Envisor: Online Environment Map Construction for Mixed Reality

Stephen DiVerdi [a,b,*], Jason Wither [a], Tobias Höllerer [a]

[a] *Four Eyes Laboratory, University of California, Santa Barbara, CA 93106*
[b] *Adobe Systems Inc., 345 Park Ave, San Jose, CA 95110*

Abstract

One of the main goals of anywhere augmentation is the development of automatic algorithms for scene acquisition in augmented reality systems. In this paper, we present Envisor, a system for online construction of environment maps in new locations. To accomplish this, Envisor uses vision-based frame to frame and landmark orientation tracking for long-term, drift-free registration. For additional robustness, a gyroscope / compass orientation unit can optionally be used for hybrid tracking. The tracked video is then projected into a cubemap frame by frame. Feedback is presented to the user to help avoid gaps in the cubemap, while any remaining gaps are filled by texture diffusion. The resulting environment map can be used for a variety of applications, including shading of virtual geometry and remote presence.

Key words: vision based tracking, environment mapping, wearable computing, anywhere augmentation

1. Introduction

The quality of augmentation possible with existing augmented reality (AR) technologies has drastically improved in recent years, thanks to advances in tracking, modeling, and rendering techniques. However, this improved quality often carries the price of increased startup effort, as these new techniques require expensive hardware and careful measurement, calibration, modeling and instrumenting of the environment before they can be used. These setup costs form a barrier to entry that hinders experimentation with AR technology by potential casual users. The goal of *Anywhere Augmentation* is to develop algorithms and applications that reduce or eliminate these startup costs by using commonly available components, ubiquitously available data sources, and online data acquisition algorithms.

One important component of modeling new scenes is the acquisition of an environment map. As an image-based representation of the light distribution around a single position, environment maps have many uses in AR systems. Most commonly, they can be used for realistic shading of virtual geometry [1,9,13] for more seamless integration of virtual objects into the physical scene. They are also useful for remote presence applications [26], as a simple way of representing a remote environment, e.g. as a backdrop in a tele-collaboration system, or in low-bandwidth first-person interfaces like QuickTime VR models [19].

In this paper, we present Envisor, a system for the automatic, online construction of environment maps using a hand-held or head-worn camera. Envisor tracks the camera's orientation using optical flow of sparse corner features for relative camera motion, and dynamically acquired landmark features to provide long-term drift-free registration. If the system is likely to be used in environments where reliable vision data cannot be depended on continuously (e.g. large occlusions, highly dynamic scenes, heavy motion blur), the camera can optionally be integrated with an auxiliary sensor such as

[*] Corresponding author.
Email addresses: stephen.diverdi@gmail.com (Stephen DiVerdi), jwither@cs.ucsb.edu (Jason Wither), holl@cs.ucsb.edu (Tobias Höllerer).

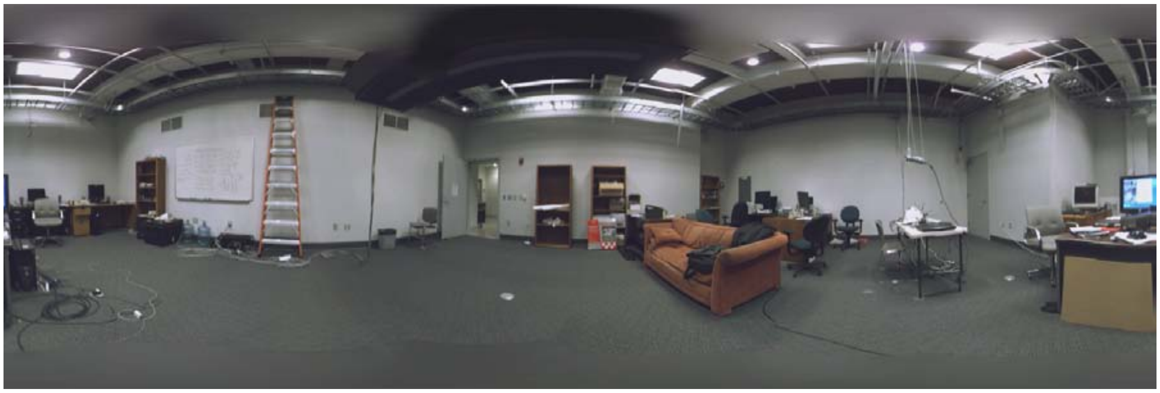

Fig. 1. A cylindrical projection of an environment map constructed using Envisor with a camera on a tripod.

a gyroscope / compass orientation tracker for increased robustness, though this is not a requirement.

To construct an environment map, Envisor's tracked video stream is projected into a cubemap frame by frame. Portions of the frame that are determined to be dynamic are masked from the projection, in an attempt to store only the background elements in the environment map. As a user may forget exactly which portions of the scene still need to be acquired, Envisor provides visual feedback in the form of arrows that indicate to the user which regions remain. To combat the inevitable gaps that will still arise, texture diffusion is used to smoothly fill in gaps to reduce their visual impact.

The contributions of this work are the combination of landmark and frame to frame components in the vision-based orientation tracking employed by Envisor, and the Envisor application for constructing environment maps online, automatically. As a demonstration of the usefulness of these contributions to the AR community, Envisor can use its acquired information to shade virtual geometry so it appears lit by the physical scene. We analyze the performance and impact of error on Envisor in a simulator and then confirm these results by capturing real panoramas with both hand-held and tripod-mounted cameras in indoor and outdoor environments (see Figures 1, 15, 16, and 17).

2. Related Work

Environment map acquisition is an important component of photorealistic AR applications. The most common approach used is to take one or more carefully calibrated photographs of a mirrored sphere, which can then be processed offline before being used for rendering [1,9]. Other options include using a camera with a fish-eye lens [13], or using an omnidirectional camera [26]. However, these approaches are all contrary to the goals of Anywhere Augmentation, because of the required special hardware, careful measurement, and offline preprocessing. Envisor works online and automatically with just a regular video camera.

Established techniques for panorama stitching or mosaicing [12,25,24,5] focus on rectifying the transformation across each entire image, over the full set of images. This creates globally optimal panoramas, but requires an offline solution, as all data must be available for processing simultaneously. There are systems that provide realtime mosaic results while waiting to compute a global refinement [2,22], by only computing local refinements online. This results in significant errors across the panorama that cannot be resolved without global optimization. Global solutions pose a problem for online acquisition systems that use video feeds for data, as all the images must be stored separately so they can all be processed and adjusted in the final optimization step. For sparse sets of images, storage is not a problem, but storing each frame of a tracked video and then computing a global solution among them is prohibitive. The goal of Envisor is to achieve tracking of sufficient quality to make global optimization unnecessary, enabling it to greedily project each video frame as it comes in, making the running storage requirement just a cubemap.

In the context of augmented reality, there is ample previous work on landmark vision-based and hybrid tracking systems. The basic approach in general is to use vision-based methods for landmark feature recognition, combined with gyroscopes for robustness. An earlier system that uses a silhouette of the horizon as a stable landmark for vision only orientation tracking was presented by Behringer [4]. More recently, Satoh et al. [21] presented an outdoor orientation tracking system that uses user-specified patches of image texture as landmarks, fused with a gyroscope. You and Neumann

[27] demonstrated a position and orientation tracker that uses offline acquired landmark features and a gyroscope in an Extended Kalman Filter framework. Most recently, Reitmayr and Drummond [20] introduced a robust 6DOF outdoor hybrid tracking system that matches video frames to a pre-acquired scene model. The limitation of each of these systems it that they depend on offline measurement of the scene before they can be used. This requirement sets up a barrier to entry that hinders casual use of these tracking solutions.

