ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Bachelor Thesis

Embedded Photometric Visual Odometry

Spring Term 2015

Supervised by:
Jörn Rehder
Pascal Gohl

Author:
Samuel Bryner

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Embedded Photometric Visual Odometry

is original work which I alone have authored and which is written in my own words.[1]

Author

Samuel	Bryner

Student supervisors

Jörn	Rehder
Pascal	Gohl

Supervising lecturer

Roland	Siegwart

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Preface

Bla bla ...

Was soll denn hier noch reinkommen?

Abstract

Kurzer Überblick/Zusammenfassung über alles was hier gemacht wird.

main points: - dense odometry - embedded

This work investigates how fancy visual algos. can leverage the power of an FPGA to run that thing on an computationally constrained platform. Here, a relatively recent way of doing visual odometry is implemented on an ARM core: Using disparity data caculated by an FPGA from a stereo camera, a frame is warped to the previous one by minimizing photometric error. This results in high precision, as the full information of a frame is taken into account. Though not the most efficient approach, it's highly parallelizable and makes good use of the FPGA ['s stereo data].

Symbols

Symbols

$\mathbf{I}(\boldsymbol{x})$	intensity image
$\mathbf{D}(\boldsymbol{x})$	disparity image
$\boldsymbol{T}$	6-DOF transformation
ϕ, θ, ψ	roll, pitch and yaw angle

stereo camera intrinsics:

f	focal length
$\boldsymbol{c}$	principal point
b	baseline

warping:

π^{-1}	back-projection operator, mapping a pixel with corresponding disparity value into 3D space
π	projection operator, mapping a point in 3D space onto the camera plane
$\boldsymbol{J_T}$	3×6 Jacobian of the transformation operator $\boldsymbol{T}$
$\boldsymbol{J_\pi}$	2×3 Jacobian of the projection function π
$\boldsymbol{J_I}$	1×2 Jacobian of the intensity image sampling

Indices

TODO

c	current frame
p	previous frame
x, y, z	world coordinates ($\in \mathbb{R}^3$, meters)
u, v	coordinates in camera plane ($\in \mathbb{R}^2$, pixels)

Acronyms and Abbreviations

TODO

ETH	Eidgenössische Technische Hochschule
ASL	Autonomous Systems Lab at ETH
IMU	Inertial Measurement Unit

SGM	Semi-Global Matching
SLAM	Simultaneous Location And Mapping

Chapter 1

Introduction

1.1 Motivation

Robots are generally constrained in their computational power and energy usage. A powerful method is to offload computation onto an FPGA, which is usually much more efficient. However, programming an FPGA isn't straightforward and integtrating it with code running on a CPU can be tricky.

In this work, a novel example of such an integration is provided by running a semi-global stereo matching [1] core developed in [2] on the FPGA and using its output for a photometric visual odometry algorithm running on a CPU.

This approach of photometric odometry does not track a sparse set of features, as is usuallty done in visual odometry, but instead warps the full image to find a perspective where the warped image matches the previous frame.

This approach is well suited for offloading to an FPGA, as most parts are highly parallelizable. Note tough, that this is not the most efficient way to do embedded odometry, as a lot more data has to be processed. The main goal was to explore how an FPGA and a general purpose processor can be integrated on an embedded device and to ascertain the potential for further optimizations by offloading more parts to the FPGA.

A visual-inertial sensor developed by the ASL [3] is used which features a Xilinx Zynq 7020 SoC consisting of a dual-core ARM Cortex A9 and an ARTIX-7 FPGA. The sensor has a wide-angle stereo camera with a resolution of 752×480 pixels and syncronized global shutters as well as a high-precision inertial measurement unit, which was not used tough. This vi-sensor is not only small and lightweight (133×57mm, 130 g), it is also power-efficient, consuming less than 10 W.

1.2 Related Work

Photometric odometry as initially developed by Comport et al. in [4] has been implemented in [5] to use data from the SGM core but running on a powerful PC instead.

In [6], feature-based odometry running on the visensor is developed, which uses the FPGA for corner detection.

TODO: googlen was es sonst noch so gibt?

TODO: expand that section with a more general overview?

