A custom, vision-based approach to inside-out tracking suitable for use in embedded systems

Report on internship at KITT Engineering

Jan Kolkmeier
Human Media Interaction Group, University of Twente
j.kolkmeier@student.utwente.nl

December 18, 2014

Contents

1. Introduction **4**
1.1. KITT Engineering 5
1.2. Motivation 5
1.3. Assignment Description 6

2. Background **7**
2.1. Pose Esitmation 7
2.2. Markers & Markerless 9
2.3. Embedded Systems 10

3. Activities **11**
3.1. System design 11
3.2. Exploring Hardware 15
3.3. Selecting the Approach 16
3.4. A Custom Marker 18
3.5. System Implementation 20
3.6. Evaluation 26
3.7. Final Demo 35

4. Conclusion **36**
4.1. Recommendations 36
4.2. Personal Reflection 37
4.3. Acknowledgements 38

References **40**

Appendices **41**
Appendix A. Circle Marker 80x80mm 41

1. Introduction

Estimating translation and orientation of an object in 3D space is a common problem for many robotics and virtual/mixed reality applications. In the field of computer vision various solutions can be found, but few non-proprietary solutions are also suitable for use with embedded, low power hardware.

In the following document, the work done and findings made during my internship at KITT Engineering[1] are presented. Goal of the internship was to explore the possibilities of a self-contained, vision-based inside-out tracking system capable of running on embedded computers.

Inside-out tracking refers to vision-based estimation of an object's pose with the camera attached to the object itself. There are approaches that make use of markers artificially placed in the environment. These markers are called *fiducials*. The camera recognises these markers and estimates its position and rotation relative to the marker. Other approaches are based on tracking natural features rather than artificial markers. In self-driving cars, for example, cameras track points around the car over several frames and make an estimation of how much and in what direction the car moved.

In the course of this internship, I worked towards building a such a vision based inside-out tracking system that would work on low powered, embedded computers. After designing an overall system, the possible approaches were evaluated.

A choice was made to use the *fiducial marker* based approach. Existing fiducial marker systems however exhibited performance bottlenecks, specifically in the marker detection, that would make their usefulness on embedded devices limited.

To circumvent this, a custom marker was designed with properties that would afford a faster detection method. The detection algorithm was implemented. A prototype of the inside-out tracking system was built using the custom marker.

Finally, this prototype was evaluated under various criteria relevant to fiducial marker systems. While this prototype had shortcomings in under some criteria, good results were obtained in the criteria of speed performance, outperforming the previously evaluated systems.

The report is structured in four chapters. In the remainder of this first chapter, I will give a motivation as to why I chose to apply for an internship at KITT, and why the

[1] `http://www.kitt.nl/en/`

particular problem (section 1.3) was interesting for me to solve. In chapter 2, I will elaborate the assignment in detail and provide some background information.

In chapter 3, the main activities performed during the internship are documented. This includes a design for the overall system (section 3.1), an overview of the used hardware and the evaluation of the possible approaches (sections 3.2 and 3.3). In sections 3.4 and 3.5, I present the custom marker design and document the implementation of a prototype. The implemented system is then evaluated in section 3.6 and the results are discussed. Finally, I show the final demo application (section 3.7).

I conclude my work done during the internship in chapter 4. Recommendations for future work are given (section 4.1) as well as a personal reflection on my time at KITT Engineering (section 4.2).

1.1. KITT Engineering

KITT Engineering is a small company with about ten people employed. Expertise's are ranging from web and embedded programming to electrical engineering, PCB design and industrial manufacturing. The company is based in Enschede, NL in the Buisness and Science Park across the campus of the University of Twente. KITT stands for '**K**reatieve **IT T**oepassingen', which is dutch for *creative IT applications*. The company lives up to its name, which is not only evident by their many collaborations with artists on art-installations, but also in their diverse work, ranging from devices for serious gaming in collaboration with university research, virtual reality applications to pipe inspection robotics and urban technology such as smart bicycle locks and public displays. KITT emphasises the use of open, reusable and modular hardware and software.

1.2. Motivation

KITT Engineering has expertise in building their applications on durable, efficient platforms that stand the 'real world' test. This is in contrast to the work I have done so far in my academic projects so far. Although I am very practical oriented, my solutions usually just work long enough to perform the next experiment or user-study. Also, I rarely had to deal with constraints in computing power, as I could usually make use of as many computers as needed in lab settings.

My motivation to do an internship at KITT was to learn about building self-contained, dedicated applications on low-power systems such as embedded platforms. The assignment - which is described in more detail in the next section - was related to computer vision and virtual reality. Two topics I have both some experience in and whose combination are becoming more and more relevant in the context of mobile and embedded

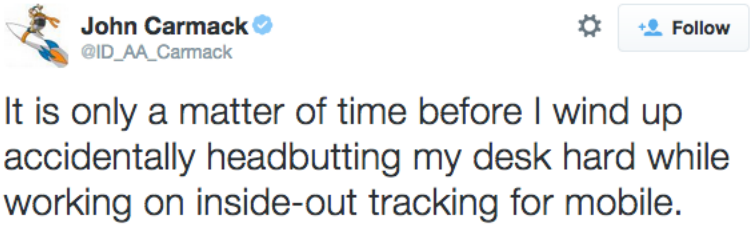


Figure 1.1.: John Carmack comically revealing that his work at Oculus VR involves inside-out tracking on mobile devices.

systems[2][3]. Gaining more insight into the limitations of these platforms and acquiring some practical development skills in the area of computer vision and embedded programming were my primary learning goals for this internship.

1.3. Assignment Description

The assignment was - in one sentence - *to explore the limits of inside-out tracking for use in embedded computers*. Inside-out tracking - also referred to as self-tracking or egomotion - means that an object's pose is tracked by vision-sensors attached to the object itself, rather than equipping the surrounding area with such sensors in order to tracking the object. In inside-out tracking, features in the environment are detected and tracked, whereas in outside-in tracking, features on the object itself are tracked. Such features might be artificially placed *fiducial markers* or natural features used in *markerless* tracking.

At the time there was no active project or research related to this problem at KITT. However, a possible application for such a system was made explicit: To equip a head-mounted display (HMD) with it and map the measured translation to translation to virtual reality scenes. This use case was kept in mind throughout exploration and development.

To highlight the relevance of this assignment, consider the *tweet* shown in fig. 1.1. It was made by John Carmack, currently CTO of the Oculus VR[4]. He states that part of his current work deals with the exact same matter.

[2]Google Project Tango: `https://www.google.com/atap/projecttango/#project`
[3]Samsung's GearVR: `http://www.samsung.com/global/microsite/gearvr/`
[4]Oculus VR: `http://oculusvr.com`

2. Background

In this chapter the problem posed in the assignment will be dissected further to give an overview of the involved challenges and some of the existing work in this area.

2.1. Pose Esitmation

The central problem of camera pose estimation using vision is calculating the extrinsic parameters of a camera. Extrinsics contain the translation and rotation in relation to some world coordinate system. Calculating the camera extrinsics first requires understanding the mapping of 3D world points onto the 2D camera plane. In this section, I review the basics of the camera model, and how they relate to camera pose estimation.

An object is located at location $X = [x_w, y_w, z_w]$ in world coordinates. The pinhole camera model gives information on which location $x = [x_i, y_i]$ in the camera image the object will end up (see figure fig. 2.1).

In mathematical terms, the camera model is a 3×4 projection matrix. Multiplying locations - or rather: vectors - with a projection matrix means transforming them from one space to another. In the case of our camera projection matrix C, this transformation behaves in such a way that it transforms a vector X from the world ($\mathbb{R}^3$) to a vector x ($\mathbb{R}^2$) in the resulting image.

To accurately model an actual camera, the values in the matrix C must be chosen in such a way that this transformation is correct. The matrix values are the product of the intrinsic parameter 3×3 matrix K, which describes parameters such as lens deformation and sensor position, and the extrinsic parameter 3×4 matrix:

$$C = K[R\ t]$$

The extrinsic parameter matrix $[R\ t]$ contains the rotation matrix R that describe the orientation of the camera, whereas the vector t is the translation vector from the world coordinate system. The (simplified) intrinsic parameter matrix K has the form:

$$K = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

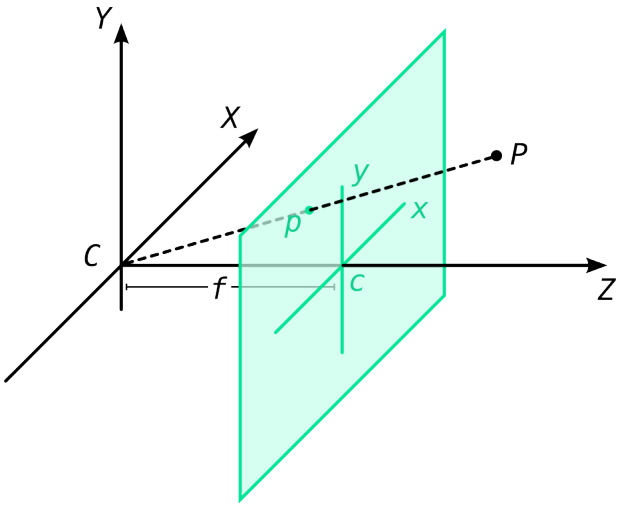


Figure 2.1.: Pinhole camera model. Point P is projected from $\mathbb{R}^3$ to point p in the 2D image plane in with the coordinate system x, y.

Here, f is the focal length and the principal point is $[u_0 v_0]^T$. Other parameters such as skew and scale factors can be encoded in other parts of this matrix. A point X in world coordinates is transformed to the point x in camera coordinates by the following model:

$$x = K[R\ t]X$$

To estimate camera extrinsics, this equation can be reformed to have $[R\,t]$ as an unknown. It can be solved on with a set of known world points (such as X) and their correlated points on the image plane (such as x). This is because one can also consider the camera intrinsics K as a known, as it can be obtained through calibration procedures (see [16]), which are part of many computer vision frameworks. These frameworks typically also contain high level functions to solve this equation. At least four correlated pairs of non-collinear points are required to obtain a solution with these methods. Note that typically, not the camera pose is estimated, but rather the pose of the object relative to the camera. For inside-out tracking, one can derive the camera world rotation R_C and t_C from the objects pose $[R_O\ t_O]$ using these equations:

$$R_C = R_O^T$$

$$t_C = -R_C * t_O$$

Figure 2.2.: Some examples of fiducial markers.

