Dual-data Filtering for Robust Position and Angular Tracking of the Oculus Rift

*

Anonymous

Abstract

We present a system to perform long-range, robust positional and angular tracking of the Oculus Rift head-mounted-display (HMD). In this case, Marker Tracking with overhead camera (pointed downwards) is used to achieve both positional and angular tracking of the Oculus Rift. Here, our contribution is to combine the resulting estimates of pose (from Marker Tracking) with low-latency inertial sensor data from the Oculus. This is motivated by our key observation that inertial sensors in the Rift suffer from low-frequency drift which accrues over time, while Marker Tracking systems suffer from high-frequency error when estimating angles. In addition the latter also suffers from glitches when tracking fails or the marker is not visible in the image frames. However, although tracking can lead to a few frames with unreliable data, they do not accrue error when tracking does resume. We combine robustly data from the Rift using computer vision based tracking methods, by using a Kalman filtering approach. Our experiments shows that this approach can deal wiht drift in yaw angle. The positional tracking enables crouching, jumping and small positional movements in games. Finally, we show acceptable delays in both the marker tracking by using color blob detection and network communication neccesary for this solution.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Display Algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity;

Keywords: Color Blob Detection, Marker Tracking, Oculus Rift DK1, Overhead Camera, Walking, Crouching, Jumping

1 Introduction

The Oculus Rift DK1 [DK1][Pohl et al. 2013] is gaining popularity as an interface for Virtual Reality (VR) and Augmented Reality (AR) applications. This system is essentially a head-mounted-display (HMD) with built-in angular tracking of rotations of the head. While this version has remarkably low-latency tracking (1000Hz), there are limitations. the residual error in the angular tracking (particularly yaw) is significant[1], even with additional calibration and magnetometer-based corrections (see fig. 5). Although the latest version Oculus Rift DK2 (not available at the time of writing) comes with infra-red (IR) LED's to perform some positional tracking, this system is limited by the view of the camera and still focuses on seated users because of the frontal facing camera.

Our main observation is that the measurement errors of the sensors in the Oculus Rift and computer vision based algorithms are complementary. While the former suffers from accrued inaccuracy over long periods of time, the latter provide more noisy measurements within a short time window. However, the Oculus Rift is able to provide low-latency reliable data in small time windows while the computer vision algorithms are more likely to yield consistent tracking over long time periods. Based on this observation, we propose a novel approach that combines both the sensor modalities.

The core contribution in this paper is a system that can perform both positional (moving, crouching, jumping) and angular tracking of head-mounted-display (HMD) within an acceptable delay. The entire system uses readily available and affortable hardware to enhance the gaming experience. Secondly, an online filtering algorithm is presented that uses the dual streams (Oculus rift and vision-based marker tracking) of data to reconcile their mutual shortcomings. Additionally, color blob tracking further reduces system latency in localising the vision-based marker processing to a subset of the camera image. We demonstrate the benefits of the above in the context of a virtual reality application using the Oculus Rift (see Supplementary Materials for a demonstration of our system).

2 Related Work

Currently we are not aware of any prior work that combines the Oculus Rift with Computer Vision in the manner we propose. Although there are several industry projects that combine the Oculus with another sensor modality, these do not use overhead camera which allows for less constrained positional tracking. Since there is not much related work to the current solution, we have looked at the two most important subproblems that have to be solved: the marker detection methods and the virtual reality helmets systems.

Marker detection: for virtual and augmented reality purposes is not novel, where multiple papers [Zhang et al. 2002],[Fiala 2005],[Garrido-Jurado et al. 2014] describe how a marker can be detected and some paper even look at optimal marker designs [Fiala 2010]. Although marker detection can be done in real-time (depending on your definition), for the HD webcam used in this project the marker detection was the major bottleneck. Color tracking is another technique for allowing tracking of objects in an image [Bradski 1998],[Simon et al. 2001],[Pérez et al. 2002], with a large body of work focusing on robustness of the color tracking under varying illumination conditions. Given that in our case we can create the marker ourselves, more simple thresholding seems to work given one simple color and computing a blob given a pre-defined threshold.

Virtual Reality helmets system: are not a recent development in consumer entertainment, despite the booming interest in the Oculus Rift. Disregarding it's prevalent use in commercial or military applications, such as pilot training and vehicle safety testing [Tomilin 1999], the use of head-mounted displays for entertainment purposes has been executed somewhat successfully almost a decade ago[Pausch et al. 1996]. However, these devices were prohibitively expensive for regular consumers up until very recently. With the advent of consumer-priced virtual reality headset systems like the Oculus Rift [DK1] and CastAR[2], and the wide commercial interest they've had, there is huge motivation to research similarly low-cost methods to improve on them. The most relevant work on positional

*e-mail:Anonymous@Anonymous.com

[1] http://www.oculusvr.com/blog/magnetometer/

[2] https://www.kickstarter.com/projects/technicalillusions/castar-the-most-versatile-ar-and-vr-system

and angular tracking of an virtual reality helmet is already quiet old. The first system dates from the 1990s [Wang et al. 1990], where head-mounted cameras are used. Improvement on both the camera system and tracking are suggested by [Ward et al. 1992],[Welch et al. 1999],[Maesen and Bekaert 2011],[Maesen et al. 2013]. In all cases, head-mounted cameras seem to be preferred over following the virtual reality helmet with an overhead camera system. Possible reasons for this are latency, where the signal of the camera has to be sent to and processed by the helmet, while with headmounted cameras, there is only latency due to processing. We however argue that given the new advances in both networking, computer vision and graphical rendering, the delay can be brought to acceptable levels, making an over-head camera a realistic possibility.

