Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for bygg- og miljøteknikk

Odd Martin Rognerud

Real-time geographical pose visualization system for the Augmented Reality Cloud

Masteroppgave i
Januar

NTNU
Kunnskap for en bedre verden

NTNU – Trondheim
Norwegian University of
Science and Technology

Real-time geographical pose visualization system for the Augmented Reality Cloud

Odd Martin Rognerud

supervised by

Terje Midtbø
Jan-Erik Vinje

January 30, 2019

Summary

This thesis explores the technologies that power the tracking and mapping under the hood of modern Augmented Reality systems, as well as previous iterations of Mobile Augmented Reality. Special attention is given to the so-called Augmented Reality Cloud and the eco-system around it, a crucial infrastructure piece for enabling persistent, interoperable and multi-user Augmented Reality experiences. Furthermore, by using principals from Computer Vision, GIS and Information Technology, a real-time system is developed that visualizes mobile device geopose on a virtual globe running in a WebGL enabled browser. Technologies applied include OpenCV, WebSockets, Cesium.js and Node.js on in three different environments; web, server and Android.

Table of Contents

List of Tables

List of Figures

Abbreviations

AR	=	Augmented Reality
VR	=	Virtual Reality
MR	=	Mixed Reality
MAR	=	Mobile Augmented Reality
OARC	=	Open Augmented Reality Cloud
WS / WSS	=	WebSocket / Secure WebSocket
SDK	=	Software Development Kit
SDO	=	Standard Development Organization
SSE	=	Server Sent Events
ENU	=	East-North-Up
ECEF	=	Earth-centered Earth-fixed
GeoPose	=	Geographical Pose
CV	=	Computer Vision
IoT	=	Internet of Things
IMU	=	Inertial Measurement Unit
MEMS	=	Micro Electro Mechanical Sensor
DST	=	Digital Signal Processing
VO	=	Visual Odometry
VIO	=	Visual-Inertial Odometry
VINS	=	Visual-Inertial Navigation system
SLAM	=	Simulatnous Location and Mapping
PTAM	=	Parallell Tracking and Mapping
V-SLAM	=	Visual Simulatnous Location and Mapping
SfM	=	Structure from motion
EKF	=	Extended Kalman Filter
MSKF	=	Multi-State Kalman Filter
MLE	=	Maximum Likelihood estimate
MAP	=	Maximum a Posteriori
BA	=	Bundle Adjustment
RANSAC	=	Random Sampling Consensus
POSIT	=	Pose from Orthography and Scaling with Iteration
ICP	=	Iterative Closest Point
RMSE	=	Root Mean Square Error
SSD	=	Sum of Squared Differences
NCC	=	Normilized Cross-Correlation

Chapter 1

Introduction

1.1 Background and motivation

Recent advancements in web technologies, 3D graphics and computational capacities have made it possible to produce richer and more nature-like visualizations. Three-dimensional visualizations have the ability to reveal even more than before about our geospatial data, such as positioning and orientation between different objects. 3D visualization is a integral part of data visualization in many sectors, including data science, construction, games and simulations. Traditionally, Geographical Information Systems (GIS) have used the 2D space of maps to visualize geospatial data. Nowadays, more and more GIS tools ship with 3D capabilites, moving away from traditional 2D maps. Todays smartphones incorporate cheap and numerous micro-electro-mechanical system (MEMS) sensors, among them compass, GPS/GNSS, gyroscop, temperature sensor and proximity sensors. These sensors supply valuable data for applications in localization, mapping, tracking, IoT, navigation and more. Mobile phone cameras in tandem with machine vision and image processing techniques play an important role in determining the relative pose between an observed object and the camera, as well as object detection and mapping. Combining the IMU, GPS/GNSS, network chip and digital camera provide large amounts of data that can be used to solve the for a highly accurate geographical pose of a device, as well as provide data for cloud based models that can improve localization upon todays GPS based systems, reducing errors down to the centimeter. The problem of accurate localization is central to Augmented Reality, which is explored as well.
Norkarts initiative in collaboration with the Norwegian Mapping Authority has shown great potential in the cross-section between Augmented Reality and geospatial applications. Other international technology firms also invest heavily in Augmented Reality and Spatial Computing, preparing for a world in which machines truly can understand its environment.

1.1.1 The Open Augmented Reality Cloud

The Open Augmented Reality Cloud, OARC, is a open-sourced, collaborative, privacy centered and interoperable *vision* of the AR Cloud in the future. The amounts of data that will be needed to build a 1:1 map of the world is staggering, requiring a careful evaluation of how to best reach a cloud infrastructure for machines to orient in the world. Adoption of AR applications struggle today. Fundamentally, there are several key issues with most AR applications:

Lack of persistence - Augmented content does not persist over time and place. No continuity is maintained in-between AR sessions because the application is not aware of its geospatial context. Augmented objects are not ensured to appear in the same spot as it previously was.

Poor immersion - Occlusion is an effect of object obstruction, that is, any augmented view should be obstructed when camera is moved outside the line of sight. Most AR apps does not occlude correctly, which ruins the immersive experience. e.g. An augmented game avatar should vanish if it is moved behind an object.

Synchronization across devices and vendors - Extending the collaborative, shared experience even further. Several vendors of augmented reality content and various devices and platforms will co-exist.

Shared experiences - AR cannot exist in a vacuum. AR applications will struggle to succeed unless experiences can be shared and multi-user interactivity in real-time. Small-scale and single-user applications will not and cannot usher the adoption of AR.

Poor User experience - AR needs to ”just work”. User experience is ruined if two or more users are required to any kind of SLAM synchronization before using AR applications, or other type of set-up.

All of the above issues can be solved by a common infrastructure known as the Augmented Reality Cloud. Research and development for the OARC is defined into three distinct parts. Object and pose detection on mobile devices, cloud infrastructure and visualization systems for research and development.

1.2 Research goals

What is the current state of mobile Augmented Reality and augmented reality cloud infrastructures.

Propose a prototype system to visualize geographical pose in a naturalistic, web-based, 3D environment, using a virtual globe.

Review different techniques and algorithms for representing mobile pose generated by cameras and sensors commonly found on smartphones.

Utilize real-time technologies to communicate between components in a multi-component system.

1.3 Structure

The structure of this project is as as follows. Chapter 2 introduces key concepts for the reader. The fundamentals of Augmented Reality and the Open Augmented Reality Cloud, state-of-the-art 3d technologies on the web, real-time technologies on the web, computer vision concepts of tracking and mapping. Chapter 3 includes a literary review of the current state in this area of research. Chapter 4 involves exploring key issues in Mobile Augmented reality with the Open Augmented Reality as a use case. Chapter 5 introduces a architecture, applications and development of a visualization system, that can be useful for further research and development on the OARC. Chapter 6 explores two ways of generating a geographical pose and visualizing the result in real-time. Chapter 7 will round off this work and discuss the take-aways.

Chapter 2

Foundations

2.1 Augmented Reality

Augmented Reality (AR) is a real-time direct or indirect view of physical real-world environment that has been superimposed upon or composed with the real world. Milgram and Kishino [56] places Augmented Reality along a virtual continuum. The continuum provides an axis along which we can place concepts similar to AR. We see that it spans from more reality-like, to more virtual-like. AR has evolved a lot since its first studied in the 1950s.

One of the fundamental problems of AR is tracking, the calculation of the 6DoF of a camera i relation to an object of interest. With the relative orientation and position, the AR system can then render virtual objects precisely.Azuma proposes a technology-agnostic definition of three characteristics.

- it combines the real and the virtual
- it is interactive in real time
- it is registered in 3D

Milgram and Kishino and Azuma have together provided the taxonomy which is currently the foundation of modern AR.

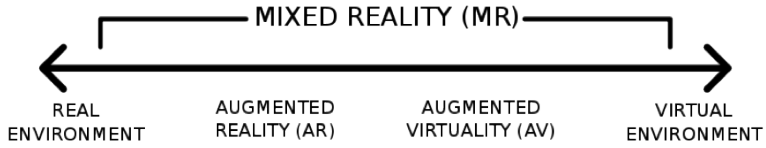


Figure 2.1: Mixed Reality spectrum [56].

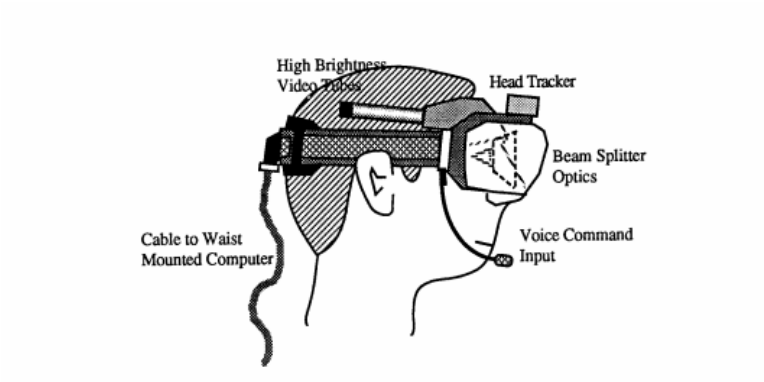


Figure 2.2: Early schematic of a HUD-based AR system [14]

Although central, vision based system are not the only areas where AR can be applied. Systems based on touch, smell, hearing and other senses have been proposed. Haptic systems, in which the sense of touch are recreated through forces, vibrations and motions, are frequently found in gaming or systems for the audio-visually impaired. In the 2002 meta cookie study Narumi et al. [61] proposed a illusion based AR/VR system in which the perceived taste of a cookie is changed by overlaying visual and olfactory information onto a real cookie. For the remainder of this work, we will consequently refer to visual Augmented Reality. Carmigniani and Furht [13] defines four main devices that augmented reality consists of, as we will discover in the following sections.

2.1.1 Displays

Severel typse of displays are used in AR. The main types are head mounted displays, handheld displays and spatial displays. Most commonly found today are variants that are handheld (eg. mobile phones) or headsets (eg. Microsoft HoloLens). Spatial AR rely on video-projections, optical elements and holograms that physically display graphical information directly on physical objects. For the remainder of this work, we will mainly refer to handheld displays.

2.1.2 Input devices

Computer vision, robotics and photogrammetry have over the years developed several different tracking methods. Researchers divide tracking into categories based on the type of equipment used. Sensor tracking methods, visual tracking methods and hybrid methods.

2.1.3 Computers

The computers are responsible for processing data from the input devices and tracking devices. Without the computers, Augmented Reality could not be able to augment data correctly. The main hurdle regarding computers is the computational load that Augmented

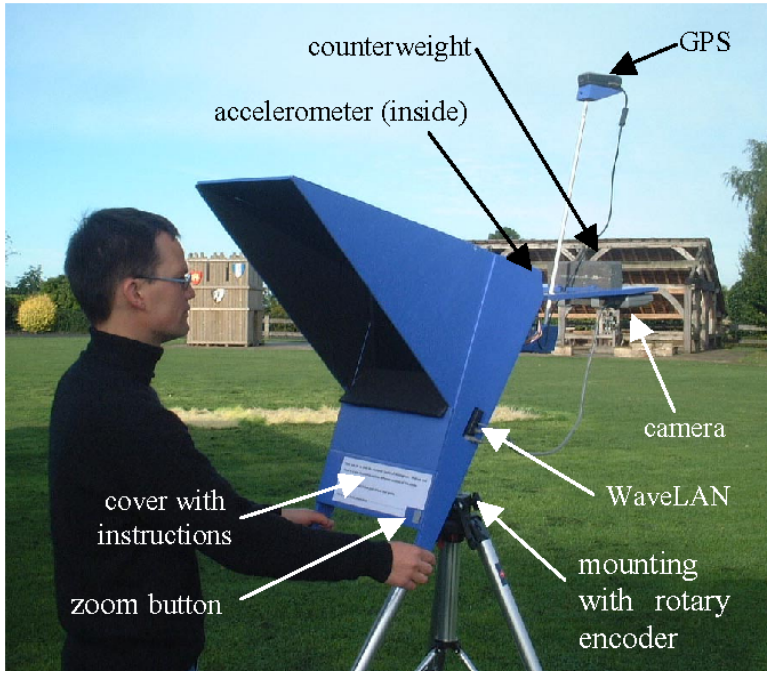


Figure 2.3: The Augurscope, a Mixed reality system [76]

Reality can impose on a system. The frame-rate is directly tied to the systems ability to process data, and systems with low or poor framerate are hardly usable. The issue with the computers of Augmented Reality system is not only computing power, but also the power consumption and heat production of the system.

2.2 Computer Vision

The field of computer vision is quintessential for the development of AR. The fundamental problems of localization and mapping is a subset of many disciplines found in CV. Understanding different concepts from CV is important to grasp how a robot can localize through visual input.

2.2.1 Fiducial markers

Fiducial markers help resolve the integration of the real world view and the super-imposed, virtual object. The markers provide a high-contrast background, such as a black square, that encapsulates a white geometric figure with predefined properties. These properties range in size, shape and color to make them easy identifiable. Markers are popular because of the their reliability in varying light conditions. Earlier AR SDK's, such as ARToolkit [41], ARTag [25] relied on marker-based tracking to determine camera pose. Easy-to-use and available toolkits such as ARToolkit and ARTag made marker-based tracking in AR

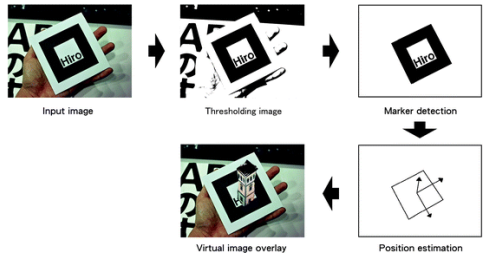


Figure 2.4: Marker detection process[40].

applications very popular. Other methods, such as feature-based and model-based tracking are more challenging because the environment is unknown. Pose estimation requires more data, estimation drift occur and the system is unable to reliably define the coordinate axis orientation. Additionally, the scale is not possible to deduce only through visual observations. Fiducial markers solve these problems by providing markers that are easily detected by machine vision techniques such as pattern recognition and image processing. Markers contain information about scale and orientation. Model-based tracking compares the images to a predefined model, while feature-based tracking detects patterns and their movements between image frames.

Kan et al. [40] provides a visual process diagram exemplifying the marker detection process. A typical marker detection process looks as follows:

- Image Acquisition - acquisition of an intensity image.
- Preprocessing - low level image processing, undistortion, line detection and corner detection
- Discriminate potential markers - fast rejection of non-markers, fast acceptance of potential markers.
- Identification and decoding of markers - template matching, decoding.
- Pose Calculation - estimation of marker pose, iterative pose calculation.

AruCo tag, developed by Garrido-Jurado et al. [30], developed a configurable fiducial marker-based system for camera pose estimations. The library is open-sourced and implemented in OpenCVs extended contributions module. Among improvements introduced in AruCo tag is:

Occlusion handling - the authors incorporates a color map of the marker board to compute an occlusion mask by color segmentation. Marker boards contain several marker in a common reference frame.

Automatic dictionary generation - previous work impose fixed dictionaries. Fiducial markers rely on unique identification by binary codes, and the accompanying markers

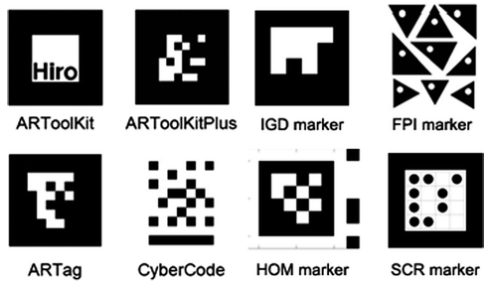


Figure 2.5: Common markers, including ARTag and ARToolkit. [40]

inter-marker distance dictates how error-prone the marker is. A algorithm for maximizing the inter-marker distance and number of bit transitions of a configurable dictionary is proposed.

2.2.2 Camera calibration

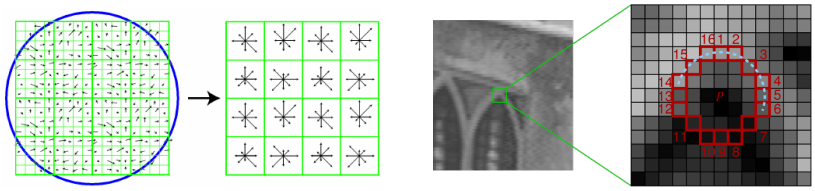
The pinhole camera model simplifies how images are captured with advanced digital cameras and enables us to map from the world frame to the camera frame. Cameras calibration is necessary to determine the relation between a cameras natural units (px) and metric units (mm). Common issues with uncalibrated cameras is distortion, because light passes through a lens rather than a pinhole. Homography estimation becomes more accurate when using undistorted images.

$$K = \begin{pmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

The intrinsic parameters are defined by Hartley and Zisserman as f_x and f_y correspond to 2D scaling, defined by the focal length. p_x and p_y correspond to 2D translation, defined by the principal point offset. s defines shear. The camera matrix K is a component of the *Projective matrix*, which is needed for projective 3D-2D mapping. The algorithm employed by OpenCV, Matlab and countless others was proposed in the famous 2000 paper *A flexible new technique for camera calibration* Zhang [88]

2.2.3 Feature detection and description

One of the widely used techniques in determining structure in a image. Feature points, also referred to as interest points in the literature, are distinct parts of images that can be found with a certain accuracy, such as edges, corners, blobs and T-junctions. Clusters of points sharing a set of mathematical constraints can form a virtual plane corresponding to the real world. Furthermore, the neighbourhood of feature points are examined to distinguish numerous feature points. Every feature point is assigned a probability, and is regarded as robust if they are transformation invariant.



(a) SIFT gradients to descriptor **(b)** FAST Corner

Figure 2.6: Various illustrations, taken from their respective papers.

The next section presents feature detectors important to CV, covering the ones important to AR. Feature detection and tracking have been subject of research for a long time. In this section we choose to skip edge detection, rather focusing on the chronological order of important feature detectors and descriptors for CV. Moravec [57] showcased the first corner detector, defining a corner as a point of low self-similarity using Sum of Squared Differences (SSD). Self-similarities is measured in a window around 4-neighborhood the current pixel. Moravecs Corner Detector is non-isotropic, unable to detect edges outside or across the vertical, horizontal or two diagonals. Harris and Stephens [34] solved this by regarding the differential intensity in all directions using the first order Taylor polynomial and partial derivatives in the image. Every window receives a score, according to some score function R. Further improvements were made with the Shi-Tomasi *good-features-to-track* [78], simply by switching the Harris corner score function $R = \lambda_1\lambda_2 - \kappa(\lambda_1 - \lambda_2)^2$ to $R = \min(\lambda_1, \lambda_2)$, where $\lambda_{1,2}$ is the eigenvalues of the structure tensor matrix from SSD.
Another breakthrough came with Scale-Invariant Keypoints, SIFT features, from Lowe [52]. Interest points are detected using the approximate Difference-Of-Gaussian (DoG) operator, by finding the local extrema in a small window. Each interest point produces a feature vector, and the local neighborhood is examined at different scales. The keypoint descriptor contains a 8 bin orientation histogram of each 4x4 sub-block in a 16x16 pixel neighborhood around a keypoint, yielding a 128 element feature vector as a descriptor. SIFT is a complex feature detector, but is still one of the most robust feature detectors today.
Although SIFT is robust and accurate, SIFT suffers from high computational cost. Speeded Up Robust Features (SURF) from Bay et al. [6] proposed a feature detector based on a Box Filter rather than Difference-of-Gaussian operator. This approach uses minimal information for features. Partly inspired by SIFT, SURF computes features more efficiently and the resulting 64-dimensional feature vector computes significantly faster. SURF performs equally good in cases of with blur or rotation, while not so good in viewpoint or illumination change. Furthermore, SIFT and SURF recently became proprietary technologies and requires a licence to use. Several derivatives of SIFT and SURF have been developed since their initial papers, too many to review here. We refer to Salahat and Qasaimeh [73] for a concise overview.

Real-time applications such as mobile SLAM and VO require even more efficient fea-

Descriptor	Type	Distance measure
BRIEF BRISK AKAZE	Binary string	Hamming distance
SIFT SURF	Floating-point vector	Euclidean, Chi Square Intersection, Bhattacharyya

Table 2.1: Descriptor matching metric overview

ture detectors, a requirement that spawned the FAST algorithm [71]. In this scheme, interest points are detected by applying a segmentation test considering a 16-pixel circle around a interest point p. If a set of n continuous pixels are either darker $I_p - t$ or brighter than $I_p + t$ the interest point plus some threshold t, the interest point is classified as a corner. Selecting $n = 12$, as proposed in the original paper, enables a high-speed test regarding the four compass directions, in which three of the four pixels must be accepted. This test quickly discard non-corners, additionally improving the speed of the algorithm. Several implementations of FAST exists, and the authors suggest applying machine learning to generalize the algorithm for $n < 12$ on a domain-specific set of images by use of a decision tree classifier and entropy minimization.