2.2. Markers & Markerless

The essential part in pose estimation is finding world points, and recognising and correlating them in the image plane of the camera. [11] surveys the several methods. They can be discriminated into two categories, those methods using *fiducial markers* and those who perform *markerless* tracking, based on natural feature detection and recognition.

Markers Fiducial markers are artificially placed images such as those shown in fig. 2.2. Various approaches exist ([4, 6, 7], to name a few). Generally, these markers can be detected by searching through the image and they can even encode information such as IDs. To estimate the pose of such a marker, the principle described above is used. Prominent features of the marker, such as the position of corners of the surrounding square relative to the marker origin, are used as 'world points'. These corners can be found back in the image, to then calculate the marker's pose. Note that such markers do not need to be pictures. For example, a constellation of infrared LEDs would behave the same way, with only a different approach to searching and identifying distinct markers, such as measuring blinking frequency of individual LEDs over time. Note that not all fiducial markers approaches are suitable for full 3D pose estimation. Some focus on 2d position and single-axis rotation[9]. Fiducial markers have also been used for inside-out tracking [1, 13].

Markerless Markerless methods for pose estimation rely on detected features in the environment, without using artificial markers. There are various methods for feature detection, most prominent SIFT (Scale Invariant Feature Transform, [12]) and SURF (Speeded-Up Robust Features, [3]), and their derivates. The idea is to find distinctive areas in a scene and describe them in such a way that they can be identified back in other images of the scene, even if perspective has changed (see fig. 2.3). The displacement of known points gives information about how the camera has moved through space. Markerless inside-out tracking has most dominantly be used in robotics for self-driving cars ([5, 14]), where it is typically referred to as visual odometry or egomotion. It is usually integrated with other odometry sensors, such as accelerometers, to compensate for drift. The advantage of this method is that it works in unknown environment - if

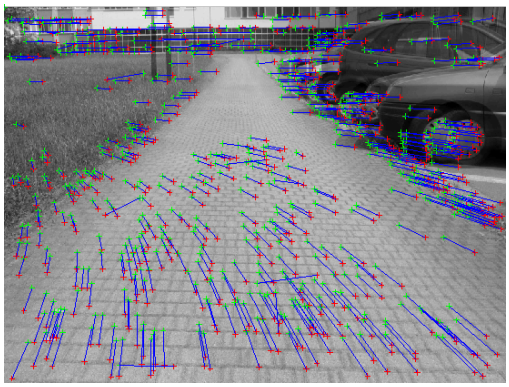


Figure 2.3.: Matching detected features from a previous frame to the current frame to estimate transition in pose.

enough features can be detected. However, it is, computationally, much more expensive than marker based solutions.

It should be noted that there is another type of 'markerless' tracking. *Image targets* are markers that are created from planar images using the same feature extraction methods as in markerless tracking. Now, the coordinates of each detected feature is described in its coordinates on the original image. The difference to fiducial markers is that it works with only parts of the image visible to the camera - as long as these parts still contain enough detectable features. This approach is typically classified under markerless tracking because the reference images do not need to be artificial - an application could extract features from a picture on a wall or from a patterned wallpaper, and track these features for pose estimation.

2.3. Embedded Systems

While there is various research in various aspects of vision based pose estimation, there are only few reports on employing these techniques on embedded systems. Some markerless approaches for visual odometry have been evaluated on flying robotic platforms[15], it is however noted that the problem in general is particularly challenging, since - unlike on ground robots, there is no other means of measuring odometry, which makes the frame-by-frame estimation of the markerless approach error prone[2].

Fiducial and image-marker based approaches on embedded devices can predominantly be found in mobile Augmented Reality (AR) applications. These, however, are typically proprietary[1][2], and geared towards Android or iOS specific platforms.

[1] `https://www.qualcomm.com/products/vuforia`
[2] `http://www.metaio.com/products/sdk/`

3. Activities

In this chapter the main activities carried out during the internship are documented. I first describe the design of an overall inside-out tracking system. It sketches a network of software and hardware components that should apply to the general problem. It should be noted that while the ambition was to realise the complete system during the internship, most other activities dealt with more specific problems of the design, leaving several other parts of that design unimplemented in the end.

These specific problems are described in sections 3.2 to 3.5 and include getting familiar with the new hardware, researching software and algorithms for the approach and designing custom algorithms before finally implementing some core parts of the proposed system in a prototype.

In section 3.6, we evaluate this prototype. Finally, in section 3.7, I present a demo application to be used with that prototype.

3.1. System design

In this section I present a hierarchy of the hard- and software components that would generally be involved in an inside-out tracking system. It was proposed as preparation to the internship, and - while not fundamentally flawed - it was certainly too ambitious. Still, it acted as a general guideline throughout the following activities.

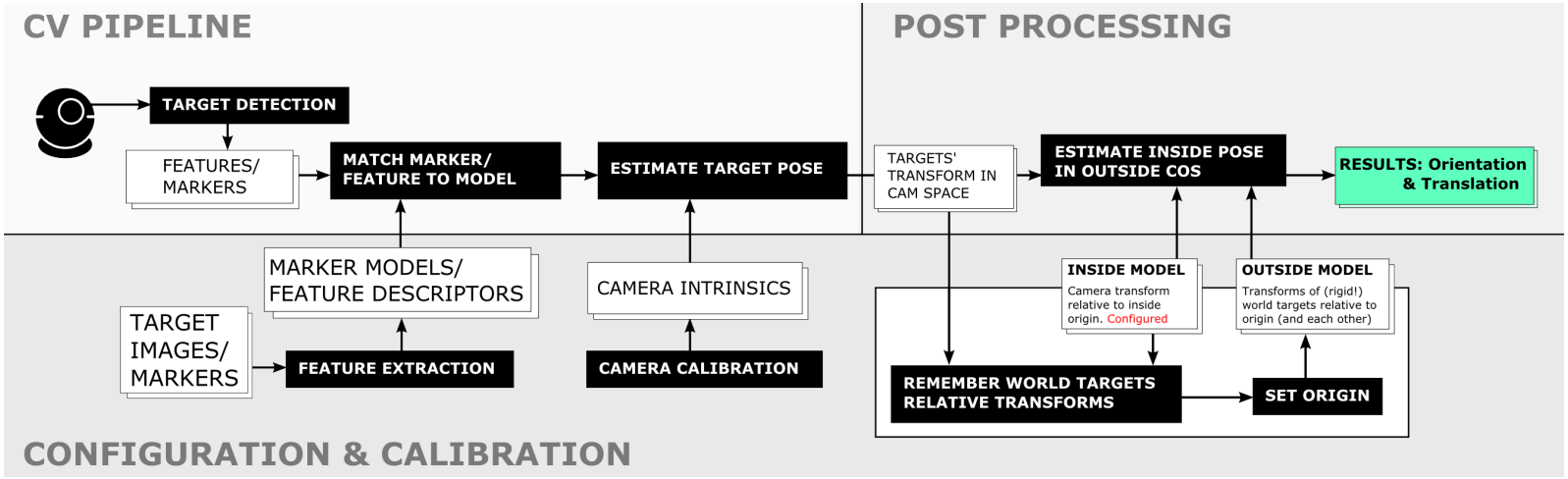


Figure 3.1.: Generalised schema of processes involved to estimate a monocular camera's pose

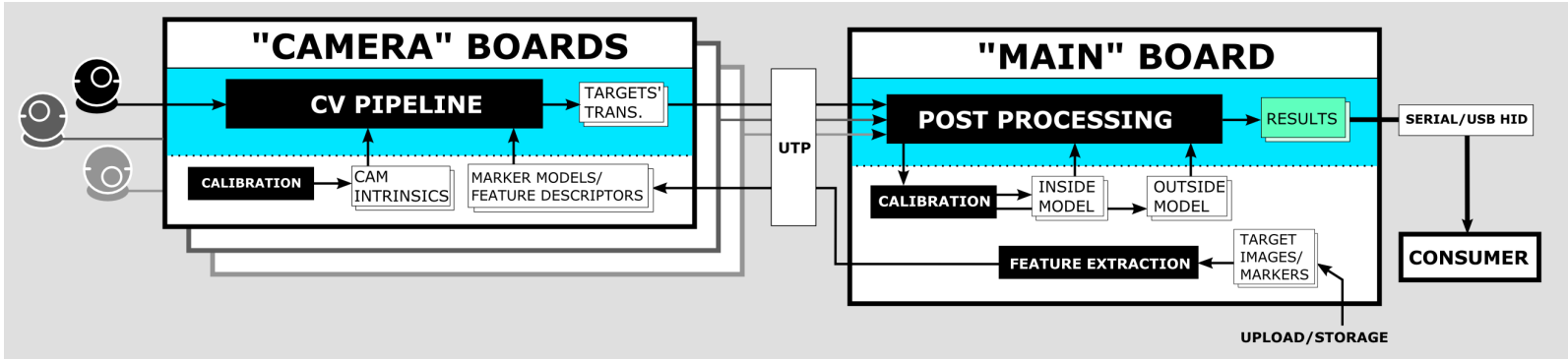


Figure 3.2.: Proposed design of a multi-camera tracking system

A first design choice was made to focus on either fiducial markers or image targets. While both have their own advantages and tradeoffs (see section 3.3), they are interchangeable in many ways. Entirely markerless tracking however was deemed to be too computationally expensive to be used in embedded devices. We can conceive how, to have tracking coverage in a wider area, we always have to have at least one marker in view of the camera. To ensure this, it was considered using more than one camera, in order to avoid having to cover the whole area that the system is to be used in with artificial markers. Being aware of the computational restrictions of embedded hardware, this also meant that each used camera would come with its own computing unit.

3.1.1. Camera Rig

We refer to multiple cameras mounted on the object to be tracked as the *camera rig*. The cameras must be rigidly mounted relative to a chosen origin of the camera rig - each camera has a offset translation and rotation relative to this origin. Estimating the pose of the camera rig - for example, when attached to a HMD in a virtual reality application - involves first estimating the camera pose in world space, and then compensating for the camera's offset in rig-space to conclude the pose of the entire rig.

Now, there are several tasks involved in such a system to compute the pose of the camera rig, with several independent computing units. In the next two sections I first describe the involved tasks for a single camera, and then how they can be implemented in a distributed system.

3.1.2. From image to pose

First, the tasks at the top left of the diagram in fig. 3.1, which deal with the main computer vision pipeline.