Chapter 2

Method

2.1 Overview

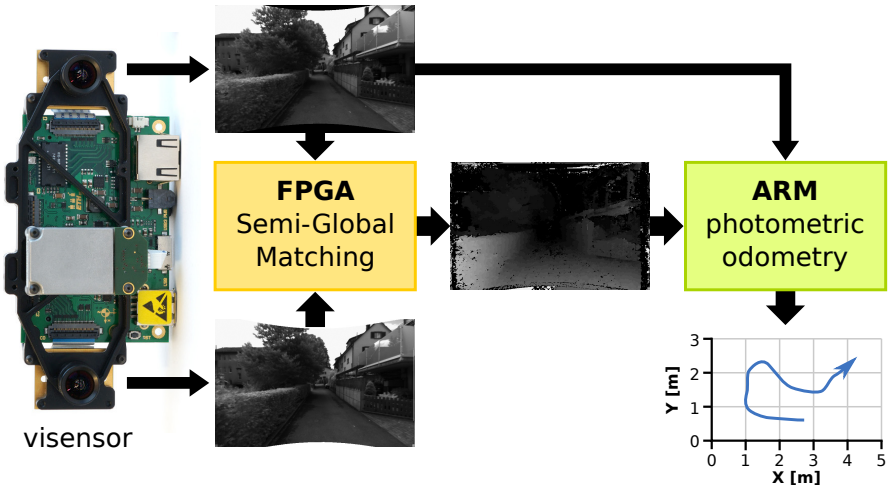

Figure 2.1: schematic overview of the whole system

The visensor provides a stream of frames, each of which consists of a stereo pair of intensity data [1].

A semiglobal stereo matching core developed by [todo: ref] running on the FPGA processes these and produces a disparity image which assigns every pixel the disparity between the two cameras. The FPGA also provides a rectified camera image.

This pair of intensity and disparity images is conceptually equivalent to a three dimensional point-cloud, as we can calculate the distance to the camera for every pixel from the disparity data. This is in turn makes it possible to render the point-cloud from an arbitrary perspective, allowing us to look at an image as if it was recorded from a different angle.

To estimate the ego-motion between two frames we can thus look for a perspective

[1] Grayscale instead of full-color images are used, because the information gain from colors is offset by the loss of resolution. However, the approach described here would work similarly colors.

that looks the same as the previous frame. The movement of the virtual camera will then correspond to the actual, physical movement of the sensor.

By subtracting the intensities from the previous frame with the intensities of the current frame sampled at the warped pixel locations a photometric error is calculated, measuring the similarity of the warped current frame with the previous one. The problem is now to minimize this error function.

Note that this approach assumes photoconsistency: Points have the same intensity, regardless of viewing angle.

2.2 Warping Pipeline

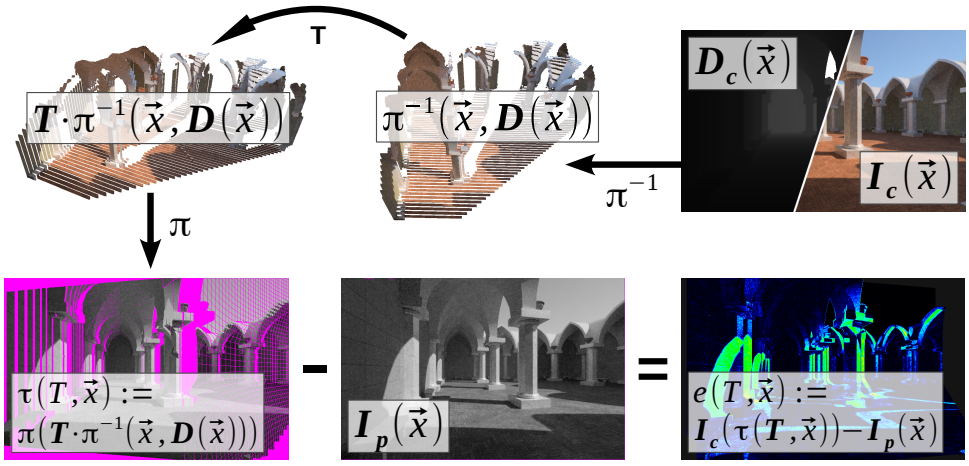

Figure 2.2: the full warping pipeline (pink pixels are not sampled by any of the warped points)

This section closely follows [5].

