

Efficient Pose Tracking from Natural Features in Standard Web Browsers

Fabian Göttl
University of Passau

Philipp Gagel
Coburg University of Applied
Sciences and Arts

Jens Grubert
Coburg University of Applied
Sciences and Arts
jens.grubert@hs-coburg.de

ABSTRACT

Computer Vision-based natural feature tracking is at the core of modern Augmented Reality applications. Still, Web-based Augmented Reality typically relies on location-based sensing (using GPS and orientation sensors) or marker-based approaches to solve the pose estimation problem.

We present an implementation and evaluation of an efficient natural feature tracking pipeline for standard Web browsers using HTML5 and WebAssembly. Our system can track image targets at real-time frame rates tablet PCs (up to 60 Hz) and smartphones (up to 25 Hz).

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**;

KEYWORDS

web-based augmented reality, natural feature tracking, webassembly, asm.js, webxr, webar

ACM Reference Format:

Fabian Göttl, Philipp Gagel, and Jens Grubert. 2018. Efficient Pose Tracking from Natural Features in Standard Web Browsers. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, Article 0, 4 pages.

1 INTRODUCTION

We witness a proliferation of Augmented Reality (AR) applications such as games [Thomas 2012] or utilities [Hartl et al. 2013] across device classes (e.g., tablets, smartphones, smartglasses) and operating systems. Software Development Kits (SDKs) such as Vuforia, Apple ARKit, Google ARCore or the Windows Mixed Reality Toolit allow for efficient creation of AR applications for individual platforms through spatial tracking components with 6 degrees of freedom (DoF) pose estimation. However, deployment of AR solutions across platforms is often hindered by those platform-specific SDKs.

Instead, existing Web-based AR solutions typically rely on location-based sensing (e.g., GPS plus orientation sensors).

In particular computer vision algorithms needed for realizing markerless Augmented Reality (AR) applications have been deemed too computational intensive for implementing them directly in the Web technology stack. While marker-based AR systems (often derivatives of ARToolkit) have been demonstrated to work in Web



Figure 1: WebAssembly-based natural feature tracking pipeline running on Google Chrome on a Microsoft Surface Pro tablet and a Samsung Galaxy S8 smartphone.

browsers [Etienne 2017], natural feature tracking algorithms are not in widespread use on standard Web browsers.

Recently, visual-inertial odometry approaches as provided by Apple ARKit or Google ARCore have been combined with custom Webviews [Google 2017a,b] to allow experimentation with Web-based Cross Reality APIs like WebXR [Group 2018]. Still, they are not available in standard Web browsers.

Providing efficient 6 DoF pose estimation using natural features in standard Web browsers could help to overcome challenges of multiple creation of platform-specific code. To this end, we present the implementation and evaluation of an efficient computer vision-based pose estimation pipeline from natural features in standard Web browsers even on mobile devices, see Figure 1. The pipeline can be tested under current versions of Google Chrome or Mozilla Firefox at www.ar4web.org.

2 RELATED WORK

There has been an ongoing interest in supporting the creation of Augmented Reality applications using Web technologies [Koopman and MacIntyre 2003; MacIntyre et al. 2011; Speigler et al. 2015].

Several architectures have been proposed that aim at separating content, registration and presentation modules for Web-based Augmented Reality solutions (e.g., [Ahn et al. 2013, 2014; Leppänen et al. 2014; MacIntyre et al. 2011]) but they typically rely on sensor-based registration and tracking. Further, efforts have been made to standardize how to make available content into Web-based

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.

Conference'17, July 2017, Washington, DC, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

