*Computer Vision Winter Workshop 2007, Michael Grabner, Helmut Grabner (eds.)*
*St. Lambrecht, Austria, February 6–8*
*Graz Technical University*

# ARToolKitPlus for Pose Tracking on Mobile Devices

Daniel Wagner and Dieter Schmalstieg

Institute for Computer Graphics and Vision, Graz University of Technology
{ wagner, schmalstieg } @ icg.tu-graz.ac.at

**Abstract** *In this paper we present ARToolKitPlus, a successor to the popular ARToolKit pose tracking library. ARToolKitPlus has been optimized and extended for the usage on mobile devices such as smartphones, PDAs and Ultra Mobile PCs (UMPCs). We explain the need and specific requirements of pose tracking on mobile devices and how we met those requirements. To prove the applicability we performed an extensive benchmark series on a broad range of off-the-shelf handhelds.*

## 1 Introduction

Augmented Reality (AR) and Virtual Reality (VR) require real-time and accurate 6DOF pose tracking of devices such as head-mounted displays, tangible interface objects, etc. Pose tracking must be inexpensive, work robustly in changing environmental conditions, support a large working volume and provide automatic localization in global coordinates. A guaranteed level of accuracy on the other hand is usually not required.

Solutions that fail to address any of these requirements are not useful for VR and AR applications. In particular for mobile AR applications, all the requirements must be met while working with very constrained technical resources. The typical mobile AR configuration involves a single consumer-grade camera mounted at a head-worn or handheld device. The video stream from the camera is simultaneously used as a video background and for pose tracking of the camera relative to the environment. This inside-out pose tracking needs to execute in real-time with the limited computational resources of a mobile device.

Tracking fiducial markers is a common strategy to achieve robustness and computational efficiency simultaneously. While the visual clutter resulting from the fiducial markers is undesirable, the deployment of black-and-white printed markers is inexpensive and quicker than accurate off-line surveying of the natural environment. By encoding unique identifiers in the marker, a large number of unique locations or objects can be tagged efficiently. These fundamental advantages have lead to a proliferation of marker-based pose tracking despite significant advances in pose tracking from natural features.

In this paper we focus on ARToolKitPlus, an open source marker tracking library designed as a successor to the extremely popular open source library ARToolKit [4]. ARToolKitPlus is unique in that it performs extremely well across a wide range of inexpensive devices, in particular ultra-mobile PCs (UMPCs), personal digital assistants (PDAs) and smartphones (see Figure 1).

The market share of smartphones with cameras which are able to execute AR applications is growing rapidly, predicted to reach one billion by 2012[1]. We therefore believe it is timely to deploy marker tracking on mobile devices, laying the foundation for mass-marketed AR. This paper describes the design requirements and lessons learned from developing such a state of the art library.

## 2 Related work

One of the first projects using camera-based 6DOF tracking of artificial 2D markers was Rekimoto's 2D Matrix Code [6] in 1996. It pioneered the use of a square planar shape for pose estimation and an embedded 2D barcode pattern for distinguishing markers. In 1999 Kato used a similar approach to develop ARToolKit [4], which was released under the GPL license and therefore became

[1] Canalys report from Nov. 2006, http://www.canalys.com/

**Figure 1:** Devices for handheld Augmented Reality: UMPC, PDA and smartphone.

enormously popular among AR researchers and enthusiasts alike. Since then, many similar square tracking libraries have emerged among which the most prominent ones are ARTag [1], Cybercode [7], the SCR marker system [11] and the IGD marker system used in the Arvika project [2].

ARToolKit is the basis for several projects concentrating on 6DOF tracking on handheld devices. The first port of ARToolKit to Windows CE led to the first self-contained handheld AR application [10] in 2003. This work evolved later into ARToolKitPlus which is described in this paper. In 2005 Henrysson [3] created a Symbian port of ARToolKit partially based on the ARToolKitPlus source code.

Other researchers tried making best use of the restricted resources of low to mid-range mobile phones by using simpler models with very restricted tracking accuracy. In 2004 Möhring [5] created a tracking solution for mobile phones that tracks color-coded 3D markers. At the same time Rohs created the Visual Code system for smartphones [8]. Both techniques provide only simple tracking in terms of position on the screen, rotation and a very coarse distance measure.

## 3 Background: marker tracking with ARToolKit

Tracking rectangular fiducial markers is today one of the most widely used tracking solutions for video see-through Augmented Reality applications. In this section, we outline the basic principle of the original ARToolKit, which was designed to run only on standard PCs. More details can be found in [4].

Before camera-based 6DOF tracking can be performed, the camera must be calibrated once as a pre-processing step. The results of this step are a perspective projection matrix as well as the image distortion parameters of the camera. Both are saved in a calibration file that is loaded later on during the start-up phase of the tracking system.