After an image is acquired from a camera, we detect (possible) targets. Those are either in the form of detected features, or possible fiducial markers. In the case of image targets, detected features must be correlated with those features extracted of all used image targets to identify the right one (if any). For fiducial markers, this is slightly simpler, as the ID is encoded in the marker itself. Given some information of the real-world scale and model of the marker, we can estimate its pose relative to the camera using the procedure described in section 2.1.

Outside model The tasks in the bottom bottom right of the diagram deal with learning the markers in the environment after they were placed or moved. The idea is that at run-time, with only a single marker in view, we assign that markers world pose to the current inverse of it's relative pose in camera rig space, placing the camera at the world origin. Now we can incrementally calculate the world pose of any other marker in view

of the camera, as long as one other marker with a calculated world pose is in view. I refer to this collection of marker world poses as the *outside model*.

Inside model In fact, if we have configured multiple cameras in the camera rig and know their relative positions towards each other, we could also set world poses of markers detected in one camera based on the known world pose of a marker detected in another. And, even more so, seeing the same marker in two cameras would allow us to estimate their relative pose in the rig as well. We can see how given this design, a powerful calibration procedure could be developed. I refer to the information about the relative poses of cameras in the camera rig as the *inside model*.

Now, in the top right of the diagram, we see the post processing tasks. Given known world poses of markers and relative positions of the cameras, we can inversely compute the final result using the detected relative positions calculated in the top left part of the diagram: the world pose of the camera rig's origin.

The tasks the bottom left of the diagram can be performed offline. If using image targets, we must first detect and extract usable features from these markers. The information is to be stored such that it can be used at run-time to correlate features in camera images with those from the image target. Similarly, for the case of fiducial markers, we need to 'extract' some information, such as the size and model used per marker ID. Another task that can be performed offline is the camera calibration, which only needs to be done once per camera.

3.1.3. Task distribution

In fig. 3.2, it is shown how these tasks could be distributed across several computing units. Consider *camera boards* and the *main board*. The camera boards on the left all perform the main computer vision pipeline of detecting and matching markers, and estimating their relative pose in their own camera space. For this, they need to have the calibration data for their individual cameras created once during a calibration procedure. Each board needs to have access to the same information about the used markers (such as feature descriptors and scale).

On the right, we have the main board. It is connected to all camera boards. It would be convenient to include the offline feature extraction of the markers on this board, as it is something that has to be shared alike with all camera boards. The main board is further responsible for the post-processing and keeps track of the outside and inside model of the system (section 3.1.2).

Lastly, the main board is responsible for integrating (possibly filtering) the information and share it with the consumer. The consumer could be connected through Wifi, ethernet, a serial port or as USB host, with the main board acting as USB HID.

3.1.4. Discussion

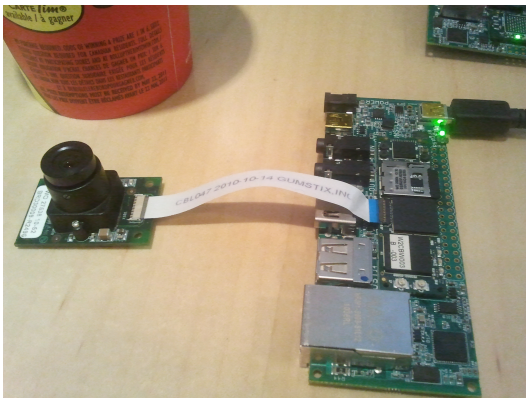
The system design presented in this section outlines the general approach of a multi-camera, multi marker inside-out tracking system, identifying tasks and assigning them to different components. One thing that is worthwhile noting is that, so far (and in the following), the term *tracking* has been used generously. The proposed system solely performs pose estimation, which means repeating the exact same task frame by frame. It 'tracks' individual markers only because it identifies them each frame anew. However, tracking usually refers to the concept of making predictions based on previous measurements to reduce computation of search for markers/features. While this can happen for each camera board individually, it might also be desirable to share knowledge from the main board with the camera boards. Especially in the case of feature detection, where reduction of the search space can give performance improvement, for example based on the knowledge which image targets cannot possibly be seen by a given camera at a given time.

3.2. Exploring Hardware

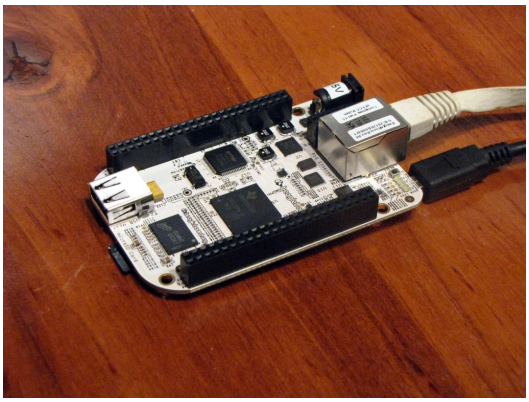
My first practical activity was getting familiar with the available hardware. In particular, I was provided with a Gumstix Tobi[1] development board fitted with a Gumstix Overo[2] COM (computer-on-module). Further, a Caspa[3] camera board was provided (see fig. 3.3a). The Overo COM has a 800Mhz ARM Cortex-A8 and 512MB ram at its disposal, as well as a C64x Fixed Point DSP clocked at 660Mhz. The camera has a resolution of 752x480 pixels and is capable of capturing images at up to 60hz.

I went through several (versions of) operating systems to find one with camera drivers

[1] `https://store.gumstix.com/index.php/products/230/`
[2] `https://store.gumstix.com/index.php/products/265/`
[3] `http://wiki.gumstix.org/index.php?title=Caspa_camera_boards`



(a) Gumstix



(b) Beaglebone

working out of the box as well as providing a reasonably recent toolchain for developing computer vision applications. Especially the latter proved difficult as I desired having a setup where I could compute the same code on both my personal computer as well as on the COM to ease the development process. However, those operating system builds that were officially supported were outdated and especially lacking in compatibility with any reasonably recent version of the OpenCV library. Eventually the Yocto[4] distribution provided a recent set of tools and a recent OpenCV version.

Now I got familiar with the Overo's integrated ISP (image signal processor). It is a pipeline in between the Caspa camera and the main processor. A tool called `media-ctl` is used to configure this pipeline. The configured pipeline then scales each image coming from the camera to a desired resolution and decodes it into a desired colour format - without using any processing power. This is a handy feature and means that images coming from the ISP have less latency than, for example, images images coming from a USB webcam, which have to be decoded on the main processor first.

Being able to open the camera and capturing images, the next step was to use them in my own applications. For this, I had to circumvent a unimplemented API in the camera driver that was causing OpenCV not to start capturing from the Caspa camera. A patched kernel module was compiled and loaded that 'faked' the response of the API request expected by OpenCV. This took quite a while, but was certainly a learning experience. Now I was able to capture camera frames reasonably fast and use them in my own applications.

After a while however, the Tobi development board started to give problems such as unpredictable power-cycles, which required me to switch to a BeagleBone[5] ARM board (see fig. 3.3b). Sadly, it does not have a ISP interface, which means I had to fall back to slower USB cameras.

3.3. Selecting the Approach

In the this section, I describe some of the considered approaches to vision based pose estimation which where evaluated. The most interesting metric next to accuracy was the speed at which it would perform on the ARM computer. At this stage, I used the rule of thumb: It should perform at '10Hz or faster' for the complete process from capturing the image to pose estimation to be considered a 'good' option.

As mentioned, the main choice was between using fiducial markers or image targets. Using IR markers was discarded early on due to the additional involved electronic component. For image targets, no preexisting complete library was tested or found. For fiducial markers, the ArUco library was tested.

[4] `https://github.com/gumstix/Gumstix-YoctoProject-Repo`
[5] `http://beagleboard.org/bone`

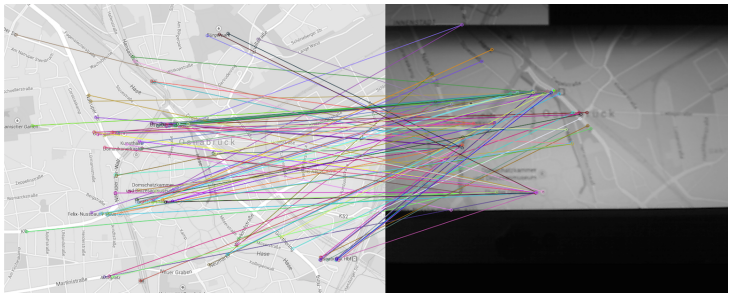


Figure 3.4.: Attempt at using feature detection and matching with OpenCV.

3.3.1. Image Targets

First attempts were made with setting up a test application for the image target approach. Features were extracted from a test image using the various methods provided by the OpenCV feature detection and description API[6]. Webcam images were captured with a printed version of the test image in view. Then, using the same technique again for extracting features from these webcam images. Matches between the features detected in the test image and the webcam image were searched (see fig. 3.4) and the pairs were used to estimate the pose of the test image in webcam space. We selected best pairs for pose estimation using the RANSAC[7] method.

With several seconds of computation, depending on the test image, the amount of features used and the used methods, performance on the tested ARM computer was several orders of magnitude off of what was desired. Furthermore, measures taken to reduce computation time would at the same time affect the accuracy of the results significantly.

Some research was done in possibilities of using the dedicated DSP (digital signal processor) on the COM, but the required amount of work was well beyond the scope of this internship, involving rewriting entire parts of the OpenCV library to use fixed point arithmetic. Further, this was something that was attempted by more knowledgable persons in the scope of a Google Summer of Code project without promising results[8].

3.3.2. Fiducial Marker

At this point, I discarded the idea of using feature-detection based image targets. Instead, focus went towards the use of fiducial markers. Besides several binary-only solutions such as ARTag[6], there are some open source libraries providing fiducial marker tracking functionality. One that would work without bigger problems was ArUco[9] (fig. 3.5).