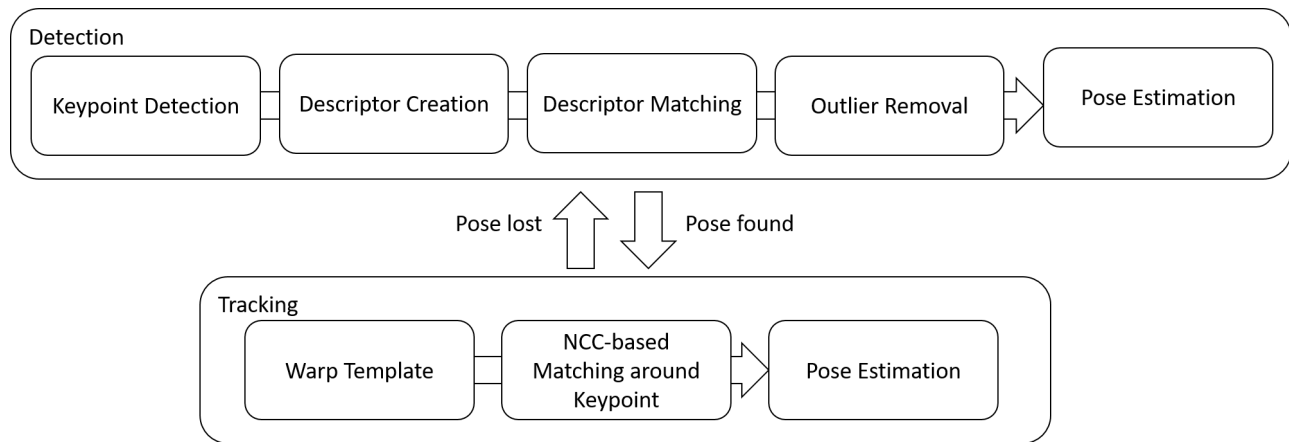


Figure 2: Overview of detection and tracking pipeline.

AR applications through XML-based formats (like ARML [Lechner 2013; Lechner and Tripp 2010], KARML [Hill et al. 2010] or the TOI format [Nixon et al. 2012]), which again focus on location-based spatial registration and often target mobile Augmented Reality browser applications [Langlotz et al. 2013, 2014; Sambinelli and Arias 2015].

While recent experimental browsers have been released with visual-inertial tracking [Google 2017a,b], on standard Web browsers computer vision-based tracking is mainly limited to fiducial-based pose estimation [Etienne 2017].

Only few works have investigated to make computer-vision based pose tracking from natural features directly available using standard web technologies. In 2011, Oberhofer et al. presented an efficient natural feature tracking pipeline with a dedicated detection and tracking phase [Oberhofer et al. 2012]. While their solution was able to run in Web browsers supporting video capture through HTML5, the implementation was purely written in JavaScript. Specifically, it was not able to achieve real-time frame rates on mobile devices. They also reported that the slowdown compared to native implementations (in Google NativeClient) was fourfold.

Recently, a commercial solution for natural feature tracking was announced [Awe.media 2017]. For this approach, no performance metrics were made available but only videos demonstrating interactive framerates on an unknown platform.

In contrast to previous approaches, we present an efficient implementation of a pose estimation pipeline from natural features that runs at real-time framerates on standard web-browsers on mobile devices with processing time for the tracking component as low as 15 ms on a tablet and 50 ms on a smartphone.

3 NATURAL FEATURE TRACKING PIPELINE

Our pose estimation pipeline combines a dedicated detection step with an efficient tracking phase as proposed for mobile platforms [Wagner et al. 2008, 2010]. Our overall pipeline is shown in Figure 2

If no pose has been found, the initial detection first extracts ORB features in the current camera frame [Rublee et al. 2011]

and matches them with features from template images using fast approximate nearest neighbor search [Muja and Lowe 2009]. This is followed by outlier removal with a threshold of three times the minimum feature distance. Based on all remaining features, we first estimate a homography based on a RANSAC scheme, transform 4 corner points of the template image using that homography and employ an iterative perspective-n-point (PnP) algorithm for the final pose estimation.

As soon as the pose is found, we can switch from the expensive detection phase into a lightweight tracking phase. This phase consists of tracking keypoints that were detected previously in the detection phase. First, we take the homography H that was determined in the previous frame. Based on this homography, we create a warped representation of the marker. To speed up computation, the warped image is downsampled by factor of 2. For each keypoint that is visible in the current frame we cut a 5×5 px image patch out of the warped image. The image patch should look similar to the patch in the current camera image. In order to save computation time, we keep track of a maximum of 25 patches or keypoints. With normalized cross correlation (NCC) we match the image patch to the current frame. Here, we use a search window of 16px length. These optimized points are stored with object keypoints together in tuples, which enables to compute an updated homography H . Similarly to the detection step, we compute the pose R and t out of H with iterative PnP.