Binary Robust Independent Elementary Features, BRIEF, described in [12] introduced binary descriptors, vastly reducing the computation times for feature matching, but performs badly under rotation. ORB, Oriented FAST and Rotated Brief, is a fusion of FAST keypoints and BRIEF descriptors with performance-enhancing modifications. ORB uses weighted centroids and moments based on image intensities of a patch around the corner to give orientation. AKAZE, proposed by Fernández Alcantarilla [23], is a a feature detector and binary feature descriptor that exploit a diffusion filtering to retain boundaries and remove noise from blurry images. The detector uses a determinant of Hessian Matrix at different scale levels to find good feature points. AKAZE is invariant to scale and rotation. Many more descriptors and keypoint detectors exist, but the ones mention are the main ones used in SLAM/VO.

2.2.4 Feature Matching

Matching is a key part of vision-based tracking. Generally, tracking is performed by applying various algorithms on the feature descriptors found after the feature detection step. The most basic matcher is the *Brute-Force Matcher*, which greedily matches a feature descriptor in image A to every feature in image B, finding the most similar one by a distance metric. Typical distance metrics can bee seen in 2.1. As the reader might recall, Hamming distance constitutes the count of differing elements in a binary sequence, by applying the XOR operation on all elements. Furthermore, l_1 and l_2 are the respective 2D and 3D Euclidean distance norms. Vector-based descriptors can also be expressed as histograms, in this case the other distance metrics exist.

Large datasets and high-dimensional features require optimized methods. Fast Library for Approximate Nearest Neighbors (FLANN) is a open-sourced library for optimized

nearest-neighbor search, frequently used for feature descriptor matching [1]. The library is included in OpenCV and is also available as a standalone library in C++, C, MATLAB and Python.

Figure 2.7: AKAZE features matching from the Graffiti sequence Oxford Dataset

2.2.5 Optical flow

Whereas Feature tracking extracts features and matches across images using corresponding binary or histogram descriptors, Optical Flow estimates motion vectors of patches around interest points according to some constraints, such as brightness constancy and the assumption that neighbouring patches move similarly. Mathematically the relation can be expressed as

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.2}$$

where I corresponds to a intensity measure of pixel (x, y) at time t. The Optical flow equation is derived from a Taylor expansion of 2.2, yielding

$$f_x u + f_y v + f_t = 0 \tag{2.3}$$

where the terms are the gradient $f_x = \frac{\partial f}{\partial x}$, $f_y = \frac{\partial f}{\partial y}$, $u = \frac{\partial x}{\partial t}$ and $v = \frac{\partial u}{\partial t}$. The variables (u, v) are the only unknown, producing what is known as the *Aperture problem*. This problem is solved by imposing further constraints on the equations, typically solved with the Lucas-Kanade (LK) method. LK assumes a 3x3 patch around the feature to have the same motions, supplying 9 equations making 2.3 solvable as a over-determined system. LSM is applied for a least-square fit solution.

LK Optical Flow is usually applied to a sparse set of corners such as promoted in the paper *Good-Features-to-Track* Shi and Tomasi [78]. Dense methods such as the Farnebck method [22], which computes Optical Flow for all points also exists, but are not suited for

[1] http://www.cs.ubc.ca/research/flann/

real-time applications. Due to the strict assumption of brightness constancy, spatial coherence and temporal persistence, the method is highly erroneous in large motion scenes, varying illumination and other dynamic scene changes. Optical Flow tracking is not dependant on apriori knowledge and does not keep a model of movement, as such it is suitable for light-weight tracking in consistent scenes.

When using the the Lukas-KAnade method together with Shi and Tomasi *Good-Features-to-Track*, it is referred to as KLT tracking. A popular variant is Pyramid KLT, which relies on feature detection in different resolution images, starting with a low resolution image at the top of the pyramid and increasing the resolution level until the bottom, original image is reached. Pyramid KLT drastically improves performance.

2.2.6 Estimation

All visual tracking is based on Maximum Likelihood Estimation, usually we want to find the model parameters such as camera pose, camera intrinsics or 3d geometry that maximize the probability of observing some measurement. Given a set of observations $\chi = \{x^i\}$, a set of parameters to estimate θ and density function

$$p(x|\theta) \tag{2.4}$$

the maximum likelihood is given by

$$p(\chi|\theta) = \prod_{i=1}^{N} p(x^i|\theta) \tag{2.5}$$

as the logarithm is monotonic we can get the same result by using the log likelihood $l(\theta)$

$$-\ln L(\theta) = l(\theta) = \sum_{i=1}^{N} \ln p(x^i|\theta) \tag{2.6}$$

MLE estimates from 2.6 are used in a wide range of statistical analyses and is the basis of all algorithms in VO/SLAM. If prior knowledge is available, we can apply Bayes theorem

$$p(\theta|\chi) = \frac{p(\chi|\theta)p(\theta)}{p(\chi)} \tag{2.7}$$

the Maximum a posteriori estimate, MAP, is then composed of the conditional given in MLE and a prior probability from what is known about the parameter θ beforehand.

$$\hat{\theta} = \max_{\theta} p(\theta|\chi) \tag{2.8}$$

Numerical methods are oftentimes used in estimating MAP, and is the de-facto standard used in 3D reconstruction and optimization of SLAM. Depending on the distribution of the prior 2.8 can take various forms such as a non-linear least-squares problem or additional constraints such as projective conditions in Bundle Adjustment.

2.2.7 Robust techniques

LSM Pose Estimation techniques are sensitive to outliers, points that do not fit the model that skew the results. Such points are also known as bad matches or false positives. Fischler and Bolles introduced the RAndom SAmple Consensus algorithm as a general parameter estimation approach in datasets with a large proportion of outliers. RANSAC is frequently used in the CV field, especially in homography and fundamental matrix estimation. An iterative approach is used, in which a subset of points in one image is matched and its transformation is computed by minimizing the reprojection error. The process is repeated N times and eventually the best-fit subset is chosen with a given statistical confidence. Choi et al. [17] provide a great overview of different variations of the original RANSAC algorithm.

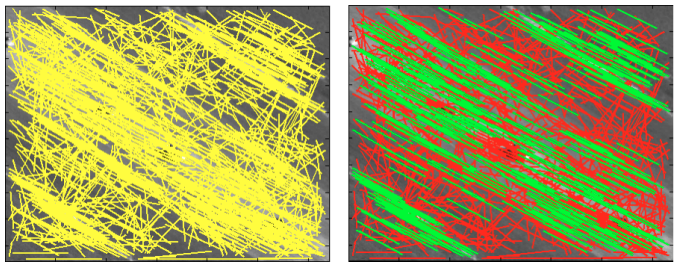


Figure 2.8: Before and after RANSAC is applied. Yellow dots are original feature matches, red outliers and finally the green outliers

2.2.8 Visual odometry

Visual Odometry is the process of estimating a robot or device ego-motion by only using the input of one or more image sequences. VO methods are divided into several categories. Monocular and stereo are the fundamental classes, according to the number of cameras used.

Earliest example of estimating robot motion through mono-camera image sequence was done by Moravec [57]. Nister et al. [63] introduced the term visual odometry, because of the similarities to the earlier robot motion tracking systems based on wheel odometry. They proposed groundbreaking methods of camera motion tracking from visual systems, and employed Fischler and Bolles [26] RANSAC algorithm for outlier rejection. Most notably, VO used during the Mars space mission as the preferred mapping mechanism [16].

Currently there are three main ways to formulate VO, namely feature-based, direct and semi-direct methods. Methods are characterized after what tracking method is used.

Direct method: This class of methods use all pixels or a sub-set of pixels and minimize error measure based on the image pixel-level intensity and/or depth. Non-linear optimization is used for camera pose and pixel depth estimation. Notable sub-classes of direct methods are dense A. Newcombe et al. [1], semi-dense Engel et al. [21] and sparseEngel et al. [20] formulations.

Method	Average pixels tracked
DTAM	200.000 px
LSD-SLAM	10.000 px
SVO	2.000 px

Table 2.2: Average denseness of methods. Similar convergence rates for all three with overlap up to 30% frame overlap

Feature-based method: Based on sparse feature-extraction and mapping across several frames. Furthermore, estimation of pose and depth is done through minimization of re-projection errors between feature pairs. High computational load due to feature extraction and matching, limiting amount of features that a system can track. Mur-Artal et al. [59] based on the ORB (Oriented FAST and rotated BRIEF) features [72] is one of the most used techniques in feature-based SLAM.

Semi-direct method: A hybrid approach borrowing from the direct methods and the feature-based methods. SVO, Semi-Direct Visual Odometry [27], is the current state-of-the-art hybrid methods for semi-direct VO. This method uses feature extraction to establish a robust area for image alignment. It has been shown to work exceptionally well, while also running on platforms with limited computational resources.

2.2.9 Pose estimation

Pose estimation is regarded as the ”basic” localization problem of AR, being subject of multidisciplinary research for a long time. Various different types of sensors can be combined to estimate pose, although vision based techniques provided good results and are most popular in recent years.

We distinguish the problem into two different cases. Firstly, consider that a 3D model is readily available or can be produced through SLAM. Then the problem becomes a classic pose estimation problem known as PnP, n-point correspondence. If no 3D model is available, the problem resolves to a camera motion estimation process. Both cases are solved using mathematical optimization.

In photogrammetry pose estimation is known as space resection, from which the problem originates. We want to compute the position and orientation of the camera given a set of correspondences between 3D features and their projections in the image plane. Robust estimation of PnP relies on robust techniques. Many different solutions exists for the PnP-problem, such as EPnP and *POSIT* as detailed in Chapter 3 of Marchand et al. [55]. In the case of an unknown 3D model, SLAM techniques can be applied.

The case of 2D-2D image correspondence, it is also possible to estimate the 3D motion through the homography. The homography links a point x_1 in image I_1 to a point x_2 in image I_2 through a set of parameters $\mathbf{h}$. 2D motion cannot reconstruct 3D scenes in every

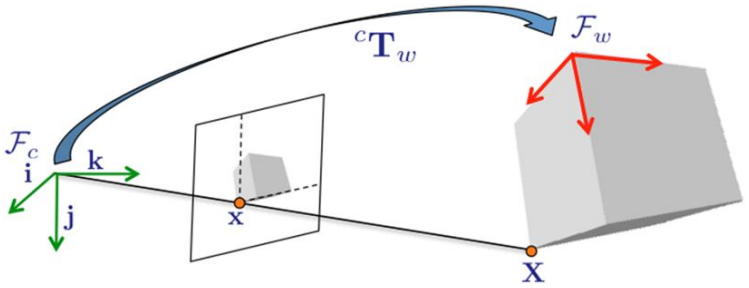


Figure 2.9: Transformation between world frame and camera frame[55]

case, however if the scene is planar the two scenes are linked through the homography H.

$$\mathbf{x}_{i}^{'} = H\mathbf{x}_{i} \tag{2.9}$$

$\mathbf{x}_{\mathbf{i}}^{'}$ and $H\mathbf{x}_{\mathbf{i}}$ are not directly equal, but they share direction and differ by a scale-factor. Thus the equation can be expressed as

$$\mathbf{x}_{\mathbf{i}}^{'} \times H\mathbf{x}_{\mathbf{i}} = 0 \tag{2.10}$$

which in turn can be written as a homogeneous linear system

$$\mathbf{A_i h} = \mathbf{0} \tag{2.11}$$

where $\mathbf{A_i}$ is a 3×9 matrix, and h is a 9-vector corresponding to the entries of H. Only two entries of $\mathbf{A_i}$ is linearly independent, since the third row is obtained from the two others. If there are more than 4 point correspondences, the system is over-determined and requires a iterative approach and a cost function to optimize. Both geometrical distance measures and statistical models such as MLE estimates are used. Robust estimation using outlier removal with RANSAC, M-estimation, LMedS or Hough-transform is common. Image stitching is a common domain where the homography is applied. More on the homography can be found in Chapter 4 of Multiple View Geometry in Computer Vision, Second Edition [35].

Theory of epipolar geometry enables the reconstruction of relative pose between images, by adhering to a geometrical constraint.

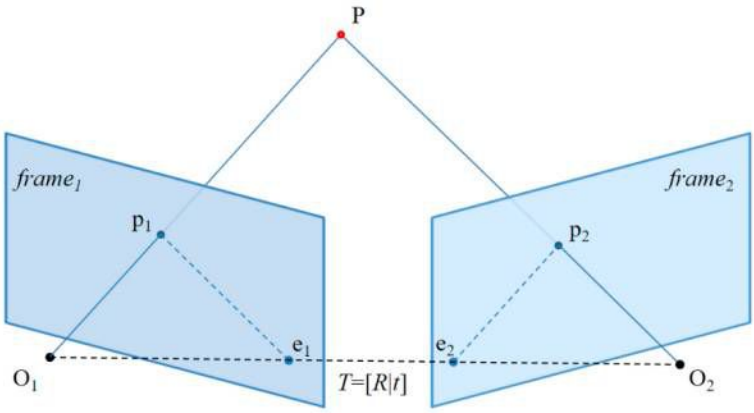


Figure 2.10: The epipolar constraint. The ray of P as from the second camera, (O_2, p_2), is projected onto O_1 as the epipolar line (e_1, p_1), greatly reducing the search-space for point-matching. O_n represents the respective n'th camera center in physical space, while the matrix $T = [R|t]$ represents the relative pose of the cameras in the scene.

- Essential Matrix E - Contains data of the Translational and Rotational offset of the relative camera pose, in global coordinates. The Singular Value Decomposition (SVD) of E yields the rotation and translation. Retrieved from F or by the 5-point algorithm; *An efficient solution to the five-point relative pose problem* [62]. Alternatively, the problem can be solved by the eight-point algorithm.

- Fundamental Matrix F - Contains data of the Translational and Rotational offset in *pixels*, in addition to the camera intrinsic parameters. Computed through the 8-point algorithm.

In brief, the essential matrix stems from the homogeneous relation of the normalized image point pair x_1, x_2

$$x_2 = \mathbf{R}x_1 + \mathbf{t} \tag{2.12}$$

by simplification, the essential matrix can be expressed as

$$\mathbf{E} = \mathbf{R}[\mathbf{t}]_\times \tag{2.13}$$

where $[\mathbf{t}]_\times$ is the vector cross product, expressed as a skew-symmetric matrix, between the 3x3 rotational matrix $\mathbf{R}$ and the 3-dimensional translation vector $\mathbf{t}$. A key issue of the triangulation process mentioned above, is that it requires enough translation or it will introduce large errors. Filtering the reprojected points by minimizing the reprojection error

In the case of 3D-2D motion approach, a third consequent frame is necessary for triangulating the 3D point. Firstly a 2D-2D relative point estimation is applied to the consecutive feature sets (f_{n-2}, f_{n-1}), then triangulating to get the 3D coordinates X_{n-1}. 3D-2D point pairs can then be extracted by the PnP algorithm with the input (X_{n-1}, f_n) to get the transformation from 2D feature points to 3D points.

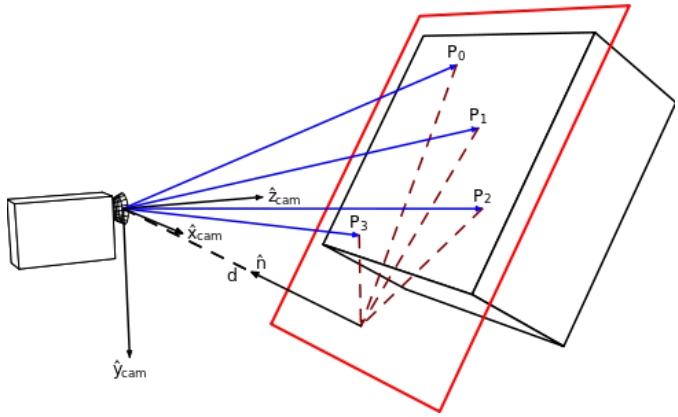


Figure 2.11: Plane captured by a camera at a distance d

As the two rays captured in each image never *truly* intersect, one must estimate the solution. Three approaches exist, according to what error chosen to minimize over; 3D error, algebraic error or reprojection error. The reprojection error is most popular, and is found by minimizing the cost function

$$\mathbf{T} = \arg\min_{\mathbf{T}} \sum |\mathbf{z} - f(\mathbf{T}, \hat{\mathbf{X}}_\mathbf{i})|^2 \tag{2.14}$$

where T is the estimated transform, $\mathbf{z}$ is the observed feature in the current frame F_n, furthermore $f(\mathbf{T}, \hat{\mathbf{X}}_\mathbf{i})$ is the reprojection function of the corresponding 3D feature point in the previous frame after applying $\mathbf{T}$. Lastly, i is the number of feature pairs.

Bundle Adjustment is the optimization of both the 3D feature points and 6DoF, while pose-graph optimization only optimizes the relative 6DoF poses of keyframes.

2.3 Visual Simultaneous Location and Mapping

A popular technique for enabling AR is SLAM, because of the inherent need of a device to know its position in the world. Visual SLAM is a sub-set of the classical problem *Structure-from-Motion*, SfM. Whereas SfM originated from the film-industry and is an off-line process, V-SLAM tries solve the mapping problem in-real time using sequential images from the same camera setup. Given an unknown environment, the device must utilize its sensors to build a map, thereafter localizing with respect to that map. Observations of the real world is not *perfect*, it is subject to uncertainty. Common sources of error are systematic errors and random errors. Furthermore, erroneous data *compound* errors. Estimation algorithms such as the pose-graph optimization, EKF, statistical models and BA are frequently used in AR to estimate positions and poses that contain error. In SLAM, every sensor observation influence the map, making it a computationally heavy task to update and maintain the map. As a result, SLAM problems are complex and heavily rely on strategic decision regarding denseness of features and refinement.

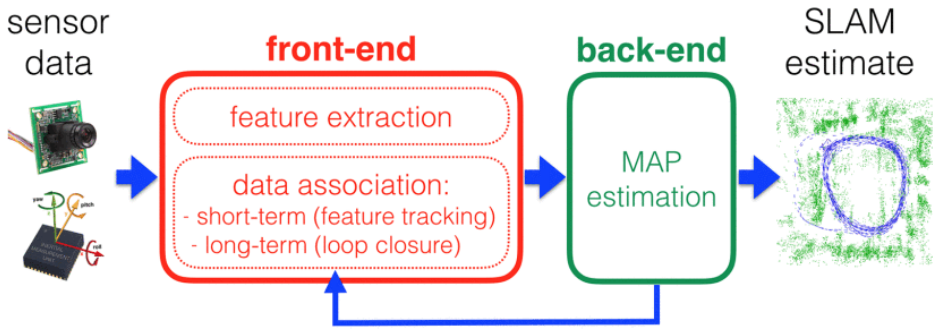


Figure 2.12: SLAM system architecture from [79]

We divide SLAM into two main categories, filtering-based SLAM and Graph-based SLAM

- Filter based methods - Classical approach that performs predictions and update steps recursively, keeping a error covariance matrix to store uncertainties of states. Variations of the Kalman filter and Particle filter are typical methods.

- Graph optimization methods - Saves keyframes and uses Bundle Adjustment optimization to minimize the errors between camera pose and/or 3D points mapped. The graph is made up from keyframes with measurements (nodes) and their geomtrical constraints(edges).

A typical SLAM architecture is divided into two distinct parts, as seen in 2.12. The front-end part of this architecture is typically handled by a device visual odometry module and a mapping module, while the backend, either local or remote, supplies optimization. Furthermore, a loop detection module and relocalization module is typically also be present.

2.13 shows how alignment can be achieved when relative pose constraints are applied to the problem.

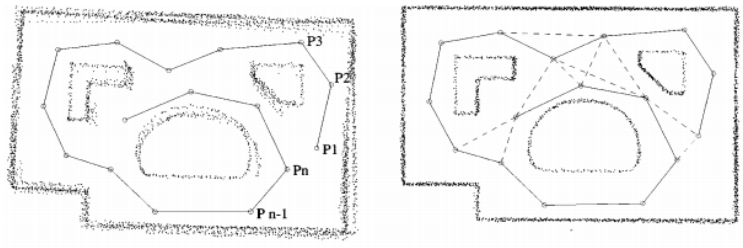


Figure 2.13: Before and after aligning range scans [53]

2.3.1 Challenges in Monocular Visual Odometry

Modern visual systems have seen a significant progress with regards to algorithmic accuracy, robustness and efficiency. Some low-level issues are not clearly defined how to resolve. Yang et al. [86] provide an overview to the issues some of the issues. Visual Odometry also suffers from the classical optical distortions common to cameras.