The basic workflow of ARToolKit at run-time is outlined in Figure 2: A camera equipped device (left top picture) reads a video stream which is rendered as a video background to generate a see-through effect on the display. The camera image is forwarded to the tracking system (middle top) which applies an edge detection operation as a first step. ARToolKit performs a very simple edge detection by thresholding the complete image with a constant value (right top picture), followed by a search for quadrangles. Resulting areas being either too large or too small are immediately rejected. Next the interior areas of the remaining quadrangles are normalized using a perspective transformation. The resulting sub-images are then checked against the set of known patterns.

When a pattern is detected, ARToolKit uses the marker's edges for a first, coarse pose detection. In the next step the rotation part of the estimated pose is refined iteratively using matrix fitting. The resulting pose matrix defines a transformation from the camera plane to a local coordinate system in the centre of the marker (see bottom right picture in Figure 2). An application can use these matrices for rendering 3D objects accurately on top of fiducial markers. The final image is displayed on the device's screen (bottom left picture).

ARToolKit can combine several co-planar markers into a multi-marker set. From an application point of view this multi-marker set is treated as a single marker and can be tracked as long as one or more markers of this set are visible. Multi-marker tracking increases the computational load but results in considerably more accurate and robust tracking.

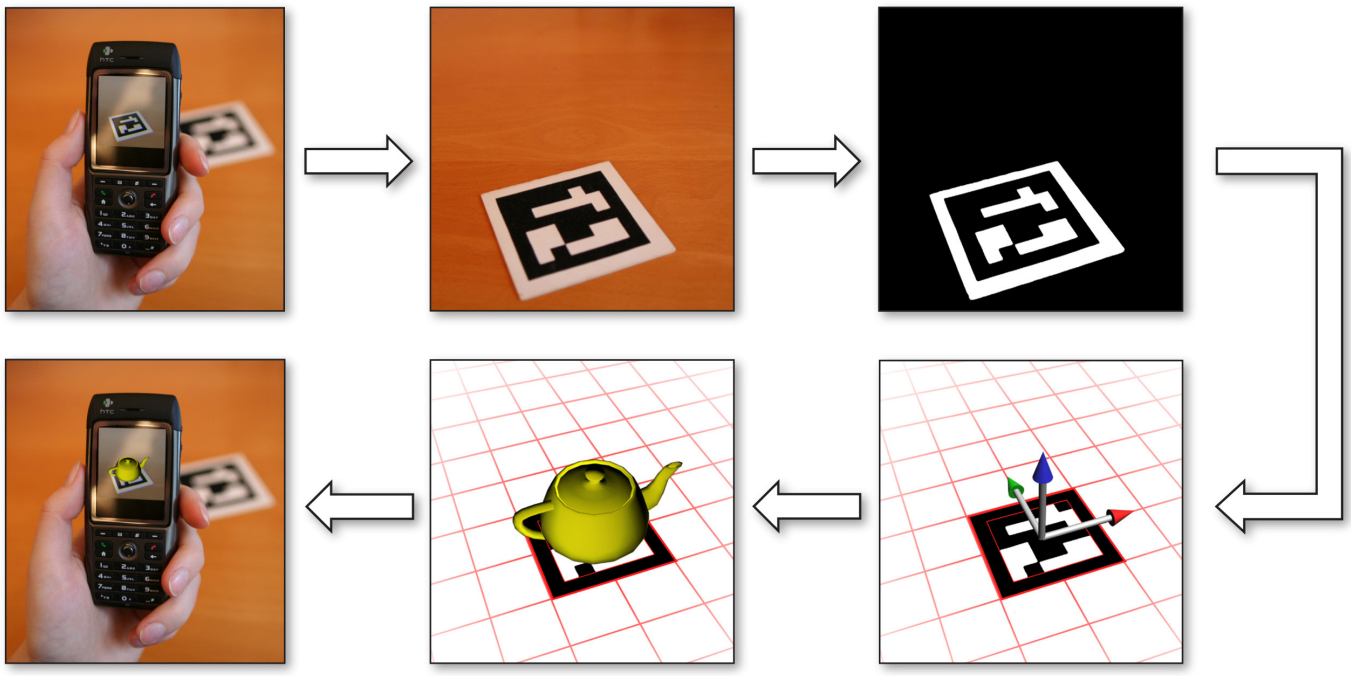

**Figure 2:** Basic workflow of an AR application using fiducial marker tracking.

# 4 Requirements for handheld tracking

Tracking on handheld devices such as PDAs or mobile phones enforces many restrictions that are not present on stationary or mobile PC-based setups. Attaching external sensors is usually not possible since these devices are typically too small and also do not expose the required hardware interfaces. Moreover, requiring external sensors defeats the objective of an off-the-shelf inexpensive solution. This leaves camera-based tracking as the only option. Consequently, the pose tracking software has to be specifically designed to run on these restricted platforms.