Most similar to Envisor is the work of Montiel and Davison in visual compassing [17]. They build off previous work on single camera simultaneous localization and mapping (SLAM) [6], using a complex Extended Kalman Filter formulation of the tracking problem to compute orientation from dynamically acquired landmark features. This work is significantly different from the algorithm we present, primarily in that our approach utilizes a more modular design, combining two configurable tracking modalities to achieve similar tracking performance. Additionally, the use of RANSAC and a larger number of simple features per frame suggests that Envisor will exhibit greater robustness to dynamic scene elements, though a direct comparison is not available.

3. Vision-based Tracking

The first contribution of this paper is the vision-based tracking technique developed to provide drift-free orientation registration. The tracking uses two separate measures, a frame to frame relative rotation and a landmark-based absolute orientation, which are combined in the final result. While the techniques described here focus on 3DOF tracking, the results are applicable to full 6DOF camera pose tracking as well.

Before tracking, the camera's intrinsic parameters must be measured in a one-time offline calibration procedure. The OpenCV [14] implementation of Zhang's technique [28] is used for this purpose, which measures the focal length, center point and radial distortion of the camera. The distortion parameters are used to correct the position of features in the image, as well as to undistort each frame on the GPU so the image will match the pinhole perspective model of OpenGL.

3.1. *Frame to Frame*

The first half of the vision-based tracking algorithm computes a relative rotation between two consecutive frames from the differences in position of a matching set of feature points in the two images. Initial features are found using Shi and Tomasi's [23] good features operator, which greedily selects a set of "corner" features, where corners are defined as image patches with strong gradients in two directions. The motion of these features between consecutive frames is determined using a pyramidal version of Lucas and Kanade's optical flow algorithm [16], which uses a hierarchy of different resolution images to efficiently match sparse image patches between two frames. See Figure 2(a) for a visualization of these features. As features are lost (moved out of the field of view, or could not be tracked), new features are added incrementally when the number of features drops below a threshold.

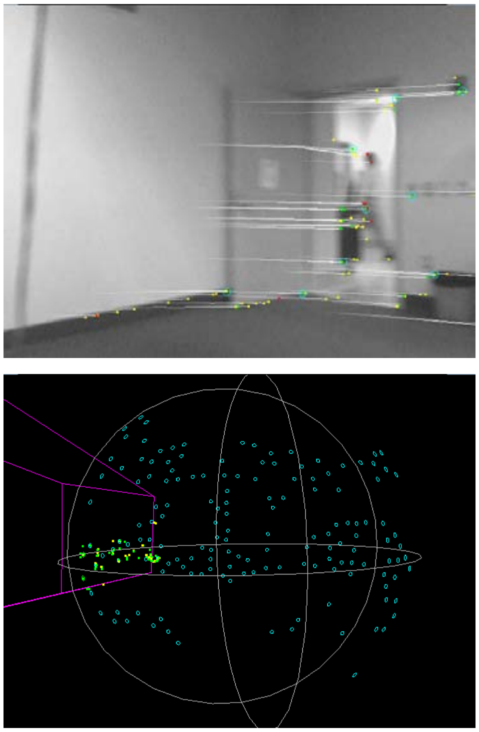

Fig. 2. *Top to bottom:* (a) A frame from a video with the tracked features drawn on top. Each point is a feature in the frame to frame tracking – red are outliers, yellow have been inliers for a few frames, and green have been inliers for many frames. Features with a circle around them have been put in the landmark map. The trail on each feature shows its recent history. (b) A visualization of the landmark map after the camera has been swept across a full hemisphere. The cyan circles show the 3D positions of each recorded landmark. The purple frustum shows the camera's current viewing volume, and the red, yellow and green points are the features currently being tracked.

With OpenGL's pinhole camera model, it is trivial to project 2D features on the image plane into 3D points on the viewing sphere. The optimal rotation between corresponding sets of 3D positions is computed using Horn's absolute orientation technique [11], which works by constructing a special 4x4 matrix from the set of feature pairs. This matrix has the property that its maximal eigenvector is a quaternion representing the optimal rotation. This approach has many nice qualities. It is a non-iterative technique that does not require further refinement and only trivial quaternion normalization as a post-processing step, though even this is not strictly necessary. The computation is very fast, and takes the same amount of time for small or large numbers of feature pairs. This is especially useful as it reduces the impact of tracking with large numbers of features. It also provides for easy integration in a RANSAC [8] implementation as both the estimation and refinement steps, which is necessary for robustness against likely outliers in dynamic, real-world environments.

3.2. *Landmarks*

The second half of the vision-based orientation tracking adds landmarks to the frame to frame feature tracking system to combat drift during long tracking runs. Drift is a problem because integration of the frame to frame tracking's relative measurements accumulates the small errors in each measurement, which can create large differences between the estimated and actual orientations. Identifying and reusing absolutely positioned landmark features addresses this problem by providing a periodic direct measurement of absolute orientation rather than relative rotation.

Existing landmark based tracking systems [6] use some sort of uniquely identifiable feature such as large image patches, or SIFT [15] or SURF [3] features for landmarks. These heavyweight features are used to recognize when the tracker is revisiting previously seen regions, as well as during the frame to frame update of currently visible features. This approach can be simplified, as the uniquely identifiable nature of landmarks are not necessary for frame to frame updates – since landmarks will agree with the motion of the rest of the scene, they will be inliers in the lightweight frame to frame tracking result discussed earlier. Therefore, optical flow based tracking is sufficient for the feature update step, and the utility of landmarks is only to uniquely identify features. Additionally, since a landmark feature does not change from when it starts being tracked to when it leaves the field of view, the landmark identification only needs to happen during feature initialization.

To first create landmarks, features that are inliers for a number of consecutive frames are promoted if they are far enough apart from existing landmarks, which are stored in a set called the map (see Figure 2(b)). Each landmark has its associated world coordinate direction vector and a feature descriptor. When a landmark is created, the patch around it in the image is used to create a SURF descriptor, using the code provided by Bay, Tuytelaars, and Gool [3]. The result is a 64 float vector that uniquely identifies that patch of image texture.

Once landmarks are in the map, they must be reacquired when they come back into the camera's field of view. If a landmark is expected to be in the field of view (by projecting known landmark locations to the camera's estimated orientation), the landmark is searched for in a small region about its expected location. To do this, Shi and Tomasi's good features operator is used to find candidate points inside a small search region, and then SURF descriptors are computed for each of those features. These descriptors are compared to the landmark descriptor and if a match is found, the feature is linked to the landmark and entered into the frame to frame tracker. After a certain number of failed attempts to reacquire a landmark feature, the landmark is determined to be lost and is removed from the map. Matching descriptors are determined by normalizing the two descriptors to have a magnitude of one and then computing the dot product, which is thresholded.

During tracking, the landmarks are used twice, once as part of the regular frame to frame rotation estimate, and separately to find an orientation estimate from just the landmarks. This separate estimate uses the same algorithm as the frame to frame tracking, but instead of computing the rotation between each landmark's position in the previous and current frame, the rotation from the landmark's world stabilized position to the current frame position is generated. RANSAC is still applied, because while landmarks are assumed to be static features, they may still change – for example, a landmark feature may be on a parked car, but after some time the car may drive away and the landmark will have changed. Landmarks that are outliers a certain number of times will eventually be discarded.