- **Photometric calibration** - Pixel intensities of the same 3D point may vary due to camera optical vignetting, auto gain and exposure controls.
- **Motion bias** - Visual Odometry methods may produce differing results when run on a image sequences backwardly or forwardly.
- **Rolling shutter** - A camera sensor system found in most mobile devices today. Pixels are exposed at different timestamps due to a rolling shutter system. Rolling shutter is used because it enables the optic sensor to capture light for a extended period of time, increasing sensitivity. Systematic errors resulting from the rolling shutter effect, such as wobble or skew, can be corrected or approximated, although in practice it can be ignored due to the high computational cost it enforces.
- **Distortion** - Since camera lenses are not perfect, distortion occurs in images. Several types exist, such as pincushion, barrel and mustache distortion. Images can be undistorted if the distortion parameters are known.
- **Motion blur** - Images appear smeared due to the long exposure or rapid movement.

2.3.2 Tight vs loosely coupled visual-inertial SLAM systems

Two main approaches exist when integrating IMU and visual data. Calculating visual-SLAM and IMU independently in separate modules is known as a tightly loosely coupled system, which suffers from drift problems. A tightly-coupled system incorporates feature information in the state vector of the sensor fusion to estimate scale. Loosely coupled systems include IMU readings as independent data in the model. Fast and lightweight systems typically opt for a loosely coupled integration while systems with more computational budget or accuracy requirements go for a tightly coupled approach. Multi-State Kalman Filtering is a traditional way of integrating the IMU, but modern methods rather use non-linear optimization for the task.

2.3.3 Loop closure

Also known as cycle closure, is the refinement step required to obtaining a consistent, global SLAM map. Tracking and mapping done over a significant amount of time is especially prone to drift, and requires loop closing. Typically done by returning to the origin, providing non-adjacent frames of the same features to the system. Loop closure is one of the key distinction between SLAM and VO.

2.4 Android Sensors

Most sensors are so-called Micro Electro Mechanical Sensors(MEMS), many of which are cheap and small. These devices capture mechanical motion and express them as electric signals. Sensors commonly found in Android devices are the accelerometer and gyroscope. Two main categories of sensors exist, physical and synthetic sensors. Physical sensors have a hardware component, while synthetic sensors are derived from one or more sensors. Android sensors generate different sensor values, which can be divided into three classes. Raw values are the unaltered values, directly passed on from the hardware component. The raw values are not directly accessible, only the corrected values. As per the documentation, the physical sensor must not be confused with the android sensor. Calibrated values are values subject to correction algorithms, that intends to remove noise. Fused values are derived from two or more sensors, in process where one sensors strength is used to cancel out another sensors weakness. This is the process known as sensor fusion 3.1.

The android sensors API consists of Sensor, SensorsManager and the SensorEventListener interface, which implements a `onSensorChangedEvent(event)` and `onAccuracyChangedE` . Each event contains a value array, as described in the documentation [2]. The first event is triggered by every sensor reading, and a $switch$ statement is a good choice for controlling the data flow here, like checking sensor type.

Next we will review the main sensors relevant for this work.

Accelerometer - Captures acceleration along x, y, and z axes by measuring forces acting on the sensor. Reports in m/s^2. As such, a phone lying still on a flat surface should report the gravitational force acting on it (as the table pushes against the phone) and in free fall no forces are acting on it.

Gyroscope - Captures angular velocity rad/s around the axes x, y and z. Possible to find current position in space given an initial position and integrating gyroscope readings. This process introduces gyroscope drift, which must be compensated for.

Magnetometer - This sensor captures the Earths geomagnetic field strength μT along x, y, and z axis. Can be used as a compass to find true north.

GPS/GNSS chip - Provides a global position in geographic coordinates and altitude in meters above the reference ellipsoid. Combined with network data, positional accuracy

[2] https://developer.android.com/reference/android/hardware/SensorEvent#values

of 5m can be achieved.

Digital camera - Provides the real-world ”canvas” of AR apps, as well as images for visual tracking. Modern smart phones often have several cameras, both back-faced and front-faced.

2.4.1 Virtual sensors

Rotation vector sensor - Synthetic sensor based on sensor fusion. As of the Android documentation, it employs a EKF-based fusion algorithm.

6DoF pose sensor - Similar to the rotation vector, but includes a delta translation of an arbitrary reference point. Available in newer Android devices (API level 25). Where possible, it uses depth sensing and the camera for improved accuracy. Each registered pose is sequenced, and contains the relative pose of the previous registered pose.

Gravity - Precise sensor of drift-free gravitational magnitude and direction. Product of gyroscope and accelerometer data.

Linear Acceleration - Produced three-dimensional acceleration, free of gravitational influence.
As a result, the acceleration sensor reading can be reconstructed by adding gravity and linear acceleration. Acceleration = Gravity + Linear acceleration.

2.14 reveals the sensor stack found on Android devices. The Hardware Abstraction Library (HAL) connects hardware to the Android platform through the interface header file sensors.h and a C++ implementation sensors.cpp from the manufacturer. Device or chip vendors can choose to supply sensor fusion implementations or Android falls back to a default implementation, which may not be optimal. The Sensor Hub layer is an optional module from the manufacturer, providing computing, monitoring and batching of sensor data.

Because of the various hardware manufacturers and device vendors operating on the Android platform, performance and accuracy can vary.

[3] https://source.android.com/devices/sensors/sensor-stack

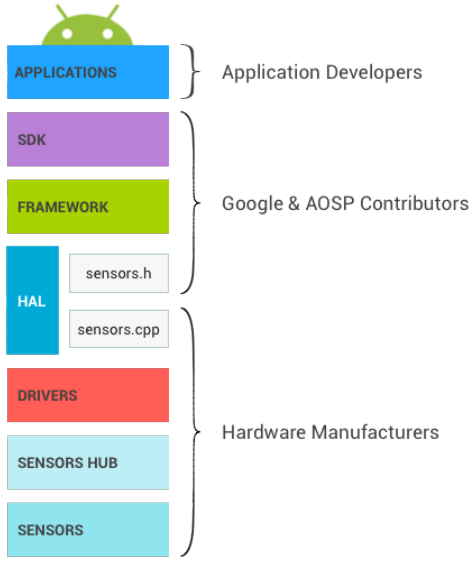


Figure 2.14: The layered architecture of Android sensors [3].

Type	Sensors	Coverage	Accuracy	Indoor / outdoor
inertial	accelerometer	anywhere	0.01m	indoor / outdoor
	gyroscopes	almost anywhere[a]	0.2 deg.	indoor / outdoor
magnetic	magnetometers	almost anywhere[a]	0.5 deg.	indoor / outdoor
electro-magnetic	GPS	almost anywhere[b]	10m∼15m	outdoor
	differential GPS	almost anywhere[b]	1m∼3m	outdoor
	RTK GPS	almost anywhere[b]	∼0.01m	outdoor
	Wi-Fi	∼90m	2.5m	indoor
	RFID	20m∼100m	1m∼3m	indoor
	UWB	10m∼100m	0.15m∼0.3m	indoor
	infrared	∼6m	0.03m∼0.1m	indoor
	Bluetooth	∼10m	0.1m∼10m	indoor
ultrasonic	ultrasonic	10m	∼0.01m	indoor

[a]blind regions including environments full of strong ambient electromagnetic fields and metal shields.
[b]blind regions including urban canyons, indoor surroundings and places out of LoS.

Figure 2.15: Different sensors available in a MAR system [15]. Modern smarthphones contains most of these sensors.

Understanding different types of errors is very important when dealing with sensor data. Common erroneous readings are a result of a possible combination of systematic error, noise , drift and bias. Systematic error affect the observation accuracy and can often times be eliminated with calibration or changing measurement scheme. Noise is a kind of random error, in which the measured value fluctuates and it be statistically quantified. Drift describes how measurements, in the long-term, can ”move away” from the real value. Drift can be caused by sensor decay or integration of sensor data. Bias is simply an offset, eg. if a device is lying flat on a table and showing anything other than $(\ 0,\ 0,\ -9.80665m/s^2\ )$. Common techniques to aid these errors are various filters commonly used in signal processing. Low-pass, high pass, moving-average for general signal smoothing and compli-

mentary, Madgewick, Kalman and particle filters.

Popular filters that can be used in sensor fusion, often used in digital signal processing:

- Madgwick filter - Madgwick et al. [54] produced a fusion algorithm that matches the Kalman filter in accuracy, while using lower computational loads and sensor frequencies.
- Particle filter - More computationally expensive and Moudlable, can perform better than the Kalman filter in non-linear systems.
- Kalman filter - Optimal in a linear system with Gaussian noise. *Can* provide good results in non-linear systems.
- Extended Kalman filter - De-facto standard in non-linear state estimation. Uses the derivatives found in the Jacobian to estimate state.
- Complimentary filter - Basic filter for the gyroscope and accelerometer -* to remove drift, as demonstrated in Figure 3.1.

2.5 Rotational representation

There are several ways to define a rotation in 3D space. Euler angles, rotation vectors, rotation matrix and quaternions all provide valid representations of a rotation. The individual attributes dictate which representation that is the most sensible for a problem.
Following Eulers rotation theorem a 3D coordinate system can be described as a single rotation about some axis, which is uniquely describred by a *minimum* of three parameters.

Euler Angles: any orientation can be obtained by combining three rotations about each of the three axis in the coordinate system. Euler angles are easy for humans to interpret but has some disadvantages. Most systems use the right hand rule and counter-clockwise rotations when defining a reference system. Many other systems exist in which the order multiplication varies.

Axis Angle: Also known as axis-angle representation. Relies on Eulers rotational theorem to show that any sequence of rotations in 3d space can be expressed as a single rotation around a single axis, and becomes $\Theta = \theta e$.

Quaternions are generally represented as $\mathbf{q} = q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k} + q_w$, often viewed as an extension of the complex number system. In a quaternion number system, $\mathbf{i}, \mathbf{j}, \mathbf{k}$ satisfy the constraint

$$\mathbf{i}^2 = \mathbf{j}^2 + \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{2.15}$$

A number of identities makes quaternions suitable to represent spatial rotations, mainly efficient multiplication and spherical interpolation (SLERP). Applying Eulers Theorem a quaternion can represent a spatial rotation and will have the form found in **??**.

Representation	Mathematical expression
Axis-Angle	$\mathbf{r} = \Theta\mathbf{e}$
Rotational Quaternion	$\mathbf{q} = \cos(\frac{\Theta}{2}) + (e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k})\sin(\frac{\Theta}{2})$
Rotation matrix about the Z-axis	$R_z = \begin{bmatrix} cos(\Theta) & -sin(\Theta) & 0 \\ sin(\Theta) & cos(\Theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Table 2.3: Overview of common rotational representations. $\mathbf{e}$ represents a unit vector.

2.6 Coordinate systems, reference frames and transformations

Coordinate systems, reference frames and transformations are heavily used in navigation, computer graphics, game development, physics, mapping and control theory. Applications in which different sensors and different platforms are used, need a way to unify observations. We say that data is observed in a specific frame, therefore we need to understand various coordinate frames. A frame is a orthonormal basis, that is, a set of three perpendicular vectors of length one. Mathematically speaking, this basis define a space. A rotation applied to specific reference frame is called an orientation.

The following section explains relevant concepts for this work.

- **Extrinsice rotation** - A rotation in relation to a fixed, often global, coordinate system.
- **Intrinsice rotation** - A rotation in relation the the rigid bodt itself. It follows that The intrinsic rotation is the *inverse* of the extrinsic rotation.
- **Geodetic frame** - Global reference frame used extensively in geodesy and navigation for precise locations on the earth, which is composed of a vertical and horizontal datum. Several datum exist, most notably WGS84 which GPS uses. WGS84 horizontal datum is latitude and longditude and vertical datum is height above or below the ellipsoid. Coordinates expressed as (ϕ, λ, h).
- Earth-Centered Earth-fixe - Also known as ECEF or the geoncetrnic frame, is centered at the earth center of mass. The frame is defined by a orthonormal basis at the center with coordinates expressed in Cartesian (x, y, z) and spins with the earth. Reference directions are the equator, the prime meridian and true north.
- East-North-Up frame - A frequently used local frame, often chosen to simplify calculations due to the magnitude of ECEF coordinates. Reference directions are defined as North, East and Up following the right-hand rule.
- North-East-Down - Popular alternative to ENU. Chosen in applications where objects tend to reside above ground. Two axis are flipped in this configuration, according to the right-hand rule.

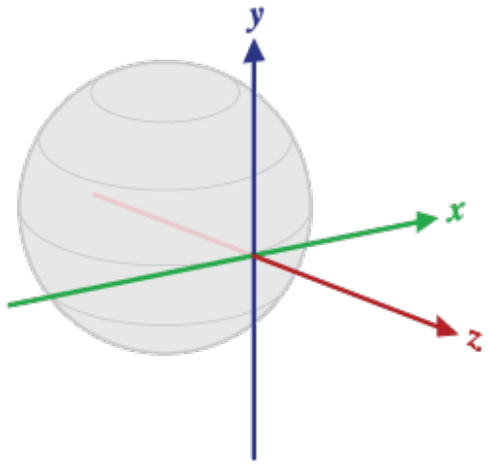


Figure 2.16: East North Up Axis.

2.7 Real-Time systems

A system is real-time if the required processing is completed in a given time constraint. Any necessary processing is done in-between rendering of frames on a screen and the user experience is not severely inhibited by processing or networking. Local and remote computing time plus latency adds up to a time constraint in which the system must operate to be regarded as real-time.
Originally, the client/server architecture of the web relied upon the client to initiate requests to the server. Technological advancements have enabled transmission and streaming of large sets of data quickly and efficiently over the internet. Mobile devices today enjoy sophisticated cellular networks and the newest technologies for transferring data in real-time over the web. In this work we consider two different means of communicating bidirectionally in a client server architecture. Bi-directional communication implies communication in both directions over a protocol layer on top of TCP/IP, which is the standard protocol used by web clients and servers used for communication. Both are a defined in the HTML standard which is currently maintained by Web Hypertext Application Technology Working Group [85]

2.7.1 Polling

The polling technique was the first attempt to achieve real time communications. This involved sending HTTP-requests frequently, to ”poll” to server for updates. If the update-rate of the server is know beforehand, this can achieve a good result. However, high or low server update scenarios are not optimal. In the case of low update rate, the server would be large quantities of empty responses. In a high update scenario, the data might already be out-dated upon arrival.

2.7.2 Long-polling

Next step towards proper real-time communication. An extension of Polling, in which the HTTP connection is held open for a set time frame if a response is not readily available. Either the connection terminates with no response after the time runs out, or the server responds. Long-polling ensures that the data is sent immediately after it is ready and provides almost real-time communications in a asynchronous fashion. Scenarios with a high data update rate that requires high volumes of requests, long-polling does not improve upon ordinary polling.

2.7.3 Server-Sent Event

Server Sent Events is a stream-based API that allows for clients to receive pushed data from a server. Both server and client must implement this event interface. The SSE relies on streaming partial HTTP responses on a open HTTP connection, after accepting a SSE initialization over a HTTP GET request. The server response with a HTTP Content-Type = text/event-stream header. The client processes data through the EventSource javascript API with a callback function. SSE works akin to a data-stream subscription, where the client listents for events. SSE is a part of the HTML5 standard.

2.7.4 WebSockets

Persistent connections over the web has provided the current basis for real-time communications. Websockets was introduced in 2012 as the first protocol to offer bi-directional data streaming. As a result, the websocket protocol is an independent TCP protocol, its only relationship to HTTP being the initial handshake request. Default port is 80 for regular connections and port 443 for secure web socket (WSS). Here follows and excerpt from page 9 of the protocol specification [24]:

```
Conceptually, WebSocket is really just a layer on top of
TCP that does the following:

o  adds a web origin-based security model for browsers

o  adds an addressing and protocol naming mechanism to
   support multiple services on one port and multiple
   host names on one IP address

o  layers a framing mechanism on top of TCP to get back
   to the IP packet mechanism that TCP is built on, but
   without length limits

o  includes an additional closing handshake in-band that
   is designed to work in the presence of proxies and
   other intermediaries
```

```
Other than that, WebSocket adds nothing.  Basically it is
intended to be as close to just exposing raw TCP to script
as possible given the constraints of the Web.
```

The performance of the different means of bi-directional client server communication over the web has been thoroughly researched. Regarding the performance, [80] found that the WS standard was superior to SSE + XHR in terms of speed. Other considerations, such as networking configurations, browser support, optimized requests or application specific needs may further favor one or the other technology. Because SSE is purely sent over traditional HTTP, its feature set is richer by default. WebSockets also require a special server implementation, giving it less freedom. Socket.IO[4], which is a abstraction on top of WebSockets, is also a very popular library on the web today. It provides transport fallback for servers that do not support web sockets, among other things.

In any case, WebSockets are chosen as the preferred mean of bi-directional communication for this case, both because of performance and ease of implementation.

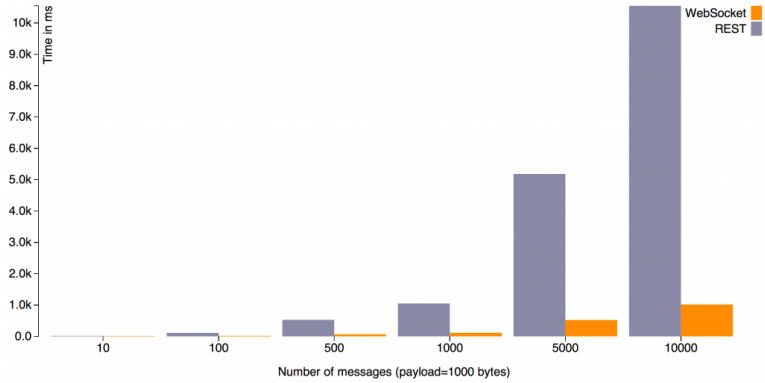


Figure 2.17: HTTP raw overhead against an echo endpoint [3].

[4]https://socket.io/

Chapter 3

Related work

3.1 Mobile Augmented Reality

Modern mobile devices are subject to ever improving hardware components that extend their capabilities. Accompanying the devices, we find a wide range of applications, that also increasingly utilize the potential of present-day, multi-sensor smart phone devices.

3.1.1 Tools for Mobile AR

Several companies produce and offer software development kits (SDK) for AR. Of the most popular ones we find Wikitude[1], Vuforia[2], Kudan[3] among many others. Their feature-set are ever growing, and will probably merge into the AR . These trackers include functionality as marker-based tracking, slam, cross-platform functionality. Object recognition, image recognition, face recognition, IMU integration and more. These technologies are proprietary and cost a considerable amount of money. Mainly focused on industry and enterprise use. Qualcomm plays an important role as the manufacturer of the most popular System on Chip for mobile phones, Snapdragon [4].

Of lower level SDKs for development and image processing, we find OpenCV, FastCV and Point Cloud Library (PCL). OpenCV is a open-source, cross-platform library that is recommended by the standards organization Khronos for CV, possibly it will its API will become the standard for CV. The code base is in C++, while wrappers expose the library to languages such as Python, Java and Matlab, giving flexibility and cross-platform funcionality
FastCV is a CV SDK developed and maintained by Qualcomm. Although FastCV has a sparser feature-set, their implementations are optimized for the mobile chips they produce.

[1] https://www.wikitude.com/
[2] https://www.vuforia.com/
[3] https://www.kudan.eu/
[4] https://www.statista.com/statistics/233415/global-market-share-of-applications-processor-suppliers/

As mathematical optimization plays an important role in estimation, mapping and localization in AR, Ceres can provide tested and efficient algorithms for general optimization problems. Ceres is a open-sourced, non-linear optimization library developed by Google written in C++. Libraries such as OpenCV and PCL also provide similar algorithms.

3.1.2 Head-Mounted Display

Mobile Augmented Reality have been around for a long time, but mostly in the defense industry [5]. These are optical displays worn on the head or as part of a helmet. We regard to types of tracking for HMDs, inside-out tracking and outside-in tracking. A inside out system registers pose by means of internal sensors on the device, yielding better pose estimates. Outside-in tracking however, applies markers *onto* the HMD of interest to track and estimates 6dof by a external system. This type of tracking is limited due to a static external observations system with a limited field of view. Modern HMD such as the Magic Leap[5] or Microsoft HoloLens[6] use inside-out tracking. A disadvantage is the computational load this puts on the system, especially in previously un-mapped environments, requiring dedicated hardware or offloading computing over a network.

