for VALERI Robot - Dynamic Control of Highly Redundandt Mobile Manipulator*, Master Thesis, KUKA Deutschland GmbH, Technicsche Universitaet Berlin, Berlin, November 2017.

[29] M. Riedel, *VALERI: OmniRob mit Linearachse und LBR*, Kurzdokumentation: Konzept und Entwicklung, KUKA Deutschland GmbH, 2013.

[30] J.S. Freudenberg, *The Virtual Spring Mass System*, EECS 461, Universty of Michigan, 2008.