Using an inverse projection derived from the standard pinhole camera model, a point $\boldsymbol{x}$ in the camera image plane can be back-projected into a point $\boldsymbol{p}$ in $\mathbb{R}^3$:

$$\boldsymbol{p} = \pi^{-1}(\boldsymbol{x}, D(\boldsymbol{x})) := \frac{b}{D(x)} \begin{bmatrix} \boldsymbol{x_u} - \boldsymbol{c_u} \\ \boldsymbol{x_v} - \boldsymbol{c_v} \\ f \end{bmatrix} \tag{2.1}$$

where b is the stereo baseline, f the focal length and $\boldsymbol{c}$ the principal point of the camera.

This point $\boldsymbol{p}$ can now be moved into a new camera frame by translating and rotating it:

$$\boldsymbol{p'} = \boldsymbol{T_R} \cdot \boldsymbol{p} + \boldsymbol{T_T} \tag{2.2}$$

Where T_R is a 3x3 rotation matrix and T_T a three dimensional translation vector. Using affine coordinates, we can write this as:

$$\boldsymbol{p'} = \boldsymbol{T}\boldsymbol{p} \tag{2.3}$$

A point in 3D space can be projected back onto the (now moved) camera image plane:

$$\boldsymbol{x}' = \pi(\boldsymbol{p}') := \frac{f}{\boldsymbol{p}'_z} \begin{bmatrix} \boldsymbol{p}'_x \\ \boldsymbol{p}'_y \end{bmatrix} + \boldsymbol{c} \tag{2.4}$$

This whole warping operator can be summarized in a warping operator τ:

$$\boldsymbol{x}' = \tau(\boldsymbol{x}, D(\boldsymbol{x}), \boldsymbol{T}) := \pi(\boldsymbol{T} \cdot \pi^{-1}(\boldsymbol{x}, D(\boldsymbol{x}))) \tag{2.5}$$

2.3 Minimization

Using τ, the error between the previous frame and the warped current frame can be defined as:

$$e(\boldsymbol{x}, \boldsymbol{T}) := I_p(\tau(\boldsymbol{x}, \boldsymbol{T})) - I_c(\boldsymbol{x}) \tag{2.6}$$

To estimate the motion between two frames, this photometric error term should be minimal for every pixel:

$$\hat{\boldsymbol{T}} = \underset{\boldsymbol{T}}{\operatorname{argmin}} \sum_{\boldsymbol{x} \in I_p} e(\boldsymbol{x}, \boldsymbol{T})^2 \tag{2.7}$$

This equation can be solved for the estimated motion by applying standard optimization techniques, for exmple Gauss-Newton:

$$\boldsymbol{J}^T \boldsymbol{J} \Delta \boldsymbol{T} = -\boldsymbol{J}^T \boldsymbol{e}(\boldsymbol{T}) \tag{2.8}$$

Here, $\boldsymbol{J} \in \mathbb{R}^{N \times 6}$ is the stacked Jacobian matrices and $\boldsymbol{e}(\boldsymbol{T}) \in \mathbb{R}^N$ the vector of the error terms of all N pixels. This equation is iteratively solved for the increment $\Delta \boldsymbol{T}$ of the motion estimation after recalculating the error term and Jacobians from the new estimation.

The Jacobian is derived by applying the chain rule to photometric error term. For a single pixel, we get a 1×6 Jacobian:

$$\boldsymbol{J} := \boldsymbol{J_I} \boldsymbol{J_\pi} \boldsymbol{J_T} \tag{2.9}$$

where $\boldsymbol{J_I} \in \mathbb{R}^{1 \times 2}$ is the image derivative of the warped previous frame and is approximated using the image's gradient:

$$\boldsymbol{J_I} := \left. \frac{\partial \mathbf{I_p}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x} = \tau(\boldsymbol{x}, \boldsymbol{T})} \approx \begin{bmatrix} \nabla_x \mathbf{I_p} & \nabla_y \mathbf{I_p} \end{bmatrix} \tag{2.10}$$