[6] `http://docs.opencv.org/modules/features2d/doc/feature_detection_and_description.html`
[7] `http://en.wikipedia.org/wiki/RANSAC`
[8] `https://code.google.com/p/opencv-dsp-acceleration/`
[9] `http://www.uco.es/investiga/grupos/ava/node/26`

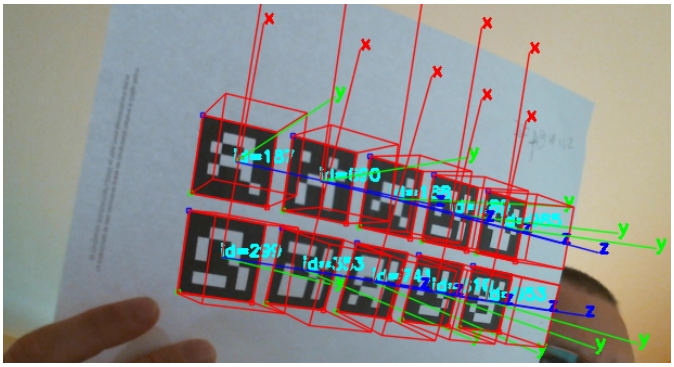


Figure 3.5.: ArUco estimating marker pose and rendering AR content in the scene.

ArUco performed performed better than my first, feature-detection based approach, however it would still perform only at less than 3Hz (using the Beaglebone and a USB camera at this point). As mentioned, my expectations at that time where slightly higher.

Looking at the source code of ArUco, there was no evident opportunity for improvement. The bottleneck seemed to be the initial detection of the marker. After some investigation, it turned out that it uses a contour-based shape detection algorithm. This would search for all connected areas in an edgemap of the captured image and approximate their outlines with vectors. Those with square outlines were the considered fiducial markers. Both, the generation of the edgemap as well as the approximation of the outlines turned out to be rather expensive computations for the ARM computers - especially in noisy environments, where many contours would have to be evaluated.

After some brainstorming with my supervisor, it was decided to implement a marker detection myself. Main motivation was to be in control of the (complexity of the) detection algorithm, which would make it easier to optimise later. This would also align better with my learning goals by sticking to the ARM architecture rather than implementing a computationally more expensive approach on the PC.

3.4. A Custom Marker

In this section I describe the design of a custom marker that must be detectable without expensive computations. I give an explanation of the detection method, as well as further motivations behind the different aspects of the marker design.

One of the suggestions of my supervisor was to look at circular markers. With nested circles of alternating colour intensity, rotationally invariant patterns could be created that would theoretically be detectable in near linear time, as a function of the amount of pixels in the image. I followed through with this idea, coming up with the marker shown in fig. 3.6, and a conceptual state-machine shown in fig. 3.8 to look for the pattern in a single row of pixels.

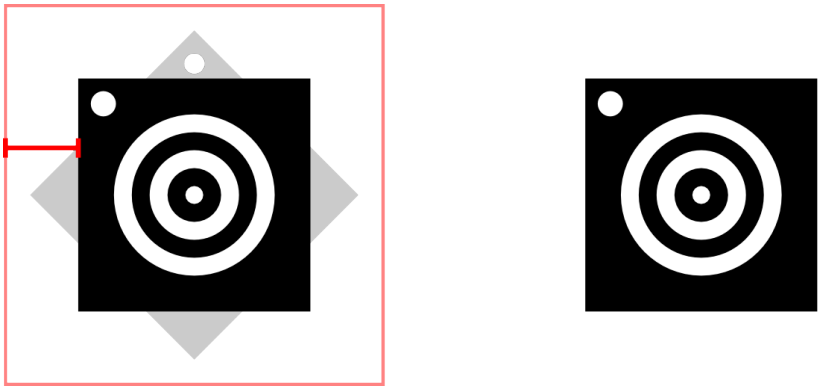


Figure 3.6.: The circle marker (right) and the circle marker with the recommended safety whitespace around it.

The marker, which I creatively call: *the circle marker*, consists of a black square with a bullseye pattern in the center. The center white dot has a diameter of a. There are two white rings around the center dot, the ring's thickness also equal to a. The space between the white rings (and between the center dot and the first ring) equal a as well. The outermost white circle is enclosed by a black square, the space between the white circle and the edge of black square equals $2a$ at its shortest point, and $\sqrt{2(6.5a)^2} - 4.5a \approx 4.7a$ at the diagonal of the square. In the upper left corner a white spot is located. To be precise, it is on the cross-section of the diagonal between the upper left corner and the bottom right, at the ratio $r = 22.98/212.13$. The diameter is 'slightly bigger than a'[10].

Motivation for this design is the idea of detecting the marker by using a simple state-machine that reads the a binary image pixel by pixel through rows and/or columns. The state-machine must only enter the accepting state when a sequence of pixels (a row or column) intersect the center of the marker as well as all alternating rings and the leading and trailing black areas created by the surrounding square. Given the rotation invariant design, this should always be the case when the marker is not otherwise occluded and the resolution of the image is high enough to reproduce the alternating pattern.

A sample is given in fig. 3.9. A line of binary pixels is only accepted (state A in the diagram) by the state-machine if there are eight flips from a groups of pixels black to white pixels or vice versa (states 2, 3 and 4). At each flip however, the ratio between the current group and the previous group must be about equal. Note that this threshold is chosen a bit more generous, as the thickness of the rings as imaged by the camera is *not* invariant under perspective transformation.

Before and after these alternating rings are accepted, the leading and trailing black pixels must have been read. Since they are by design significantly wider than the rings, the amount of black pixels must be bigger than the following or previous white pixels

[10]The white corner dot was placed 'intuitively' in an experiment. It never got moved to a more defined location.

respectively (states 1 and 5).

In an implementation of this, we would further keep track of the indexes of the first and last ring that was detected. The distance of these indexes can give information about the location and size of the marker, which is useful to find the the corners of the enclosing black square.

For finding the exact corners, I have not designed another algorithm. I rather use an the existing contour detection method. Although it is expensive to run on an entire image, given the information of the marker location, we can reduce the operating area for the contour detection significantly. This is also why part of the circle marker's design is to include sufficient whitespace around the black square (see fig. 3.6). Remember: we do not know yet the rotation of the marker. Now, with enough whitespace, when a circle is detected, we can safely search in a more generous area so we do not have to worry about either leaving out parts of the marker or, conversely, include parts of the environment instead.

The white marker in the corner can be used to tell the orientation of the marker. For this, the implemented detection algorithm must find the corners of the enclosing square, and then interpolate the pixel coordinates between the two pairs of diagonally opposing corners with the known interpolation ratio r (and at the opposite side: $1 - r$). At one of these four interpolated pixel coordinates should be a (group of) white pixels. This is where the upper left corner of the marker is. Note that this still works independent of the perspective transformation when looking at the marker from a side. This is because the cross ratio of three points on a line is invariant to projective transformation, as shown in fig. 3.7.

3.4.1. Discussion

A marker was designed with properties that should make detectable in a fast manner using the searching state-machine. Note that, although there are several circular fiducial markers[13, 10, 8], as far as I could tell, none use a comparable method for detection. Most rely on detecting connected areas with equal centroids, or on on contour detection on the entire image.

At this point, the marker is lacking one significant property: it does not carry any information such as an ID. It was argued that it could be added in future iterations. In the following section, I describe the my implementation of an inside out tracking system that works with one *circle marker*.

3.5. System Implementation

A software prototype was implemented to test the previously designed marker detection approach and to implement the functionality of an inside-out tracking system that es-

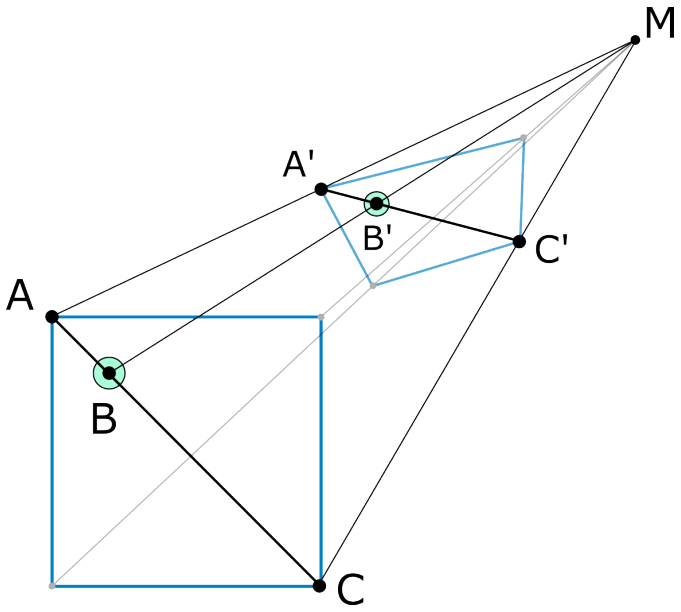


Figure 3.7.: The cross ratio of three points on a line is invariant to perspective transformation: $cr(A, B, C) = cr(A', B', C')$, with A and C being the upper left and lower right corner of the marker, and B being the location of the white corner spot.

timates the camera pose based on the detected circle marker. Reflecting on the system design in section 3.1, the implemented prototype focusses mainly on the tasks inside the computer vision pipeline, namely marker detection, matching marker features (corners) to the marker model, and estimation of the camera pose based on the matches.

3.5.1. Setup

The entire software was realised using the C++ API of the OpenCV library. The main marker detection and pose estimation procedure was extracted and built into a library I named KCVJ, standing for **K**ITT **C**omputer **V**ision **J**an. The KCVJ library in is easily included to third applications and can work with images from disk, captured from webcams and videos alike.

3.5.2. Circle Marker Detection

I implemented the state machine for detecting the circle marker in a sequence of pixels as described in section 3.4. However, some extensions were necessary for it to work reliably, as described in the following paragraphs.