3.1.3 GPU acceleration on mobile systems

Parallel computing is often applied to offload computationally heavy tasks from the CPU to the GPU or other type of dedicated hardware. In general, GPUs are optimized for certain computational tasks, such as rendering graphics. The CPU has fewer cores and solves tasks sequentially, while the GPU contains more cores to efficiently compute massive calculations in parallel. Genereal purpose computing on mobile GPUs (GPGPU) is possible using graphics APIs such a OpenGL ES, OpenCL, CUDA for Nvidia architectures, Android native RenderScript or iOS native Metal. GPGPU optimized mobile systems have been shown to increase algorithmic performance [84]. As AR inherently involves image data, some tasks, such as feature extraction or matching, may be offloaded to the GPU for increased framerates. Hardware accelerated computer vision and neural networks is in the works with the OpenVX standard. Qualcomm aims to provide optimized chipsets for CV tasks in the future, where dedicated hardware can accelerate mobile capabilities[7].

3.1.4 Mobile sensor fusion

Achieving a reliable estimation for the device orientation is not an easy task. A mobile device is equipped with several sensors, many of which can be combined to eliminate sensor bias. Indoor navigation, autonomous vehicles, GPS/GNSS and other areas of signal processing heavily use sensor fusion for modelling the environment or process of interest. Traditionally a filter-based approach is used for estimating system state from multiple sensor readings.

[5] https://www.magicleap.com/
[6] https://www.microsoft.com/en-us/hololens
[7] https://www.qualcomm.com/products/snapdragon-xr1-platform

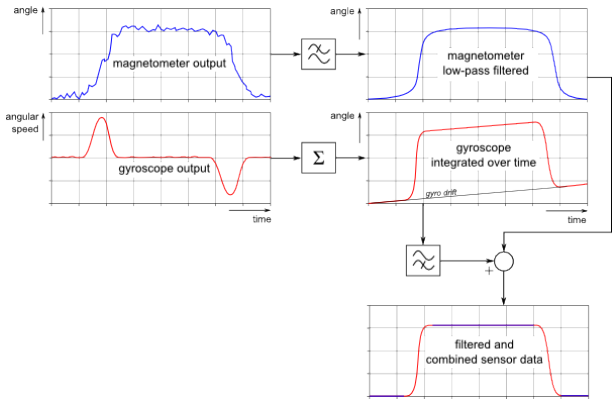


Figure 3.1: Fusion of gyroscope and magnetometer for improved sensor readings using a complimentary filter [48]

Gośliński et al. [32] conducted experiments comparing different algorithms for Android orientation estimation, including both the Extended Kalman Filter (EKF) and the Adaptive Extended Kalman Filter (AEKF) on sensor readings. Using a industrial rated orientation sensor, the Xsens MTi orientation sensors, for a truthful reference point the estimates was mapped onto the same reference for comparison. The experiments showed that orientation can precisely estimated using EKF and AEKF based algorithms on raw acceleration, magnetometer and gyroscope data. CPU and battery usage was found negligible for the incorporation of this estimation process on a modern mobile phone running Android.

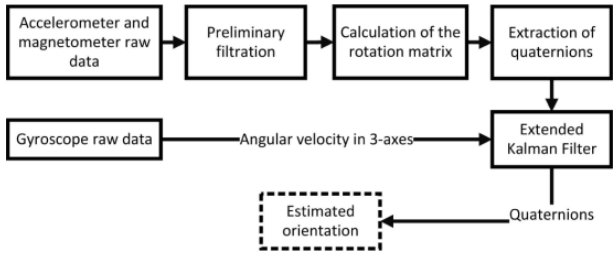


Figure 3.2: Estimation process used by Gośliński et al. [32]

Other work such as Piao and Kim [67] have also adapted the front-end of *ORB-SLAM* [59] to tightly-couple IMU readings to the VO system. Qin et al. [69] applied both loosely and tightly coupled approaches in their system. The loosely coupled system was used to initalize the system, recovering scale, bias, gravity and velocity estimates. After initialization a tightly coupled system kicked in, providing high-accuracy, robust state estimation. The authers went with a Bundle Adjustment problem formulation instead of EKF.

3.1.5 Mobile Pose tracking

Estimating mobile pose is a key task in mobile augmented reality systems, and has been extensively researched in the field of computer vision, robotics and autonomous systems. As the computing power, mobile network capabilities and sensors in smartphones improve, more and more of the traditional computer vision techniques gets available on the smarthphone platform.

Schall et al. [75] developed a MAR platform consisting of a ultra-mobile processing unit (UMPC), IMU, GPS, barometer and a camera all mounted together on a mobile platform. Multi-sensor fusion was performed in real-time. A online modem supplied networking capabilities such as online corrections for the RTK-GPS. Tracking is done as 2D-2D picture correspondence between the generated map and camera images, matching with FAST [71] features and a 8x8 pixel normalized cross-correlation (NCC) matching.

Feature tracking is key for obtaining the relative orientation and position of a object and the device. Traditional mobile tracking mainly use markers or delegating the heavy computing tasks to a remote computer. Newer trends in tracking use either direct or feature-based approaches, of which either can be dense or sparse. Matching is done either 3D-3D, 3D-2D, 2D-2D accordingly.

Earlier approaches relied heavily on detection of features such as edges [46] and contours [65], but these efforts suffer from occlusion and cluttering. More recent approaches use techniques such as key-point based tracking [33], where the tracking accuracy and robustness are notably better. Due to high computational costs, this approach is does not meet the real-time performance criteria of the mobile platform. Wagner [83] found success modifying existing methods such as SIFT [52]and Ferns [64].

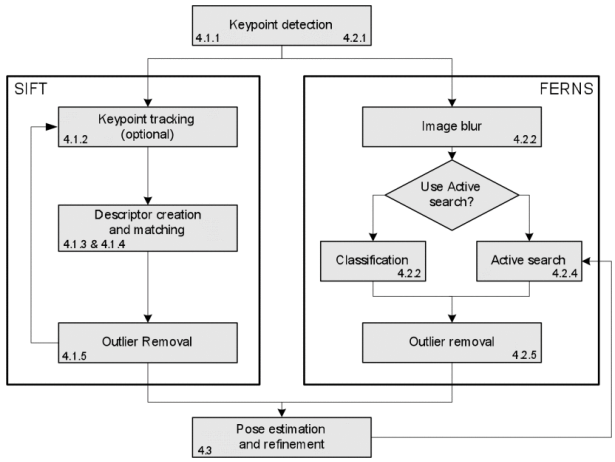


Figure 3.3: Pipeline of keypoint detection [83]

3.1.6 Global Localization SLAM

Ventura et al. [82] implemented a key-frame-based monocular SLAM with a mobile phone

supported by a server-side map registration process for global localization. Key-frames are supplied to the server by a client SLAM system. Upon arrival of new key-frames the server registration is recomputed and a global estimate returned. The authors expand upon the idea of loop closing. Instead of detecting overlaps in a single, local SLAM map, they are interested in detecting overlap of the entire local map and a previously known, global 3D map.
The client-side SLAM system is based on PTAM [47] and SIFT features, while improving upon the initialization process using principles from [58]. Global points are represented as the mean descriptor the corresponding feature on the server.

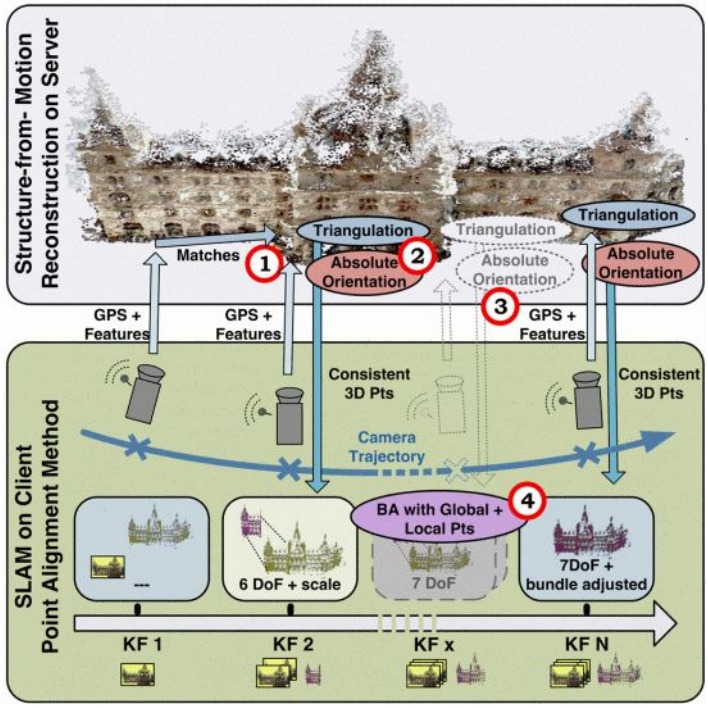


Figure 3.4: Global pose estimation using offline key-frame registration [82]

Two different scenarios were tested, small-scale indoor and large-scale outdoor. In the outdoor experiment, $10,000m^2$ reconstruction generated a 3.9GB descriptor database. Database partitioning based on the three different areas in the experiment, decided according to client GPS readings. Initialization took on average 12.6 seconds and time-to-first-localization was 5.3 seconds.

3.1.7 Monocular visual-inertial odometry for Mobile Augmented Reality

One of the earliest implementations of a visual, marker-less tracking systems running on a mobile is [47], porting the PTAM system [45] to an Apple iPhone 3G. The authors used two simultaneous threads, one to handle frame-to-frame tracking and camera registration, while a background thread performed BA to optimize the map. Actively removing unnecessary key-frames and measurements was key for making it work at acceptable framerates on a mobile phone. Other measures such as full-frame rotation estimation, due to lack of native gyroscope, and reduced image resolution (240x320 px). While several limitations were uncovered, Klein and W. Murray [47] proved that keyframe-based SLAM is feasible on mobile phones, opening up future work for mobile tracking and mapping.

Leutenegger et al. [50] proposed a system that recently was open-sourced, named OKVIS (Open Keyframe-based visual-inertial system). OKVIS uses Harris corners [34] and BRISK descriptors [49], which are extracted along the IMU-generated gravity direction. The non-linear optimization process proposed demands significant computing times, lowering its viability on mobile or embedded systems.

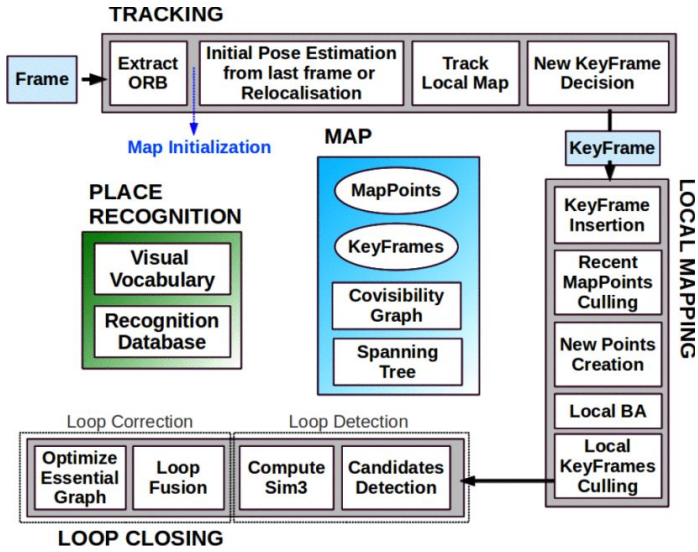


Figure 3.5: Original ORB-SLAM architecture [59]

VINS-mobile, based on [69] and earlier works of the same authors, propose a visual-inertial navigation system based on monocular vision. Advantages of integrating IMU measurements include stand-in motion tracking in circumstances where the visual part fails, for instance low-texture areas, changing illumination or motion blur. Features such as robust initialization, camera-IMU extrinsic calibration, online re-localization and pose graph optimization, as well as pose graph reuse. Pose graph reuse enables the system to save, load and merge local pose graphs. System state estimation is highly non-linear, so

advanced non-linear optimization techniques is applied. In comparison to OKVIS [50], which is more suited for stereo systems, VINS-mono is especially designed for mono camera systems.

VINS-mobile is based on KLT sparse optical flow tracking using *Good-Features-To-Track*, maintaining a minimum number of 100-300 uniformly distributed features per image. RANSAC is applied with the fundamental matrix model of motion estimation. Keyframe selection is based of two criteria, namely a constraint on average feature parallax between current image and latest keyframe. Secondly, rotational parallax can also occur, but can be remedied with short-term Gyroscope data.

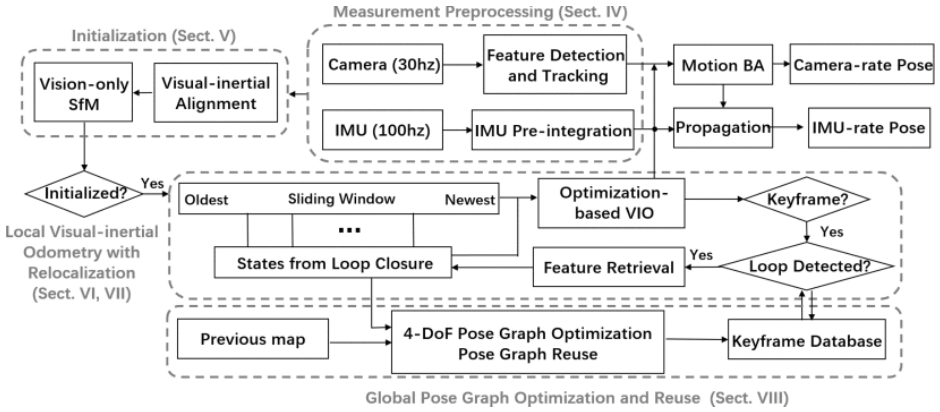


Figure 3.6: VINS-mono pipeline [51]

IMU pre-integration can be found in the appendix of the paper, and mainly involves integrating the IMU sensor measurements in the time frame between images, assigning each image a rotation, position and velocity, while continuously rep-proagating IMU new measurements. Backend SLAM, namely the loop closing and pose graph optimization is based on BRIEF descriptors from the keyframes and DBoW2 [29] for converting the given description to a bag-of-word representation. Compared to OKVIS [50], VINS-mono shows better performance and boasts more features.

Delmerico and Scaramuzza [19] tested state-of-the-art, publicly available, algorithms from both loose/tight and filtering / optimization VIO pipelines. Several hardware platforms were analyzed to discover per-frame processing time, CPU and memory load. Although the study was performed with autonomous flying robots, their results have carry-

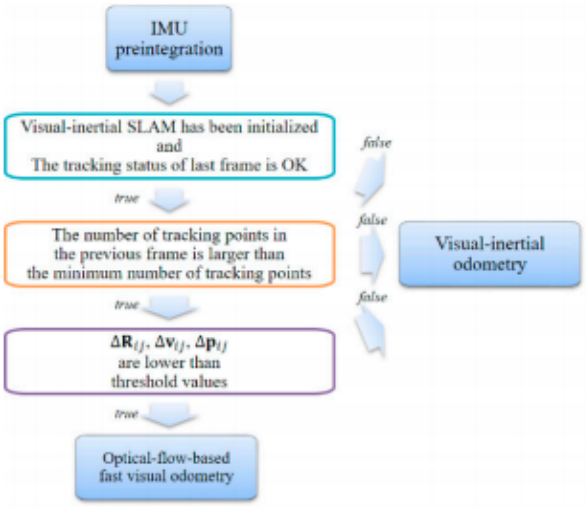


Figure 3.7: Adaptive flowchart execution of [67]

over value to mobile applications due to similar computing constraints.

3.1.8 State-of-the-Art Mobile SLAM

Most noteworthy break-through came with PTAM Klein and W. Murray [47], the first case of simultaneous tracking and mapping on a mobile device, using different threads. The tracking thread was responsible for finding features, updating the map and drawing graphics. Map updates is delegated to a second, parallel thread, which waits for new keyframes, adds points to the map while optimizing and maintaining the map.

Mur-Artal et al. [59] developed a ORB-SLAM, a feature-based, real-time SLAM system for monocular devices, later expanded to support RGB-D and stereo cameras, loop-closing and re-localizing in [60]. ORB-SLAM open-sourced, making it a very popular choice for researchers in many different fields. Three parallel threads are utilized, expanding upon PTAM [47]. Tracking and mapping is implemented using ORB features [72] features in two different threads and bag-of-words place recognition [29] for efficient re-localization and loop closure in the third thread. Key-frames are generously kept according to a survival of the fittest approach, while being efficiently discarded if unwanted.

Engel et al. [20] is a direct and sparse method for VO, i.e. it does not rely on feature detectio and tracking. This approach have been extended to include pose graph optimization, loop closure and IMU integration. This approach is not intended for mobile phones. Requires photometric calibration and a good lens. From the same authors we also have LSD-SLAM [21], Large-Scale Direct SLAM, is a direct method based on mapping by depth estimation and minimazation of photometric error. LSD-Slam has been developed for mobile as MobileSLAM [8] running realtime with 30fps.

[8] https://github.com/xorthat/MobileSLAM

[27] proposes a half-way solution between direct and indirect methods, mainly aimed at drones. It uses both image intensities *and* features to improve performance. SVO 2.0 improves upon the existing [28] by using edge tracking together with features to improve tracking in textureless environments.

An adaptive method developed by Piao and Kim [67] utilize different tracking modules according to observations have been found to increase accuracy. The authors implemented fast optical-flow-based fast VO module and a adaptive policy, while integrating it with the existing VIO system ORB-SLAM[59]. If the number of key-points tracked in the previous frame is adequate and the IMU change in velocity, rotation and position since the last frame is small since the last frame.

3.1.9 CNNs for pose estimation

The 2015 study *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization* [42] proposed a deep neural network for monocular cameras to estimate 6DoF pose. Using GoogLeNet as a basis for the pose regression network, the system achieves real-time performance of 20fps and localization error of approximately 2m and 3 degrees for large scale, outdoor scenes and 0.5m and 5 degrees accuracy indoors. The authors also found that context and field of view was more important than image resolution for relocalization based on monocular images. The neural net inputs down-scaled 224x224 pixel images and regresses the cameras 6DoF pose relative to a scene. This is done by learning high-level local features that in many cases are more robust to illumination change, motion blur and changing weather. As an example, they show that blurring a landmark increase apparent contour size, making the system believe it is closer. For the paper the authors released a urban localization dataset *Cambridge Landmarks*, consisting of 5 scenes. The dataset was created with traditional SfM techniques. PoseNet is the state-of-art pose estimation based on Deep Learning.

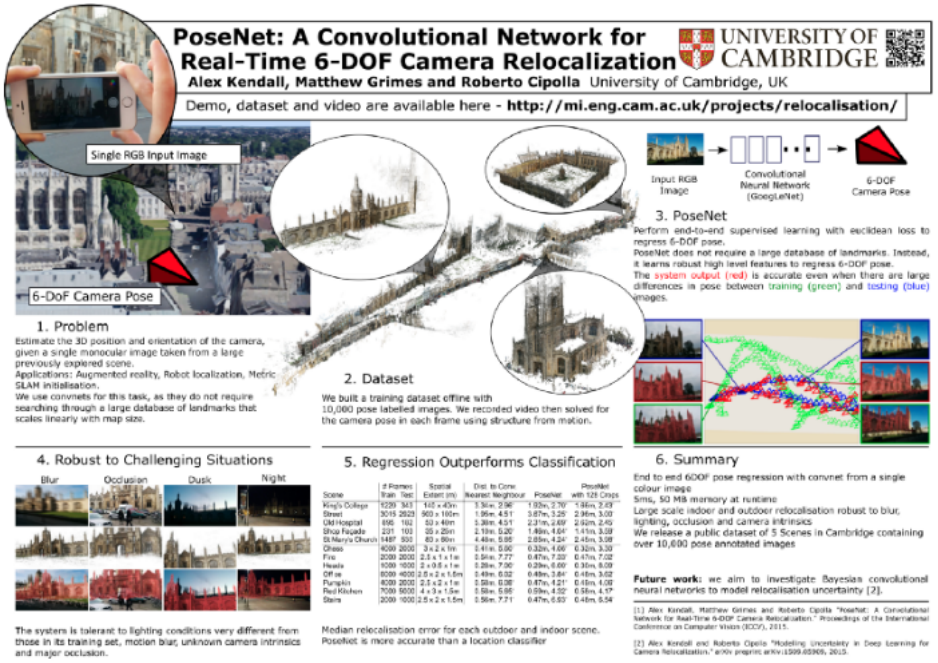


Figure 3.8: PoseNet overview [42]

3.1.10 Depth from motion

Some MAR systems, such as the now abandoned Google Tango devices or the Iphone X, come with depth sensors, while other devices can be extended with additional hardware [9]. These systems use expensive hardware and is not as easily scalable as depth-estimation from monocular cameras readily available in millions of devices already on the market today.