The term $\boldsymbol{J_\pi}$ is the 2×3 Jacobian of the projection function 2.4, evaluated at the warped 3D point:

$$\boldsymbol{J_\pi} := \left. \frac{\partial \pi(\boldsymbol{p})}{\partial \boldsymbol{p}} \right|_{\boldsymbol{p} = \boldsymbol{T} \cdot \pi^{-1}(\boldsymbol{x}, \mathbf{D}(\boldsymbol{x}))} = \begin{bmatrix} f/\boldsymbol{p_z} & 0 & -f\boldsymbol{p_x}/\boldsymbol{p_z}^2 \\ 0 & f/\boldsymbol{p_z} & -f\boldsymbol{p_y}/\boldsymbol{p_z}^2 \end{bmatrix} \tag{2.11}$$

$\boldsymbol{J_T}$ is the 3×6 Jacobian of the transformation operator T and the most costly term to compute:

$$\boldsymbol{J_T} := \left.\frac{\partial(\boldsymbol{T}\boldsymbol{p})}{\partial \boldsymbol{T}}\right|_{\boldsymbol{p}=\pi^{-1}(\boldsymbol{x},\mathbf{D}(\boldsymbol{x}))} \tag{2.12}$$

Chapter 3

Optimizations and other improvements

The algorithm described in the previous section works well but suffers from slow performance and does not nearly run in realtime on an embedded device. Therefore, some optimization strategies are applied:

3.1 Image pyramids

A common optimization technique is the use of multiple resolutions: Images are repeatedly downscaled by a factor of two (essentially quartering the number of pixels) by averaging over a 2 x 2 block to generate a stack of increasingly smaller images (a 'pyramid').

The minimization is run on the smallest set of images and the resulting value is used as an initial value for the next bigger set of images.

This greatly reduces the number of iterations required and enhances the convergence radius.

The image pyramid can also be used to trade a bit of accuracy for even more performance gain by simply aborting early and not using the full resolution at all. Throwing out the one or two uppermost levels usually incurs negligible loss of accuracy. See also section 5.1.

3.2 Pixel selection by image gradient

We can further optimize away pixels which do not strongly influence the minimization such as points in homogenous image regions where $\nabla \mathbf{I} \approx 0$ and therefore $\boldsymbol{J_I} \approx 0$.

This is already provided to some extent by the semi-global matching alorithm, as pixels without strong gradients are usally hard to match and therefore often don't provide a disparity value.

3.3 Handling outliers

By robustly weighting the photometric error terms outliers can be dampened to reduce the influence of occlusions, moving scenery or other noise, such as errors from the semi-global matcher. This improves quality and stability with negligible performance penalty.

Another idea (which wasn't fully investigated) to handle occlusions is to use a Z-buffer to eliminate points which are behind others when warped.

3.4 Further possible optmiziations

A few things which have not been investigated in this work but which provide good avenues for further optimizations:

3.4.1 integration of IMU

Modern visual odometry and SLAM systems such as [7] incorporate data from an inertial measurement unit using various complicated filtering schemes to increase accuracy.

In contrast, using integrated acceleration values for pohotometric odometry would be trivial by providing a good initial guess for the minimization process and could greatly speed up perfomance by reducing the number of required optimization iterations.

3.4.2 Keyframes

Instead of matching the previous frame, keyframes can be used which are only updated when the relative motion gets too big. This way, drift can be reduced and even completely eliminated when being more or less stationary.

3.4.3 Offload more work to FPGA

Large parts of the photometric odometry algorithm are highly parallelizable and would profit from an implementation running on the FPGA. This is the main point of this work and is further discussed in section 5.2.

Chapter 4

Implementation details

A few things that are only tangentially related to the core algorithm but which nevertheless might be encountered in an actual implementation:

4.1 Representation of transformation

A six degree of freedom transformation can be represented in multiple ways. While translations are very straightforward, rotations can be represented in numerous ways (Euler-angles, quaternions, rotation matrices, etc.) and proper derivation of the Jacobians can be tricky.