## 4.1 Hardware

Although the general hardware concepts are similar, a mobile phone's or PDA's hardware configuration is in many ways very different from standard PC hardware.

Most mobile phones use ARM compatible CPUs running at a rate of 100-600 MHz, which are primarily optimized for low power consumption. Despite a strong interest in multimedia applications, these CPUs usually do not have floating-point units (FPUs). Instead, floating-point operations are emulated in software which makes them about 50 times slower than integer operations. Even integer divisions are generally emulated in software. Due to size, cost and power restrictions, mobile phone CPUs usually do not possess parallel execution units. Thus there is no multi-core or hyper-threading technology, and designs with a single arithmetic/logical unit (ALU) are prevalent.

Memory is a scarce resource on mobile phones. To reduce costs and conserve battery power, mobile phones possess as little memory as possible. High end devices typically have up to 64 Mbytes of RAM, but the limited operating system design often constrains applications to use no more than a few Mbytes. To conserve even more power, memory access is very slow, and every cache-miss stalls the CPU considerably.

Most mobile phones today include built-in cameras, but image quality of these cameras is usually poor compared to PC-based cameras used for computer vision. While today's mobile phones can record still pictures at megapixel resolutions, internal bandwidth limits video to typical resolutions of 320x240 or even 160x120 pixels. In order to reduce costs and size, most mobile phones use tiny, low-quality lenses, which can cause strong vignetting (radial luminance fall-off to the corners, see left image in Figure 4). Finally, most cameras provide the video stream in proprietary or unusual pixel formats such as YUV12.

## 4.2 Software

Most mobile phones today use proprietary, closed operating systems and are not accessible to system-level programming. Among the open, programmable operating systems, the most important ones are Symbian, Linux and Windows CE. While these three platforms are fully programmable, they are mutually incompatible, which makes the design of cross-platform software extremely challenging. Even within multiple device models supporting the same operating system, small incompatibilities, in particular concerning low-level hardware features prevent an acceptable level of binary compatibility, and frequently require recompilation even after minor code modifications.

# 5 ARToolKitPlus

For the last 3 years have we developed ARToolKitPlus, a tracking library specifically targeted at mobile devices. Although originally based on ARToolKit, the current version of ARToolKitPlus shares almost no code with its ancestor ARToolKit. To our knowledge ARToolKitPlus is unique on mobile devices in terms of performance, supported platforms and features. Sections 5.1 to 5.10 describe in detail how the aforementioned restrictions are met to make ARToolKitPlus a practical tracking solution on handheld devices. ARToolKitPlus is freely available[2] under the GPL open source license.

## 5.1 Fixed point

The lack of an FPU is probably the single, most important issue for floating point intensive software on mobile phones and PDAs. To determine the time spent on floating point operations, custom code instrumentation was applied to reveal the most prominent bottlenecks. Tests showed that floating-point usage slowed down especially the pose estimation part of ARToolKit on mobile devices. Replacing the native C float data-type with a system-wide C++ class (emulating all operations with fixed point arithmetic) failed due to strongly varying requirements on precision and numeric range along the pipeline. Instead many functions had to be re-implemented using hand-written fixed point code after determining local range and precision requirements.

## 5.2 Pixel formats

Supporting the native pixel formats of phone cameras is crucial for high performance tracking. Converting to a common format costs too much performance, especially due to the severe memory bandwidth limitations on these devices. Some camera formats already provide data in a format that is ideal for tracking, such as the YUV12 format common on phones. YUV12 stores luminance (Y) at full resolution (8-bits), followed by two chrominance components (UV) at half resolution (effectively 2-bits each). Naturally 8-bit luminance images provide a suitable format for pose tracking from back-and-white markers while minimizing memory footprint. In contrast, formats such as RGB565 require the use of lookup tables for fast format conversion.

## 5.3 Id-markers

ARToolKit uses template matching of the 2D image surrounded by a black border after quantizing the image to a matrix of programmer-specified resolution. Unfortunately this restricts the amount of concurrently usable markers at run-time considerably due to increasing

[2] http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php

costs of searching in the image database: N visible and M known markers require 4*M*N template matching operations. ARToolKitPlus introduced binary marker patterns (similar to those of ARTag) which implicitly encode the marker's id with built-in forward error correction (CRC). Detection of id-markers is always faster than for template-markers since no image matching is required. Currently ARToolKitPlus supports up to 4096 id-markers. More markers could be supported at the cost of decreasing the id-detection robustness.

Id markers offer several more advantages over template markers (besides better performance): Although ARToolKit allows the user to choose almost any image for marker patterns, most users still choose their patterns out the small set of markers that comes with the ARToolKit distribution. With id-markers, the user does not have to provide marker images, but can freely choose any marker from a fixed set of 4096 patterns. In contrast to template markers, the user is not required to train ARToolKitPlus with new patterns since any valid marker is implicitly known to the system. The encoded id is highly redundant and is therefore robust against 90° rotation steps, which is a natural problem with square template markers.