3.3. *Hybrid Tracking*

The two vision-based tracking modalities - frame to frame and landmark tracking - are combined to produce a higher quality final tracked result. The landmark vision tracking provides an absolute orientation, but is not available each frame as enough landmarks may not be visible. The frame to frame vision tracking provides an angular velocity estimate with a low amount of error assuming slow camera motion, but integration over time creates drift. When a landmark measurement is available, it is used as the current orientation estimate, and when enough landmarks are not available, the relative measurement of the frame to frame tracking is integrated into the previous orientation estimate. This way, the benefits of both modalities are gained in the final tracker.

Since vision-based tracking is not always reliable in some environments (with insufficient texture or highly dynamic content), other sensors can optionally be used as well when the vision sensors fail. For example, an InertiaCube 2 provides an absolute orientation measurement that is always available but has greater error than the vision measurements. It could be used when the vision tracking fails to improve robustness. A simple way to integrate it would be to use its relative rotation measurements to update the orientation estimate when vision-based tracking is not available. More sophisticated sensor fusion could also be employed, but we leave this for future work.

4. Environment Map Construction

The second main contribution of this work is the Envisor application, which uses the tracked video stream to create an environment map of the surrounding scene online and fully automatically. The first step of this process is to project each frame of the video into a cubemap based on the tracked orientation estimate. To guide the acquisition process and reduce the likelihood of missed regions, feedback is given to the user to indicate which regions of the scene still need to be acquired. Finally, since small gaps will be unavoidable, a texture diffusion process is used to blend surrounding pixels into those gaps, reducing their visual impact.

4.1. *Cubemap Projection*

Envisor uses an OpenGL cubemap to store the generated environment map. Each frame, the orientation and intrinsic parameters of the camera are used to render the video image into the cubemap using a frame buffer object. This is done by computing the direction vectors from the camera's position to the corners of the camera image and drawing a texture mapped quad into each cubemap face using those direction vectors as the corners of the quad (cubemaps are indexed by direction vector, so this puts the projected image in the correct position). An alpha mask is used that gradually falls off around the border, to make sure each frame smoothly blends into the cubemap, reducing the jarring effects of any inconsistencies.

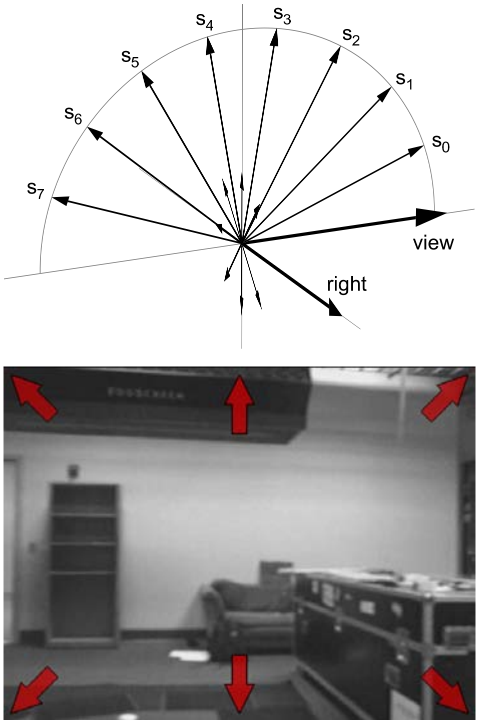

Fig. 3. *Top to bottom:* (a) Gap searching proceeds by rotating the view vector around each of 8 evenly-spaced vectors perpendicular to the view direction. Here, the view vector is rotated about the right vector. Along each sample vector s_i, the cubemap is sampled to see if there is a gap. (b) Eight arrows are drawn around the periphery to show the user where gaps remain. The intensity of an arrow is based on how many of the sample vectors along that direction detected gaps. In this example, the user has just completed a circular sweep, so there are no gaps to the left or right.

4.2. *Gap Avoidance*

Enabling user avoidance of gaps means giving the user useful feedback during the acquisition process that allows him or her to more intelligently direct the camera. In a wearable context, a user interface that simply presents the user with the cubemap and allows panning around the view is too complex, as it would require significant cognitive load and manipulation of the wearable input device. Instead, the feedback should be more tailored to low cognitive load with no interaction requirement. Envisor presents a passive display of a set of arrows around the current view that indicate which directions gaps are present along. See Figure 3(b) for an example image. As there are fewer unfilled pixels along a certain direction, that arrow will become more transparent until it disappears when all the gaps are filled.

Creating these arrows requires efficiently sampling the pixels along each direction and testing for gaps (to make this determination easy, the starting cubemap is

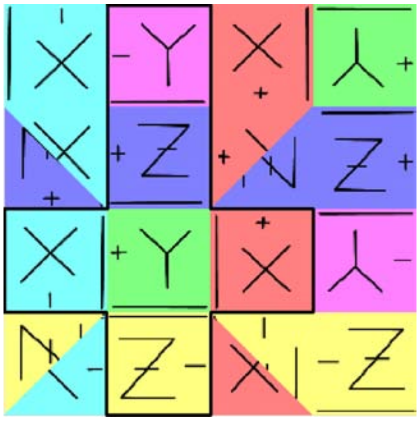

Fig. 4. Layout of the cubemap faces in the atlas texture. The black outline marks where the cubemap faces are sampled from when applying the texture diffusion.

set to black with alpha values of 1, and when pixels are drawn in from the video image, their alpha values are set to 0). To sample the cubemap along each direction, the camera's view, right and up vectors (d, r and u respectively) are extracted from the camera's extrinsic pose. The right and up vectors can be used to create 8 cardinal directions around the viewing direction. Then for each of these axes, the view vector is rotated about the axis incrementally in the range of [0..180] degrees (see Figure 3(a)).

An obvious way to implement this would be to read back the pixels from the cubemap along the sample vector directions, but this pixel readback is extremely slow as it breaks GPU pipelining by introducing a stall. Since the panorama construction has heavy CPU and GPU use, good pipelining is critical for performance, so keeping the computation on the GPU is important. To accomplish this, the cubemap is sampled by drawing a point with the sample vector as its texture coordinate. The series of samples are combined by blending with an additive blend function, and a simple fragment shader is used to test the sampled values to see if they are gaps or not and output 1 or 0 appropriately. The result is 8 pixels in an offscreen buffer, each with an alpha value that represents the weight for the corresponding direction. Then when drawing the arrows in the user interface, the geometry simply has this texture applied with the appropriate pixel's coordinate passed as the texture coordinates for the entire arrow.

4.3. *Gap Filling*

Even with user feedback, gaps are still likely to occur, though they may be smaller. It is important to do something to fill these gaps, as they create a very distracting visual artifact (the appearance of obvious "holes" in the environment map and shaded geometry). The fast image inpainting algorithm of Oliveira et al. [18] is a good candidate to use for the gap filling, but it requires user input to place diffusion boundaries. The technique can be adapted to be fully automatic however, with just the diffusion component and no boundaries. The advantages of this approach are numerous. First, it can easily be implemented on the GPU, which is in line with the goal of avoiding readback of GPU data to the CPU. Second, it is an incremental algorithm, so a little work can be done each frame without impacting the overall performance. And finally, a modified version can be made to gracefully handle updates of gap regions as new projected pixels are incrementally filled in. As this will happen often, it is important for the gap filling to be able to respond to the new data immediately.