If the distance between the camera pose of the recent and previous frame is within a given threshold (e.g., the translation threshold for targets in DIN A4 was empirically determined as 5 cm), we handle the pose as valid. At the next frame we start over again in front of this phase. If the pose is invalid, the detection phase is started again.

3.1 Implementation

Our pipeline utilizes WebAssembly and OpenCV for efficient processing of tracking data. We use the framework Emscripten, to compile C++ code with OpenCV embeddings to WebAssembly. Alternatively, Emscripten is able to output code in a subset of Javascript called asm.js. The camera is accessed through `getUserMedia` and

writes the image into a HTML5 canvas element. The data is passed to the C++ level through heap memory and processed through the above mentioned pipeline. Finally, the computed pose is passed back and utilized in rendering a 3D scene with WebGL.

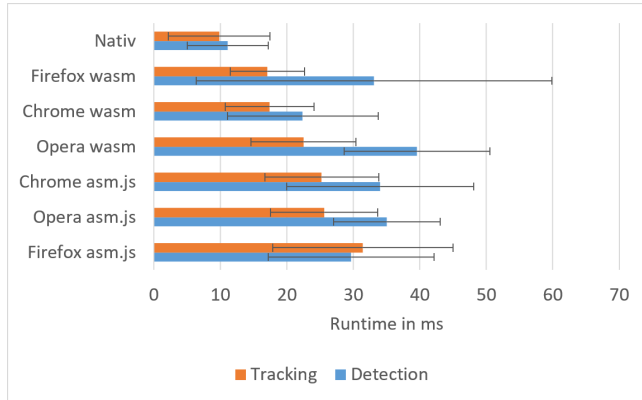


Figure 3: Runtime performance of the tracking and detection steps on a Microsoft Surface Pro.
 wasm: WebAssembly, asm.js: A low-level Javascript subset

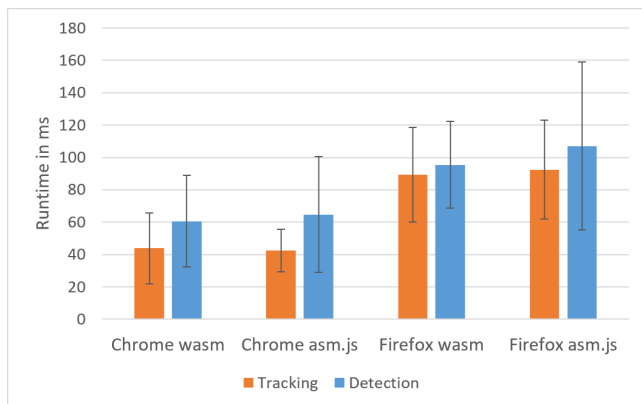


Figure 4: Runtime performance of the tracking and detection steps on a Samsung Galaxy S8.
 wasm: WebAssembly, asm.js: A low-level Javascript subset

4 EVALUATION

We conducted a performance evaluation of the pipeline on a Tablet PC and a smartphone with two Web browsers on each platform (Mozilla Firefox version 59 and Google Chrome version 64). Additionally, we evaluated Opera 52 on the Tablet PC.

The Tablet PC was a Microsoft Surface Pro with an Intel Core i5-6300U (Dual Core with 2,40 Ghz) processor and 8 GB RAM running Windows 10. The smartphone was a Samsung Galaxy S8 with an Octa-core (4x2.3 GHz Mongoose M2 and 4x1.7 GHz Cortex-A53) processor and 4 GB RAM running Android 7.0 (Nougat). The camera resolution was set to 320x240 px.

Figure 3 shows the runtime on the Tablet PC and Figure 4 for the smartphone. Please note, that in Figure 3 there are additional bars indicating the performance of the native code.

Compared to the native C++ implementation on a Microsoft Surface Pro the average runtime for detection increases by 100% on Chrome to 200% for Firefox. In contrast, for tracking the average runtime increases by 70% for Firefox and by 75% for Chrome.