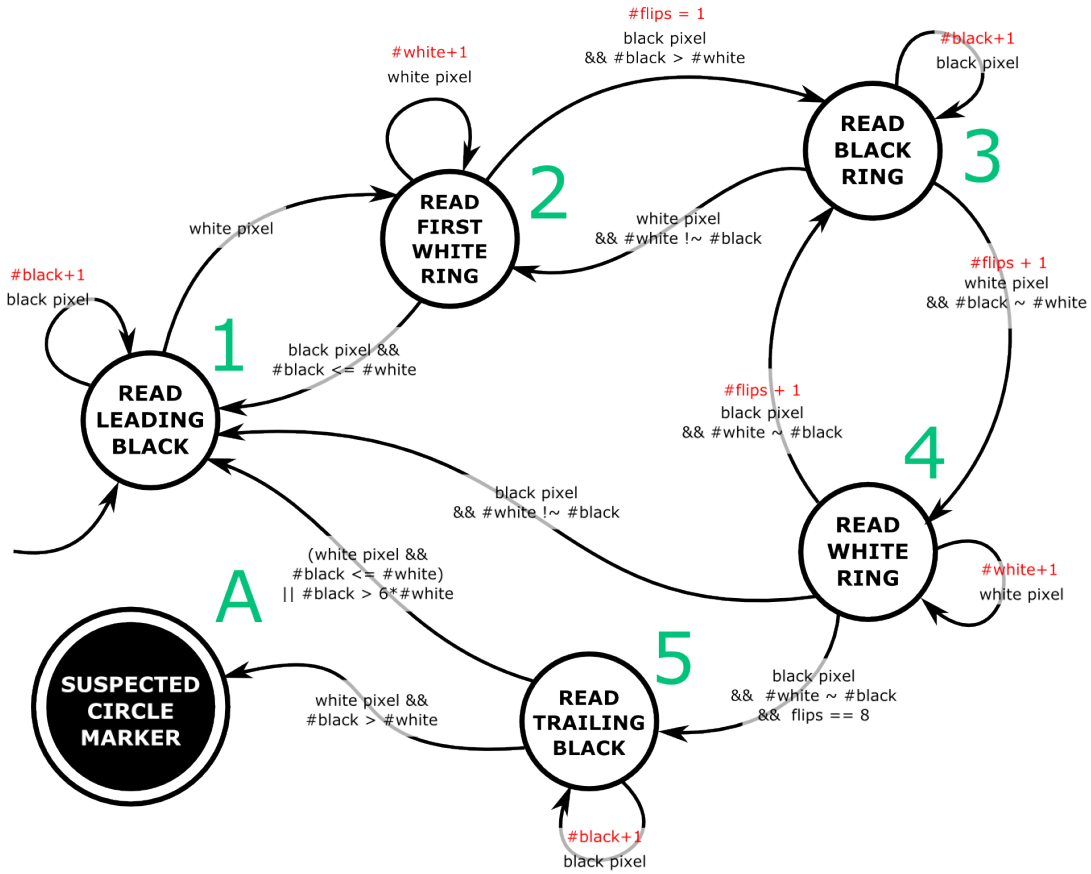


Figure 3.8.: The state-machine that accepts the sequence of pixels generated when walking through the center of the *circle marker*. One omission in this diagram is that always one of the two variables `#black` and `#white` which represent the counted consecutive pixels, has to be reset when entering a new state.

Figure 3.9.: An example of a line parsed by the state-machine, numbers inside the black and white pixels refer to the parsing state of the state-machine in fig. 3.8.

Searching The detection algorithm searches in down-sampled, binary-thresholded version of the captured image (see fig. 3.10). We call the scale by which it's size is reduced the *search scale*. First tests yielded many false positives in noisy areas. First attempts were to be more strict on the amount of pixels needed to accept the 'rings' - this however yielded too many false negatives, as the actual circle marker would not be recognised anymore at all scales.

Eventually, the solution was to do two searches, one in horizontal image lines (rows) and the other in vertical image lines (columns). We suspect circles only in those areas where both in vertical and horizontal directions the circle detection produced a positive search result.

Further improvements where to reduce the search on the second pass (when searching vertically) to only those columns where the first pass had already positive results. For this we make use of the first estimates of the center and width of the circle marker based on the distance between indexes the first and last ring.

Another effective filter against noise was to require at least two suspected circles with similar size to be detected very close to each other in the first pass.

Finding Corners We rely on OpenCV's contour detection[11] to find the outlines of the black square of the marker. An area around the detected circle is chosen as ROI (region of interest). Size of the ROI depends on the dimensions of the detected circle in horizontal and vertical direction. The output of the contour detection is a list of all detected shapes inside the ROI, each shape is defined by a list of points that mark its outline. Note that the contour detection can be configured to only return the extreme outer contours in the ROI, instead of making a hierarchical search, looking for contours inside contours. This way, we do not have to worry to have the inner rings of the circle marker in the results.

To find the four corner points of the marker, we consider all returned contours that contain exactly four points. We calculate the area of those shapes, and assume that the shape with the biggest area is the black square of the marker.

Refine Corners With the four marker corners found in the binary, scaled image, I employ sub-pixel refinement[12] of corner locations. This method looks for the exact corners inside the full-resolution image. It starts at the initial approximated corners, and iteratively tries to make a better estimate on the corner location based on nearby edge-gradients. This should improve the overall result of the pose estimation (see fig. 3.11).

[11] `http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours`

[12] `http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=cornersubpix#cornersubpix`

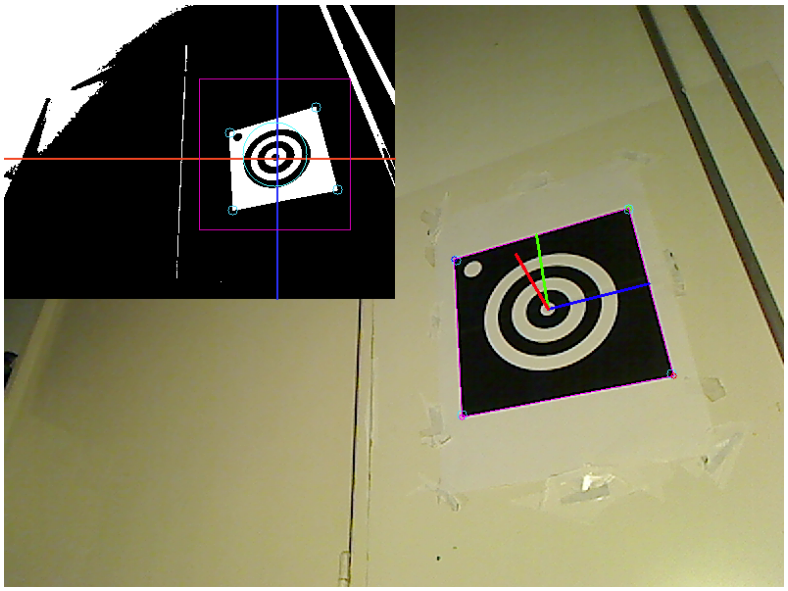


Figure 3.10.: The debugging output of the application. The scaled binary search image in the top left, with the ROI in which the circle marker is expected represented by the magenta square. On the right side the detected & refined corners of the marker in the high-resolution image.

Figure 3.11.: Refining corners. The center of the cyan circle marks the approximation from the binary search image, the refined location is at the center of the magenta circle.

Sort corners The - now refined - list of points returned from the contour detection are sorted in clockwise order. However, which of the four corner points was detected first depends on the rotation of the marker on the camera plane. The procedure described in section 3.4 is employed to look for the white corner spot, and sort the corners accordingly to fit the model of the marker.

3.5.3. Pose Estimation

Given the model of the marker and the corresponding corners in the image, pose estimation can be performed as described earlier. I made use of OpenCV's `solvePnP`[13] function, which estimates translation and rotation of the marker in camera space. We perform the calculations described in section 2.1 to retrieve the pose of the camera in marker coordinates (see fig. 3.10). With this calculated, the main part of the computer vision pipeline as proposed in the system design is implemented.

3.5.4. Post-processing, inside and outside models

The post-processing tasks, such as procedures for auto-calibrating world poses for newly detected markers (see section 3.1.2) are only marginally implemented. Only the functionality to set the current camera pose as the origin by setting the inverse pose of the currently detected marker as it's world position is implemented. So the outside model can be created for a single marker only. The inside model of the camera rig is restricted to the translation of the single camera. While a lot of the functionality that needed to realise the complete calibration of the inside and outside model is already implemented, it should be noted that the classes in the current implementation do not reflect the many-to-many relationship between multiple cameras and multiple markers well.

3.5.5. Camera Calibration

Lastly, I should note that an existing camera calibration solution was adapted and extended so it would work on the used ARM computers. Additional options were added to disable the GUI, as well as options to perform the calibration at lower camera resolutions instead of the maximum. The calibration software requires images taken from the camera with a chessboard pattern in view from different orientations and positions (see fig. 3.12). These images can be taken 'on the fly' from a connected webcam. It turned out that the 'on the fly' detection of the chessboard pattern was too heavy for the ARM computers at least at high image resolutions. Another option was added to take the pictures first and perform the chessboard detection and calibration procedure after.

[13] `http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp`

Figure 3.12.: Calibrating a camera using the chessboard pattern.

3.5.6. Discussion

The implemented prototype was built to test the concept of the designed circle marker. Consequently, this prototype only partially implements aspects of the original system design. The system was never realised in the architecture proposed in section 3.1.3. Further, it inherits the limitations of the designed marker. Most notably that markers cannot have IDs. So far, I have not shown any metrics of how well the implemented system performs under different circumstances. This is the focus of the next section.

3.6. Evaluation

The implementation of the circle marker detection and pose estimation was evaluated according to several performance criteria. [6] proposes a list of eleven evaluation criteria for fiducial marker systems listed below. However they - surprisingly - do not include any criteria about the *precision* of the estimated pose. It was added as a twelfth item.

1. the *false positive* rate,
2. the *intermarker confusion* rate,
3. the *false negative* rate,
4. the *minimal marker size*,
5. the *vertex jitter* characteristics,
6. the *marker library size*,
7. *immunity to lighting conditions*,
8. *immunity to occlusion*,
9. *perspective support*,
10. *immunity to photometric calibration*, and
11. the *speed performance*
12. the *pose estimation precision*

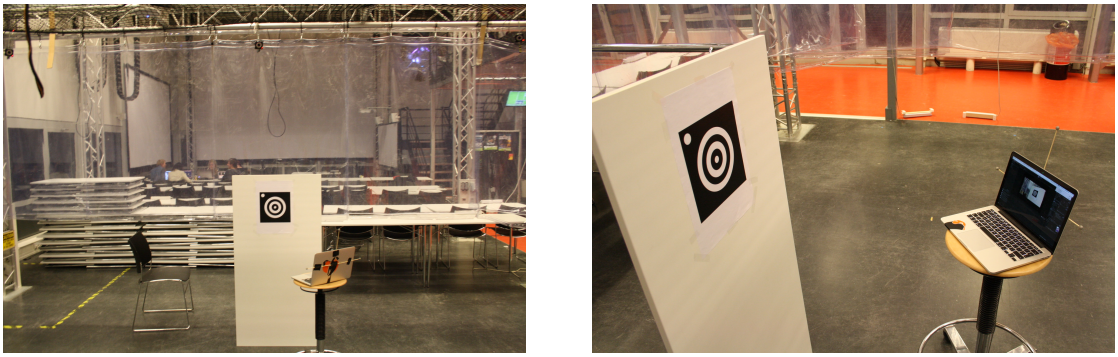


Figure 3.13.: Creation of the testset. The position of the camera was tracked with the optitrack system. OptiTrack markers visible attached to the laptop lid. Circle marker is fixed to a white surface.