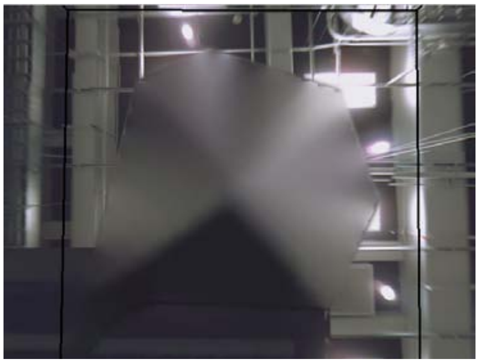

Fig. 5. An example of gap filling. At the end of an acquisition, gaps may remain, especially around the north and south pole. Here, the north pole gap is filled.

The basic approach is to use a fragment shader that implements a diffusion function. Pseudocode for the shader is as follows.

```
center = sample texture at center of kernel
avg = 0
count = 0
dist = MAX
for each pixel in kernel
{
  samp = sample texture at kernel pixel
  dist = min( dist, samp.a )
  if( samp.a <= center.a )
  {
    avg += samp
    count += 1
  }
}
avg /= count;
avg.a = dist + 1
if( count == 0 || center.a < avg.a )
  output = center
else
```

```
    output = avg
```

This shader operates as follows. First, all pixels in the cubemap are initialized with an alpha value of 1, while projected pixels are set to have an alpha value of 0. During diffusion, the alpha value is used to encode the distance from a filled pixel – pixels with alpha 0 are filled, pixels with alpha 1 have not been diffused into yet, and alpha values in between designate the distance that pixel is from the nearest filled pixel. The shader computes an RGB value that is the average of all the surrounding pixels that have a distance less than or equal to the distance of the center pixel. It also finds the minimum distance to the center pixel. The new output value is then the average RGB and the minimum distance plus one for the alpha value.

One advantage of this implementation is that it does not repeatedly recompute pixels that have already been diffused, which would result in their colors slowly fading due to the global average value. Also, by using the distance as part of the criteria for deciding to update a pixel, when new regions of the image are filled, the new smaller distances of nearby pixels will insure they get updated. See Figure 5 for an example.

The remaining difficulty is to implement the diffusion on a cubemap, with correct diffusion between cubemap faces. The solution is to do the diffusion in a regular 2D texture that has been filled in with the cubemap faces laid out so the boundaries of the embedded faces meet as they do on the actual cube. The particular layout used is from Gu [10], which can be seen in Figure 4. Before a diffusion step is computed, the cubemap is drawn into the atlas texture as shown. Then the diffusion is computed by drawing each face back into the cubemap with the diffusion shader enabled, sampling from the atlas texture. This way pixels at the boundary of one face will correctly sample from the abutting faces as needed.

To save on performance, the texture diffusion is actually computed on a subsampled cubemap with faces sized 64x64 pixels (while the full resolution cubemap is 512x512 or greater). As the diffused texture is very low frequency, this does not impact the visual quality.

5. Results

See Figure 1 for an example of an environment map constructed by Envisor from a camera on a tripod. While a few misregistrations are evident, particularly in regions that are close to the camera, they are minor. Since the camera was on a tripod, it was unable to acquire the scene directly above or below its position, so the texture diffusion process has filled in those gaps.

stage	**Athlon**	**Xeon**	**Pentium**
video decoding	13.0	8.5	11.2
undistortion	0.3	0.3	0.3
preprocessing total	13.3	8.8	11.5
KLT tracking	24.5	15.7	28.6
RANSAC	0.3	0.5	0.7
landmarks	40.1	33.9	24.5
tracking total	65.2	50.5	54.2
cubemap update	2.7	2.3	3.7
total	81.2	61.6	69.5

Table 1
Average times (in ms) of the various stages of Envisor, on three computers. The preprocessing and tracking are broken up into their component stages, and timings are presented for each stage as well as the frame total. The final total is the start to finish for each frame of the test application.

To demonstrate an application of the constructed environment maps, we implemented a simple material shader that uses a blurred environment map as a glossy reflectance map for rendering virtual geometry [7]. See Figure 6 for an example of virtual geometry shaded using the panorama from Figure 1.

5.1. *Performance*

This work was tested on three machines: a desktop with a 2.1GHz AMD Athlon CPU and an NVIDIA GeForce FX 6200 graphics card, another desktop with a 3.0GHz Intel Xeon and an NVIDIA GeForce 7800 GS, and a laptop with a 2.0GHz Intel Pentium M CPU and an NVIDIA GeForce Go 6600. The camera used was a Unibrain Fire-i400 camera with a 4mm lens. In general, we experience around 15 frames per second in the testing application, which runs off of a pre-recorded MPEG encoded video and accompanying metadata file. See Table 1 for more detailed timing data. The GPU implemented steps of the technique are not accurately represented in the timing data because of the difficulty in accurately measuring the stages separately given the GPU's heavy pipelining. Because of the way the GPU stages are implemented however, they are able to completely overlap the CPU portions of the algorithm, and so do not impact the final per-frame running time. This was confirmed by comparing timing data with and without the GPU components of the application, which were not significantly different.

Unfortunately, some of the steps that take the most time are fixed costs. When SURF features are used for

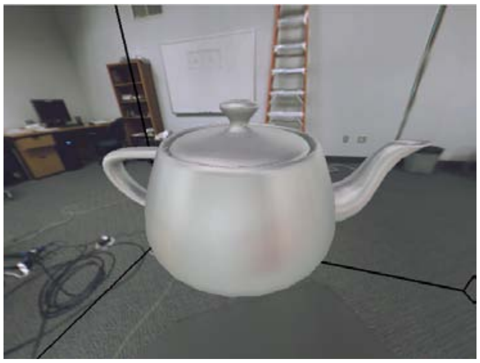

Fig. 6. Virtual geometry shaded using the acquired environment map from Figure 1. The environment map is filtered first to create the appearance of a silver material with a glossy finish. The teapot is superimposed over the acquired environment map.

landmarks, there is a step to compute the integral image of the video frame. This expensive operation is performed even if only one SURF descriptor is needed. However, better performance can be achieved by not doing every step of the algorithm every frame. For example, only looking for new landmarks every 3 frames yields a large improvement to the average framerate. Similarly, one of the slowest components of the KLT tracking is initialization of new features, which can also be done every few frames. By distributing periodic workloads across frames (interleaving KLT feature initialization and landmark searching, e.g.) the performance can be increased. How aggressively this can be done depends on the expected speed of camera motion and dynamic nature of the scene. For faster camera motion, features will be in the field of view for fewer frames, and so all tracking operations must happen frequently. More dynamic scenes will need better robustness to outliers, which requires more features processed more frequently. These considerations are important on a per-application basis.

The ability to tune the performance of the frame to frame and landmark feature tracking separately is an additional advantage of the approach to tracking used in this work. The level of configurability to particular application needs is very high – for example in a scene with consistent strong texture and very few dynamic elements, landmark tracking updates by themselves may be sufficient, without relying on frame to frame measurements to fill gaps. Alternately applications such as fully immersive VR or games that only needs angular velocity input could use only the frame to frame updates without the landmark corrections. This advantage is in contrast to black box tracking solutions that only rely on one tracking modality.

5.2. *Tracking*

For testing purposes, we used a selection of different cameras: a Unibrain Fire-i, a Unibrain Fire-i400, and a Point Grey FireFlyMV. They cover a range from consumer level to mid-range lab cameras. Mostly, we used the Fire-i400 because it has the widest field of view, at 51°. A wider field of view means the feature tracking is less likely to get distracted by large occluders such as a person walking by, and that potentially faster motion can be tracked. It also improves the robustness of the tracking against regions of uniform texture, as such regions will have to be much larger to fill the camera's field of view. However, the most significant impact of the wider field of view was that it made environment map construction much faster as fewer sweeps around the scene were needed, which definitely improved the usability of Envisor.