Fortunately, odometry works in a relative fashion without any absolute orientation and steps are very incremental as photometric odometry cannot handle more than a few degrees of rotation. This implies we do not have to deal with gimbal-lock and other mathematical hurdles of working in $SO(3)$.

4.2 Image scaling

Care has to be taken to properly downscale image coordinates when working with image pyramids. This can be done by scaling the camera intrinsics properly: Halving the image width also means halving the focal length and doubling the baseline.

Downscaling usually implies filtering and doing so alters the 3D structure. For this reason, [ref to comport ICP] does not downscale the disparity values and samples them at the full resolution. When matching pose on the camera plane instead of in 3D space, downscaling the disparity values works as well [1].

4.3 Ignore invalid pixels

Pixels that do not have a disparity value (because the SGM algorithm couldn't find any correspondence) can obviously be ignored. So can pixels which are satu-

[1] It might be worth investigating how much of an effect on performance and quality downscaling the disparity images has. Disparity values are only read at integer coordinates and downscaling them might not be worth the runtime penalty.

rated or underexposed: They do not provide valid disparity data and are often not photoconsistent either.

4.4 Don't remove too many pixels

The optimizations described in section 3 can substantially reduce the pixel count, so much so that there are not enough for stable performance. Especially filtering pixels based on their image gradient as explained in section 3.2 requires a well-chosen threshold, as image gradients depend on the scene. [ref to ICP] proposes the calculation of a histogram to select the N best pixels. An easier approach is to simply restart the current iteration with a lower threshold when the number of pixels gets too low and increasing it after a step with enough pixels. The same problem also applies to other optimization parameters such as the size of the image pyramid.

Chapter 5

Results

5.1 Qualitative assessment

To assess the quality of photometric approach to visual odometry, a circual trajectory of an indoor office scene was processed offline with a high quality SGM from OpenCV and run trough both the very accurate ASLAM algorithm [7] and the photometric odometry method described in this work.

After a traveled distance of about 12 m a drift of about 20 cm has been found. On faster trajectories with larger movement between frames, the Gauss-Newton optimization can diverge resulting in a completely wrong trajectory.

An important finding, which was already noted in [TODO: can't find a paper mentioning this], is that processing the camera images in their full resolution is not a necessity and as can be seen in figure 5.1 even using only a twice downscaled image still results in precise tracking while drastically improving performance.

5.2 Timing

Perfomance of the different parts of the algorithm have been timed by counting cpu cycles while running the full photometric odometry on a single ARM core on the vi-sensor while moving around slowly. The time for the SGM is not shown in figure 5.2, as this part is running on the FPGA with the full 30 FPS provided by the cameras.

Using early abort in the image pyramid we can achieve an average run-time performance of about 5 Hz.

This value not only depends on the algorithm's parameters such as max. pyramid levels, but also on the scene (more 'dense' environments which provide more structure converge faster) and on the movement speed. The bigger the steps, the more iterations are required for convergence, which is counterproductive as a longer running time implies an even bigger step when moving. This could potentially be addressed by incorporating integrated acceleration values of an IMU.

Another important point is that at least half the time is spent on calculating the image pyramid, warping pixels and computing the Jacobians. These parts are highly parallelizable because every pixel is completely independent of each other. This offers great potential for offloading further work onto the FPGA, or the second