Believable AR experiences relies on consistent and robust augmentations. Interacting realistically with the environment requires precise occlusion of virtual objects. A prerequisite for this is a depth map, either by a depth sensor or depth from motion. Extensive research has been done on depth estimation from cameras in the field of 3D reconstruction, computer vision, as well as deep learning. State-of-the-art research from Google [81], showcasing keyframe-based, real-time, dense, depth estimation on a mobile phone using a planar, bilateral solver (a keyframe selection strategy) on a single CPU core, can be used to handle occlusion problems efficiently in the future. The system is also independent of the underlying VIO or SLAM implementation. Other depth-estimators using deep neural networks Godard et al. [31] and Kendall et al. [43] is outperformed by Valentin et al. [81], as seen in 3.9.

[9] https://structure.io/

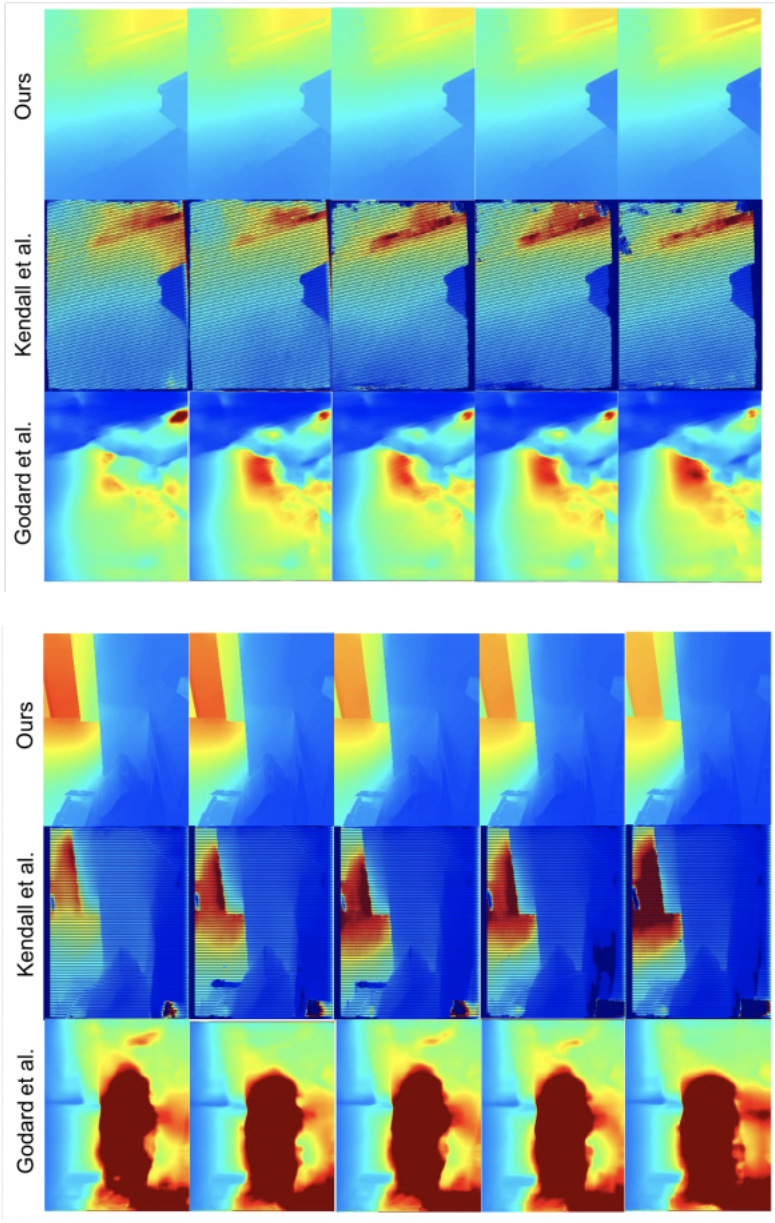


Figure 3.9: Side-by-side comparison of depth estimation on the Middlesbury Stereo Dataset from the Google team [81] paper.

3.2 3D Visualization on the web

Many studies have been done on 3D visualization on the web. We find a two different classes of visualization toolkits for 3D web applications, virtual globes and generic graphics engines. The two differ mainly by the level of graphics abstractions and including GIS tools. A large amount of virtual globes are aimed at 3D GIS applications, and naturally bring a higher level of abstraction and useful GIS tools. Generic web 3D libraries bring a lower level abstraction to WebGL.

3.2.1 Imperative 3D content

The emergence of HTML5 and WebGL sparked the development of many different 3D web visualization libraries. 3D graphics on the web is more available than ever, libraries/API's such as Cesium.js [10], three.js [11], X3DOM [12], OSG.js[13], WebGlGlobe[14], Web World Wind [15] and Google Earth Engine [16] exist today to visualize a broad range of 3D geospatial data.

3.2.2 Declarative 3D content

Behr et al. [7] proposed a unified integration model between X3D and HTML, in a similar manner that 2D graphics as SVG was integrated with HTML. Achieving a tight integration, an in-place, declarative, rendering of 3D content on the web that is standardized is important to evolve and improve HTML. [39] provides a introduction to the state of affairs of integrating a 3D standard in HTML. X3DOM is still evolving [8].

Online virtual globe-style visualizations have also gained traction over the last years. Google Earth was once the most popular virtual globe, but is since discontinued in favor of Google Earth Engine (GEE). GEE leverages cloud-computing services and analysis capabilities and can be regarded more of a remote sensing platform. A web-based code editor can be used to access the GEE Javascript API.

NASA World Wind [9] started out as a C# based desktop application, but is today extended onto Java desktop(2006), Android(2012), iOS(2013) and Javascript(2014) as pointed out by [68]. Web World Wind is the web-based version, which is based on HTML5 and WebGl. NASA World Wind aims to bring large scale, geographical data and GIS functionalities to the layman and to foster collaboration between scientists, developers and other stakeholders.

Cesium virtual globe

Other smaller, open-sourced web globes include the OpenSceneGraph extension osgEarth and OpenWebGlobe. Further comparisons of the web globes can be found in [44].

[10] https://cesiumjs.org/
[11] https://threejs.org/
[12] https://www.x3dom.org/
[13] http://osgjs.org/
[14] https://experiments.withgoogle.com/chrome/globe
[15] https://worldwind.arc.nasa.gov/web/
[16] https://earthengine.google.com/

Earlier web-based 3D visualizations such as [36], relied on plug-ins and java3D for persistent and extended functionality in 3D web GIS.

Chapter 4

Use-Case

This chapter presents the Open Augmented Reality Cloud as a use case for Mobile Augmented Reality and spatial computing. We will review the current state of AR-Cloud vendors and look at the importance of non-proprietary, crowd-sourced AR-cloud. Several issues arise, such as privacy, interoperability, research and development. We explore these key issues in the next chapter.
An outline of some specific use cases that are imagined AR Cloud infrastructure and improved positiong:

- Modelling of constructional or architectural components in a virtual environment based on crowd-sourced point-cloud.
- Deviation of geospatial data. A crowd-sourced sparse point-cloud can be updated far more frequently than remote sensing, areal imagery or advanced mobile mapping units can. Geospatial data administrators and managers can find discrepancies in their data.
- Reliable data source for emergency response vehicles.
- Spatial queries of semantically categorized features.

Figure 4.1: Newly established Open Augmented Reality Cloud organization [1]

- Aid any kind of visual-mapping system on various autonomous robots such as UAV's, cars or robots.

4.1 Current state of the AR-Cloud ecosystem

An increasing amount of AR organizations realize that cloud infrastructure is required to succeed with adoption of both developers and consumers of AR content and AR-driven applications. Since Ori Inbar coined the term AR Cloud [38], the concept has gained a lot of attention. Major players such as Apple have publicly stated their interest in succeeding and becoming market leaders in Mobile AR. Although Google scrapped their Tango project, most of the tech is retained on their ARCore platform. AR as a whole is widely regarded as a disruptive technology, impacting the core of how we access and share information. The capital that is being moved into the AR space is significant, with the latest reports valuing the total AR market at over 10 billion USD[2].

4.2 Interoperability and standards

Interoperability is the degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context. As a consequence of interoperability, systems can exchange syntactic and semantically meaningful information in a certain context.

In the context of AR, [66] defines Open and Interoperable AR as:

```
Complete end-to-end systems in which
modular components can be supplied
by multiple vendors and still have the
same workflow and experience quality
```

Open and interoperable AR provides many benefits. As with ordinary data interoperability, it increases a users discovery, sharing and consuming of AR data

Some work has been done to gather enterprise AR technology such as The Augmented Reality for Enterprise Alliance (AREA) [3]. Founded in 2013, AREA is a global non-profit organization to help enterprises maximize the impact of AR by sharing resources, research, dialogue, guiding adoption and clearing the path for interoperable AR-enabled enterprise systems. Architectural proposals such as [74]
One of tha major challenges of AR aand VR has been the lack of standards for hardware, software and interfaces. Several SDO's (Standards Development Organizations) work on specifications for AR, 4.1 gives an overview of the standards landscape of AR.

A recent report on the interoperability challenges we find in AR can be found in [66]. Several areas of potential for interoperability is recognized, and is highly relevant for the

[1] https://www.openarcloud.org/
[2] https://www.marketsandmarkets.com/PressReleases/augmented-reality-virtual-reality.asp
[3] http://thearea.org/

Name	Governing Body
KML	OGC
Web Map Service	OGC
GML	OGC
GeoLocation API	W3C
GeoRSS	GeoRSS
GeoJSON	
RFC5870	IETF

Table 4.1: Location standards

Name	Governing Body
SensorML	OGC
Sensor Observation Service	OGC
Device Motion Event Specification	W3C
Device Orientation Event Specification	W3C

Table 4.2: Sensor API on the web

Open AR Cloud. Although this reports is focused on enterprise AR, many points are highly valid.

Camera Calibration - Cameras is the most important sensor of any AR system, regardless of platform. Detecting and tracking the user pose is done mainly through visual input. Interoperable AR needs a generic pose metadata that can be used by a standardized set of systems and AR engines. Generic pose metadata is reliant on a vendor-neutral, camera calibration process and a way of encoding this data. All AR engine should require their developers to include this camera calibration process. Photometric calibration is also an issue, as errors amount quicker with uncalibrated cameras. However, online calibration is both possible for photometric and geometric calibration of cameras [10] is made possible, a huge win for direct methods.

Interfaces for Vision-Based Tracking Components - Vision-based tracking is highly dependant on optimized algorithms. As a result of the diverse set of AR-platforms, there will never be a single CV-algorithm to standardize across. However, the field of AR would benefit form interfaces between AR execution engines and the tracking algorithms.

Khronos group - The Khronos Group develop royalty-free open standards for 3D graphics, Virtual and Augmented Reality, Parallel Computing, Neural Networks, and Vision Processing. A handful standards are especially relevant for Augmented Reality. Further relevant standards for AR can be found in 4.3. The following five standards aim to provide performance, power and portability to AR, while reducing platform fragmentation. Khronos standard adopters must pass conformance tests defined by Khronos to be formally compliant to a specification. conformant products and their vendors can be found online [5]

Open Geospatial Consortium - The Open Geospatial Consortium is a non-profit organization dedicated to developing open specifications for geospatial services and technolo-

[5] https://www.khronos.org/conformance/adopters/conformant-products

Table 4.3: AR relevant SDO's

SDO	AR relevance	Ratified standard	Standards in progress
MIPI Alliance	Hardware interfaces	Camera: CSI-2/3 Display: DSI-2 Sensor: I3C Low Latency Interface: LLI Physical layer: C-PHY, D-PHY, MPHY	Touch interface
Khronos Group	Specifications to connect software to silicon	glTF WebGL Vulkan OpenVX	OpenXR StreamInput OpenKCam
Open Mobile Alliance (OMA)	Defines open mobile service specifications	MobileAR Enabler 2 / 3	MobileAr Enable 1/4
IEEE	Broad range of industry standards	50 standards relevant to AR[4]	ARLEM VR/AR Working Group (VRAR) IEEE P2048.*
ISO	Wide range of IT technology standards	MPEG-* X3D H-Anim	3D Media Formats Point Cloud Compression XR reference model
OGC	Standards to "geo-enable" the web	ARML2.0, GML, IndoorGML, CityGML	POI, Point Cloud
World Wide Web Consortium (W3C)	Web-based standards for content and tools	WebRTC, CSS, SVG, Generic Sensor API, HTML5	WebVR, Generic Sensor API, XR Services Community Group

gies. OGC specifications is primarily focused on interoperability, management, search, delivery and presentation of geospatial data. Specifically for AR, OGC provide the Augmented Reality Markup Language (ARML). Currently at version 2.0, it allows description of virtual objects in a AR scene based on anchors, as well as interactions such as events. ARML focuses on mapping georeferenced *Points-of-Interest* and their metadata, and mapping between POI content providers and the AR application. The specification itself borrows heavily from other OGC standards such as KML and GML. The object model uses concepts such as *Features*, *Anchors* and *VisualAssets*. Features alike to GML features with accompanying anchors. Visual assets describe appearances of digital objects. The Anchors describe location and spatial relationship between the real world, scene and virtual object. Similar but independent specifications to ARML are MobAR from Open Mobile Alliance and ARAF from ISO.

- glTF - GL Transmission Format, run-time asset delivery.
- WebGL - Javascript binding to OpenGL ES graphics API
- Vulkan - Next generation graphics API.
- OpenVX - Vision-based processing acceleration standard.
- OpenKCam - Low-level camera and sensor API (in progress).
- StreamInput - General-purpose framework for advanced sensor discovery (in progress).

glTf serves 3D scenes and models effectively and optimizes run-time processing of interactive 3D applications. With a wide industry support, glTf is analogous to image JPEG format[6].
WebGL provides a hardware acceleration compliant graphics API for web browsers through the HTML5 canvas element. Binding to OpenGL ES with Javascript, WebGL is responsible for bringing plugin-free 3D the web [7].

4.2.1 Vulkan

Vulkan was released in 2016 and includes a high-efficiency, platform-agnostic graphics API, that handles a wider array of platforms compared to OpenGL. Whereas OpenGL was designed mainly for heavy workstation and CPU work, Vulkan meets the demands of modern platforms, such as mobile and embedded devices, as well as traditional PC's. Vulkan-based AR presentation systems will provide lower overhead, portability, low latency and predictable performance, as is recognized as critical for AR. Vulkan can be regarded as a lower-level version of OpenGL.

[6] https://www.khronos.org/gltf/
[7] https://www.khronos.org/webgl/

4.2.2 OpenVX

OpenVX was developed to accelerate computer vision and neural networks, ushered by the need of lower power consumption and high performance on a wide array of execution architectures. Specifically designed low-power, real-time and embedded systems, it is ideal for mobile augmented reality. OpenVX uses a graph as its algorithmic pipeline, consisting of language-angostic nodes. Each node is an instance of a CV function (eg. KLT Optical Flow implementation), typically provided by the hardware vendor. As such, a vendor supplies a OpenVX implementation for its The node contains data references, return values and performance information. The API aims for efficient transfer of execution nodes between CPU and GPU. User-defined nodes, unlike standard vendor-defined nodes, can also be developed under certain restrictions. User-defined nodes are necessary to create advanced CV applications using OpenVX, as the core function set is small. This framework gives OpenVX flexibility that supports optimization of performance and power consumption. Currently, there is not yet a OpenVX implementation supplied for Qualcomms Snapdragon chips, which are commonly found in Android smarthphones. While OpenVX have overlapping functionality with OpenCV, they are complimentary to each other. OpenCV is a *de-facto* standard and a open library for CV algorithms, developed and maintained by the community without any formal specification. OpenVX is a more optimized, fully specified and narrower API. A neural network extenson to OpenVX was introduced in 2018.

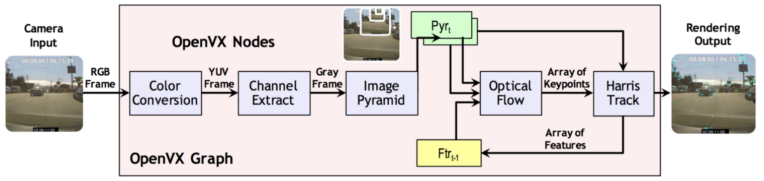


Figure 4.2: Khronos OpenVX Graph

4.2.3 StreamInput and OpenKCam

StreamInput aims for a general, low-level, cross-platform advanced sensor API. It should provide sensor discovery and sensor synchronization for advanced multi-sensor applications such as mobile AR. Uses standardized UST timestamps on every sensor reading, simplifying fusion processes. StreamInput also proposes *Context awareness* in the API, enabling developers and vendors to query the device contextual queries such as ”am i in a car” or ”am i in a hand or a pocket”.
OpenKCam is intended as a open standard for low-level control of mobile and embedded cameras. Proposed by the Khronos group in 2013, it highlights uses-cases such as Image Signal Process control (ISP), rolling shutter elimination, camera parameter retrieval and motion sensor synchronization with StreamInput are highly relevant for Augmented reality. The ISP can offload Together with OpenVX, these standards can be used together for enabling a interoperable AR future. Both standards are not yet fully specified.

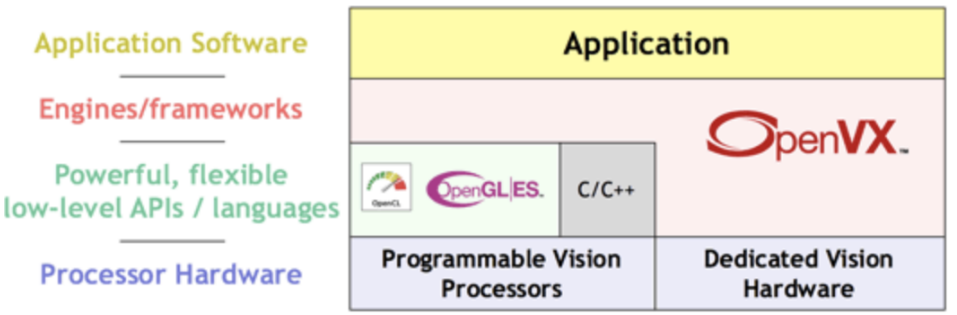


Figure 4.3: OpenVX Framework

4.2.4 OpenXR

The AR and VR industry is highly fragemented. Proprietary solutions induce costs to port applications from one run-time platform to another. OpenXR is a application-facing API for and VR, proposed by Khronos at SIGGRAPH 2018. Initially more focused on VR games standardization, it hopefully can be extended to AR applications running on different AR engines in the future.
[8]

4.2.5 Visual Positioning standards

The primary constraint of visual positioning is a map of distinct features. such features originate from different sources, such as 2D images, 3D point clouds, RGB-D images and more. Many different platforms supply these images such as mobile phones, robots, UAV's and other devices. Fantasmo.io plans to open-source its Camera Positioning Standard[9] (C.P.S.), hoping to unify the next generation of 3D spatial mapping. Built on open and interoperable principles, C.P.S. supplies a general feature-based map and infrastructure for data exchange on common CV platforms.

4.2.6 Social implications

Mobile systems are particularly exposed to societal acceptance issues, as the use cases often arise in social settings. Interactions with AR applications disrupting in the public space, especially so in small-scale areas such as public transport or at a shopping street. Researchers have found that interaction with AR systems have to be subtle to be socially acceptable [18]. Large-scale, outdoor smart phone AR interaction can possibly impose negative social reactions, as the user interface requires one to point cameras at objects. Safety concerns such as physical fatigue or reduced awareness arise when using MAR systems. Physical intrusion imposed on others while holding a smart phone continuously and over long stretches of time may cause social backlash.

[9] https://www.camerapositioning.io/about/

4.3 AR Cloud Vendors

The following section contains an overview of vendors and technologies currently supporting AR Cloud solutions or will in the near future. The vision being that all vendors will participate in a open, interoperable and private cloud eco-system. Business use-case heavily influences the approach from a technology point-of-view, furthermore most vendors keep their solutions proprietary and only available through APIs and SDK's.

We only regard vendors based on marker-less methods, as it is most relevant for the OARC.

4.3.1 AR Cloud solutions

This section provides a brief review of business attempting to innovate with regards to the AR Cloud and Mobile AR in general. As most of the tech is proprietary, it is very limited what we know of technical details surrounding their solutions. A summary and reference is provided for each vendor.

4.3.2 Google ARCore

Have introduced their own process named Concurrent Odometry and Mapping, similar to SLAM. Google keeps this tech proprietary and patented, and the patent reveals that it utilized traditional SLAM with IMU integration. Recently announced "Cloud Anchors" in an attempt to enabling multi user experiences. These anchors are generated in the cloud on Google servers, but require a pre-scan of a user that wants to resolve an anchor for others to use. Other users need to relocalize in that space, that is, point cloud matching.