### 5.4 Automatic thresholding

In stationary setups lighting can often be controlled to provide well balanced brightness throughout the complete environment of interest. In mobile setups, which can easily span several rooms, floors or even combined indoor/outdoor areas, tracking must adapt to changing lighting conditions. Although many cameras possess auto-gain features today, the final image brightness can still vary heavily which causes severe problems with constant threshold values. Global thresholding, the typical solution for this problem, is computationally too expensive and therefore not suitable for embedded platforms.

Instead ARToolKitPlus includes a simple, yet very effective heuristic for automatic thresholding (see Figure 3) which imposes no measurable performance loss. Instead of looking at the whole image, only the last seen marker is considered. After a marker was found, the median of all extracted marker pixels is calculated and used as a threshold for the next image to process. If the heuristic fails because no marker is found, ARToolKitPlus randomizes the threshold in such a case for every new frame until a new marker is detected. Empirical tests show that after a marker gets lost it takes only a few frames to find a new, working threshold.

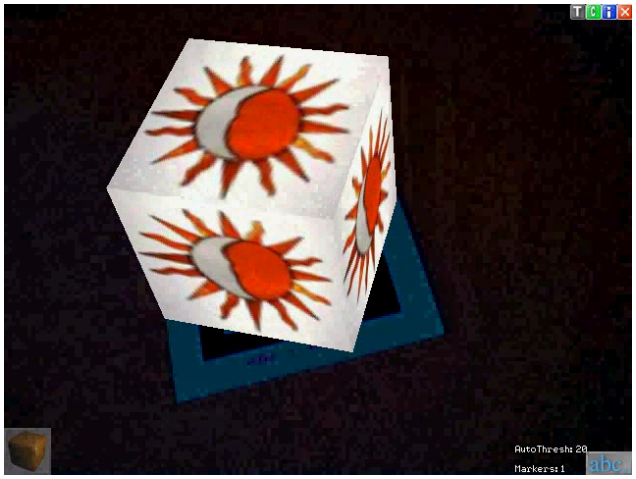

**Figure 3:** Automatic thresholding for tracking in extremely dark environments.

### 5.5 Vignetting

Some cameras in mobile phones today exhibit strong vignetting (see left image in Figure 4). Thresholding such an image with an image-wide constant value results in an image as can be seen in the middle picture of Figure 4. If a marker is close to the border in such an image, it will overlap with the dark areas that were classified as black and the marker would therefore not be detected anymore. To prevent this, ARToolKitPlus provides a simple vignetting compensation feature: The user can specify a radial fall-off from the centre of the image to the corners. This fall-off is specified numerically rather than using an image mask in order to minimize memory bandwidth usage. After activating vignetting compensation even strongly tampered images are thresholded correctly (see right picture in Figure 4). Vignetting compensation adds only a minimal performance penalty.

### 5.6 Camera calibration

Precise camera calibration is crucial for accurate 6DOF optical pose tracking. ARToolKitPlus supports the original

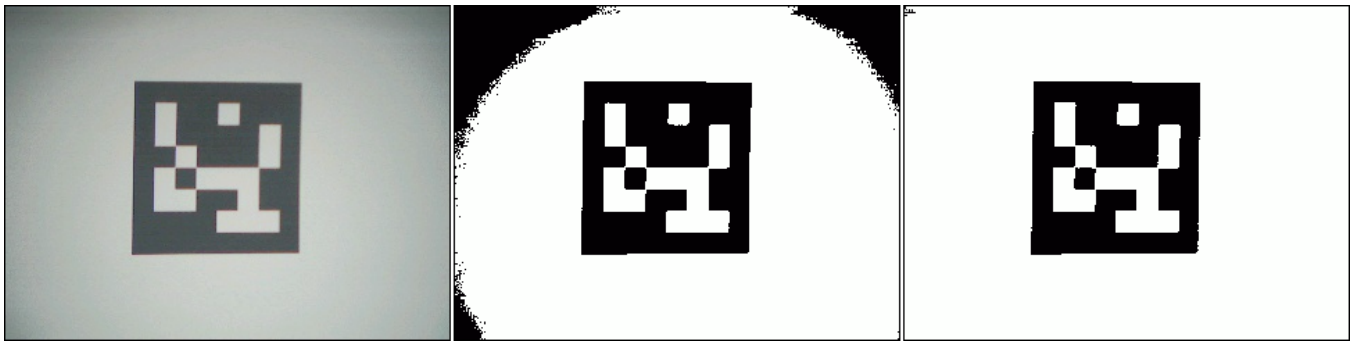

**Figure 4:** Vignetting. Left: original camera image. Middle: constant thresholding. Right: thresholding with vignetting compensation.