The accuracy of the tracking was tested by moving the camera through a circular sweep at roughly 20°per second, on a tripod in a large room under ideal tracking conditions. The tracking is accurate enough that after completing the loop, the camera is off of the original orientation by 0.2°. This is significant, as it means that the characteristic discontinuity at the end of a closed-loop in panorama stitching, which generally requires a global refinement to the stitching, is not as important for Envisor. This accuracy is also sufficient for the landmarks from the beginning of the sweep to be reclaimed as they come back into view. This means that as long as good tracking conditions are maintained, Envisor is capable of long-term drift-free orientation tracking. However, because the landmark features are originally initialized off of the frame to frame tracking results, if the relative orientation gets distracted the landmarks will incorporate that error into their positions. If this error is too large, the reacquisition of old landmark features will fail, causing them to be discarded and new landmarks acquired in their place. Because of these limitations, the vision based tracking alone is not robust to poor tracking conditions such as total occlusion. Under good conditions, the tracking is successful indefinitely.

5.3. *Environment Mapping*

The quality of the resulting environment map from Envisor depends heavily on the quality of the tracking data obtained. People are very sensitive to small registration errors when tracking results can be compared directly, side-by-side, as they are in an environment map at the borders between projected frames. Visible gaps and jumps negatively affect the appearance of a panorama. While the tracking presented here is able to rely on a variety of different modalities, only the frame to frame relative updates create seamless blending within the environment map, as they directly compute the optimal transform between two frames. If there are errors in the tracking from bad video data or random noise in the landmark orientation measurement, this will result in discontinuities in the environment map. However, depending on the target application, these discontinuities may not be a problem. For example, applications that use environment maps as a backdrop may find small errors acceptable. Shading of virtual geometry that is not completely specular and smooth will also not be adversely affected by these errors.

One of the problems facing environment map construction is the changing exposure and white balance of automatically adjusting cameras. As a camera moves from a bright region to a darker one, or vice-versa, it takes some amount of time to adjust to the new illumination, which means that revisiting the same portion of a scene may result in different pixels values than were previously acquired. This problem is evident when the camera is re-swept over a region, and the border between the old and new data is clearly visible due to brightness and hue differences. In low dynamic range environments, such as an office with fluorescent lighting that creates significant ambient illumination, the camera's automatic adjustment can be turned off with no ill effects (see Figure 1). However, in high dynamic range environments such as outdoors or indoors with very localized light sources, the automatic exposure adjustment is important for tracking because it ensures that the image features always have good contrast. Over or under exposure will reduce the quality of the computed optical flow, hurting tracking performance. Alternately, if a camera supports reading the adjusted parameters per-frame, a color model can be fit to these parameters that would allow manual normalization of the images on the CPU or GPU, so the tracking can always have an optimal exposure image, while the environment map always has normalized intensities. Unfortunately, our cameras do not support this feature.

6. Error Analysis

In scenes with good conditions for vision-based tracking (with sufficient texture for optical flow and enough coherent motion for RANSAC), the most significant sources of error are due to motion blur and translation of the camera. Other contributing factors are image noise and distractions in the scene.

6.1. *Motion Blur*

Motion blur is caused by the camera moving too quickly within the scene, stretching point features into lines across the frame. Theoretically, the maximum rate of rotation that can be tracked is limited by the camera's field of view and the framerate. For our testing setup, we used a 51°field of view camera and had a framerate of 10Hz. If half the image needs to remain in the field of view between consecutive frames for tracking to succeed, then that results in a theoretical maximum angular velocity of 255°per second. Realistically, motion blur causes optical flow to fail at much lower speeds. In practice, we find that angular velocities of up to 60°per second can be tracked by the frame to frame tracking, while landmark tracking is successful at angular velocities of up to 30°per second. The reasons for the slower maximum for landmark tracking are that the landmarks require features to be tracked successfully for a number of consecutive frames before they can be promoted to landmark status, and that the blurring decreases the quality of the computed SURF descriptor, which interferes with reinitialization. The most effective way to increase the maximum trackable angular velocity is to lower the camera's exposure time, reducing the motion blur effect. High speed cameras and brightly illuminated scenes will both improve this result.

6.2. *Camera Translation*

Translation error is introduced because the assumption that the camera rotates about its optical center is not correct, especially for hand-held or head-worn cameras. To quantify the effect this has on the tracking, we can estimate the relationship between translation and measured rotation. Let p be a 3D point at the center of the camera image plane at distance n from the camera, corresponding to a 3D point P that is distance D from the camera. If the camera undergoes a translation of distance T, perpendicular to the viewing direction, then the point P will appear to move T in the opposite direction (in camera coordinates), becoming Q, with corresponding image plane point q. See Figure 7 for an illustration. The distance the point in the image plane will have appeared to move is

$$\overline{pq} = \frac{nT}{D} \tag{1}$$

If the camera had instead rotated by θ degrees about an axis perpendicular to the viewing direction, the distance between p and q would instead be

$$\overline{pq} = n \tan\theta \tag{2}$$

Therefore, for a translation T, the apparent rotation θ can be computed as

$$\theta = \tan^{-1}\frac{T}{D} \tag{3}$$

Pure translation is unlikely, however. More likely is that the camera will rotate about a point that is not the optical center, which will cause both rotation and translation simultaneously (see Figure 8). Assume the pivot about which the camera rotates is distance d in front of the camera's optical center (negative values of d mean the pivot is behind the optical center). Then for a rotation of θ, the translation of the camera T will be

$$T = 2d\sin\frac{\theta}{2} \tag{4}$$

This translation T will cause an additional rotation measurement of $\delta\theta$, and total measured rotation θ_m

$$\theta_m = \theta + \delta\theta = \theta + \tan^{-1}\frac{2d\sin\frac{\theta}{2}}{D} \tag{5}$$

For small values of θ, $\sin\theta \approx 1$ and for small values of x, $\tan^{-1}x \approx x$. Therefore, for small rotations,

$$\theta_m \approx \theta + \frac{2d}{D} \tag{6}$$

The significance of this approximation is that the error is based only on the ratio of the distances from the pivot to the camera and the camera to the 3D feature. This result was confirmed with synthetic experiments, in which an off-center virtual camera was rotated within a set of points spaced 5° apart on a 10m sphere. The camera's pivot-offset was varied from 5mm to 5m, and it underwent 360 1° rotations in each configuration. Results can be seen in Figure 9, which roughly agrees with the predicted model.

In real world scenarios, indoor scenes tend to range between 2m and 10m from the camera, versus outdoor scenes which range between 5m and 100m. For handheld cameras, an offset of 20cm is reasonable, in which case the synthetic test results estimate total error (after a full circle) on the order of 1° indoors, versus 0.1° degrees outdoors. A camera on a tripod has an offset of 5cm or less, which results in errors of 0.1° indoors or 0.01° outdoors.

test	pivot	scene	measured error	predicted error
	(cm)	(m)	(deg)	(deg)
A	3.5	1	6	12
B	12.3	1	33	44
C	3.5	2	2	6

Table 2
Measured and estimated errors for real tests with a camera rotating through a full circle about a pivot point offset from its optical center, in a nearby scene. The error is the yaw value of the orientation (pitch and tilt were negligible) when the camera is returned to the initial orientation.