In the following I elaborate on these criteria and, where applicable, I describe the method in which they where tested and report the results. Based on the design limitations of having a library of only one marker, we can immediately skip criteria 2 and 6.

A test set is created with actual camera images. Ground truth was ascertained with a professional tracking system - the Naturalpoint OptiTrack[14]. An OptiTrack marker was attached to a laptop, with the center of marker being in the center of the laptop's webcam. The laptop was moved through the capture space on a chair with wheels, swivelling the chair to keeping the marker in view of the camera (see fig. 3.13). The area covered was roughly 400x270cm. In intervals, sample images were taken and the output of the OptiTrack system at that moment was recorded. The precision of the OptiTrack system is reported to be in the *sub-millimeter* range[15]. The resolution of the laptop webcam was 1280x720 pixels.

The results of these evaluations are more thoroughly discussed in the following section (see section 3.6.10).

For all other tests, unless reported otherwise, the following setup was used. A printed circle marker with a size of 25x25cm; Images captured from a Logitech QuickCam®Pro 9000[16] at a resolution of 800x600 pixels; Binary search image used for scaled to 320x240 pixels (factor 0.4); Processing is performed on the Beaglebone ARM board with a 700Mhz CPU and 256MB RAM.

3.6.1. False positives

False positives are markers detected in the scene that are not actually markers, typically due to noise or other structures that resemble the marker that is to be detected.

[14] `http://www.naturalpoint.com/optitrack/`
[15] `https://www.naturalpoint.com/optitrack/about/press/20100727.html`
[16] `http://support.logitech.com/product/3056`

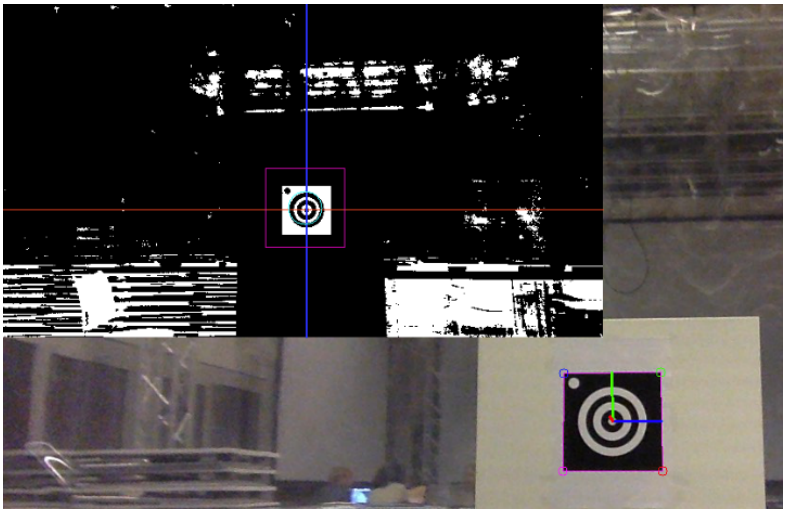


Figure 3.14.: The noisy background in the test set was no problem for the circle marker detection.

In the test set, not a single false positive was detected. The set does contain quite some background noise that is not filtered by the thresholding for the binary image (see section 3.6.1). Additional test with intentionally noisy scenes shown in fig. 3.15 did not produce false positives.

3.6.2. False negative

[6] report the false negative rate of their approach based on theoretical reasoning on the limits implied by their design. Given the prototypical nature of my approach, this is not really feasible at this point.

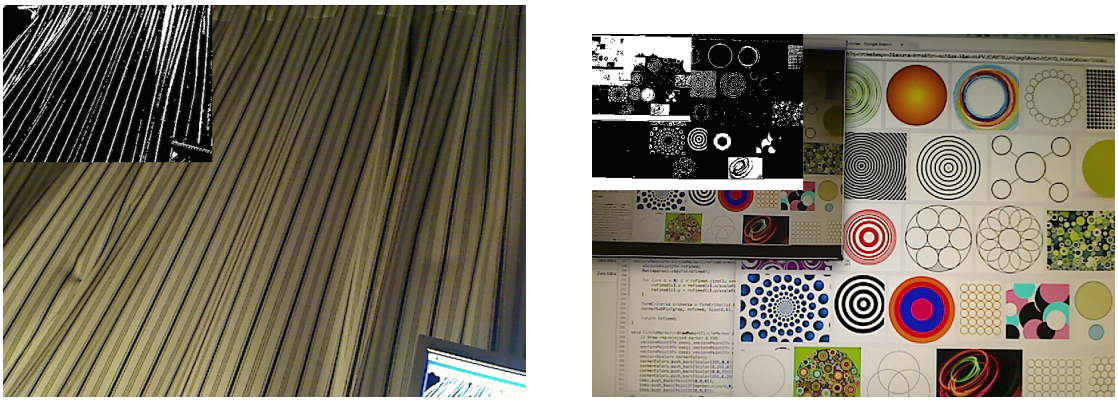


Figure 3.15.: Deliberately noisy areas did not produce false positives.

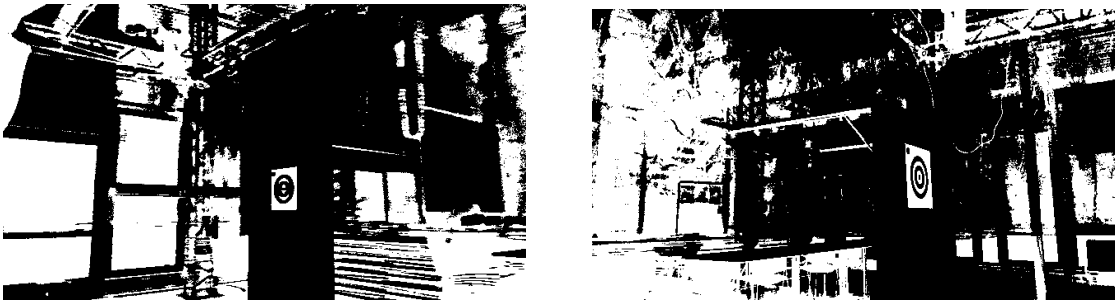


Figure 3.16.: False negatives due to extreme perspectives and motion blur.

Observing the output of my software while moving the camera, it can be seen that there is a high number of false negatives. Most of them introduced due to motion blur or extreme perspectives onto the marker (see fig. 3.16). Other false negatives are a little less obvious, and need further examination.

In the test set (which is also heavily affected by motion blur), we have 74.09% false negative at a search scale of 0.3. At a search scales of 0.5, 0.7 and 1.0, this the rate of false negatives reduces to 43.35%, 39.03%, and 36.27% respectively.

3.6.3. Minimal marker size

Based on the designed state-machine, the theoretical shortest sequence of pixels that is accepted as a marker must have at least 13 pixels wide to reflect the alternating rings as well as the leading and trailing black border. In practice however, this is slightly more, depending also on the perspective. Qualitative testing showed that it at 45 pixels upwards detection becomes stable. [6] actually report 13 pixels as their minimal marker size.

The measure of minimal marker size is useful, as it allows estimations of the maximal physical distance between the camera and the marker - depending on the physical marker size and the camera properties.

3.6.4. Vertex jitter

Vertex jitter is the noise in marker corner positions. This affects the result of the pose estimation directly. [6] report a standard deviation between 0.03 and 0.09 pixels, depending on the camera used. We compute the vertex jitter by calculating the standard deviation of the marker corners over 200 frames. The webcam is stationary about $1.5m$ away from the printed circle marker.

The measured vertex jitter has a standard deviation of 0.11 pixels.

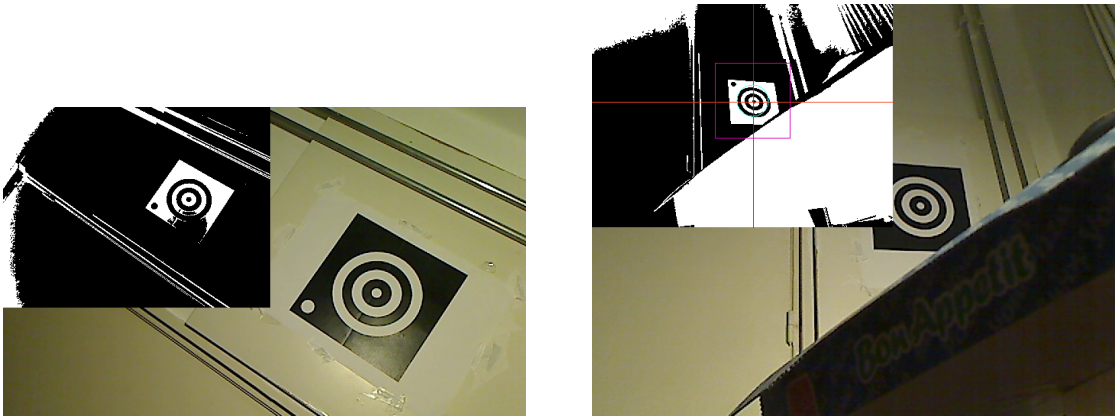


Figure 3.17.: Left: Marker cannot be detected due to reflections. Right: Marker is detected, but corners cannot be approximated because parts of the black square are occluded by a pizza box.

The jitter is in the same order of magnitude as reported in [6]. However, I noticed that the jitter, which is also clearly visible in the debugging visualisation of my application, has effects on the estimated camera pose when the camera is further away from the marker.

For the 25x25cm marker, I observed a standard deviation of the pose estimation of 17mm at a position 75cm away from the marker. At a position 200cm away from the marker, the standard deviation increased to 116mm.

3.6.5. Immunity to lighting conditions & Immunity to occlusion

Our software uses a simple and fast thresholding operation to acquire the binary image used for the detection algorithm. A side effect of this is that the detection is prone to reflectivity and variations in ambient lighting. Bright spots on the marker make it undetectable.

Occlusion is something that was not dealt with, either. Occluding or damaging the rings in the circle markers means that the marker cannot be properly detected under all perspectives. If the black square around the marker is occluded in such a way that it does not have contour with four corners anymore anymore, the marker might still be detected in many cases. The corners, however, will not be detected reliably anymore, resulting in no or wrong estimations (see fig. 3.17).