ARToolKit calibration tool as well as the more accurate and more convenient to use GML MatLab Camera Calibration Toolbox[3]. To calibrate a mobile phone's camera, the user has to take several pictures of a checker board (see Figure 5), which allows then semi-automatic camera calibration on the PC.

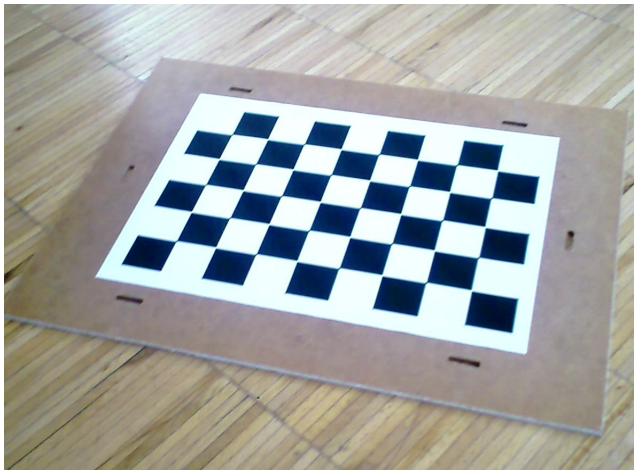

**Figure 5:** Checker board for semi-automatic camera calibration using the MATLAB camera calibration toolbox.

Precise lens undistortion is usually computationally expensive. ARToolKitPlus can use a lookup table to speedup this process. Since even the generation of this lookup table can take up to 10 seconds on low-end devices, it can be cached using the phone's file storage. When ARToolKitPlus searches for the cached lookup table at start-up, it either loads it or automatically creates and stores it for the next start-up.

### 5.7 Portability

As already mentioned in section 4.2, today's mobile phones run a wide variety of system software. Hence, portability is of high concern. ARToolKitPlus does not include code for camera access or 3D rendering. Its only interfaces for data I/O are a pixel buffer for image input and 4x4 floating point matrices (compatible with the OpenGL matrix format) for tracking results output. It is therefore only limited by the amount of supported input pixel formats. ARToolKitPlus is implemented in pure C++ and is consequently highly portable. Currently Windows XP, Windows CE, Symbian and Linux are supported which covers the majority of today's development and target devices. While a Java port would extend the range of supported mobile devices considerably, our informal experiments have shown that the performance of Java on today's mobile phones does not allow interactive frame rates.

[3] GML MatLab Camera Calibration Toolbox: http://research.graphicon.ru/calibration/gml-matlab-camera-calibration-toolbox.html

### 5.8 Custom memory management

Memory is not just a scarce hardware resource on mobile devices but often restricted even further due to deficiencies in mobile operating systems. It is therefore crucial to provide application developers with maximum control over memory de/allocation. Hence all memory management in ARToolKitPlus can be customized by the user. On most platforms ARToolKitPlus uses the standard memory de/allocation functions per default. On Windows CE ARToolKitPlus' memory manager allocates memory outside the process' memory slot thereby keeping this scarce resource available for DLLs and unmanaged memory allocations. Since ARToolKitPlus' memory footprint is fix and known at compile-time, the requirements for such a custom memory manager are minimal.

### 5.9 Optimizations

Over the years, many optimizations were applied to ARToolKitPlus. Besides rewriting major parts of floating point intensive code with fixed point counterparts, ARToolKitPlus makes heavy usage of inline expansion and pre-processor techniques. E.g. we use the pre-processor to generate separate functions for each supported pixel format, which allows switching between those formats at runtime with no performance penalty.

ARToolKitPlus uses lookup tables wherever possible. The pose estimation algorithm intensively uses trigonometric functions that were accelerated with sine and cosine lookup tables. The lens undistortion method of ARToolKitPlus is specified using higher order polynomials which introduce high computational costs at runtime and were therefore replaced by a lookup table too. Matrix fitting requires perspective projection, including (fixed point) divisions which are not implemented in hardware on most ARM CPUs. Replacing these divisions with another lookup table resulted in further significant speedups.

Lookup tables can usually not provide the same exact results as algorithmic methods. Special care was taken to always provide enough accuracy so that final results are indistinguishable.

### 5.10 Robust Planar Pose Tracking

Although the focus of ARToolKitPlus is on mobile devices, improvements for desktop-based setups were added too. The "Robust Planar Pose Tracking" (RPP) algorithm [9] by Schweighofer et al. provides improved pose estimation quality with less jitter and improved robustness. RPP takes into account that two local minima exist for the pose estimation error function and specifically deals with these two errors to always find the optimal solution.