3 Angular and positional tracking data

For the angular and positional tracking both a standard webcam (Logitech C920 HD) and the sensor information of Oculus Rift DK1 are combined. The webcam tracks an augmented reality marker [Garrido-Jurado et al. 2014], which gives us estimates of both the angular and positional data of the Oculus. Because the detection of the augmented reality marker is not fast enough (especially in the HD1080p recordings), this step has been improved by using color detection (Section 3.1) to find subregions, executing the marker detection algorithm on a small portion of the image (Section 3.2). The Oculus Rift DK1 is connected to the laptop, which directly receives the angular data (Section 3.3). Computer vision processing is performed on a server which sends the data over a wireless connection to a laptop (using UDP), where timestamps (NTP) are used to synchronise the Oculus and webcam data (Section 3.4) On the laptop, both the webcam and Oculus data are combined using a Kalman Filter (Section 3.5)

3.1 Fast color blob detection

Our focus is the creation of a fast and reliable detection with center location a known color feature. Many simple blob detection methods have been developed over the years [Williams 1990], with much attention directed to human skin color [Bradski 1998]. However, skin color of humans can vary greatly, often making this an unreliable feature. In our case, with easilly accessible printed markers, the only variations are caused by the light reflections of the surface, while color is pre-defined. The two main reasons to perform color feature detection is that this is more robust and faster than structured marker detection.
The following methodology (see Figure 3) has been developed to perform fast feature detection:

- Convert the RGB image to HSV
- Thesholding in the HSV space to obtain a binary image
- Compute the integral image of the binary image
- Check using the integral image for possible blob locations
- Refine the blob locations

The main challenge is to develop a fast method that obtains the region which contains the color based on the thresholded images. To achieve this, first the integral image is computed for the binary image. This allows us to compute the histogram on both the rows and columns of the image, where using the integral image we only have to perform 4 array references to compute the number of black pixels in the image[3]. By thresholding (with $T = 10$ pixels), we remove the noise and obtain possible regions that can contain a blob feature. However, if there are multiple blobs in the image, the integral image can be used to check if the regions indeed contain a significant number of black pixels. Finally, a refinement needs to take place, where for each detected blob we verify if it can pruned by checking if the outer rows and columns have enough pixels (with $T = 10$ pixels). The rows and columns with less pixels than the threshold are removed, which gives us the final squared blob position in the image.

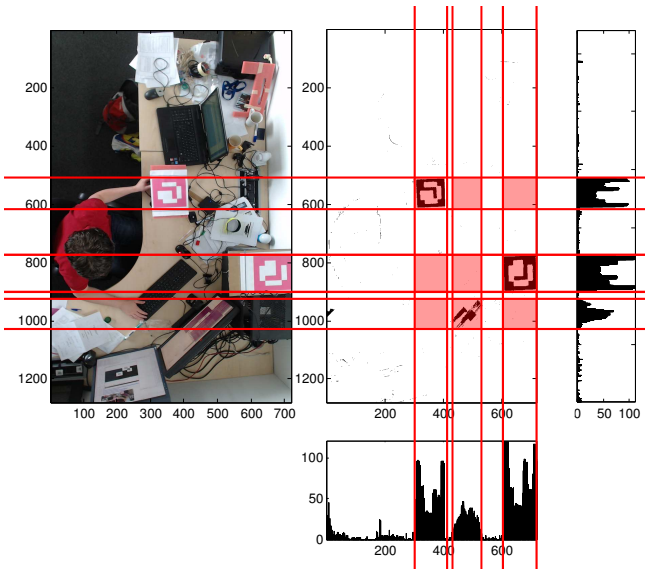


Figure 1: *Visualisation of our color blob tracking works. Thresholding of the image (left) result in binary mask (right). Based on the integral image, fast computation of the histograms is possible. Thresholding based on histograms gives us the region of interest (based on horizontal and vertical lines), where for the red region using the integral image the number of white pixels can be computer using 4 array references.*

While this method performs efficiently on desktop and laptop devices, a further optimized method can be employed to reduce impact of latency of tracking on low-powered devices, such as mobile phones and tablets. Here, our GPU accelerated method follows [Williams 1990] in the computation of the first moment of the integral of the thresholded color difference image to recover the centre of the blob area. A GPU reduction performs the summation hierarchically with accumulations from successive (16 bit half-float format) render target passes with OpenGL ES2.0. Such processing is extremely fast with simple addition operations aligned to the maximum number of simulatanous texture samples per fragment shader pass of the device. The primary performance harzard is the potential for delays in transfer of results to CPU memory. This is avoided by having the render target results remain on GPU memory for texture reads by successive passes, until the final reduced buffer to sum is a single value or otherwise very small buffer of values. We demonstrate the robustness of this fast feature tracking method in an iPad app available on the AppStore [SkyeWars 2013] (Figure 2).