Figure 5 indicates the average performance of the complete tracking pipeline (after an initial detection step). The figure indicates that accessing the camera through getUserMedia is substantially slower on Firefox compared to Google Chrome. Both on the Surface Pro and on the Galaxy S8 Google chrome needs on average 1 ms to deliver the camera image, whereas on Mozilla Firefox it is 6 ms on the Surface Pro and 8 ms on the Samsung Galaxy S8.

This indicates, that if the initial detection phase is completed, our pipeline can run up to 70 Hz under Chrome on a Surface Pro tablet and 20 Hz on a Galaxy S8. However, under Firefox it only runs at 30 Hz on a Surface Pro and 7 Hz on a Galaxy S8.

For robustness, we measured degrees until tracking failed. On multiple targets the minimum angle (starting from the horizontal plane) required for tracking was on average 17°(sd=3) both on the Tablet PC and the smartphone.

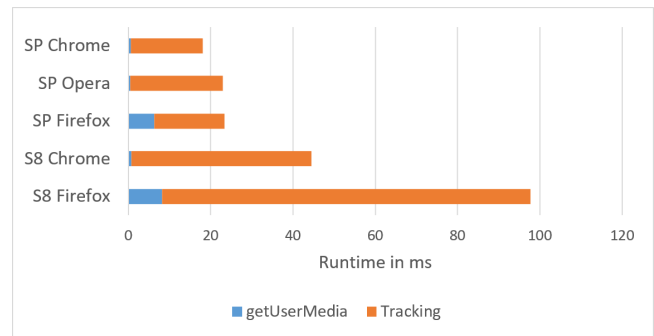


Figure 5: Runtime performance of the tracking pipeline including access to the camera image through getUserMedia.
 SP: Microsoft Surface Pro, S8: Samsung Galaxy S8.

5 DISCUSSION

The results indicate that under optimal conditions, the proposed pipeline can run efficiently on standard Web browsers, both on Tablet PCs as well as on recent smartphones. The initial detection step is rather slow (between 3 Hz on a smartphone and 12 Hz on a tablet PC) with a noticeable speedup as soon as the tracking phase enters after the target was found initially. The real life runtime performance of the pipeline depends on how often a switch between both phases is necessary. We found empirically that even if the tracking phase fails the detection phase quickly re-initializes the pose and, hence, is only active for 1-2 frames.

We investigated the pipeline performance under WebAssembly and the JavaScript subset asm.js. WebAssembly builds are faster than asm.js. In contrast, WebAssembly builds are slower for the detection step in Firefox (by 9%) and Opera (by 12%). We assume

that some optimizations for WebAssembly are not applicable in the detection step.

One issue we noticed during our evaluation, is the strong dependency of the pipeline runtime overall performance on the employed browser. Specifically, while Google Chrome can provide fast access to camera images in approximately 1 ms, Firefox can take up to 26 ms on a smartphone to access the image data.

To be fair, this issue is not specific to our implementation, but applies to other vision-based pipelines that require live camera access, as well.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we presented an implementation and evaluation of an efficient natural feature tracking pipeline for standard Web browsers using HTML5 and WebAssembly. Our system can track image targets at real-time frame rates tablet PCs (up to 65 Hz) and smartphones (up to 25 Hz).

In future work, we want to combine our pipeline with efficient large scale image search and further optimize the tracking parameters (e.g., number of keypoints, search window size) on a per target basis. We also see potential for integrating it with Semantic Web-based Augmented Reality Systems [Nixon et al. 2012] or to utilize WebAssembly to enable new Web-based AR user experiences, e.g., through around-device interaction [Grubert et al. 2016].