The RPP algorithm was ported to C++ and added to ARToolKitPlus' set of pose estimators, running well on standard PCs. Unfortunately, due to the high numerical precision requirements of this algorithm, a fixed point port suitable for mobile devices is currently not feasible.

## 6 ARToolKitPlus data flow

ARToolKitPlus can be seen as a data flow system where image data enter on one side and pose matrices exit on the other end. On its way data passes through several sub-sections which can be configured independently by the user at start-up. Figure 6 shows a simplified version of this data flow. In each section exactly one algorithm is executed.

Pixel data can be passed into ARToolKitPlus in six different formats ranging from 32-bit formats such as RGBA over 24-bit and 16-bit RGB formats down to 8-bit greyscale. Per default, pixel data is thresholded with a constant value that remains unchanged as long as the user does not change it manually. Alternatively ARToolKitPlus can use its automatic thresholding feature to select a better threshold value for the next image. Both variants can be combined with the vignetting compensation algorithm (see Figure 4).

ARToolKitPlus supports three types of marker detection: Template matching compares the marker's interior area against images in the internal database loaded at start-up. Alternatively users can activate one of two ID-based marker detection algorithms: While "Simple ID" supports only up to 512 markers, the more advanced BCH encoding allows up to 4096 markers.

Next, all detected markers are undistorted. The distortion parameter evaluation can either be performed for all pixels of the markers' edges at runtime or using a look-up table that is created or loaded at start-up. Users can also select to bypass the undistortion at all which should only be done if the image itself is undistorted before it is passed to ARToolKitPlus.

In the final step the poses of all valid markers are estimated. In the default settings ARToolKitPlus uses floating point code for the standard single- and multi-marker tracking algorithms on PCs and the fixed-point counter parts on embedded devices. The Robust Planar Pose estimator is only available in floating point and therefore not suitable for embedded platforms.

## 7 Performance

To test ARToolKitPlus' performance for practical applicability, we benchmarked it on several handheld devices. ARToolKitPlus is primarily CPU bound. So even though all these devices run Windows CE, they represent a good overview of what is currently available on the market. Additionally we ran the benchmarks on a PC as a comparison of the processing power on handhelds to a typical PC-based setup. Since several of these devices are available under different brands, we also list the OEMs' code names. All builds were created using the Microsoft ARM and x86 compilers with full optimization activated (/Ox). Where possible we used the Intel compiler suite to compare different compilers. Benchmarks were performed on the following devices:

- **i-mate SP5** (codename HTC Tornado) is a typical smartphone with a 200 MHz Texas Instruments OMAP850 CPU.
- **HTC MTeoR** (codename HTC Breeze) is a fast Smartphone device with a 300MHz Samsung S3C2442 CPU.
- **HTC TyTN** (codename HTC Hermes) is a PocketPC phone with a 400MHz Samsung S3C2442 CPU.
- **Gizmondo** is a mobile gaming console with an nVidia GoForce 4500 3D chip (not used in the benchmark), a built-in camera and a Samsung S3C2440 400 MHz CPU.
- **T-Mobile MDA Pro** (codename HTC Universal) is a high-end PocketPC phone with an Intel XScale PXA270 CPU running at 520MHz.
- **Dell Axim X51v** is a high-end PocketPC PDA with an Intel 2700G 3D chip (not used in the benchmark) and an Intel XScale PXA270 CPU running at 624MHz.
- **Intel 2 GHz Core Duo** represents a standard PC-based setup. On this device we ran ARToolKitPlus with regular floating point code.

We tested three different scenarios: one using single marker tracking and two using multi-marker tracking. In ARToolKit and ARToolKitPlus multi-marker tracking is implemented by first tracking all markers separately, then combining all tracking results and finally optimizing for the complete set. Because of the last step, tracking a multi-marker set with N visible markers is considerably slower than tracking N independent markers.

Due to the aforementioned optimizations, the current version of ARToolKitPlus is roughly 50 times faster on mobile devices than the initial port. Consequently, as can be seen in Table 1, single marker tracking represents no major bottleneck on any of the tested devices. It is interesting to notice that with single marker tracking the