[3]The sum of all values in rectangle is given by 4 values of the integral image: $(x_{br} - x_{bl} - x_{tr} + x_{tl})$, where t, b, l, r are respectively top, bottom, left and right

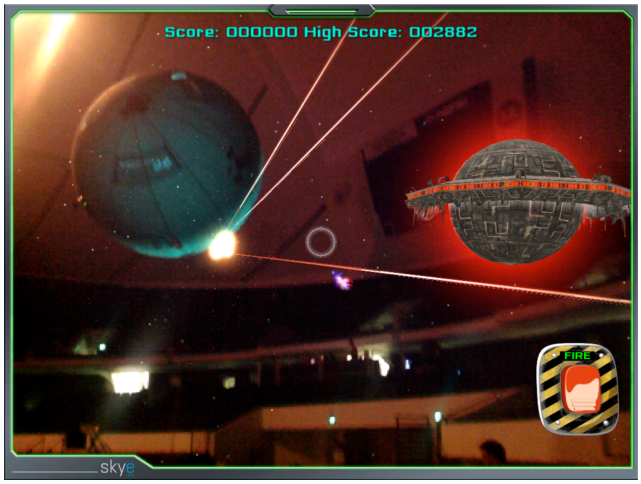


Figure 2: *Fast blob feature tracking on a GPU as applied on mobile devices to tracking an illuminated aerial robot with moon coloring in the augmented reality game,* **Skye Wars***. The converging lasers and explosion visual effect illustrates the tracked location of the moon's cresent within a challenging illumination environment (the Anaheim Arena).*

3.2 Angular and Positional tracking: From the Marker Detection

Marker tracking is performed with the Aruco library [Garrido-Jurado et al. 2014], where markers are found in combination with the color blob detection. By using the color blob detection detection, we limit the search area of the Aruco library, making it in most cases faster. The camera can be calibrated either with a checkerboard or the code provided by Aruco, which allows us to obtain the angular and positional information based on the known size of the marker. The angular position of the marker is converted to euler angles. In this work, we chose a overhead camera, where the yaw will be the most accurately estimated angle. This is because, given the appearance of the marker, we only need two points to accurately estimate yaw, while for pitch and roll 3 points are needed. To compute the postion and angles, the four points of the square are required. We considered using multiple markers, which will give some robustness in detection (if the single marker is missed) and improving the accuracy by taking the average angles of the markers. However in this case, the markers often became too small in terms of pixels for accurate detection, which is the reason that in the final setup a single marker of 19×19 centimeters is used which can still be detected at a distance of 3 meters given a (Logitech C920 HD) webcam. In the case that we lose the marker, no angular information will be provided by the webcam to the laptop, so we will rely on entirely on the Oculus data. Without the marker, it is impossible anyway to provide angular information because a homography needs to be estimated which can only be done by finding four points of the square. For the positional data, if the marker detection fails, we can still compute an estimate based on size and location of the blob. There are however circumstances in which also the color blob is also not detected (i.e. where the user will walk out of the camera view or the marker is perpendicular to the image plane). In scenarios where the gameplay uses the positional data, a clearly defined field of view needs to be set both for obvious safety reasons and for gameplay experience.

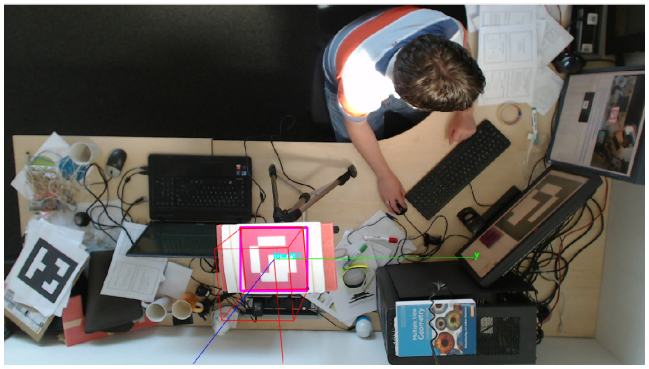


Figure 3: *Example of the output of Aruco given the marker detection after performing color detection*

3.3 Angular tracking: From Oculus Rift inertial sensors

The Oculus Rift inertial sensors will give us pitch, roll and yaw. Given our initial experiments (see Section 5.1), most of the drift seems to be in the yaw. Although this is not visible for the user of the Oculus except when they have finished using it. Still, by having the users walking around it would be impractical if the yaw angle drifted slightly, also because the camera view should be in relation to an area in the real world, which should be similar to the field of view of the camera.

3.4 Synchronisation of signals

Both the Camera and Oculus Rift are recorded on two different devices, where these signals need to by synchronised. This is achieved by first synchronising the clock of both computers using NTP to the same time server. Both the computer attached to the webcam and the laptop that receives the Oculus Rift sensor data give this data a timestamp (given the NTP synchronised clocks). The timestamp data is matched on the laptop that controls the Oculus Rift sensor, where the timestamp with the most delay is used as direct input in the Oculus Rift rendering module. We discovered that the synchronisation is not necessary. In this case, both the Oculus and Network data are collected by seperate threads. On the client side and we used the most recent sensor data which worked as well. The experiments are all performed with synchronised data.