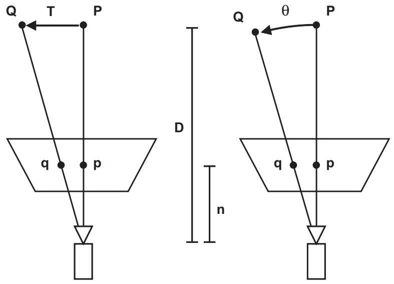

Fig. 7. Comparison of camera translation and rotation on scene features. *Left to right:* (a) As the camera undergoes translation T, point P in the scene moves the same distance. The corresponding projected points p and q show the motion of the feature in the image. (b) Point q can also be generated by rotating the camera through angle θ.

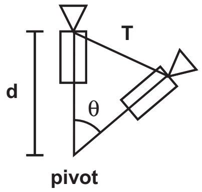

Fig. 8. Induced translation from rotation of the camera about a pivot point that is offset from the optical center.

To further confirm these results, controlled real world scenarios were measured with different camera pivot offsets. The camera was set on a tripod surrounded by real objects (walls and chairs) a known distance away. The camera was then slowly rotated through a full circle and the final error was measured. Data can be seen in Table 2.

Aside from the fact that the translation error model over-predicts the error, the measured errors are partially smaller than expected due to the difficulty in ensuring that features are only chosen from near-scene objects. In portions of the constructed scene, far field objects were partially visible and some features ended up in those regions, meaning they did not exhibit the expected parallax of the near field objects. This has the effect of lessening the measurement error per-frame.

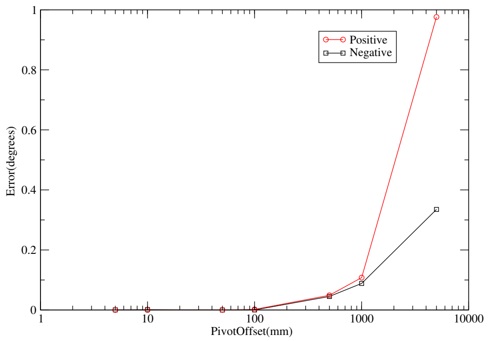

Fig. 9. Error in rotation measurements in synthetic test of off-center rotation inside a 10m sphere. Error is the average over 360 1° rotations.

6.3. *Image Noise*

Noisy image data and motion blur both have the effect of randomly perturbing the detected positions of each image feature. To measure the impact of these sources of error, Gaussian noise was added to feature positions before running different components of the tracking. In each test, the camera was first held stationary for 360 frames, and then rotated by 360 1° increments in a synthetic scene consisting of features 10m away and 5° apart from one another. The average error in detected rotation was measured versus the amount of noise added.

First, the absolute orientation computation was tested in isolation (see Figure 10). The error increases roughly linearly with the magnitude of the noise in the feature tracking. This is the expected behavior, as the orientation computation generates the optimal rotation for the entire set of features, and random noise in the input points will directly translate to random noise in the output rotation. Because these errors are random, they will not systematically affect the orientation estimate, but they will cause it to drift over time.

The second test measured RANSAC's performance at finding a coherent estimate under noisy conditions (see Figure 11). The error in orientation increases faster than for absolute orientation by itself, because RANSAC is likely to find a subset of the features that move coherently (in some random direction) and use them exclusively for the estimate. Therefore, in the case where increased image noise is expected, the RANSAC match threshold should be adjusted to allow for the expected random motion of features. Since in general the feature tracking provides subpixel accuracy, the default match threshold is set to the number of degrees spanned by one pixel, which has performed well empirically.

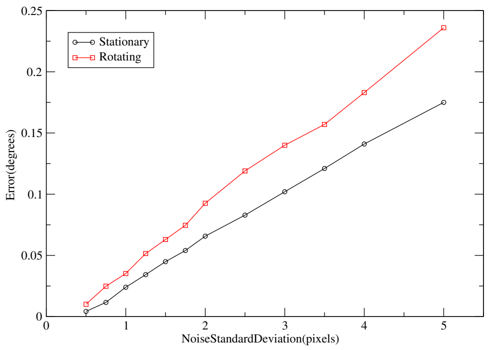

Fig. 10. Average error in the computed absolute orientation during 1° rotations versus pixel noise added in to feature positions.

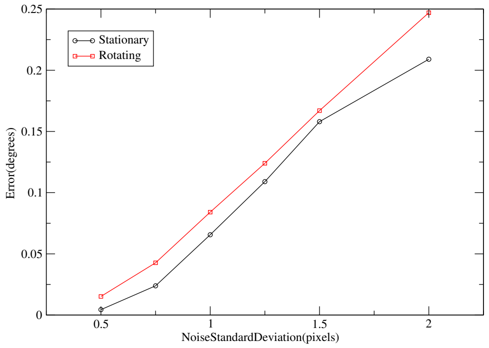

Fig. 11. Average error in the coherent RANSAC rotation estimate during 1° rotations versus pixel noise added in to feature positions, simulating the effect of noisy or motion blurred images.

6.4. *Distractions*

Large distractions that move in the video independently of the camera motion (people, cars, etc.) can interfere with the orientation computation if they overwhelm RANSAC's ability to ignore outliers. To test the performance with large distractions, significant coherent noise (standard deviation of 10 pixels, randomly chosen per-frame) was added to a random subset of the features in each frame before the coherent rotation estimation step.

The results of this test can be seen in Figure 12. The effect of outliers is nearly negligible up until a certain percentage, at which point it increases dramatically. This is the expected behavior for the RANSAC algorithm – outliers can be discarded without affecting the estimate until there are so many outliers that a dominant set of inliers cannot be found. Once that threshold is reached, it is likely that RANSAC will converge on the

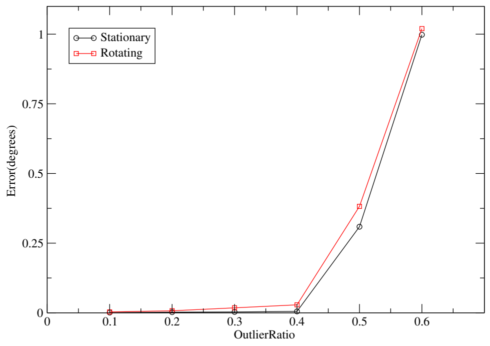

Fig. 12. Average error in the measured rotation during 1° rotations versus portion of features with significant coherent noise added, simulating the effect of large distractions in the scene.

incorrect set of coherent motion. This possible failure mode will occur when a single object, such as a nearby person or a moving car, occludes most of the camera's field of view. More likely in dynamic scenes is that a large number of small distractions will be present in the scene, each with their own motion. In this case, so long as the percentage of inlier features is still above the RANSAC threshold, the correct estimate is still most likely to be found. For scenes with more dynamic content, the RANSAC threshold should be adjusted accordingly.

6.5. *Summary*

In practice, the largest overall source of error is from translation of the camera, especially in the case of hand-held camera motion. Adequate illumination and slow camera motion make image noise insignificant, and with a wide field of view camera (50° or greater), lack of texture is rarely a problem. In crowded areas, distractions can be a problem if there is not enough static imagery visible in each frame.

Tracking error impacts the ability of the landmark features to correct for drift in long acquisitions. Landmark reacquisition searches for features within a region around the expected position of the landmark. The default search region has a radius of 15 pixels, which roughly corresponds to an error of 1°. If the accumulated tracking error is less than this region, then landmarks can be reacquired and will correct the error, enabling long-term tracking without drift. If the expected tracking error is greater then the search radius should be increased to ensure that landmarks can still be reacquired.