REFERENCES

- Sangchul Ahn, Heedong Ko, and Steven Feiner. 2013. Webizing mobile AR contents. In *Virtual Reality (VR), 2013 IEEE*. IEEE, 131–132.
- Sangchul Ahn, Heedong Ko, and Byoungyun Yoo. 2014. Webizing mobile augmented reality content. *New Review of Hypermedia and Multimedia* 20, 1 (2014), 79–100.
- Awe.media. 2017. Bring your images to life. <https://awe.media/blog/bring-your-images-to-life>. (2017). Accessed: 2018-03-02.
- Jerome Etienne. 2017. AR.js - Efficient Augmented Reality for the Web. <https://github.com/jeromeetienne/AR.js>. (2017). Accessed: 2018-03-02.
- Google. 2017a. Quickstart for AR on the Web. <https://developers.google.com/ar/develop/web/quickstart>. (2017). Accessed: 2018-03-02.
- Google. 2017b. WebARonARKit. <https://github.com/google-ar/WebARonARKit>. (2017). Accessed: 2018-04-22.
- Immersive Web Community Group. 2018. WebXR Device API. <https://immersive-web.github.io/webxr/>. (2018). Accessed: 2018-04-22.
- Jens Grubert, Eyal Ofek, Michel Pahud, Matthias Kranz, and Dieter Schmalstieg. 2016. Glasshands: Interaction around unmodified mobile devices using sunglasses. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces*. ACM, 215–224.
- Andreas Hartl, Jens Grubert, Dieter Schmalstieg, and Gerhard Reitmayr. 2013. Mobile interactive hologram verification. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*. IEEE, 75–82.
- Alex Hill, Blair MacIntyre, Maribeth Gandy, Brian Davidson, and Hafez Rouzati. 2010. Khroma: An open kml/html architecture for mobile augmented reality applications. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*. IEEE, 233–234.
- Rob Kooper and Blair MacIntyre. 2003. Browsing the real-world wide web: Maintaining awareness of virtual information in an AR information space. *International Journal of Human-Computer Interaction* 16, 3 (2003), 425–446.
- Tobias Langlotz, Jens Grubert, and Raphael Grasset. 2013. Augmented reality browsers: essential products or only gadgets? *Commun. ACM* 56, 11 (2013), 34–36.
- Tobias Langlotz, Thanh Nguyen, Dieter Schmalstieg, and Raphael Grasset. 2014. Next-generation augmented reality browsers: rich, seamless, and adaptive. *Proc. IEEE* 102, 2 (2014), 155–169.
- Martin Lechner. 2013. ARML 2.0 in the context of existing AR data formats. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2013 6th Workshop on*. IEEE, 41–47.
- Martin Lechner and Markus Tripp. 2010. ARML-AR augmented reality standard. *coordinates* 13, 47.797222 (2010), 432–440.
- Teemu Leppänen, Arto Heikkinen, Antti Karhu, Erkki Harjula, Jukka Riekki, and Timo Koskela. 2014. Augmented reality web applications with mobile agents in the internet of things. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*. IEEE, 54–59.
- Blair MacIntyre, Alex Hill, Hafez Rouzati, Maribeth Gandy, and Brian Davidson. 2011. The Argon AR Web Browser and standards-based AR application environment. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*. IEEE, 65–74.
- Marius Muja and David G Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (I)* 2, 331–340 (2009), 2.
- Lyndon JB Nixon, Jens Grubert, Gerhard Reitmayr, and James Scicluna. 2012. SmartReality: Integrating the Web into Augmented Reality.. In *I-SEMANTICS (Posters & Demos)*. Citeseer, 48–54.
- Christoph Oberhofer, Jens Grubert, and Gerhard Reitmayr. 2012. Natural feature tracking in javascript. In *Virtual Reality Short Papers and Posters (VRW), 2012 IEEE*. IEEE, 113–114.
- Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, 2564–2571.
- Fernando Sambinelli and Cecilia Sosa Arias. 2015. Augmented Reality Browsers: A Proposal for Architectural Standardization. *International Journal of Software Engineering & Applications* 6, 1 (2015), 1.
- Gheric Speiginger, Blair MacIntyre, Jay Bolter, Hafez Rouzati, Amy Lambeth, Laura Levy, Laurie Baird, Maribeth Gandy, Matt Sanders, Brian Davidson, et al. 2015. The evolution of the argon web framework through its use creating cultural heritage and community-based augmented reality applications. In *International Conference on Human-Computer Interaction*. Springer, 112–124.
- Bruce H Thomas. 2012. A survey of visual, mixed, and augmented reality gaming. *Computers in Entertainment (CIE)* 10, 1 (2012), 3.
- Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. 2008. Pose tracking from natural features on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 125–134.
- Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. 2010. Real-time detection and tracking for augmented reality on mobile phones. *IEEE transactions on visualization and computer graphics* 16, 3 (2010), 355–368.