3.5 Robustification using dual-track filtering

After synchronising the signals of the Camera and Oculus Rift, the two signals need to be combined, where the Kalman filter [Kalman 1960] is used for this purpose. The first challenge is in alignment of the angles between the Oculus Rift and Marker tracking. The Oculus Rift starts with yaw at zero, while the yaw of the camera depends on the Marker Position in the scene. To correct for this, a difference vector δ is computed between the angle of the Oculus Rift and the marker tracking, which is used in the remainder to correct the marker tracking angular data.
The Kalman filter created in this research is able to deal with angular information, where the distance between 5 and 355 is 10 degrees, instead of 350 degrees. The estimation of the next state is based on the velocity. The Oculus Rift and marker tracking is combined in the measuring stage, where for both sensors the standard deviations of the noise are indicated. In the case of pitch and roll, the measure of the Oculus have a lower standard deviation, while for the yaw the marker tracking deviation is set to

be lower.
Our version of the Kalman filter (Algorithm 1) gives a good framework for combining the noisy observations of both the Oculus and marker tracking. In addition to dealing with the noisy data, the Kalman filter is also able to deal with the fact that the camera might not observe the marker. In this case, we only use the Oculus Rift data. We considered also using a Kalman filter for the positional data of the marker tracking. However losing the marker was in most cases either because the marker could not be viewed anymore due to the fact that the marker went out of the image plane or due to severe pose (i.e perpendicular to the image plane). For both cases, further predictions of the Kalman Filter are not that useful.

Data: prediction: y_t, measuments: z_t, oculus: $o_{t,1:3}$, vision: $v_{t,1:3}$
Result: Kalman filter: x_t
Prediction using velocity: A, Observation process: H,
Measument noise: R, System noise: Q;
while *program runs* **do**
$y_t = Ax_{t-1}$;
$y_t = \mathbf{value2angle}(y_t)$;
$E_t = AP_{t-1}A^T + Q$;
$K_t = E_tH^T(HE_tH^T + R)$;
$t_{t,1:3} = \mathbf{angulardiff}(o_{t,1:3}, y_{t-1,1:3})$;
if *notcameraworking* **then**
$d_{1:3} = \mathbf{angulardiff}(o_{t,1:3}, y_{t-1,1:3})$;
$v_{t,1:3} = v_{t-1,1:3} + d_{1:3}$;
$t_{t,4:6} = z_{t,4:6} - \mathbf{angulardiff}(v_{t,1:3}, o_{t-1,1:3})$;
$v_t = \mathbf{value2angle}(v_t)$;
end
$t_{t,4:6} = \mathbf{angulardiff}(v_{t,1:3}, y_{t-1,1:3})$;
$x_t = y_t + K_tt_t$;
$x_t = \mathbf{value2angle}(x_t)$;
$P_t = (I - K_tH)E_t$;
end
Algorithm 1: Kalman filter for angular rotation, by using **angulardiff** to compute the closest distance between two angles and **value2angle** which converts any number not between $[0 - 360]$ to the correct angular value.

4 System Architecture and Testbed Configuration

We opted to use Unity and its dedicated Oculus Rift SDK plugins in order to set up the software side of our testbed, which were easy to manage and modify. Given Unity's accessible prototyping capabilities we were also able to quickly build a sample game environment for demonstration purposes.

The headset itself was mounted on a tripod and fastened securely to ensure stability. The tripod was aligned so that the orientation data provided by the headset was as close to 0 degrees as possible on every axis. The tripod's orientation and position was then marked, using tape for guidelines, so that it could be returned to its original position whenever required. The headset was calibrated with the tripod attached in the test environment in order to account for any magnetic interference from local artifacts (steps were taken to isolate the system from devices that could provide magnetic interference, within reason) and the earth's magnetic intensity and flow at the laboratory's location.

This system is targeted to be capable of supporting multiple users in a "drop in and play" environment. The usb 2.0 wired camera must be attached to a host machine for visual processing. Likewise, the Oculus Rift headset required its own host machine for rendering. To accommodate these constraints, our system was set up to support a single-server-multiple-clients networking protocol. The camera is attached to the server (since we expect both to remain stable throughout the session) and carries out all visual processing, transmitting camera data regularly to the clients. The clients are laptops attached to the system's users which are also connected to the individual headsets. Unfortunately, the Oculus Rift DevKit 1 requires an external power source which would need portable battery source of modest rating to fully untether our experimental setup. The networked solution is however, wireless, in order to sufficiently test the latency and engineering issues surrounding a fully-mobile system.

5 Experiments and Results

Several experiments have been performed to understand the performance of the Oculus Rift and combine it with the Webcam data. During these experiments, we mainly looked at the difference between the sensor in the Oculus Rift compared with the Webcam data. We verified that the positional data was correct by placing the marker on different known positions and recorded these positions. Since positional marker tracking [Zhang et al. 2002],[Fiala 2005],[Garrido-Jurado et al. 2014] is pretty well understood, and because positional tracking depends on the camera setup and scene, we will not go into details on this issue. For the purpose of this project, the positional tracking was deemed as accurate enough and any small errors dependent on the camera setup and hardware used. Given this project, the more interesting scientific question are:

- What is the Angular Drift of the Oculus Rift?
- How can we combine the Angular measure of the Oculus Rift and Webcam?
- Can we improve the Yaw drift of the Oculus Rift with the Webcam?
- What kind of Latency does this system have?

5.1 Verifying Angular Drift of Oculus Rift

Oculus VR's Principal Scientist, Steve LaValle[4], provides average and worst-case yaw drift values for the Oculus Rift DevKit 1 with magnetometer correction. For their experiments, the user sat down and played a virtual reality game for over twenty minutes. They continuously tracked the headset's orientation data and compare it against a ground truth obtained by measuring head orientations using an OptiTrack motion capture system. The Oculus Rift team discovered a drift of approximately 3.7 degrees on average with worst-case performance within 10 degrees.