The impact of tracking error on the acquired environment maps is that it creates visible discontinuities between frames that are spatially adjacent but not temporally adjacent. In our experience, registration errors on the order of 0.5-1° create distracting artifacts in the final environment map. Therefore, the sources of error for a scene need to be mitigated so the total accumulated error is less than 1°. For indoor scenes, this requires the use of a tripod (or careful hand-held rotation). The RANSAC parameters need to be adjusted to accommodate the expected levels of image noise and distractions. With these considerations, we have been able to achieve sufficient accuracy for environment acquisition in both indoor and outdoor scenes (see Figures 1, 15, 16, and 17).

7. Alternative Landmarks

SURF features were selected to use as landmark descriptors because of their robustness to changes in the scene (rotation, illumination, etc.). However, because of their significant performance penalty, we examined alternative descriptors. One of the main useful aspects of SURF (and SIFT and similar descriptors) is their robustness against viewpoint changes, but for panorama creation the viewpoint is static, so less complex descriptors can be considered as replacements.

We used texture patches to compare with SURF descriptors. A texture patch is a small square of pixels copied out of the video image, centered on the feature position, and stored in a 1D array. Pixels are sampled from the video with bilinear interpolation. We experimented with patch sizes ranging from 11x11 to 25x25, but did not see significant performance differences in them and settled on 25x25.

To provide some robustness against global illumination changes, the array is scaled so the maximal element equals 1. To further provide robustness against rotation along the viewing direction, patches are always extracted at an orientation that matches the global up vector. This way, regardless of the orientation of the camera when it extracts a patch, it will always get the same descriptor. The twist angle used is from the camera's current estimated orientation, since landmark extraction happens after the rest of the orientation update has completed.

Texture patch similarity is computed as

$$f(x,y) = 1 - \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \qquad (7)$$

where n is the number of pixels in the patches and x and y are length n descriptor arrays in the range [0,1].

This is a slight variation on the standard sum of squared distances, adjusted so that a value of 1 means exactly alike and smaller numbers mean more dissimilar.

7.1. *Performance*

The texture patch descriptors are a dramatic speed improvement over SURF. On our 2.1GHz AMD Athlon CPU, computing SURF landmarks takes a total of 40.1ms per frame, whereas texture patches take 0.4ms. The two order of magnitude improvement comes partially from the simplicity of the texture patch features, as well as the integral image computation that is part of the SURF algorithm and is a significant overhead for small numbers of features per image. Indeed, the enhancements made to the SURF algorithm are geared towards the use case of computing many features in a single frame, whereas the scenario in Envisor is to compute a few features every couple frames.

7.2. *Analysis*

It is worthwhile to examine how image conditions impact the different descriptors respectively, to see what cost the speed improvement of the texture patch descriptors comes with. Towards this end, two experiments were conducted, in which descriptors were acquired for a number of points in an image with good quality, and then the descriptors were reacquired under less ideal conditions and the resulting similarity was measured. Ideally a gradual falloff of similarity with image conditions would be found, which means that threshold parameters can be adjusted for conditions to achieve similar results.

In the first experiment, the second set of descriptors are offset in a random direction by some pixel offset before being reacquired. The results are in Figure 13. As can be seen, both descriptors are unaffected for offsets up to 0.5 pixels, but beyond that, SURF descriptors drops quickly while texture patches fall off more slowly. This slow fall off is likely because the sum of squared distances metric will only contribute large differences to the sum for edge pixels (near high gradients), which will exhibit large differences for small offsets. Most pixels however are in smoother regions and so will not impact the similarity much.

The second experiment measures the effect image noise has on descriptors by adding white noise to each pixel intensity. The results are in Figure 14. While SURF descriptors again drop off quickly, afterward SURF and texture patches decline at the same rate. The high similarity values for small amounts of noise is a desirable feature of the texture features, as camera data always includes some low sensor noise, especially in darker environments.

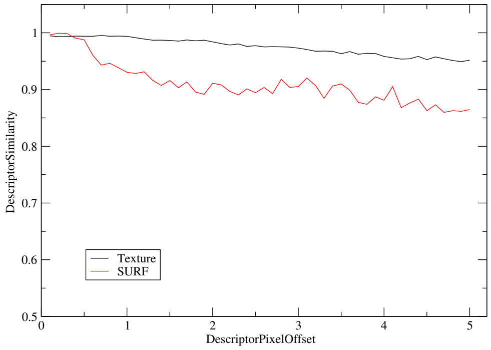

Fig. 13. Average similarity of between original descriptors and descriptors computed with different amounts of pixel intensity noise added, simulating the effect of image noise on landmark reacquisition.

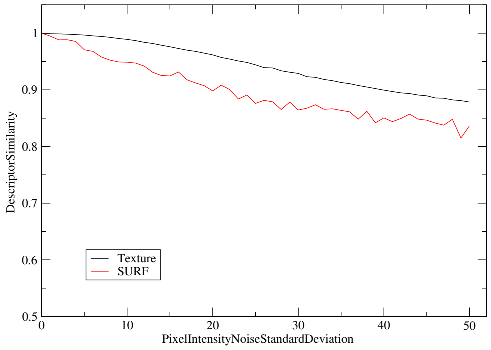

Fig. 14. Average similarity of between original descriptors and descriptors computed with different amounts of pixel intensity noise added, simulating the effect of image noise on landmark reacquisition.

Normally, the SURF descriptor reacquisition threshold is set at 0.95. The data in Figure 14 shows how to adjust this threshold to account for different levels of expected image noise. Additionally, these experiments show that texture patches require a higher threshold to maintain similar probability of landmark reacquisition as the SURF descriptors. In practice, setting the threshold to 0.98 yields tracking performance on par with SURF descriptors on the same camera video.

8. Conclusions

In this paper, we presented Envisor, an application for the online and automatic creation of environment maps as part of a wearable augmented reality system. The two main contributions of Envisor are a modular orientation tracking algorithm that provides configurable, long-term and drift-free tracking, and a technique for using this tracked video feed to automatically create environment maps online. We have demonstrated the quality of Envisor's results, and discussed the implications of its performance in a variety of contexts.

To improve Envisor's environment map construction, we would like to find a camera that supports reading the exposure parameters each frame, so we can implement an image normalization procedure and get good tracking and good environment maps in high dynamic range scenes. Long-term, we would like to extend these techniques into 6DOF camera tracking and online scene model building.

In the meantime, Envisor is another tool in the growing toolbox of Anywhere Augmentation technologies. It allows users with standard hardware to quickly acquire useful information about their environment, enabling experimentation with more advanced augmented reality techniques.

Acknowledgments

Special thanks to Simon Julier. This research was in part funded by a grant from NSF IGERT in Interactive Digital Multimedia #DGE-0221713 and a research contract with the Korea Institute of Science and Technology (KIST) through the Tangible Space Initiative Project.

References

[1] K. Agusanto, L. Li, Z. Chuangui, and N. Sing. Photorealistic rendering for augmented reality using environment illumination. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2003.

[2] P. Baudisch, D. Tan, D. Steedly, E. Rudolph, M. Uyttendaele, C. Pal, and R. Szeliski. Panoramic viewfinder: providing a real-time preview to help users avoid flaws in panoramic pictures. In *Proceedings of the Conference of the Computer-Human Interaction Special Interest Group of Australia*, 2005.

[3] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision*, 2006.

[4] R. Behringer. Registration for outdoor augmented reality applications using computer vision techniques and hybrid sensors. In *Proceedings of Virtual Reality*, 1999.