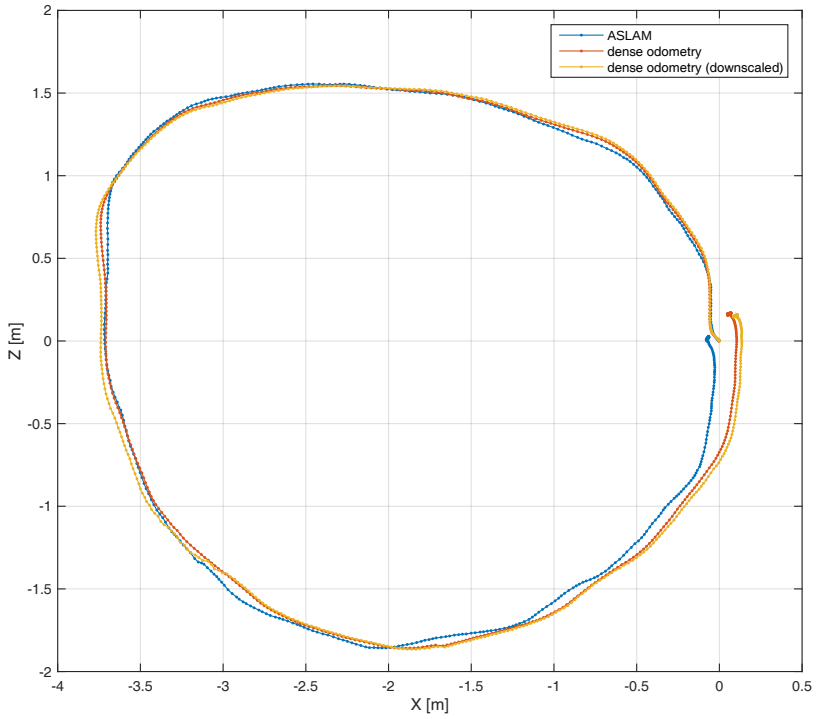

Figure 5.1: qualitative comparison of photometric odometry to ASLAM. The red trajectory was computed using full image resolution, while the orange one only uses $1/16^{th}$ of the available pixels by aborting early in the image pyramid, thus greatly speeding up runtime performance at negligible loss of accuracy.

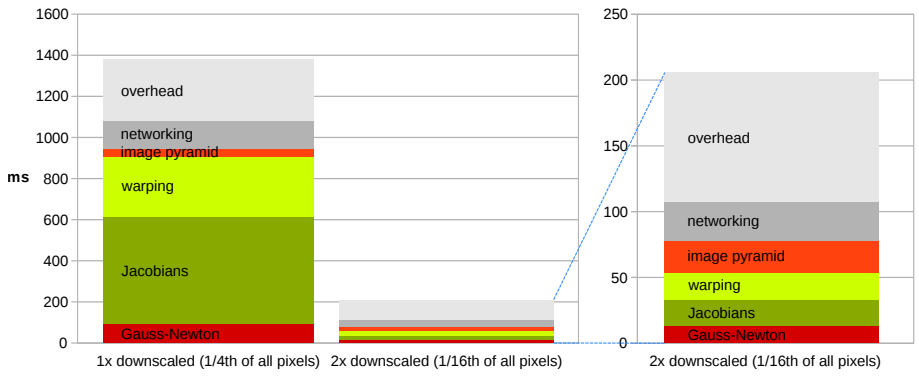

Figure 5.2: perfomance breakdown using early abort of the image pyramid

ARM core.

Chapter 6

Conclusion

Embedded odometry is feasible, but needs more optimization to run in realtime. E.g. by offloading XY to FPGA.

- better error rejection - use IMU - user arm neon / more of fpga - use keyframes

This work has shown that running photometric odometry on a computationally constrained platform is feasible and a perfomance of 5 Hz can be achieved by running semi-global matching on the FPGA and a few optimizations, mainly early abort in the image pyramid.

It also shows clearly, that there is still a big potential for moving even more parts onto the FPGA and achieving odometry out-of-the (small and lightweight) box.

Bibliography

[1] H. Hirschmüller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2. IEEE, 2005, pp. 807–814.

[2] D. Honegger, H. Oleynikova, and M. Pollefeys, "Real-time and low latency embedded computer vision hardware based on a combination of fpga and mobile cpu," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 4930–4935.

[3] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with fpga preprocessing for accurate real-time slam," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 431–437.

[4] A. I. Comport, E. Malis, and P. Rives, "Accurate quadrifocal tracking for robust 3d visual odometry," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 40–45.

[5] S. Omari, M. Bloesch, M. Burri, P. Gohl, M. W. Achtelik, and R. Y. Siegwart, "Real-time dense stereoscopic visual odometry."

[6] M. T. Dymczyk, "Visual-inertial motion estimation on computationally constrained platforms," 2014.

[7] S. Leutenegger, P. T. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, "Keyframe-based visual-inertial slam using nonlinear optimization." in *Robotics: Science and Systems*, 2013.

Appendix A

Irgendwas

Do you want doku for the code here? how to compile, configure and run?