In order to simulate approximately 20 minutes of standard gameplay the tripod oscillated on yaw at a moderate pace, with occasional rotations on pitch and roll. Orientation data from the headset was recorded continuously at 60Hz and at 90 second intervals for 15 intervals, with 80 seconds per interval dedicated to moving the headset in a natural way (totaling to 20 minutes of movement across the entire run) and 10 seconds to realign the headset to the starting position. By realigning the headset to its original position where the sensor indicated 0 degrees on each axis, and recording the stipulated orientation at that point, we merely calculated the difference between the original measurement and the current measurement in order to calculate the drift. The experiment was repeated three times to calculate a satisfactory average; one set with the magnetometer correction on and one with the correction off.

[4] http://www.oculusvr.com/blog/magnetometer/

Our yaw drift experiments with magnetometer correction showed worse results than the Oculus VR team, deviating by approximately 8.22 degrees on average after the 20 minutes of standard gameplay. Our average maximum error value was near the value presented by Oculus VR, at 10.51 degrees. It should be noted that although we ran our experiments with the utmost care, it is possible that there is slight deviation from the true value compared to a computerized ground-truth method such as the OptiTrack motion capture system. Drift from the rest position angle, with magnetometer correction, tends to oscillate around zero. Without magnetometer correction, the average yaw drift by the end of the experimental run was approximately 64.13 degrees. Drift in these cases steadily increases, so the maximum average drift value is equivalent to the average yaw drift by the end of the experimental run.

Given that an top-down camera is used to track the marker, the most accurate angle given by the marker detection is the yaw, because only 2 points of the markers are needed to estimate it's position. Figure 4(b) shows that the estimation for yaw by the camera is smoother than for the another angles. In case of pitch and roll, the camera estimates are less accurate as can be observed in Figures 4(a) and 4(c), where the angular estimation of the cameras contains lot of noise, which is filtered by using our Kalman filter. The Kalman filter takes into account the noisy measurement of pitch and roll, while at the same time is able to give the camera measurement in yaw higher priority. There are two reasons that pitch and roll are more noisy for the camera. The first reason is that 3 points of the marker are neccesary to to estimate those angles. The second reason is that the board on which the marker was attached sometimes bends a little which has effect on the geometric assumption used in computing those angles (this can be improved in our next prototype).

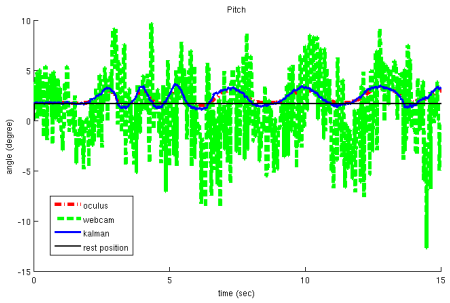
5.2 Filtered Orientation Estimation

The same procedure is used to validate our headset's yaw-drift in order to test our complete system. The headset's orientation data, the webcam's orientation estimates, and the resulting Kalman filtered data are all recorded at 60Hz.

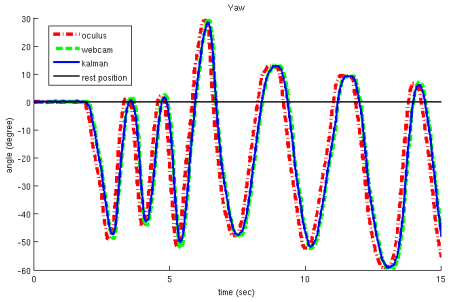
We were also interested in the system's performance in "non-standard" gameplay scenarios with severe motion. In order to simulate "non-standard" motion the headset was rotated rigorously along each axis – although mostly on yaw – for brief periods followed by long periods of stable rest. Occasionally, the marker was occluded, either manually or just due to the headset's orientation, to induce tracking failure. This is highly relevant for augmented reality scenarios, where the capability of additional unrestricted movement through space would require a system that would be able to track the user's orientation in difficult situations.

The Kalman filtered data performed well in all scenarios. This was least noticeable for the standard 20 minute gameplay experiments where adjustments were on the order of tenths of a degree. We expect that as gameplay extends past 20 minutes, and yaw-drift accumulates, the filtered data will lay closer to the true orientation than the raw headset data. More interestingly, and confirming our assumption above, the Kalman filtered data also performed well in the severe, "non-standard" gameplay scenario. The headset's orientation data with magnetometer correction had drifted by approximately 20 degrees in the worst-case while the filtered data showed a drift of approximately 5 degrees for the same instance. Without the magnetometer correction, the headset's final drift for standard gameplay motion was approximately 25 degrees, while the filtered data reported a drift of approximately 8 degrees.