3.6.6. Perspective support

Naturally, for pose estimation, markers must be detected with some degree of freedom in perspective.

Qualitative analysis show that the marker is well recognised under angles between 30 and 40 degrees onto the orthogonal axis of the marker - depending on the scale of the marker. It should be noted that, due to a shortcoming in the design of the marker, the rotation around the orthogonal axis is somewhat limited. If the white corner spot is straight above the center of the circle marker, false negatives can be observed.

This is because the corner spot is too close to the outer white ring. This causes the leading black area not to be accepted as such by the detection state-machine.

3.6.7. Immunity to photometric calibration

Proper camera calibration is not needed for *detection* of the circle marker, but it is vital to estimate the correct pose of the camera. Other aspects of photometric calibration such as colour accuracy are less relevant, since we deal with binary and grayscale image for detection and refinement of our marker corners only.

3.6.8. Speed performance

I measure the speed performance in terms of the time it takes to process single frames. This test is repeated with different scale settings for the binary image using the test set. Besides the total processing time, we also measure the processing time taken up by the individual sub tasks, namely grabbing the frame from disk or camera, generating the scaled binary image, detecting circles, detecting approximate corners locations, refining corner locations, and estimating the pose. An additional test was performed to compare the speed on a regular computer as well.

The total processing time needed to detect markers and estimate poses can be seen in fig. 3.18. Processing times are shown for the first 100 frames of the test set, and at four different scale settings. Mean processing times for scale=$[0.3, 0.5, 0.7, 1.0]$ were $[72.0, 135.5, 204.5, 419.76]$. Mean processing times against the amount of pixels are shown in fig. 3.19.

Per sub task, at a search scale of 0.3, the percentages of mean processing time over all 578 frames is computed. Grabbing the frame from disk: 73.75%; Generating the scaled binary image: 18.17%; Detecting circles: 7.44%; Detecting approximate corners locations: 0.08%; refining corner locations: 0.01%; and Estimating the pose: 0.56%. All 578 frames were processed in a total of 169.67 seconds, which means per second, 3.4125 frames were processed.

When reading data from the USB webcam (800x600 pixels, search scale=0.3) instead of from disk, an average of 7.14 frames can be processed per second. On a Macbook Pro, the result is 24.4 FPS, with the limiting factor being the refresh rate of the camera (detection and estimation is performed in less than 3ms).

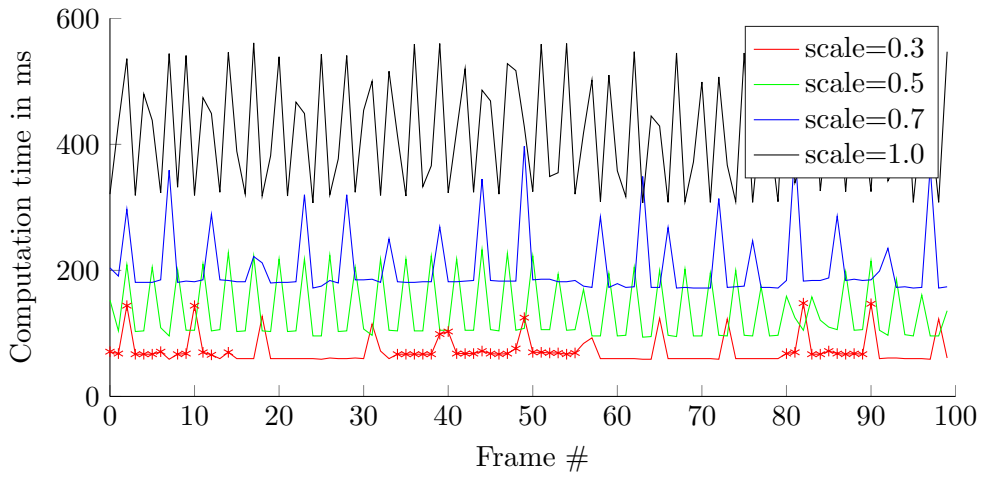


Figure 3.18.: Combined processing time of detection and pose estimation with different scale settings of the binary search image. Sampled from the first 100 frames of the test set. For scale=0.3, those frames have been marked where a circle marker was detected.

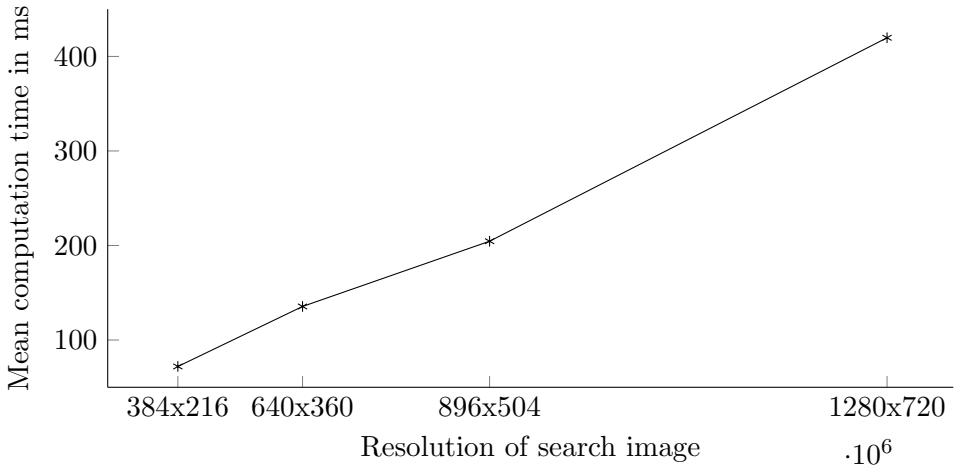


Figure 3.19.: Mean computation time against the resolution of the search image.

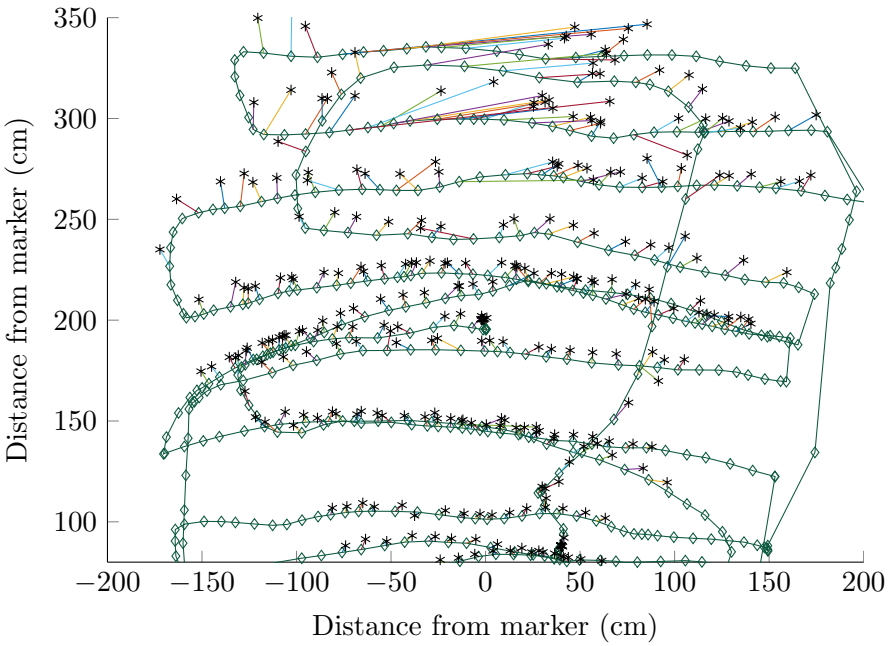


Figure 3.20.: .

Latency of the system has not been calculated. It can be seen that it is most caused in image acquisition, and depends on the used platform.

3.6.9. Pose estimation precision

We measure the error in the estimations made on the test set, using the data from the OptiTrack as ground truth. With a search scale of 0.5, in 328 of the 579 frames the marker was detected and a position could be estimated. The overall mean error was 12.40cm. Looking at those estimations that are between 80-200cm away from the target, the mean error is 5.24cm. This test is visualised in fig. 3.20.

One last observation was that the detection of the white corner spot was successful whenever the four corners of the marker were also detected. There were no incorrect orientations of the marker detected.

3.6.10. Discussion of the Results

The implementation was tested according to various criteria in the previous sections. While some observations have already been briefly discussed there, I would like to address some of them in more detail in this section.

False negatives We have seen that the detection algorithm would not detect the circle marker in a high percentage of the images in the test set. Increasing the scale of the search images improved this somewhat. Based on observation, many of the false negatives could, however, be attributed to motion blur. At small scales, the rings in the circle marker are especially likely to vanish under motion blur, as seen in fig. 3.16. While this means that the marker design does exactly not attribute towards robustness against motion blur, improvement should also be sought in better cameras. Especially those that expose settings such as exposure time to the user.

Speed performance The overall processing speeds where positive. The detection algorithm was designed to perform with linear complexity in terms of the amount of pixels, which could be confirmed in fig. 3.19. On average, our detection and estimation ran at more that twice the speed of the ArUco library tested insection 3.3.2 - although the initial desired performance of 10Hz was not met. It is remarkable that most of the time in the application is spent reading the images from disk or from the camera. On a different note, there where some irregularities detected. On a frame by frame basis, the computation times would show peaks (see fig. 3.18).

Spikes in processing It seems unlikely that the spikes are caused by instabilities in the circle detection algorithm, for example due to noisy conditions, given periodic nature of the spikes. There also does not seem to be a correlation between the spikes and wether or not a marker was actually detected. A suspicion was that spikes were due to background tasks on the computer, however, since spikes become denser with higher scale rates, it seems more likely to be related to memory access, either due to a bug in the code or other limitations of the ARM computer. This is something that would need to be examined in more detail.

Error in pose estimation The pose estimation errors are rather high at $5 - 12$cm, depending on the distance. Some of this error must can be explained with a bias in the ground truth - there must have been some offset between the OptiTrack marker and the camera location. On the other hand, the overall magnitude of this error is nicely explained by the effect of the vertex jitter on the pose estimation (see section 3.6.4), which was between 1cm at small distances, and 12 at bigger distances. This is not directly a part of the marker *detection*, but rather of the approach on pose estimation. Four corners is the absolute minimum for 6DOF pose estimation. A marker design with more than four corners that would making the overall pose estimation less sensitive to noise in individual corners detection, resulting in better estimations also at smaller scales (i.e. due to higher distances).

