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.
[1]
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
$\boldsymbol{J_T}$	6×3 Jacobian of the transformation operator $\boldsymbol{T}$
$\boldsymbol{J_P}$	3×2 Jacobian of the projection function π
$\boldsymbol{J_I}$	2×1 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
IMU	Inertial Measurement Unit
SGM	Semi-Global Matching
UAV	Unmanned Aerial Vehicle

Chapter 1

Introduction

1.1 Motivation

Robots are computationally and power consumptionally contstraint. So, we need algos that run fast, even on weak processors. Promising approach: Use FPGAs. However, not as easy to use and integrate, not suitable for every sort of problem.
In this work, a novel example is provided of leveraging the power of an FPGA to run things so far requring a huge PC on an embedded device. A semi-global stereo matching [ref to SGM?] core developed in [ref. to dominik] provides disparity data 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 a very efficient approach, 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 code to the FPGA.
In our case, a visensor developed by the ASL [ref to visensor] is used which features a Zynq board with a dual-core ARM and a XY FPGA [ref to zynq]. This sensor possesses a stereo camera with syncronized global shutters as well as a high-precision inertial measurement unit, which was not used tough.
- embedded visual odometry feasible? -
Hier kommt die Einleitung! Yes, really.
normally: odometry uses sparse features, often tracked over multiple frames
here, we use full image and try to warp it so that it fits the previous frame this eliminates all that tedious search for good and consistent features and makes use of all available data we need depth data, so we use a stereo camera
so far, this has all been done by others [sammy's stuff, comport et al.]
one of the main points of this work is to do all this on an computationally constraint platform, namely the visensor [reference here]. This thing features a ZynQ board [moar reference] with a dualcore ARM processor and an FPGA, two synchronized global-shutter cameras and a few other nifty things we don't use here (or can't even use, as somebody screwed up the FPGA configuration xD).
So, the goal is to make use of the disparity data provided by the FPGA and integrate this thing with the other, more general purpose thing. We want to determine how feasible such an approach is, and we therefore want to know how good the performance of this is. So expect to see some timing measurements soon (tm).
main goal: explore embedded visual odometry

how to offload computation onto an FPGA (here: visensor with XY FPGA and Z ARM (Zynq))
in this case: FPGA runs stereo matching (already done by Pascal et al.), use this data to do odometry on ARM (specifically, 'dense' odometry)

1.2 Related Work

Hier kommt Comport und Sammy hin...
vorallem: Marcin!
TODO: googlen was es sonst noch so gibt?

Chapter 2

Method

2.1 Overview

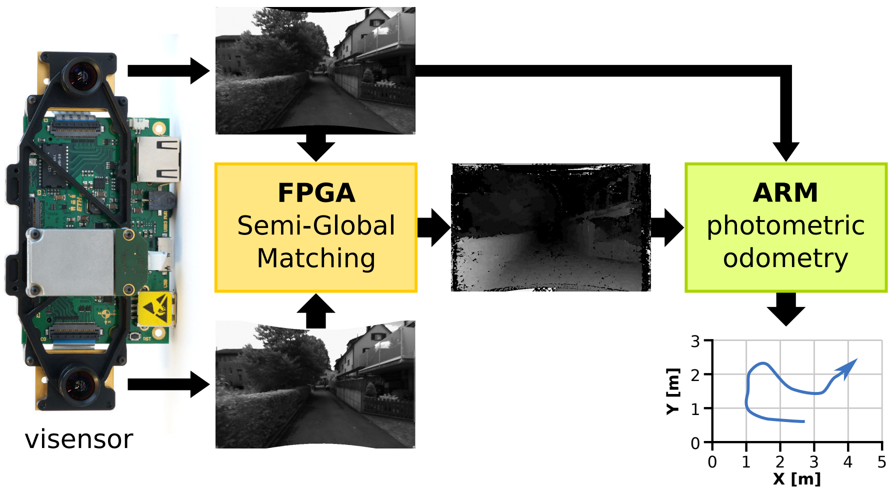

Figure 2.1: schematic overview of the whole system

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 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.

[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.