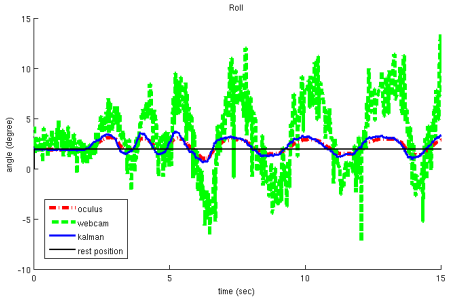
In Figure 5, we show three experiments with both the Oculus Rift (red), Camera Position (green), Kalman Filter (blue) and Rest Po-



(a) Pitch



(b) Yaw



(c) Roll

Figure 4: *The pitch, yaw and roll estimates of both the Oculus Rift sensors, Marker Tracking and Kalman Filter. It shows that the Marker Tracking is not so accurate for pitch and roll, while being accurate and smooth in yaw estimation of the Virtual Reality Helmet.*

	Desktop PC	Apple MacBook Pro
Network Latency	0.54 ms	
Marker Detection	9.085 ms	*8.194 ms
Color Marker Detection	**4.01 + 3.68** ms	**5.54 + 6.67** ms
GPU Feature Detection	na	**1.54 + 6.67** ms

Table 1: *The latency due to the network and computer vision (color marker detection or only marker detection), this table shows that the latency introduced by our solution is under the advised latency of 20 ms. It also shows that the color blob detection methodology is able to speed up the Marker Detection and an even better speedup can be achieved with the GPU color feature tracking method. In case of the combination of color detection and marker detection, first the speed of the color detection is given and than of the marker detection. [*] In this case, marker detection does not search on lower resolutions for markers.*

sition (black). Figure 5(a), the magnetometer in the Oculus Rift deals very well with the drift showing that all the signal are nicely aligned. Without the magnetometer (Figure 5(b)), a large drift in the Oculus Rift can be observed, where both the camera position and Kalman filter give a better estimate. Also in case of more severe movements (see difference in y-axis in Figure 5(c)), we observe more drift even with a magnetometer, where the marker detection and Kalman filter correct the drift in yaw as observed from the rest periodes.

5.3 Measurable Latency

The latency of the entire system is an important issue in Virtual Reality, where the Oculus Rift developers are aiming to get the latency to under 20 milliseconds[5]. Although the latency in rendering, streaming to screen and communication of sensor information is normally also present, we focused only on the additional latency parameters in our system. The biggest latency in our system can be expected in the network communication and the computer vision processing. In Table 1, the network communication and vision computation speed are shown. The Desktop Computer used in these experiments has as processor an Intel Core i7-3770 CPU @ 3.40GHz, while the Apple MacBook Pro has a Intel Core i7-3720QM CPU @ 2.60GHz. The Table shows that first performing Color Blob Detection and using only a small region for Marker Detection has a some speed gain over only using Marker Detection. Although for the Apple MacBook Pro there seems to be no speed gain, in reality the Marker Tracking library uses a trade-off between speed and resolution, where on the Apple MacBook Pro is does not find marker located further for the camera due to their lower resolution, which can be fixed by using Color Marker Detection.
In our case, the network latency is measure by using ping over 500 packages (mean = 5.27, median = 0.54) on a local network utilising a wifi connection.

6 Application Domains

We believe that there is still unexplored potential in using existing virtual reality headsets for low-cost shared augmented reality experiences. By fully exploiting all the capabilities of our yaw-drift corrected system we are able to track multiple users' position, height, and orientation using only additional cheap materials that are quick to set up. Additionally, the system is robust even when poorly constructed, and calibration in new lighting conditions only takes a few minutes.

Although we currently use a dedicated desktop or laptop machine for vision processing (see Figure 6), it would be easy to develop a portable solution using small-sized computers, like the single-board Raspberry Pi. Given the Raspberry Pi has similar GPU capability as mobile devices used in figure 2 we assume its processing capability is approximately up to the task. In order to accomplish a fully portable system, a portable power source would have to be provided for the headset itself.

Our system can be applied to several existing or upcoming headset-based virtual reality solutions. The Oculus Rift DevKit 2 will come with positional tracking through an external near Infrared CMOS Sensor intended to be placed in front of a seated user. The headset itself will come with infrared emitters on the face of the device but not on the top or back straps, restricting positional tracking to a viewing plane which is only functional when viewed directly from the front. This is sufficient for the seated experience the Oculus Rift is targetting, where the user is expected to remain relatively stable, but restrictive in comparison to our system. It is also uncertain whether a single CMOS sensor would be able to detect multiple users or whether the new positional data is being used to correct for yaw-drift. Our system could equally be used to cheaply enhance the DevKit 2's capabilities.

Finally, the crux of our solution is the real-time filtering of multiple streams of data. This allows some flexibility in our choice of tracking and detection method, as long as it can reliably detect the user's orientation and translation in real-time from an eagle-eye vantage point. For example, cost permitting, a spaced LED board solution could be used with a low-persistence camera to track bright light sources in low-light imagery, making the system more robust to changes in lighting conditions.