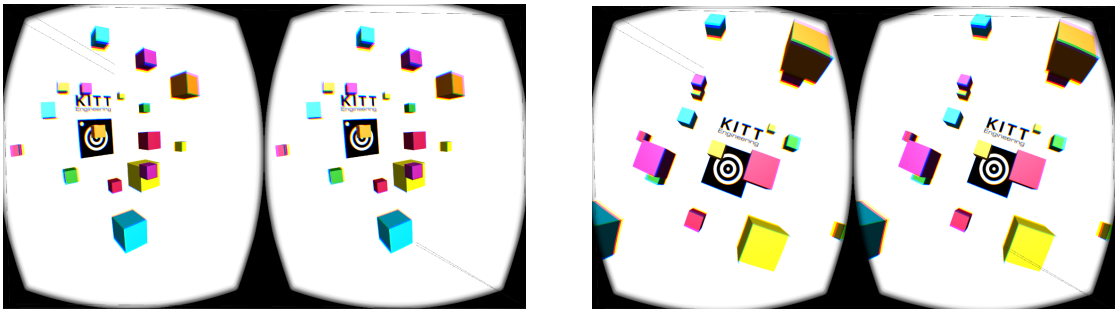


Figure 3.21.: Demo Application rendering stereoscopic images of the scene. Camera perspective is changed according to the result of our pose estimation software.

3.7. Final Demo

As a final demonstration of the results of my work during this internship, the prototype was extended with with a simple UDP socket. This way, other applications could read the results of the pose estimation. The position is given in three parameters, (x, y, z). For the rotation, we calculate and give three euler-angles (rx, ry, rz) from the 3×3 rotation matrix handled internally.

A Virtual Reality application was built to consume the pose information. The result is shown in fig. 3.21. The 3D scene contains some freely placed cubes of various colours, a representation of the circle marker, and the KITT Engineering logo. The camera perspective is adjusted according to the result received from the pose estimation software. The output is rendered in stereoscopic images, suitable for the Oculus Rift HMD.

The VR application should run on a regular computer to which the Oculus Rift is connected. The tracking software is shipped using the cross-platform build system *CMake*[17] and was successfully compiled on all major platforms (Mac/OS X, PC/Windows & Linux, ARM/Linux). The VR application makes requests to a configured IP and port of the tracking software - on the same computer, or on a different computer in a local network.

I should note at this point that the experience does not yet compare to a professional tracking system. Especially the bad performance when moving the camera is irritating, and the high position estimation error is clearly noticeable. When using a low-power computer to perform the pose estimation, the higher latency and lower frame-rate do also not favourably attribute to the experience.

[17] `http://www.cmake.org/`

4. Conclusion

In the course of this internship, I worked towards building a vision based inside-out tracking system for embedded computers. After designing the system, possible approaches were evaluated. The choice was to use a*fiducial marker* based approach. Existing fiducial marker systems would exhibit performance bottlenecks in the detection of such markers. A custom marker was designed with properties that would afford a faster detection method. The detection algorithm was implemented. A prototype of the inside-out tracking system was built using the custom marker. This prototype was evaluated under various criteria relevant to fiducial marker systems. Good results were obtained in the criteria of speed performance over the previously evaluated systems. Some aspects such as unique marker IDs were not part of this prototype and were not evaluated. Particularly problematic for the application is the high error in pose estimation, especially at bigger distances. Another concern is the high ratio of false negatives caused by motion blur due to a moving camera. The ratio of false positives on the other hand is extremely low.

Finally, a demo was built to put the prototype of the system to the intended use. This demo works as one would expect given the limitations of the prototype.

I would conclude that the overall goal of building a vision based inside-out tracking system was reached, although the prototype does not entirely implement the system as originally (and ambitiously) designed. At one point, a conscious choice was made to focus on the development of one aspect of the system, and make it perform well. This made some aspects of the original system impossible to implement, as they relied on features that were, for sake of time, not included to the custom solution. This was namely the feature of having uniquely identifiable markers. As mentioned, also the more specific goals of the custom solution were met. The circle marker approach shows promising performance in terms of processing speed, and it seems to outperform other solutions under this criterion.

4.1. Recommendations

Before we discuss the general recommendations on how best to proceed with building an inside-out tracking system, I first want to give some recommendations about improving the current circle marker approach.

As noted, it can be expected that using more than four detectable corners for pose

estimation improves the precision in pose estimation. More corners could be introduced by changing the black square to another shape around the circle marker. This should to some extend be possible without compromising the circle marker detection. One could also combine this approach with the implementation of an ID system. It was shown that we could reliably interpolate between the four detected corner points. This could be used to also look for information outside the area of the marker. This could, for example, be done in the form of smaller dots hat represent a binary code. These dots could serve as additional corners for the pose estimation.

Depending on the achievable precision, filtering the results should be considered.

As far as the performance goes, a first step is to move the system back to COMs with integrated image signal processor such as the Gumstix. Better performance is expected due to the faster pipeline from camera to decoded pixel data in memory. Not only does this mean that frames can be computed at a higher refresh rate - if the processing can keep up, also the latency of the system will be reduced.

A worthwhile effort would be the creation of additional test cases with ground truth. Not only to measure the effects of changes made, but also to compare them against other systems. For this, the various different markers would have to be included into the test scene as well.

On a similar note, I would also recommend to design such a system in a general way, such that components as marker detection and pose estimation can be replaced by different implementations. My original system design in gives a good starting point.

Lastly, while I found OpenCV to be a suitable library for the prototype I built, there are other image processing libraries around, such as TI's IMGLIB[1]. Although typically with a limited set of features, these libraries are optimised for their specific target platform, and may even have support for outsourcing calculations to the DSP.

4.2. Personal Reflection

I thoroughly enjoyed my internship at KITT Engineering. This can be attributed to both the friendly and helpful atmosphere inside the company, as well as to the interesting and challenging assignment. Many of the expectations I had prior to the internship where met.

Wide insight in working with embedded devices was gained. Both in the 'mundane' activities of configuring and using an embedded Linux operating system, as well as the more advanced topics such as using ISPs and custom kernel modules.

I further acquired knowledge about the limitations of these devices while developing my own software. In particular, I developed a piece of computer vision software that

[1] `http://www.ti.com/tool/sprc264`

runs with acceptable performance, and outperforms (at least in some aspects) generic solutions that were designed for other platforms. On the other hand, I have liked to gain some experience in actually optimising existing code on a lower level, or even make use of the other processing units such as the DSP to improve the performance of the system. Looking back now, this however would have been far beyond my abilities, anyway.

Another learning objective was to gain practical insight and experience in developing computer vision applications. This objective was certainly fulfilled. The implemented application was advanced and went way beyond the reuse of existing solutions - which I was accustomed to in my previous computer vision related debaucheries.

The work done in the context of this internship was certainly different from my studies at university. While a part of the assignment was to perform research towards the design and implementation, I did focus more on an actual application and development based on the acquired knowledge. And I followed through with the implementation.

This reflection would be incomplete without addressing the fact that the finalisation date of the internship in general, and of this report in particular, had suffered under other university-related activities that I chose to follow through in parallel. Although it was in agreement with my supervisors, I would not consider this best practice. Finalising in a more concise and dedicated period would have been preferable to me. I am confident that this realisation as well the other good experiences made during my time at KITT Engineering will be of benefit in my academic and professional future.

4.3. Acknowledgements

Thanks to Ronald for is knowledgable advice and feedback on computer vision matters. Thanks to everybody at KITT for making my time at their company so enjoyable and for all the cups of powdered soup I consumed. Special thanks to my supervisor Peter for his inspiring advice and for sharing his expertise with me, to Thomas for all his support and arrangements, and of course to Andries for facilitating this interesting assignment in the first place.

Bibliography

[1] Fakhreddine Ababsa and M Mallem. A robust circular fiducial detection technique and real-time 3d camera tracking. *Journal of Multimedia*, 3(4):34–41, 2008.

[2] Markus Achtelik, Abraham Bachrach, Ruijie He, Samuel Prentice, and Nicholas Roy. Stereo Vision and Laser Odometry for Autonomous Helicopters in GPS-denied Indoor Environments. *SPIE Defense, Security, and Sensing. International Society for Optics and Photonics*, 7332:733219–733219–10, May 2009.

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[4] Filippo Bergamasco and Andrea Albarelli. RUNE-Tag: A high accuracy fiducial marker with strong occlusion resilience. *Computer Vision and Pattern Recognition*, 2011.

[5] Wilhelm Burger and Bir Bhanu. Estimating 3D egomotion from perspective image sequence. *IEEE transactions on pattern analysis and machine intelligence*, 12(11), 1990.

[6] Mark Fiala. Designing highly reliable fiducial markers. *IEEE transactions on pattern analysis and machine intelligence*, 32(7):1317–24, July 2010.

[7] S. Garrido-Jurado, R. Muñoz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, June 2014.

[8] LB Gatrell, WA Hoff, and CW Sklair. Robust Image Features: Concentric Contrasting Circles and Their Image Extraction. *Robotics-DL tentative. International Society for Optics and Photonics*, 1992.

[9] Martin Kaltenbrunner and Ross Bencina. reacTIVision: a computer-vision framework for table-based tangible interaction. *Proceedings of the 1st international conference on Tangible and embedded interaction*, 2007.

[10] Johannes Köhler, Alain Pagani, and Didier Stricker. Detection and identification techniques for markers used in computer vision. *OASIcs-OpenAccess Series in Informatics*, 19:36–44, 2011.

[11] Vincent Lepetit and Pascal Fua. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and trends in computer graphics and vision*, 1(1):1–89, 2005.

[12] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[13] L. Naimark and E. Foxlin. Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. *Proceedings. International Symposium on Mixed and Augmented Reality*, pages 27–36, 2002.

[14] D Nistér, O Naroditsky, and James Bergen. Visual odometry. *Computer Vision and Pattern Recognition*, 00(C), 2004.

[15] Paul Robinette and Thomas R. Collins. Feature detection and SLAM on embedded processors for micro-robot navigation. *SPIE Defense, Security, and Sensing. International Society for Optics and Photonics*, page 874106, May 2013.

[16] Zhengyou Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE*, 1998, 2000.

A. Circle Marker 80x80mm