4.3.3 Apple ARKit

Apple recently acquired Metaio and FlyBy Media, strengthening Apples tech in SLAM/VO, sensor fusion, 3D tracking and image recognition. ARKit 2.0, released in June 2018, includes support for the ARWorldMap. The ARWorldMap supports saving and sharing anchors, objects and maps, through iOs peer-to-peer framework `MultipeerConnectivity`. Other features of ARKit 2.0 is object detection and tracking. Developers can supply ARKit features describing a given object, and ARKit can in turn track this object. Improved environmental texturing with a trained neural network and a new AR 3D object format, USDZ, also appear in the feature list. Apple takes AR seriously, showcasing that they are the current market leaders on the AR mobile development platform.

- Immersal[10] - Finnish startup for markerless, multi-user AR with a patent pending visual positioning system. Expands upon ARKit and ARCore with their own AR Cloud SDK.
- YouAR[11] - Cross-platform, interoperable multi-user AR backed by a Global AR Cloud database. Tech is based on monocular SLAM with environmental occlusion

[10] https://immersal.com/
[11] https://youar.io/

tech, without the need for extra hardware. Persistence is hosted through Aamazon Web Services. Already showcased cross-platform, real-time AR interactivity between ARCore and ARKit platforms.

- Placenote[12] - Previously known as vertical.ai. Provdies a point cloud mapping and localization persistent in the cloud. Small-scale
- Scape[13] - London-based computer vision startup. Provids a AR cloud version of its Visual Positioning Service and Vision Engine, with the goal of enable visual robots to understand its surroundings. Currently live in select cities, Scapes tech lets you query the Visual Engine with images and video to determine position in sub-cm accuracy. Technology is proprietary, accessed through a SDK. Available on iOs, Android and Unity platforms.
- 6D.ai[14] - Based out the computer lab of Oxford University, 6d.ai have strong academic ties. Provides a SDK that currently is in beta, currently supporting iOs. On device computing is done as much as possible, which is great for the privacy aspect.
- Bluevision labs[15] - Recently acquired by the ridesharing company Lyft, as a part of their autonomous vehicle ambitions.
- Project Whare - Newly unveiled project from Samsung. Aims at solving the persistence and multi-user AR problem. Solution is compatible with ARKit and ARCore, as well as Unity.
- Fantasmo.io[16] - Open-sourced approach to map the world in a crowd-sourcing fashion. Will release open standards and tools for capturing and mapping the world in 3D. Fantasmo relies on geo-referenced photos stored in the cloud and using their proposed Camera Positioning Standard[17] (C.P.S) to relocalize. C.P.S aims to become the 6DoF equivalaent of GPS.
- Ubiquity6[18] -
- Niantic Real World Platform - Rcently acquired the startup Escher Reality [19]. Aims to help Niantic bring persistent, contextualized and semantic AR to its users. Heavily focused on AR and geospatial games. Niantic calls it a world-scale persistent state engine.
- Selerio.io [20] - Incorporates semantic meaning to their maps, aiming to solve smart AR. By using neural networks in addition to SLAM, Selerio can infer contextual meaning and depth prediction through a neural networks. Limited by having to pre-map an area, and poor performance in large scenes.

[12] https://placenote.com/
[13] https://scape.io/
[14] https://www.6d.ai/
[15] https://www.bluevisionlabs.com/
[16] https://fantasmo.io/home
[17] https://www.camerapositioning.io/about/
[18] https://ubiquity6.com/
[19] http://www.escherreality.com/
[20] https://www.selerio.io/

- Sturfee[21] - Launched SDK with access to city-scale AR cloud services, currently supporting 10 cities in the US.

4.4 5G and Edge computing

Physical infrastructure is important for the AR Cloud, given that the data must be transmitted, processed in real-time, offload computing resources and re-localize many users at once. New technological advancements in networking, such as the next generation mobile network 5G and state of the art *Edge Computing*. In the literature there is overlap with the term *Fog Computing*, which is a more generalized concept than Edge Computing. The Open FogConsortium [22] defines it as

a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Thing.

while The European Telecommunications Standards Institute defines Mobile Edge Computing as

"offers IT service and cloud-computing capabilities at the edge of the mobile network in an environment that is characterized by proximity, ultra-low latency, and high bandwidth. Furthermore, it provides exposure to a real-time radio network and context information."

Edge computing brings the cloud "closer" to the user, and is seen as the natural evolution of mobile base stations. Edge servers bring computational resources and services physically closer, drastically reducing latency. [77] showcases the use of Mobile Edge Computing in a industry support Augmented Reality system. The authors developed a intermediate edge server that received compressed video and IMU data to track with a EKF based approach. A remote operator would received the video and annotate still frames which the edge server in turn Augmented matched and registered in 3D for the user. Total end-to-end latency was 50 ms which can be further improved when 5G technology is fully realized.

Next generation network technology, 5G, will have larger bandwidth and virtually no latency, promising speeds 100 times faster than current 4G wireless networks. Once it is implemented and new generation devices that supports it are on the market, 5G will provide the possibilities for AR, VR, IoT, autonomous vehicles and many other technologies. ETSI are currently working on standardizing concept of Mobile Edge Computing (MEC) [37].

5G and Edge computing are, along with AR, key emerging technologies that will complement each other in the near future.

[21] https://sturfee.com/

[22] https://www.openfogconsortium.org/

4.4.1 Privacy

The problem of privacy and security exist in any modern networked system in which sensitive data is processed, as no system is 100% secure. Especially in visual system, large amounts of data is captured and processed, and possibly held by third-party services. Coud services are increasingly used compared to conventional in-house servers, providing storage, computing and other services at demand. MAR applications rely on positional and image data, while future collaborative MAR applications will expose even more data to service vendors. Concerns are raised regarding trust [11], devices such as Google Glass sparked the discussion on facial recognition capabilities of AR applications Acquisti et al. [2]. Their research proposed openness, use limitation, purpose specification and collection limitation as a general guideline framework to remedy the issue. Other approaches such as Shu et al. [79], a system in which privacy disconsent could be expressed with hand gestures. Edge computing can potentially prove to be a privacy hurdle as offloading high definition images, video streams or sensor data to a Edge server exposes sensitive data. Trusted Execution Environments, Public Key Infrastructure and encryption are some tactics that can be utilized as proposed in [87].
Vendors in the AR Cloud space should strive for maximal on-device storage and computing, where possible. As Edge Computing matures, servers from the cloud-periphery will hopefully provide services compliant privacy standards. The OARC aspire to uphold the following core values [23].

- Be open, transparent and interoperable
- Be Guided by immutable values
- Use standards created by consensus of diverse active community
- Ensure authenticity and accountability via distributed data and technology
- Respect and protect the privacy and rights of individuals
- Preserve the real and intellectual property of creators
- Foster innovation and opportunity
- Incentivize users to generate data and content
- Comply with local and global rules and regulations

[23] https://www.openarcloud.org/

Chapter 5

Proposed solution

To solve the general problem of increased mobile localization accuracy through the use of GPS/GNSS, inertial sensors and CV techniques, we propose a architecture for Android based smartphones, using block diagrams. Additionally, the system design includes a client/server based prototype for visualization of geographic pose provided by the Android client. We do not propose to solve the whole pipeline with object detection, localization and mapping. Rather, we want to showcase a real-time system for visualizing the pose of a imagined client providing data to improve a part of the AR Cloud.

Section 5.1 shows the architecture of such system. The mobile client is intended to run on different mobile platforms, capturing high definition images and video that can be processed remotely to improve a global map and retrieve its improved positioning through keyframe matches. The geopose client architecture can be found in Section 5.2.

5.1 Visualization client

The following sections shows how a translation from mobile sensor data, e.g. the geopose, to the virtual, 3D environment on the browser. The goal is to visually orient and place the mobile device precisely in the virtual environment. Using mathematical theory and techniques we recognize a set of transformations to properly align a device in the virtual environment to the corresponding real-world pose.

5.1.1 Choosing a Client library

We review three different WebGL based libraries to find the most suited for the geopose visualization. The following table shows a analysis from previous work [70].

The Cesium application programming interface is rich and well-suited for solving problems in the geospatial domain. At the core, Cesium resembles a generic graphics engine. Layers such as renderer, scene and primitives compose the graphics stack. The Cesium API consists of many functions to manipulate WebGL on a reasonably high level,

Platform	**Web**		
Name	Cesium.JS	Three.JS	X3DOM
Type	Library	Library	Framework
Documentation & Userbase	Code samples Well documented Tutorials Sandcastle application Simulation and GIS	Code samples Well Documented General graphics and games	Well documented
Support & maintenance	>27.000 commits >3.900 stars 173 contributors 87 releases Frequent releases Exceptional contributors Forum	>26.000 commits >48.000 stars 1031 contributors 92 releases Active community and forum	>6000 commits>540 stars 48 contributors 13 releases Mailing list Forum
Standards compliance	Activly adopts and contributes to standards OGC and Khronos standards	Khronos standards	ISO compliant Actively adopts and contributs to standards
Graphics library	WebGL	WebGL	WebGL
License	Apache 2.0	MIT	MIT/GPL
Spatial reference	Yes	No	Yes
Graphics API level	High	Medium	High

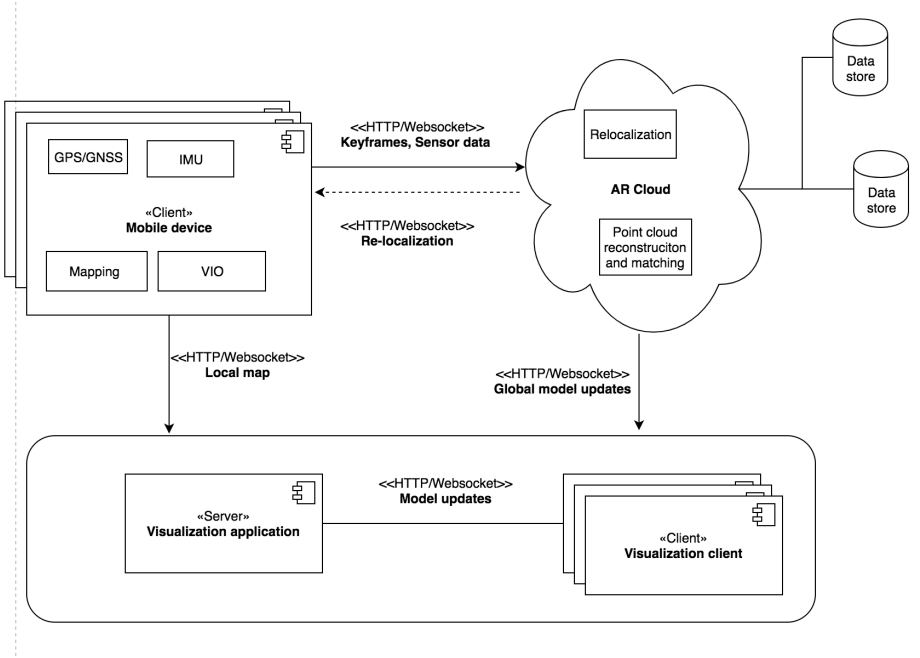


Figure 5.1: Conceptual system of a visualization system for the AR Cloud

while also encapsulating a solid math module that includes vectors and matrix algebra.

A proper GIS 3D visualization includes terrain model. Cesium includes Ion, a cloud based tile, imagery and terrain services based on free and open data. To access these services, a free Cesium Ion account is needed. After signing up a Ion, an access token must be retrieved and passed on to the Cesium.js implementation. The token gives access to data residing on users cloud account. Several default dataset are available from the go, with options to upload own data. We use the default world terrain. Cesium fuses several terrain providers into a *Quantized-mesh* terrain tileset that is optimized from streaming over networks. A qunatized mesh is simply a multi-resolution quadtree pyramid of heightmaps. Terrain resolution is varying, from 1-2m in England to 30m in the rest of Europe.

```
1 Cesium.Ion.defaultAccessToken = config.ionToken;
2 export const viewer = new Cesium.Viewer('cesiumContainer', {
3     terrainProvider: new Cesium.CesiumTerrainProvider({
4         url: Cesium.IonResource.fromAssetId(terrainId.WORLD_TERRAIN)
5     }),
6     timeline: false,
7     animation: true,
8     navigationHelpButton: false
9 });
```

Listing 5.1: Initialization of Terrain resource from Cesium Ion cloud services

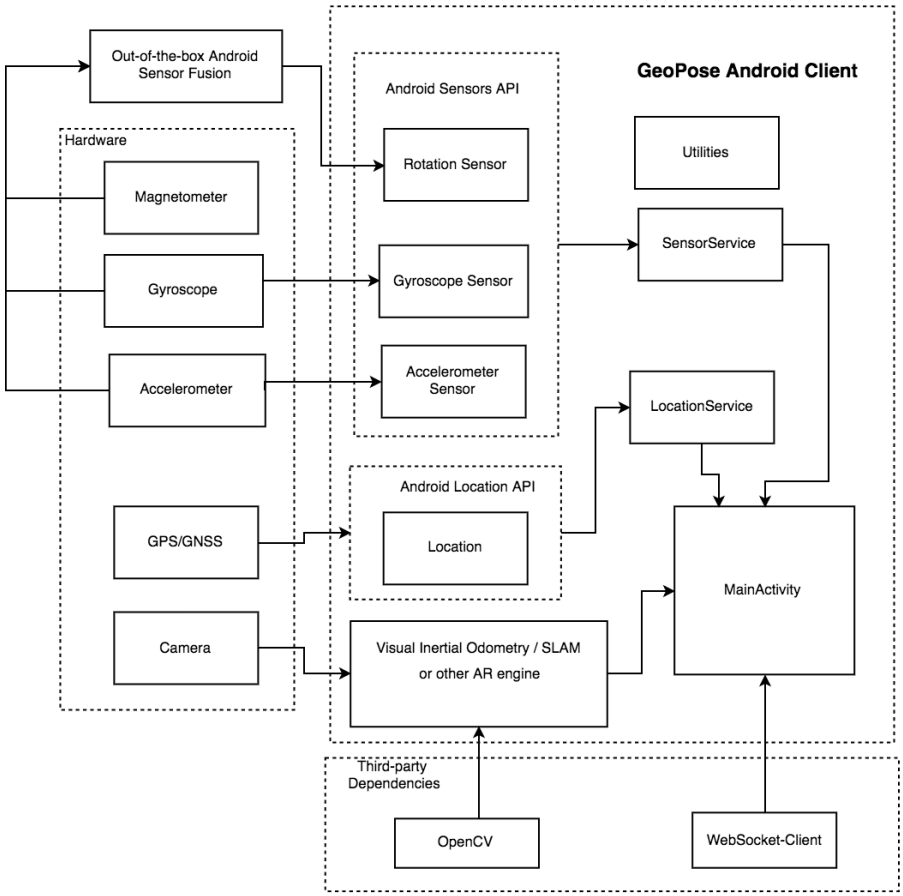


Figure 5.2: Block diagram of Mobile client device

Source	Attribution
NZ 8m Digital Elevation Model (2012)	LINZ - Land Information New Zealand
Digital Elevation Model (DEM) of Australia derived from LiDAR 5 Metre Grid	Based on Geoscience Australia material
LIDAR Composite DTM - 1m & 2m	Environment Agency copyright and/or database right 2015. All rights reserved.
Digital Elevation Model over Europe (EU-DEM)	Produced using Copernicus data and information funded by the European Union - EU-DEM layers
CGIAR SRTM	CGIAR-CSI
GTOPO30, SRTM, and National Elevation Dataset (NED)	Data available from the U.S. Geological Survey

Table 5.1: Attributions of datasets used by Cesium Ion to generate the World Terrain.

Dependency	Version
OpenCV w/ contrib	3.4.1
Socket.IO client	0.5.2
Google Gson	2.5.8

Table 5.2: Android client dependencies

The only dependency needed for the Cesium web visualization client itself is the Socket.IO-client package, easily retrieved through the node package manager NPM[1].

5.2 Geopose client - Android

OpenCV with the contributions module[2] was imported as a native module dependency in our project. Android allows native code (C++) to run and communicate with Java code. Java Native Interface serves as a wrapper around the native code of OpenCV. This version includes modules and extra features that are not yet added to the official OpenCV package. Other dependencies except OpenCV are added normally through the Gradle build system for Android Studio.

5.2.1 Android permissions

Firstly, the Android requires explicit access to use certain services such as networked communication, camera location. A simple `checkPermissions()` on launch which

[1] https://www.npmjs.com/

[2] https://github.com/mainvooid/opencv-android-sdk-with-contrib

loops through an array of permission ID's, querying the the user for permissions that needs to be granted. The Android Manifest.xml file, which contains essential information for the system, also needs to have a declared tag for each permission.

5.2.2 Background services

Two backgroundservices are defined, a SensorService and a LocationService. Each runs on its own thread in the background, where it continously listens for a change broadcast by the Android Sensors API or Location API. Sensor readings can be intercepted in the *OnSensorChanged* method and processed here. Due to the high frequency, we chose to compute a running average of the synthetic `TYPE_ROTATION_VECTOR` over $0.5s$ to even the readings [3].
The location service simply listens for changes in location that crosses a certain threshold. Both services broadcast their readings further to the main thread, which holds the socket connection. Braodcasting between threads is done with the Observer pattern implemented with the `LocalBroadcastmanager`.

5.2.3 Android MainActivity

This class holds the Android Activity that is launched on start. In short, here is the Socket client initialized and the OpenCV interface to the camera implemented. The socket itself only needs the socket.io server URL and handlers for received socket events. Emitting events is done by simple JSON strings constructed with the Gson Library[4].

5.3 Server solution

A server is needed to update the visualization with real-time readings from the geopose client and fetch high-accuracy height data from a remote API, as well as serving 3D content. Systems dependencies are listed in figure **??**. The server environment chosen is Node.JS, a JavaScript run-time environment based on Chromium V8 designed to execute outside of the traditional browser. Node.JS is asynchronous, non-blocking and lightweight, with loads of modules. We employ the http server framework Express to serve our files, as a proxy and keep the websocket implementation. `Promise` based approach is a common way to implement asycnhronous code that reads like synchronous code. A `Promise` is a proxy value that might be fulfilled or rejected according to the response from the remote service or request. Below is a server code snippet that intends to read a heightmap from a remote service or file system, with a set bound in UTM coordinates. In our case, the heightmap is hex encoded png with dimensions 500px by 500px. The ground sampling distance is 1m, as seen in 5.3. Each asynchronous operation returns a `Promise` which in turn executes `.then(function())` when the promise resolves. `img.scan()` and `img.getPixelColor()` is supplied by the jimp[5] library.

[3] https://www.johndcook.com/blog/standard$_deviation/$
[4] **https://github.com/google/gson**
[5] https://www.npmjs.com/package/jimp

Dependency	Version
Express	4.16.14
MongoDB	3.1.10
Socket.Io	2.2.0
Jimp	0.6.0
proj4	2.5.0

Table 5.3: Dependencies in the server implementation

```
const getHeightGrid = (imagePath, west, south, east, north) => {
const promise = jimp.read(imagePath).then((img) => {
    const grid = {
        north,
        east,
        south,
        west,
        heights: [],
        height: img.bitmap.height,
        width: img.bitmap.width,
        cellSizeEW: (east - west) / img.bitmap.width,
        cellSizeNS: (north - south) / img.bitmap.height
    };
    img.scan(0, 0, grid.height, grid.width, (x, y) => {
        const rgb = jimp.intToRGBA(img.getPixelColor(x, y));
        grid.heights.push(rgbToHeight(rgb));
        if (x === grid.width - 1 && y === grid.height - 1) {
            console.log('img scan complete');
        }
    });
    return grid;
});
return promise;
};
```

The height is computed by `rgbToHeight` which computes the height as from the combined RGB values. The data is served from a WMS server, encoded as a .png image file.

```
const rgbToHeight = (rgb) => {
    const r = rgb.r * (2 ** 16);
    const g = rgb.g * (2 ** 8);
    const b = rgb.b * (2 ** 0);
    return (wmsMaxHeight - wmsMinHeight) * ((r + g + b) / maxRgbAsNumber)
        ↪ + wmsMinHeight;
};
```

5.3.1 Geodetic space to image pixel space

Bilinear interpolation is the natural extension of linear interpolation, in which linear interpolation is performed both in the horizontal and vertical direction. Common areas of application are computer vision and image processing.