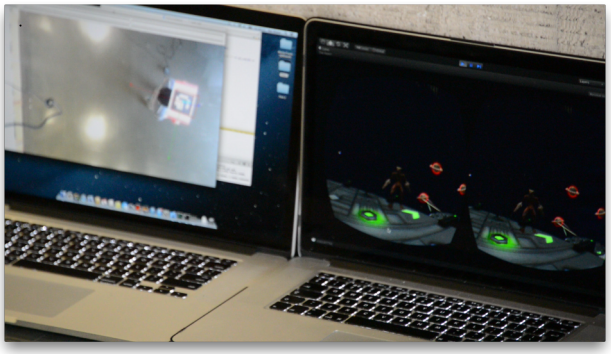
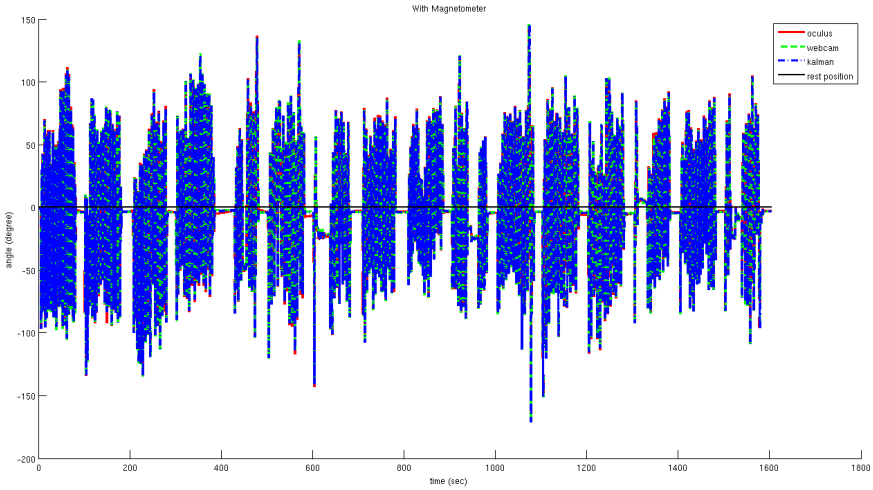


Figure 6: *Example of the system running on two MacBook Pros, where the user has a large space to walk around while in the Skye Wars game environment.*

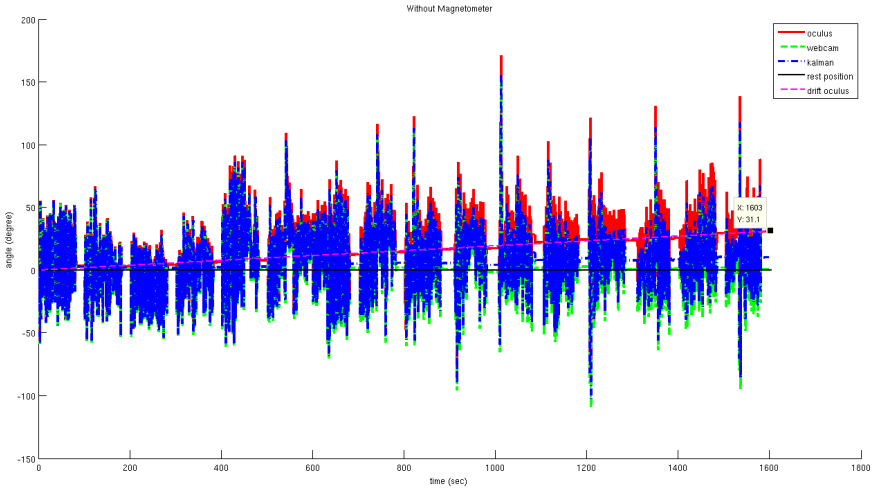
7 Discussion

Although our developers did not experience any significant amount of nausea (at least not more than to be expected from current virtual reality headsets), malaise in virtual reality systems is an endemic problem that must be addressed carefully. Due to time limitations, we were unable to run user studies to ensure our system's usability. However, what we have shown is that such a system is possible to construct in a relatively short amount of time with low-cost materials, but perfecting our system to consumer standards would take much longer. Future work may include extensive user studies and

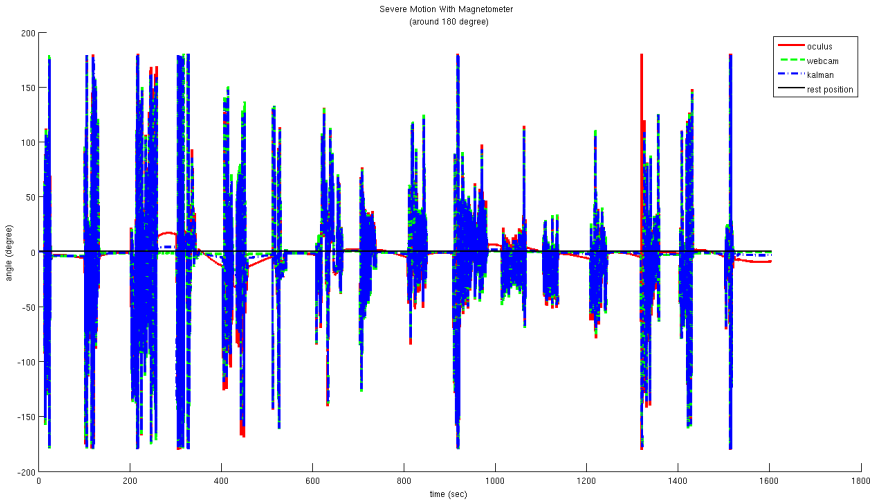
[5] http://oculusrift-blog.com/john-carmacks-message-of-latency/682/



(a) With Magnetometer



(b) Without Magnetometer



(c) Severe Motion With Magnetometer

Figure 5: *Yaw estimation is shown for both the Oculus Rift with and without magnetometer and with severe movements, in the top, middle and bottom graphs respectively. The top, middle and bottom graphs. The top figure shows that there is not much drift in Oculus Rift estimation and all signals are well aligned. The middle figure shows that in the rest periodes a much larger drift (see our purple line estimate of the drift nicely aligned with the red line) and demonstrates that the Kalman filter is able to correct for this drift. The bottom figure shows that severe movements also create more drift in the Oculus Rift where both the camera data and the Kalman filter correct for this drift.*

algorithmic tweaking to develop a solution with comparable accuracy to cumbersome or costly ground-truth alternatives. In addition to user studies, more extensive testing is required to ensure our solution's stability in a variety of gameplay scenarios. Users will typically follow the motions encouraged by the individual game, which may not be as encompassing as the full range of movements and environments we would like to test against. On the other hand, the gameplay can also limit the allowable positional movement by, for example, placing the gamer on a platform surrounded by a deadly pit.