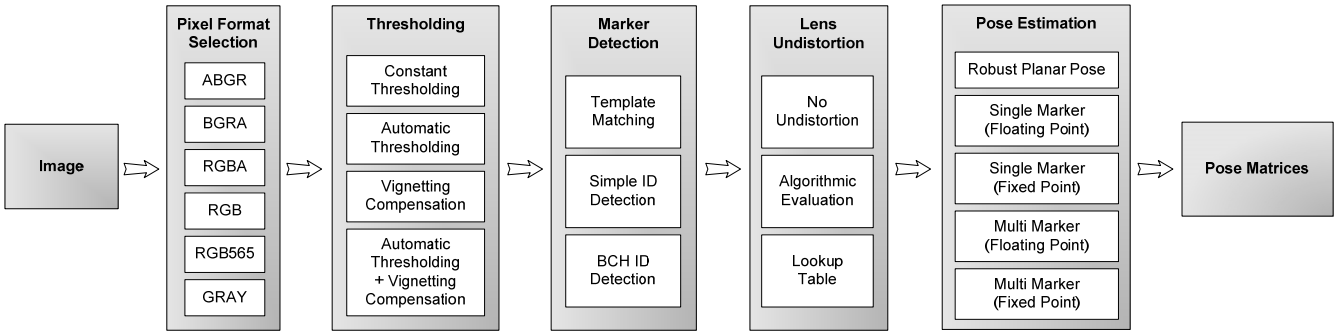


**Figure 6:** Simplified data flow in ARToolKitPlus. In each block, one algorithm is executed.

Intel compiler gains some speed advantage over the Microsoft compiler on those CPUs which can run that code. (Non-Intel CPUs required disabling some optimization flags of the Intel compiler or the generated code would not run).

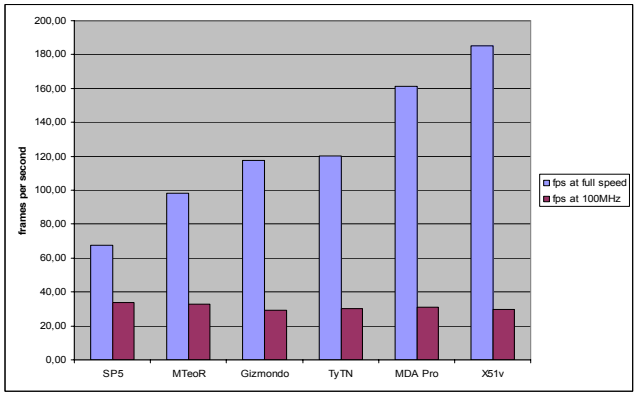


**Figure 7:** Frames per second for single marker tracking on embedded devices (less is better).

Multi-marker tracking puts a severe burden on the processing power of today's mobile devices. While tracking a multi-marker set with four visible markers still performs satisfactory on most devices, the cost for tracking ten visible markers is too high for acceptable frame-rates – considering that tracking is only a one small part of a practical application. It is interesting to notice that on all embedded devices the code generated with the Intel compiler performs worse than the code generated with the MS compiler, which is in contrast to the results of single marker tracking. The reason for this behaviour is not revealed yet.

Almost all smartphones and PDAs today use ARM based CPUs. Furthermore, ARToolKitPlus is almost fully CPU bound and hardly memory-bandwidth bound at all. Hence it is not surprising that the tracking performance on the devices in this benchmark increases linear with the CPUs' clock rates. As can be seen in Figure 7, all devices process 31.12 (+/- 1.76) frames per second at a normalized speed of 100 MHz.

## Lessons learned

From our experience with pose tracking and Augmented Reality in general on mobile devices we arrived at the following set of guidelines and lessons learned:

- **Sequential vs. parallel:** Even though ARM CPUs usually do not have parallel execution units, many operations such as reading the camera or waiting for network reply can be successfully accelerated using multi-threading because they are I/O bound.
- **Camera resolution:** Some high-end phones can deliver video streams at resolutions up to 640x480 pixels. In practice though, other than when using high quality PC cameras, there is only a minimal improvement in tracking quality. The reasons for this are the low quality lenses and camera sensors with high noise levels.
- **Multi-marker tracking:** Using high quality cameras on PCs allows stable single marker tracking. Larger markers can sometimes compensate for the lower image quality of mobile phone cameras, but user interface designs often prevent this. Multi-marker tracking provides highly stable tracking – at the expense of higher computational costs though.
- **Id-based markers:** With a growing number of markers known to the tracking system, the process of template matching can seriously degrade overall performance. Id-based markers do not share this weakness and are always faster to detect than template markers.
- **Camera pixel formats:** Several pixel formats already provide ideal access to the image data for ARToolKitPlus. E.g. as described in Section 5.2, the YUV12 pixel format stores the image in gray values which allows faster tracking, even though the image must be converted to RGB for displaying video background anyway.
- **Compilers:** As shown in Section 7 some compilers can increase tracking speed in certain situations. In older versions of ARToolKitPlus which contained more floating point code, speedups up to 70% were noticed. Unfortunately these compilers are often expensive and generated code only works on specific CPUs.