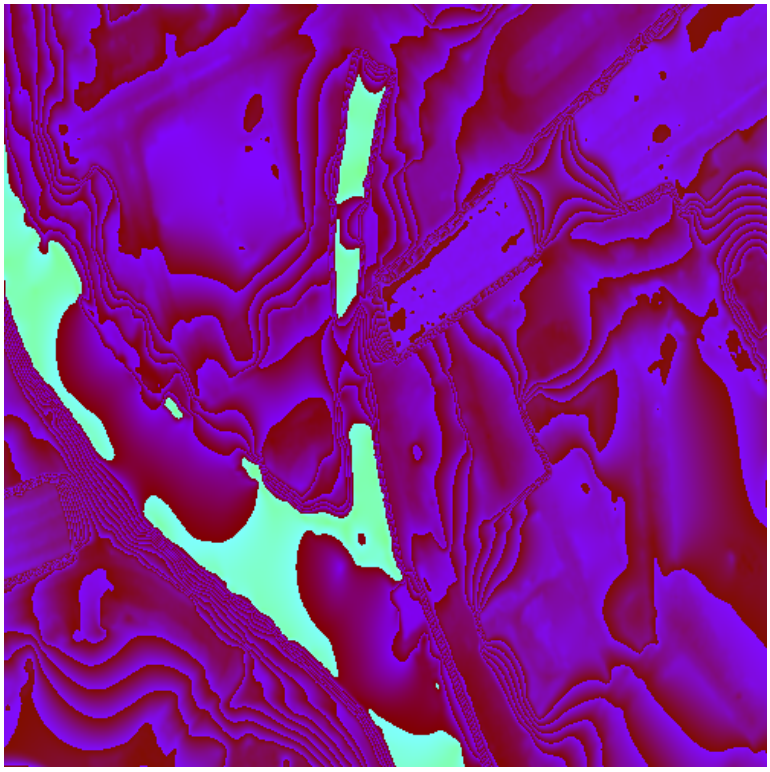


Figure 5.3: Heightmap from the WMS service. Green-like areas signify bodies of water with a elevation value of 0.

To be able to use the generated heightmap, we have to transform geodetic coordinates to pixel coordinates. We firstly transform from Geodetic coordinates to the UTM map projection plane. The library proj4.js[6] provides a open-source implementation of common coordinate system transformations. Proj4 requires source and target reference systems as a coded WKT string with a EPSG coded reference system. A map projection to UTM zone 33N from the default WGS84:

```
    const targetSrs = '+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units
        ↪ =m +no_defs';
    ...
    const pUtm = proj4(targetSrs, {
            x: p.lng,
            y: p.lat
        });
```

The resulting javascript object with key (x, y) is furthermore input into a function that bilinearly interpolates the given (x, y) coordinate pair in the heightgrid. The following code snippet shows the function that takes a geodetic coordinate pair and produces its orthometric height according to the heightmap.

```
const getHeightFromLatLng = (lat, lng) => {
    const gridPromise = getHeightGrid(imgPath, west, south, east, north)
        .then(grid => {
            const result = generate2DPoints([{
                lat,
                lng
            }], proj4transformCode);
            return generateHeightsForPointsFromGrid(result.utmPointArray,
                ↪ grid, geoidHeight);
    });
return gridPromise;
};
```

The `generateHeightsForPointsFromGrid()` function takes a `pointArray`, `grid` and `geoidHeight`. Since we our visualization software use the WGS84 Ellipsoid as the height reference to model the earth, we need ellipsoidal heights.

$$h = N + H \tag{5.1}$$

Where h is the ellipsoidal height, g the geoidal seperation and H the orthometric height. Furthermore, the bilinear interpolation step is done as following since we know the grid cell length and the four corners $(h0, hx, hy, hxy)$ the point (x, y) falls between, yielding

```
    const bilinearInterpolation = (h0, hx, hy, hxy, x, y) =>
    h0*(1 - x)*(1 - y) + hx*x*(1 - y) + hy*y*(1 - x) + hxy*x*y;
```

The combination of the preceding functions enable us to map Latitude and Longitude to a high accuracy height.

[6] http://proj4js.org/

Chapter 6

Experiment

In this chapter, we explore some approaches to visualize geographic pose in the Geopose system explained in the previous chapter. For the experiment the following devices was used. LG G6 smartphone, locally running server and a Cesium visualization client. Server and visualization clients is running on MacBook Pro with a 2.8 GHzz Intel Core i7, Radeon Pro 560 dedicated graphics and 16 GB of RAM running macOS High Sierra. Development IDEs are Android Studio and WebStorm developed by JetBrains [1].

6.1 Camera calibration

We want to perform 3D-2D mapping between the camera frame and global frame. By obtaining the camera *intrinsics* and distortion coefficients, pose recovery is made possible. The coefficients camera matrix K, as defined in 2.1, are found by detecting a circle grid pattern. which are needed for pose recovery. A camera-calibration program was developed for Android, based on the OpenCV implementation. In calibration mode, the program calls `findCirclesGrid()` on each frame, attempting to detect the grid. If detected, a colored graph shows up on the screen, connecting the circle centers. Touching the screen will save the circle centers to the `CameraCalibrator` class instance, which after sufficient corner captures can be used to calculate the coefficients. 20 or more pattern detections from different angles is sufficient, each capture supplies one equation. After calibration, the user can verify the result by choosing a side-by-side view of original and undistorted images.

The Camera calibration can be improved by removing outliers from the source images used in the calibration. As we have implemented a phone-based calibration, this option is not available. An alternative could be using desktop environment a implementation in Matlab or OpenCV, which can provide greater flexibility.

[1] https://www.jetbrains.com/

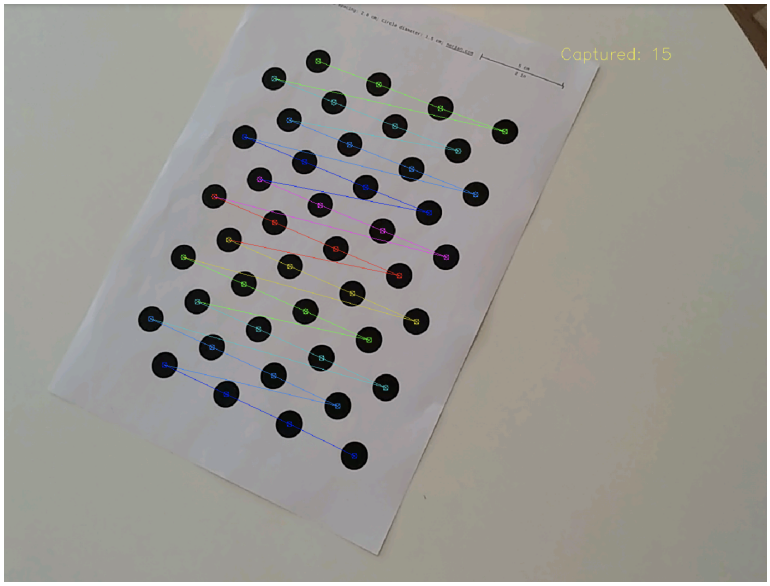


Figure 6.1: Tracking the calibration board using the geopose calibration activity

Sensor	Name	Sampling rate	Manufacturer	Version
Accelerometer	LGE Accelerometer	$\approx 50MHz$	BOSCH	2062500
Gyroscope	LGE Gyroscope	$\approx 50MHz$	BOSCH	2062500
Magnometer	LGE Magnometer	$\approx 50MHz$	AKM	1
Rotation Vector	LGE Rotation Vector Sensor	$\approx 50MHz$	Qualcomm	2
Barometer	BMP280 Pressure	-	BOSCH	1040300

Table 6.1: Details of LG G6 sensor capabilities

LG G6	Spec
CPU	Qualcomm MSM8996pro
OS	Android 8.0 Oreo, SDK level 26
Memory	4GB RAM
Display	5,44”, 1440x2703px
Camera	13MP (IMX258)

Table 6.2: Caption

After a successfully calibrating the results read

$$K = \begin{bmatrix} 1755.984413656498 & 0 & 960 \\ 0 & 1755.984413656498 & 540 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion coefficients:

$$\kappa = \begin{bmatrix} 0.3192679200839998 \\ -1.219662462985774 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

A simplified system is proposed as a proof-of-concept, in which a marker-based approach is used in place of a off-line, pre-captured 3D point map and relocalization.
We start by downloading a free 3D COLLADA model to represent the a mobile phone. In order to use it in Cesium, it is converted to glTf with an online tool[2]. Placed on the server, it is immediately piped through a HTTP stream to the client when requested.

```
router.get('/3dmodel/phone', (req, res) => {
    const model = fs.createReadStream('src/data/phone.glb');
    model.pipe(res);
});router.get('/3dmodel/cesiumMan', (req, res) => {
    const model = fs.createReadStream('src/data/cesiumMan.glb');
    model.pipe(res);
});
```

Listing 6.1: HTTP End point on our local server

The .glb format is binary glTf, optimized for streaming over networks. Read- and writestream lets us process the data as it arrives. In this case, its not necessary to process the phone model. Piping read and write streams is a typical way of handling data with Node.js. On the visualization client this data is read as a `Promise` which we wait for to resolve by doing a `.then(data => doSomething...)` once its ready.

6.2 Approach 1 - IMU-based geopose

The first step here is aligning the two frames. As the model itself and the emitted geopose quaternion are in different frames, they will appear wrong in the visualization. Because of the non-commutative nature of quaternions, multiplication order matters.

- Establish local ENU frame - `Transforms.eastNorthUpToFixedFrame()` supplied with a point returns a 4x4 transformation matrix which transforms from ENU to ECEF.

To make quaternion multiplication a little easier we implemented a helper function to find a quaternion rotation between two vectors. The formula is based on linear aglebra, and results in a normalized rotational quaternion. The dot product between the vectors yield an axis of rotation, and the magnitude the gives amount of radians to rotate.

[2]https://blackthread.io/gltf-converter/

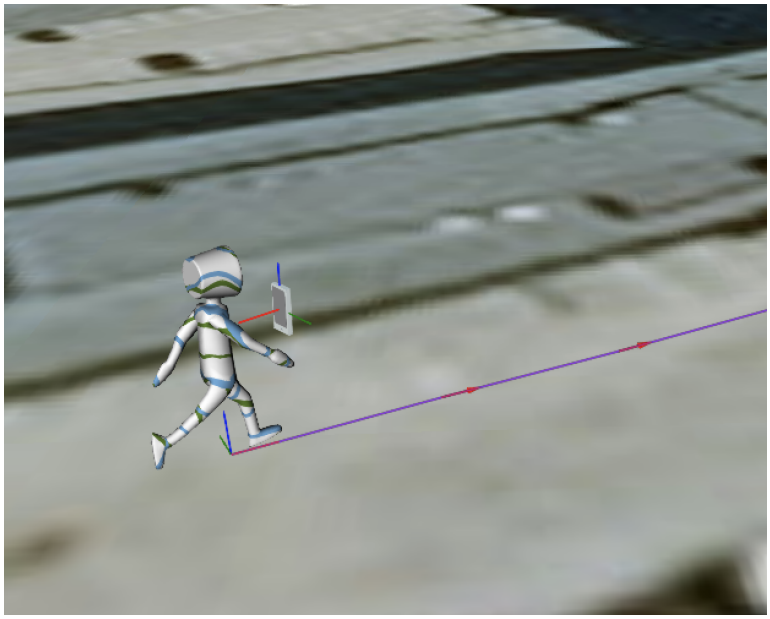


Figure 6.2: 3D models and their reference frames.

```
export const quaternionFromTo = (u, v) => {
    const kCosTheta = Cartesian3.dot(u, v);
    const k = Math.sqrt((Cartesian3.magnitude(u) ** 2) * (Cartesian3.
        ↪ magnitude(v) ** 2));
    let rotAxis = new Cartesian3();

    if (kCosTheta / k === -1) {
        rotAxis = Math.abs(u.x) > Math.abs(u.y)
            ? new Cartesian3(-u.y, u.x, 0) : new Cartesian3(0, -u.z, u.y);
    } else {
        Cartesian3.cross(u, v, rotAxis);
    }
    const result = new Quaternion(rotAxis.x, rotAxis.y, rotAxis.z,
        ↪ kCosTheta + k);
    Quaternion.normalize(result, result);
    return result;
};
```

Furthermore, we have to align the phone model to the correct intrinsic frame as reported by Android studio, that is relative to the natural orientation of the screen when its held upright with default screen orientation. The X axis points to the right horizontally, the Y axis up, and the z axis completes the system by the right-hand rule. The code snippet below shows the process of aligning the phone correctly to the local ENU frame. `q_enu_ecef` is the transform from ENU to ECEF at the point we are. `Quaternion.multiply(q_left, q_right, destination)` is the signature of the Quaternion multiplication function from the Cesium API.

```
    // Phone realigned to Android sensors API frame
```

```
    const rot1 = quaternionFromTo(Cartesian3.UNIT_X, Cartesian3.UNIT_Z);
    const rot2 = quaternionFromTo(Cartesian3.UNIT_Z, Cartesian3.UNIT_Y);
    Quaternion.multiply(initPhoneOrientation, rot1, initPhoneOrientation);
    Quaternion.multiply(initPhoneOrientation, rot2, initPhoneOrientation);
    Quaternion.multiply(q_enu_ecef, initPhoneOrientation, phoneOrientation
        ↪ );
```

An IMU-only based pose estimation yields geopose by combining the fused rotation vector provided by the Android sensors API. The application receives sensor readings and combines them with the location readings to construct a JSON object. Every sensor update the geopose is emitted to the server, which further emits it to the visualization client.

6.3 Approach 2 - Marker based geopose

This test will have a more CV approach, by applying the camera calibration to extract relative pose from fiducial markers. the idea is that relocalization is done through markers instead of keyframes. Immediately after a marker enters the view of the camera, the app emits an event. The server then matches the marker ID to (lat, lng, h) coordinate triplet. The snippet below shows how the server handles events. `socket.on()` implies incoming event and `io.emit()` broadcasts a event.

```
    socket.on('tracking marker', (data) => {
        io.emit('update pose', data);
        // Pass rotation and translation on to viewer
    });
    socket.on('tracking changed', data => {
        const marker = JSON.parse(data);
        console.log('marker id: ' + marker.id + ', status: ' + marker.
            ↪ status);
        if (marker.status === "gained tracking") {
            getHeightFromLatLng(point.lat, point.lng)
                .then(data => {
                    io.emit('marker geolocation', {
                        id: marker.id,
                        lat: data[0],
                        lng: data[1],
                        h: data[2]
                    })
                })
                .catch(err => console.log(err))
        };

        if (marker.status === "lost tracking") {
            io.emit('lost marker', marker.id)
        }
    })
```

Listing 6.2: Server socket endpoints

We will use the `Aruco` markers [30] that were specially developed for AR applications to achieve this. A minimum dictionary of 3X3 bits and 50 total markers is chosen for the largest inter-marker distance. This approach searches every incoming frame for markers, by detecting the corners. Images are analyzed by adaptive thresholding to segment

each marker, and each potential marker contour is extracted from the image. Candidates with contours that are very unlikely to be markers are discarded, after which the tracker analyzes the inner codification. The perspective transformation from the camera calibration now enables us to extract black and white bits after perspective distortion is gone. Now the bits are counted and matched to the given dictionary. all of this is done in the `detectMarkers()` function of the Aruco implementation. The below code snippet shows the meat in the Aruco tracker.

```
1        @Override
2    public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
         ↪ inputFrame) {
3        Mat img = inputFrame.rgba();
4        Imgproc.cvtColor(img, rgb, Imgproc.COLOR_RGBA2RGB);
5        img.copyTo(imgCopy);
6        ArrayList<Mat> corners = new ArrayList<>();
7        Mat ids = new Mat();
8        Scalar borderColor = new Scalar(0, 250, 0, 0);
9
10       Aruco.detectMarkers(rgb, dictionary, corners, ids);
11       if(corners.size() > 0) {
12           Aruco.drawDetectedMarkers(rgb, corners, ids, borderColor);
13           Aruco.estimatePoseSingleMarkers(corners, 0.05f, cameraMatrix,
                 ↪ distortionMatrix, rvecs, tvecs);
14           emitPose(ids);
15           drawMarkerAxis(ids, rgb, rvecs, tvecs, 0.05f);
16           updateTrackedMarkers(ids);
17       } else {
18           for (Marker m : trackedMarkers.values()){
19               String lostMarkerJson = Util.serializeMarker(m.getId(),
                     ↪ tvecs, rvecs, STATUS_LOST);
20               socket.emit("tracking changed", lostMarkerJson);
21               trackedMarkers.remove(m.getId());
22           }
23       }
24       return rgb;
25   }
```

Listing 6.3: Android client image pipeline

The application keeps track of which markers are in view at all times, and emitting those relative poses through the socket. The transmitted `tvecs` and `rvecs` contains vectors of the relative translation and rotation of each marker. The rotation is represented by a quaternion.

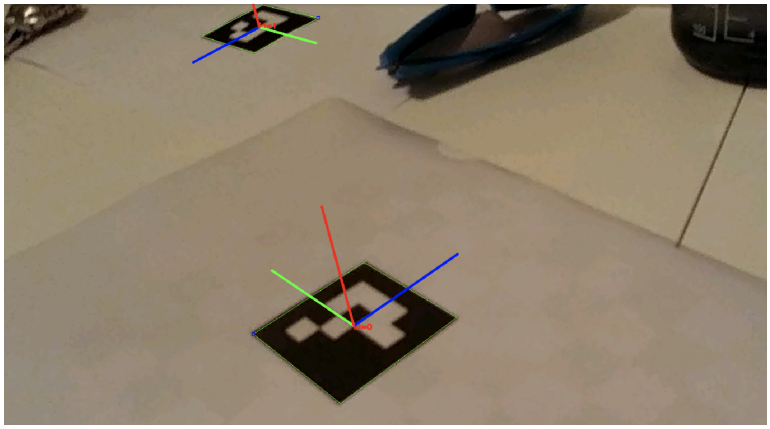


Figure 6.3: Detected Aruco Markers

Chapter 7

Discussion and conclusion

Although AR Cloud solutions are receiving attention by established tech firms, start-ups and the open-source community alike, it is far from being realized as one common ecosystem in which different solutions interact seamlessly.

ARKit seems to deliver the most complete AR platform as per early 2019, with ARCore right behind. Start-ups will either succeed on their own or be acquired for their tech, as Bluvision labs, Escher Reality or Metaio. In any case, AR Mobile tech moves fast and nobody can really say what will happen in the future. Few would disagree that mobile AR and the AR Cloud are at a similar stage as the internet was 30 years ago.

Pose estimation on a multi-sensor device such as a mobile phone is a advanced topic that spans many areas of research. We have reviewed some of the core methods used in state-of-the-art AR engines today, with the intentions of establishing a open-sourced alternative. More specifically, a system for real-time visualization of global poses provided by mobile devices implemented for further research and development.
Relocalization within a prior point cloud is the one of the biggest issues concerning multi-user and persistent Augmented Reality, mostly because of ambiguity and computational cost.
The problem of scale ambiguity remains an issue for re-localizing within the AR cloud. In the monocular case, there exists two ways to determine scale from images: by using the dimension of some known objects or by moving the camera. Moving the camera includes off-line keyframe matching, loop-closing, moving at a fixed height, or fusing data with external sensors found in the IMU or GPS. Alternativly, external sensors such as a depth camera or laser ranging can be used, but comes with a power, financial, storage and range cost.

Much of the reviewed research stems from Robots and computer vision, where the core problems of AR also are being handled. Classical Geographical Information Technology, Computer Vision and Deep learning are now converging at the intersection some might

call *Spatial Computing*. Spatial computing, combined with crowd-sourced data capturing, might in the future produce a global 3D semantic map for devices to perceive, taking us one step further into *Hyper-reality*.

Determining a state-of-the-art VIO/SLAM system for a open-source crowdsourced mapping of the world is not easy. The research is only a demonstration of what is possible, often times done in extremely well-defined or constrained enviroments. As such it is much left to do after deciding what kind of approach to take when designing such a system. The devil is in the details, especially regarding the fine details of *exactly* how to keep or discard keyframes, system initialization, relocalization and more. ORB-SLAM[59] and LSD-SLAM[21] are probably the most suited foundations to build an more complete system that can reconstruct the environment semi-densely. These two pipelines do not incorporate IMU readings by default, so one must look to variants such as OK-VIS [50] as well. In the future we will se lots of possibilities with the emergence of MEC and 5G networks, as well as improved sensors. Standards from Khronos that are especially suited for Mobile AR is OpenKCam, OpenStream and OpenVX. Adoption of these standards could enable developers to gain even more juice out of the hardware. In any case, monocular tracking and mapping suffers from having only one camera, and will always be one step behind stereo systems or systems with depth sensors.