The effectiveness of different real-time filtering and tracking algorithms could also serve as the body for future publications. Our current system uses an adaptation of the Aruco marker tracking library enhanced with fast feature tracking based on color. In this case, integration of the color blob detection with the marker detection can be improved using only the resolution suggested by the color detection. We also use a simple Kalman filter to adjust the headset's orientation data over time with the camera's tracking data, which could subsume other methods such as Low/High Pass Filter combination or Extend Kalman/Particle Filter.

The Oculus Rift DevKit 2 comes with built-in USB ports, which will ease the development of a portable system based on LED lights, ultrasound emitters, or other client-side powered tracking methods. In this case, they still focus on a seated user. Our focus, however, is on developing a very low-cost solution that doesn't require special ceiling markers or expensive tracking devices that are generally not available at the consumer level. Another low-cost alternative for tracking would be to construct a cheap sphere dipole or tripole [Greenspan and Fraser 2003] with different colored spheres for general spherical color tracking [Sỳkora et al. 2008] or the fast-feature color tracking used in our current study. We expect that this system would be less susceptible to failure on steep orientation angles in pitch and roll. Additionally, it may be less cumbersome to attach to the headset and it would be easier to account for steep angle failure by adding more cheap color markers at problem orientations.

References

BRADSKI, G. R. 1998. Real time face and object tracking as a component of a perceptual user interface. In *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on*, IEEE, 214–219.

DK1, O. R. Oculus rift-virtual reality headset for 3d gaming. Available: http://www.oculusvr.com.

FIALA, M. 2005. Artag, a fiducial marker system using digital techniques. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, IEEE, 590–596.

FIALA, M. 2010. Designing highly reliable fiducial markers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 32*, 7, 1317–1324.

GARRIDO-JURADO, S., MUOZ-SALINAS, R., MADRID-CUEVAS, F., AND MARN-JIMNEZ, M. 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition 47*, 6, 2280 – 2292.

GREENSPAN, M., AND FRASER, I. 2003. Tracking a sphere dipole. In *16th International Conference on Vision Interface*.

KALMAN, R. E. 1960. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering 82*, 1, 35–45.

MAESEN, S., AND BEKAERT, P. 2011. Scalable optical tracking-a practical low-cost solution for large virtual environments.

MAESEN, S., GOORTS, P., AND BEKAERT, P. 2013. Scalable optical tracking for navigating large virtual environments using spatially encoded markers.

PAUSCH, R., SNODDY, J., TAYLOR, R., WATSON, S., AND HASELTINE, E. 1996. Disney's aladdin: first steps toward storytelling in virtual reality. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, 193–203.

PÉREZ, P., HUE, C., VERMAAK, J., AND GANGNET, M. 2002. Color-based probabilistic tracking. In *Computer visionECCV 2002*, Springer, 661–675.

POHL, D., JOHNSON, G. S., AND BOLKART, T. 2013. Improved pre-warping for wide angle, head mounted displays. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, ACM, New York, NY, USA, VRST '13, 259–262.

SIMON, M., BEHNKE, S., AND ROJAS, R. 2001. Robust real time color tracking. In *RoboCup 2000: Robot Soccer World Cup IV*. Springer, 239–248.

SKYEWARS. 2013. Skyewars [apple itunes appstore game]. *40th SIGGRAPH Computer Animation Festival, Retrieved from http://farpeek.com/index.php/apps/7-skyewars*.

SỲKORA, D., SEDLÁČEK, D., AND RIEGE, K. 2008. Real-time color ball tracking for augmented reality. In *Proceedings of the 14th Eurographics conference on Virtual Environments*, Eurographics Association, 9–16.

TOMILIN, M. 1999. Head-mounted displays. *Power 10241024*, 10241024, 640480.

WANG, J.-F., CHI, V., AND FUCHS, H. 1990. *A real-time optical 3D tracker for head-mounted display systems*, vol. 24. ACM.

WARD, M., AZUMA, R., BENNETT, R., GOTTSCHALK, S., AND FUCHS, H. 1992. A demonstrated optical tracker with scalable work area for head-mounted display systems. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, ACM, 43–52.

WELCH, G., BISHOP, G., VICCI, L., BRUMBACK, S., KELLER, K., ET AL. 1999. The hiball tracker: High-performance wide-area tracking for virtual and augmented environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, 1–ff.

WILLIAMS, L. 1990. Performance-driven facial animation. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '90, 235–242.

ZHANG, X., FRONZ, S., AND NAVAB, N. 2002. Visual marker detection and decoding in ar systems: A comparative study. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, IEEE Computer Society, 97.