| | Single Marker | | Multi Marker (4 markers) | | Multi Marker (10 markers) | |
|---|---|---|---|---|---|---|
| Device | MS compiler | Intel compiler | MS compiler | Intel compiler | MS compiler | Intel compiler |
| i-mate SP5 | 14.8 ms | 13.3 ms | 66.4 ms | 78.4 ms | 234.1 ms | 273.8 ms |
| HTC MTeoR | 10.2 ms | n/a | 44.6 ms | n/a | 153.3 ms | n/a |
| Gizmondo | 8.5 ms | n/a | 34.5 ms | n/a | 122.7 ms | n/a |
| HTC TyTN | 8.3 ms | n/a | 34.9 ms | n/a | 128.1 ms | n/a |
| MDA Pro | 6.2 ms | 6.0 ms | 24.1 ms | 29.5 ms | 83.4 ms | 99.1 ms |
| Dell X51v | 5.4 ms | 5.1 ms | 20.7 ms | 23.25 ms | 69.8 ms | 81.2 ms |
| PC | 0.55 ms | 0.43 ms | 6.26 ms | 2.77 ms | 17.53 ms | 8.3 ms |

**Table1:** Benchmarks performed on images with one, four and ten markers. The latter two images were tracked with a multi-marker set of 12 markers of which four and ten were visible.

- **Developing on the PC, final testing on the device:** Debugging on embedded devices is cumbersome and on some platforms such as Symbian not possible without expensive tools. Doing as much work as possible on the PC should be preferred since it results in faster development cycles and often even cleaner code due to the increased portability requirements.

## 8 Future work

We believe that tracking performance of ARToolKitPlus reached a level where no significant improvements are expected to happen from optimizations alone. Instead we want to concentrate on features that fundamentally improve the tracking quality in the future. In particular, the current method for edge detection is a major weakness of ARToolKit and ARToolKitPlus. In the future we plan to add more powerful edge detection and reconstruction modes that should work better in difficult lighting conditions even allowing partially occluded markers. Moreover, we intend to experiment with a combination of fiducial marker tracking and selective natural feature tracking based on fast feature detectors.

In the future, more phones will include multimedia CPU extensions with capabilities such as video encoding and decoding, which could be used for computationally expensive image space operations such as precise pixel flow detection. Currently these functions are usually not accessible due to unavailable vendor-specific APIs. We therefore welcome the upcoming OpenKODE[4] initiative that will provide a common standard for programming mobile phone hardware.

## 9 Acknowledgements

The authors want to thank Hirokazu Kato and Mark Billinghurst for the development of ARToolKit and making it publicly available under the GPL license. Some parts of this work were inspired by ARTag developed by Mark Fiala. A big thank you goes to Thomas Pintaric who wrote the MATLAB camera model support and ported the RPP to C++ for integration into ARToolKitPlus. We would also like to thank all users of ARToolKitPlus who fixed bugs and contributed patches for new platforms. This project was funded in part by Austrian Science Fund FWF under contracts no. L32-N04 and Y193.

[4] OpenKODE: http://www.khronos.org/openkode

**References**

[1] Mark Fiala, ARTag, An Improved Marker System Based on ARToolkit. National Research Council Canada, Publication Number: NRC: 47419, 2004.

[2] Wolfang Friedrich, ARVIKA - augmented reality for development, production and service. *Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR), 2002.* pages 3-4, 2002, Germany.

[3] Anders Henrysson, Mark Billinghurst, Mark Ollila. Face to Face Collaborative AR on Mobile Phones. *Proceedings International Symposium on Augmented and Mixed Reality (ISMAR'05)*, pages 80-89, 2005, Austria.

[4] Hirokazu Kato, Mark Billinghurst. Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99).* pages 85-94, 1999, USA.

[5] Mathias Möhring, Christian Lessig, Oliver Bimber. Video See-Through AR on Consumer Cell Phones. *Proceedings of International Symposium on Augmented and Mixed Reality (ISMAR'04)*, pages 252-253, 2004, USA

[6] Jun Rekimoto. Matrix: A Realitime Object Identification and Registration Method for Augmented Reality. *Proceedings of Asia Pacific Computer-Human Interaction (APCHI) 1998*, pages 63-68, 1998, Japan

[7] Jun Rekimoto, Yuji Ayatsuka. CyberCode: designing augmented reality environments with visual tags. *Proceedings of Designing Augmented Reality Environments (DARE) 2000*, pages 1-10, 2000, Denmark.

[8] Michael Rohs, Beat Gfeller. Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. *Advances in Pervasive Computing, Austrian Computer Society (OCG)*, pp. 265-271, Austria, 2004.

[9] Gerald Schweighofer, Axel Pinz. Robust Pose Estimation from a Planar Target. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2024-2030, 2006.

[10] Daniel Wagner, Dieter Schmalstieg. First Steps Towards Handheld Augmented Reality. *Proceedings of the 7th International Conference on Wearable Computers (ISWC 2003)*, pages 127-135, 2003, USA.

[11] Xiang Zhang, Yakup Genc, Nassir Navab. Mobile computing and industrial augmented reality for real-time data access. *Proceedings of Emerging Technologies and Factory Automation (ETFA), 2001*, pages 583-588, 2001, France.