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

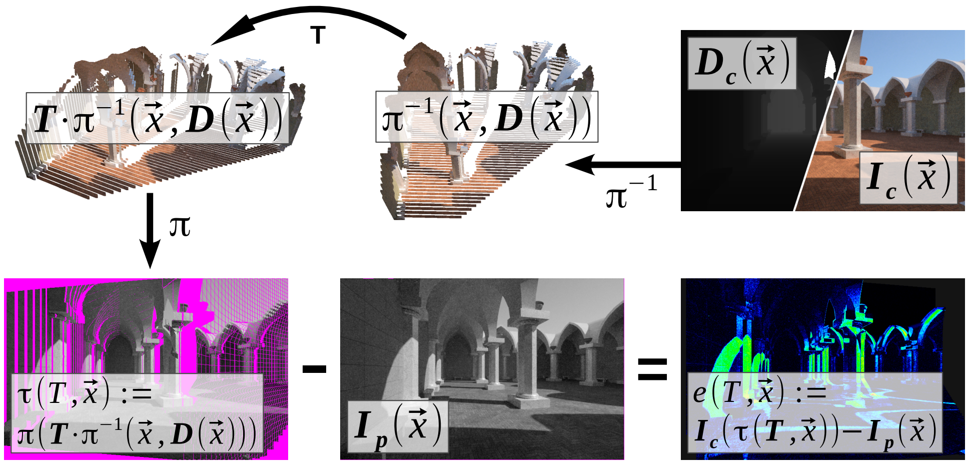
T
$T \cdot \pi^{-1}(\vec{x}, D(\vec{x}))$
$\pi^{-1}(\vec{x}, D(\vec{x}))$
$D_c(\vec{x})$
$I_c(\vec{x})$
π^{-1}
π
$\tau(T, \vec{x}) := \pi(T \cdot \pi^{-1}(\vec{x}, D(\vec{x})))$
-
$I_p(\vec{x})$
=
$e(T, \vec{x}) := I_c(\tau(T, \vec{x})) - I_p(\vec{x})$

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

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:

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

TODO: we have pixels with depth data, which we can project pixels into 3D space where they can be warped trough time and space. Then, we reproject them back onto the camera's image plane and get a new picture which is the old picture viewed from a different perspective ("warped").
Now we can compare this warped image with a previous frame (called 'The Keyframe') by simply calculating the squared error of the pixel intensities. We now have a function e(T) which we can minimize using classical approaches, like Gauss-Newton: Derive that thing ($J_I * J_P * J_T$) and use $J^T * J * \varDelta T = J^T * e(T)$.
To speed things up (and hopefully to increase convergence radius, but I somehow doubt this a bit), we use an image pyramid: scale down images, find minimum, scale up a bit again, improve transformation estimation, repeat.
- note on occlusions -¿ could use a Z-Buffer or something, but how to actually implement? (open question!) - still would need to handle outliers due to noise and moving things (people...)

2.3 Minimization

2.4 Optimizations and other improvements

Works, but not real-time capable so far. Too much data to process...

2.4.1 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 [TODO: ref to results]

2.4.2 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.

2.4.3 Weighting of errors

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.

2.4.4 Keyframes

Instead of matching the previous frame, a keyframe can be used which is 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.
This has not been investigated in this work.

2.5 Implementation details

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

2.5.1 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 $SO(3)$.

2.5.2 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 [2].

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

2.5.4 Don't remove too many pixels

The optimizations described in section 2.4 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 2.4.2 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.

[2] 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.

Chapter 3

Results

Hier kommen: - timings - a few notes about simulation - qualitiative evaluation - circles
To assess the accuracy of photometric odoemtry, [We/I] ran in circles and measured the final offset. For high quality depth images, generated offline by OpenCV [insert exact algo here] we get an error of [about 10-20cm for a circle with diam 3.5m/length=???]. Using the more noisy depth output of the FPGA, we get an error of [TODO].

Chapter 4

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

Bibliography

[1] M. Raibert, *Legged Robots That Balance.* Cambridge, MA: MIT Press, 1986.

Appendix A

Irgendwas

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