We have also explored two simple ways of visualizing a real-world pose that we call the geopose. Ideally, it should have be tested with several connected mobile clients at once, all connecting to the same server. The Cesium viewer has a wide range of options for visualizing error metrics or other data, that could be useful in future improvements to the visualization client.

Bibliography

[1] A. Newcombe, R., J. Lovegrove, S., J. Davison, A., 11 2011. DTAM: Dense tracking and mapping in real-time.

[2] Acquisti, A., Gross, R., Stutzman, F., 12 2014. Face Recognition and Privacy in the Age of Augmented Reality. Vol. 6.

[3] Arun Gupta, 2014. REST vs WebSocket comparison benchmark.
URL http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks

[4] Azuma, R. T., 8 1997. A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments 6 (4), 355–385.
URL https://doi.org/10.1162/pres.1997.6.4.355

[5] Barfield, W., Caudell, T., 1 2001. Fundamentals of Wearable Computers and Augmented Reality.

[6] Bay, H., Tuytelaars, T., Van Gool, L., 2006. SURF: Speeded Up Robust Features. In: Leonardis, A., Bischof, H., Pinz, A. (Eds.), Computer Vision ECCV 2006. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 404–417.

[7] Behr, J., Eschler, P., Jung, Y., Zöllner, M., 2009. X3DOM: A DOM-based HTML5/X3D Integration Model. Proceedings of the 14th International Conference on 3D Web Technology 1 (212), 127–135.
URL http://portal.acm.org/citation.cfm?doid=1559764.1559784%5Cnhttp://dl.acm.org/citation.cfm?id=1559764.1559784

[8] Behr, J., Jung, Y., Franke, T., Sturm, T., 2012. Using Images and Explicit Binary Container for Efficient and Incremental Delivery of Declarative 3D Scenes on the Web. In: Proceedings of the 17th International Conference on 3D Web Technology. Web3D '12. ACM, New York, NY, USA, pp. 17–25.
URL http://doi.acm.org/10.1145/2338714.2338717

[9] Bell, D. G., Kuehnel, F., Maxwell, C., Kim, R., Kasraie, K., Gaskins, T., Hogan, P., Coughlan, J., 2007. NASA World Wind: Opensource GIS for Mission Operations. In: 2007 IEEE Aerospace Conference. pp. 1–9.

[10] Bergmann, P., Wang, R., Cremers, D., 2018. Online Photometric Calibration of Auto Exposure Video for Realtime Visual Odometry and SLAM. IEEE Robotics and Automation Letters 3, 627–634.

[11] Bermejo, C., Huang, Z., Braud, T., Hui, P., 2017. When Augmented Reality meets Big Data. In: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). pp. 169–174.

[12] Calonder, M., Lepetit, V., Strecha, C., Fua, P., 9 2010. BRIEF: Binary Robust Independent Elementary Features. Vol. 6314.

[13] Carmigniani, J., Furht, B., 2011. Augmented Reality: An Overview. In: Furht, B. (Ed.), Handbook of Augmented Reality. Springer New York, New York, NY, pp. 3–46.
URL https://doi.org/10.1007/978-1-4614-0064-6_1

[14] Caudell, T., Mizell, D., 2 1992. Augmented reality: An application of heads-up display technology to manual manufacturing processes. Vol. 2.

[15] Chatzopoulos, D., Bermejo, C., Huang, Z., Hui, P., 2017. Mobile Augmented Reality Survey: From Where We Are to Where We Go. IEEE Access 5, 6917–6950.

[16] Cheng, Y., Maimone, M., Matthies, L., 2005. Visual odometry on the Mars Exploration Rovers. In: 2005 IEEE International Conference on Systems, Man and Cybernetics. Vol. 1. pp. 903–910.

[17] Choi, S., Kim, T., Yu, W., 2009. Performance Evaluation of RANSAC Family. In: BMVC.

[18] Costanza, E., Inverso, S., Pavlov, E., Allen, R., Maes, P., 1 2006. eye-q: eyeglass peripheral display for subtle intimate notifications.

[19] Delmerico, J., Scaramuzza, D., 5 2018. A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots.

[20] Engel, J., Koltun, V., Cremers, D., 2018. Direct Sparse Odometry. IEEE Transactions on Pattern Analysis and Machine Intelligence 40, 611–625.

[21] Engel, J., Schöps, T., Cremers, D., 2014. LSD-SLAM: Large-Scale Direct Monocular SLAM. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.), Computer Vision ECCV 2014. Springer International Publishing, Cham, pp. 834–849.

[22] Farnebäck, G., 2003. Two-Frame Motion Estimation Based on Polynomial Expansion. In: Bigun, J., Gustavsson, T. (Eds.), Image Analysis. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 363–370.

[23] Fernández Alcantarilla, P., 9 2013. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces.

[24] Fette, I., Melnikove, A., 2011. The WebSocket Protocol.
URL https://tools.ietf.org/html/rfc6455

[25] Fiala, M., 1 2004. ARTag, An Improved Marker System Based on ARToolkit.

[26] Fischler, M. A., Bolles, R. C., 6 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Commun. ACM 24 (6), 381–395.
URL http://doi.acm.org/10.1145/358669.358692

[27] Forster, C., Pizzoli, M., Scaramuzza, D., 2014. SVO: Fast semi-direct monocular visual odometry. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). pp. 15–22.

[28] Forster, C., Zhang, Z., Gassner, M., Werlberger, M., Scaramuzza, D., 2017. SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems. IEEE Transactions on Robotics 33 (2), 249–265.

[29] Galvez-López, D., Tardos, J. D., 2012. Bags of Binary Words for Fast Place Recognition in Image Sequences. IEEE Transactions on Robotics 28 (5), 1188–1197.

[30] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., Marín-Jiménez, M. J., 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition 47 (6), 2280–2292.
URL http://www.sciencedirect.com/science/article/pii/S0031320314000235

[31] Godard, C., Aodha, O., J. Brostow, G., 9 2016. Unsupervised Monocular Depth Estimation with Left-Right Consistency.

[32] Gośliński, J., Nowicki, M., Skrzypczyński, P., 2015. Performance Comparison of EKF-Based Algorithms for Orientation Estimation on Android Platform. IEEE Sensors Journal 15 (7), 3781–3792.

[33] Hare, S., Saffari, A., Torr, P. H. S., 2012. Efficient online structured output learning for keypoint-based object tracking. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition. pp. 1894–1901.

[34] Harris, C., Stephens, M., 1 1988. A combined corner and edge detector.

[35] Hartley, R., Zisserman, A., 2000. Multiple View Geometry in Computer Vision, 2nd Edition. Cambridge University Press.

[36] Hobona, G., James, P., Fairbairn, D., 2006. Web-based visualization of 3D geospatial data using Java3D. IEEE Computer Graphics and Applications 26 (4), 28–33.

[37] Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., Young, V., 2015. Mobile Edge Computing A key technology towards 5G.

[38] Inbar, O., 2017. AR-Cloud.
URL https://medium.com/super-ventures-blog/arkit-and-arcore-will-not-usher-massive-adoption-of-mobile-ar-da3d8

[39] Jankowski, J., Ressler, S., Sons, K., Jung, Y., Behr, J., Slusallek, P., 2013. Declarative Integration of Interactive 3D Graphics into the World-wide Web: Principles, Current Approaches, and Research Agenda. In: Proceedings of the 18th International Conference on 3D Web Technology. Web3D '13. ACM, New York, NY, USA, pp. 39–45.
URL http://doi.acm.org/10.1145/2466533.2466547

[40] Kan, T.-W., Teng, C.-H., Chen, M. Y., 2011. QR Code Based Augmented Reality Applications. In: Furht, B. (Ed.), Handbook of Augmented Reality. Springer New York, New York, NY, pp. 339–354.
URL https://doi.org/10.1007/978-1-4614-0064-6_16

[41] Kato, H., Billinghurst, M., 1999. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In: Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99). pp. 85–94.

[42] Kendall, A., Grimes, M. K., Cipolla, R., 2015. PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization. 2015 IEEE International Conference on Computer Vision (ICCV), 2938–2946.

[43] Kendall, A., Martirosyan, H., Dasgupta, S., Henry, P., 10 2017. End-to-End Learning of Geometry and Context for Deep Stereo Regression.

[44] Keysers, J. H., 2015. Digital Globe Review 2015. Tech. rep.
URL http://www.crcsi.com.au/assets/Uploads/Globe-review-paper-March-2015.pdf

[45] Klein, G., Murray, D., 12 2007. Parallel Tracking and Mapping for Small AR Workspaces.

[46] Klein, G., Murray, D. W., 2006. Full-3D Edge Tracking with a Particle Filter. In: BMVC. pp. 1119–1128.

[47] Klein, G., W. Murray, D., 10 2009. Parallel Tracking and Mapping on a Camera Phone.

[48] Lawitzki, P., 2012. Application of Dynamic Binaural Signals in Acoustic Games. Ph.D. thesis, Stuttgart Media University, Stuttgart.

[49] Leutenegger, S., Chli, M., Siegwart, R. Y., 2011. BRISK: Binary Robust invariant scalable keypoints. In: 2011 International Conference on Computer Vision. pp. 2548–2555.

[50] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., Furgale, P., 12 2014. Keyframe-based visualinertial odometry using nonlinear optimization. The International Journal of Robotics Research 34 (3), 314–334.
URL https://doi.org/10.1177/0278364914554813

[51] Li, P., Qin, T., Hu, B., Zhu, F., Shen, S., 2017. Monocular Visual-Inertial State Estimation for Mobile Augmented Reality. In: 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR). pp. 11–21.

[52] Lowe, D. G., 2004. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60 (2), 91–110.
URL https://doi.org/10.1023/B:VISI.0000029664.99615.94

[53] Lu, F., Milios, E., 1997. Globally Consistent Range Scan Alignment for Environment Mapping. Autonomous Robots 4 (4), 333–349.
URL https://doi.org/10.1023/A:1008854305733

[54] Madgwick, S. O. H., Harrison, A. J. L., Vaidyanathan, R., 2011. Estimation of IMU and MARG orientation using a gradient descent algorithm. In: 2011 IEEE International Conference on Rehabilitation Robotics. pp. 1–7.

[55] Marchand, E., Uchiyama, H., Spindler, F., 2016. Pose Estimation for Augmented Reality: A Hands-On Survey. IEEE Transactions on Visualization and Computer Graphics 22 (12), 2633–2651.

[56] Milgram, P., Kishino, F., 1994. A taxonomy of mixed reality visual displays. IEICE Transactions on Information Systems Vol E77-D (No. 12), 13211329.

[57] Moravec, H. P., 1980. Obstacle Avoidance and Navigation in the Real World by a }Seeing Robot Rover.

[58] Mulloni, A., Ramachandran, M., Reitmayr, G., Wagner, D., Grasset, R., Diaz, S., 2013. User friendly SLAM initialization. In: 2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR). pp. 153–162.

[59] Mur-Artal, R., Montiel, J. M. M., Tardós, J. D., 2015. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics 31 (5), 1147–1163.

[60] Mur-Artal, R., Tardós, J. D., 2017. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. IEEE Transactions on Robotics 33 (5), 1255–1262.

[61] Narumi, T., Nishizaka, S., Kajinami, T., Tanikawa, T., Hirose, M., 2011. Meta Cookie+: An Illusion-Based Gustatory Display. In: Shumaker, R. (Ed.), Virtual and Mixed Reality - New Trends. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 260–269.

[62] Nister, D., 2004. An efficient solution to the five-point relative pose problem. IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (6), 756–770.

[63] Nister, D., Naroditsky, O., Bergen, J., 2004. Visual odometry. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004. Vol. 1. pp. I–I.

[64] Ozuysal, M., Fua, P., Lepetit, V., 2007. Fast Keypoint Recognition in Ten Lines of Code. In: 2007 IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–8.

[65] Payet, N., Todorovic, S., 2011. From contours to 3D object detection and pose estimation. In: 2011 International Conference on Computer Vision. pp. 983–990.

[66] Perey, C., 2017. Program on Technology Innovation: Enterprise Augmented Reality Vision, Interoperability Requirements, and Standards Landscape. Tech. rep., EPRI, Palo Alto, CA.

[67] Piao, J.-C., Kim, S.-D., 11 2017. Adaptive Monocular VisualInertial SLAM for Real-Time Augmented Reality Applications in Mobile Devices. Vol. 17.

[68] Pirotti, F., Brovelli, M. A., Prestifilippo, G., Zamboni, G., Kilsedar, C. E., Piragnolo, M., Hogan, P., 2017. An open source virtual globe rendering engine for 3D applications: NASA World Wind. Open Geospatial Data, Software and Standards 2 (1), 4. URL https://doi.org/10.1186/s40965-017-0016-5

[69] Qin, T., Li, P., Shen, S., 2018. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. IEEE Transactions on Robotics 34 (4), 1004–1020.

[70] Rognerud, O. M., 2018. Real time visualization of semantic geospatialobeservations and semantic models for theAugmented Reality Cloud. Ph.D. thesis, NTNU.

[71] Rosten, E., Drummond, T., 2006. Machine Learning for High-Speed Corner Detection. In: Leonardis, A., Bischof, H., Pinz, A. (Eds.), Computer Vision ECCV 2006. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 430–443.

[72] Rublee, E., Rabaud, V., Konolige, K., Bradski, G., 2011. ORB: An efficient alternative to SIFT or SURF. In: 2011 International Conference on Computer Vision. pp. 2564–2571.

[73] Salahat, E., Qasaimeh, M., 2017. Recent advances in features extraction and description algorithms: A comprehensive survey. 2017 IEEE International Conference on Industrial Technology (ICIT), 1059–1063.

[74] Sambinelli, F., Sosa Arias, C., 1 2015. Augmented Reality Browsers: A Proposal for Architectural Standardization. Vol. 6.

[75] Schall, G., Wagner, D., Reitmayr, G., Taichmann, E., Wieser, M., Schmalstieg, D., Hofmann-Wellenhof, B., 2009. Global pose estimation using multi-sensor fusion for outdoor Augmented Reality. In: 2009 8th IEEE International Symposium on Mixed and Augmented Reality. pp. 153–162.

[76] Schnädelbach, H., Koleva, B., Flintham, M., Fraser, M., Izadi, S., Chandler, P., Foster, M., Benford, S., Greenhalgh, C., Rodden, T., 2002. The augurscope: a mixed reality interface for outdoors. In: CHI.

[77] Schneider, M., Rambach, J., Stricker, D., 2017. Augmented reality based on edge computing using the example of remote live support. In: 2017 IEEE International Conference on Industrial Technology (ICIT). pp. 1277–1282.

[78] Shi, J., Tomasi, 1994. Good features to track. In: 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 593–600.

[79] Shu, J., Zheng, R., Hui, P., 10 2016. Cardea: Context-Aware Visual Privacy Protection from Pervasive Cameras.

[80] Słodziak, W., Nowak, Z., 2016. Performance Analysis of Web Systems Based on XMLHttpRequest, Server-Sent Events and WebSocket. In: Grzech, A., Borzemski, L., Świątek, J., Wilimowska, Z. (Eds.), Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology ISAT 2015 Part II. Springer International Publishing, Cham, pp. 71–83.

[81] Valentin, J., Dryanovski, I., Afonso, J., Pascoal, J., Tsotsos, K., Leung, M., Schmidt, M., Guleryuz, O., Khamis, S., Tankovitch, V., Fanello, S., Kowdle, A., Izadi, S., Rhemann, C., T. Barron, J., Wadhwa, N., Dzitsiuk, M., Schoenberg, M., Verma, V., Turner, E., 12 2018. Depth from motion for smartphone AR.

[82] Ventura, J., Arth, C., Reitmayr, G., Schmalstieg, D., 4 2014. Global Localization from Monocular SLAM on a Mobile Phone. IEEE Transactions on Visualization and Computer Graphics 20 (4), 531–539.
URL https://doi.org/10.1109/TVCG.2014.27

[83] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., Schmalstieg, D., 2008. Pose tracking from natural features on mobile phones. In: 2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality. pp. 125–134.

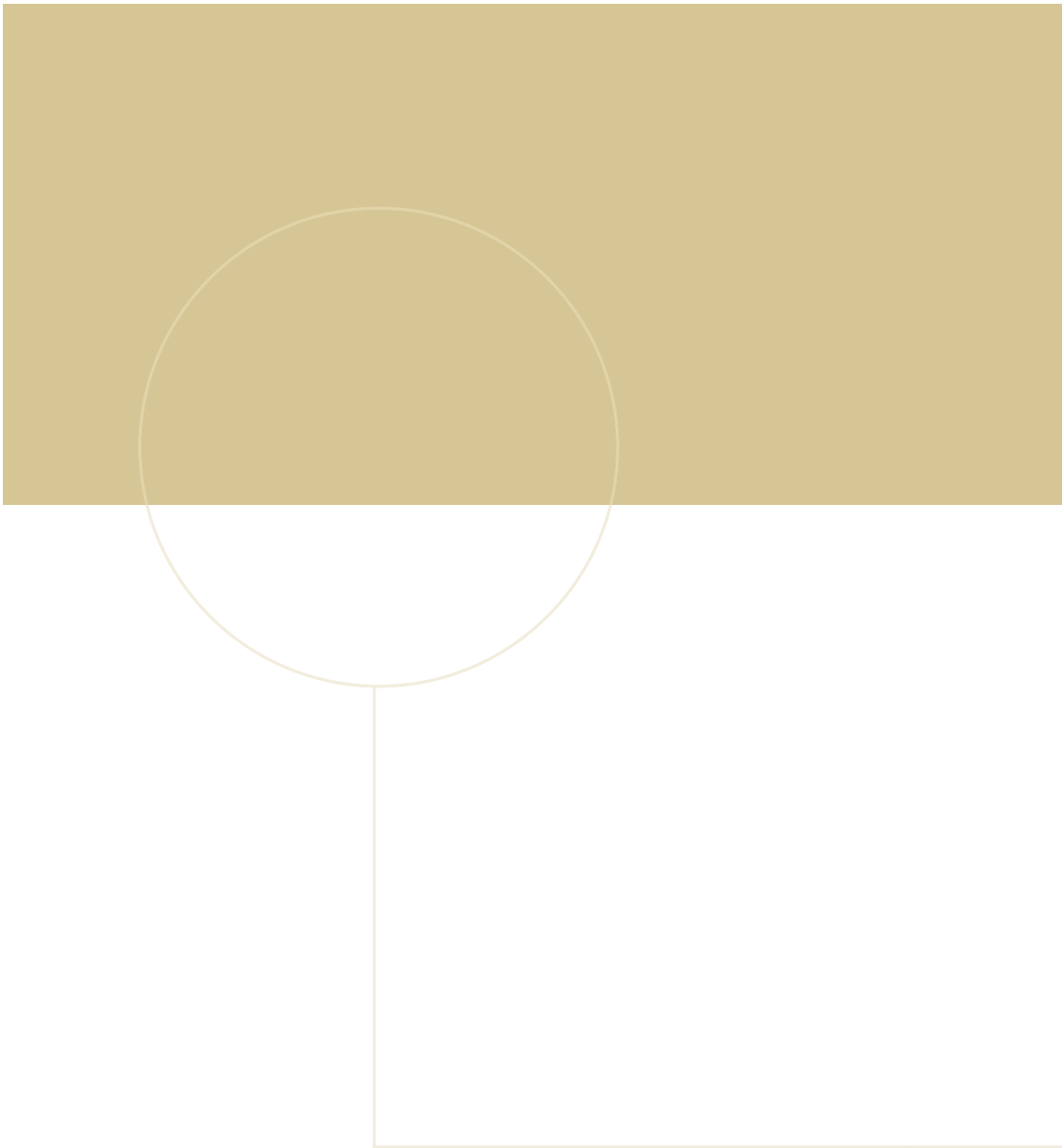
[84] Wang, G., Xiong, Y., Yun, J., Cavallaro, J., 3 2014. Computer Vision Accelerators for Mobile Systems based on OpenCL GPGPU Co-Processing. Vol. 76.

[85] WHATWG, 2018. Server Sent Events specification.
URL https://html.spec.whatwg.org/multipage/server-sent-events.html

[86] Yang, N., Wang, R., Gao, X., Cremers, D., 6 2018. Challenges in Monocular Visual Odometry: Photometric Calibration, Motion Bias and Rolling Shutter Effect. Vol. 3.

[87] Yi, S., Li, C., Li, Q., 6 2015. A Survey of Fog Computing: Concepts, Applications, and Issues.

[88] Zhang, Z., 2000. A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (11), 1330–1334.



NTNU

Kunnskap for en bedre verden