[5] M. Brown and D. Lowe. Recognising panoramas. In *Proceedings of the International Conference on Computer Vision*, 2003.

[6] A. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the International Conference on Computer Vision*, 2003.

[7] S. DiVerdi and T. Höllerer. Combining dynamic physical and virtual illumination in augmented reality. Technical report, University of California, Santa Barabara, 2004. UCSB//CSD-04-28.

[8] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24, 1981.

[9] T. Grosch. PanoAR: Interactive augmentation of omni-directional images with consistent lighting. In *Proceedings of Mirage*, 2005.

[10] X. Gu. GPU-based conformal flow on surfaces. Technical report, State University of New York at Stony Brook, 2006. Computer Science.

[11] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4, 1987.

[12] S. Hsu, H. Sawhney, and R. Kumar. Automated mosaics via topology inference. *Computer Graphics and Applications*, 22(2), 2002.

[13] M. Imura, Y. Yasumuro, Y. Manabe, and K. Chihara. Fountain designer: Control virtual water as you like. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2006. Demo.

[14] Open source computer vision library reference manual, 2000. Intel Corporation.

[15] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 20, 2003.

[16] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.

[17] J. Montiel and A. Davison. A visual compass based on SLAM. In *Proceedings of the International Conference on Robotics and Automation*, 2006.

[18] M. Oliveira, B. Bowen, R. McKenna, and Y. Chang. Fast digital image inpainting. In *Proceedings of the International Conference on Visualization, Imaging and Image Processing*, 2001.

[19] Quicktime vr, 2007. Apple, http://www.apple.com/quicktime/tech-nologies/qtvr/.

[20] G. Reitmayr and T. Drummond. Going out: Robust model-based tracking for outdoor augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2006.

[21] K. Satoh, M. Anabuki, H. Yamamoto, and H. Tamura. A hybrid registration method for outdoor augmented reality. In *Proceedings of the International Symposium on Augmented Reality*, 2001.

[22] H. Sawhney, R. Kumar, G. Gendel, J. Bergen, D. Dixon, and V. Paragano. VideoBrushTM: Experiences with consumer video mosaicing. In *Proceedings of the Workshop on Applications of Computer Vision*, 1998.

[23] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1994.

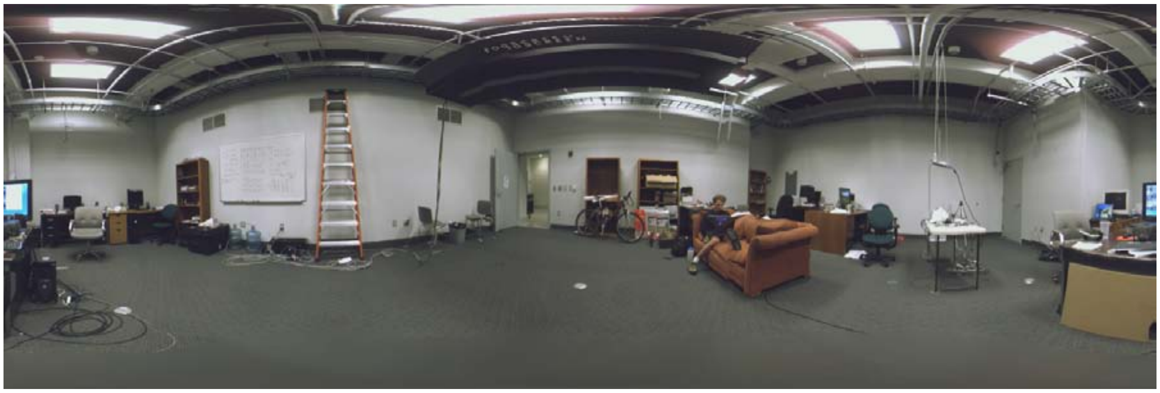

Fig. 15. A panorama carefully constructed with a hand-held camera indoors.

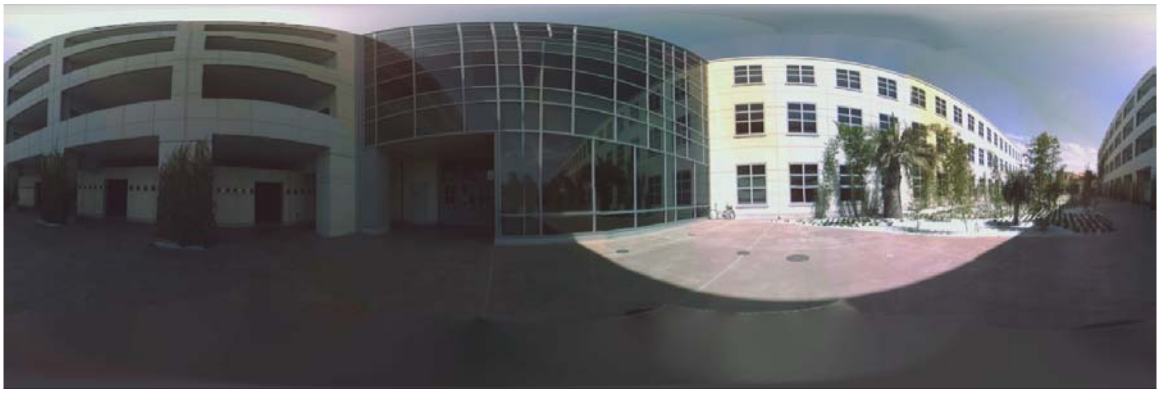

Fig. 16. A panorama constructed outdoors with a hand-held camera.

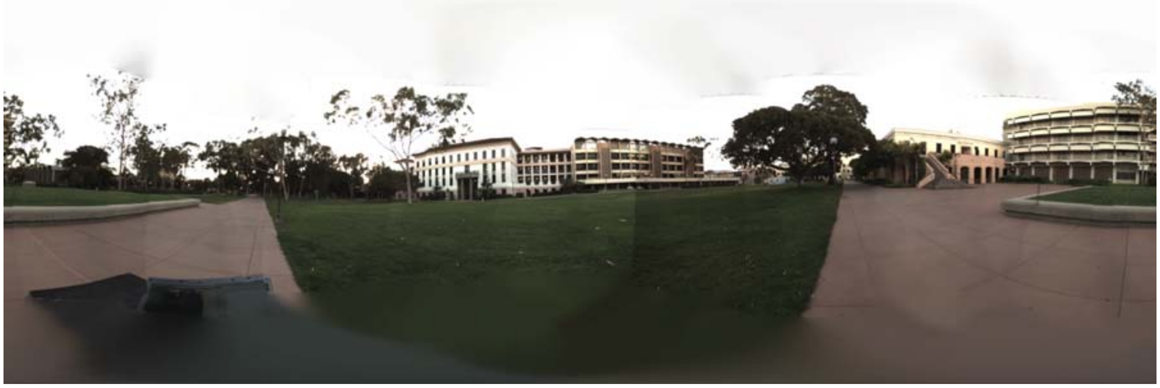

Fig. 17. Another outdoor panorama. While camera exposure was constant, the intensity discontinuity came from the scene darkening during acquisition.

[24] D. Steedly, C. Pal, and R. Szeliski. Efficiently registering video into panoramic mosaics. In *Proceedings of the International Conference on Computer Vision*, 2005.

[25] R. Szeliski and H. Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of SIGGRAPH*, 1997.

[26] M. Uyttendaele, A. Criminisi, S. Kang, S. Winder, R. Szeliski, and R. Hartley. Image-based interactive exploration of real-world environments. *Computer Graphics and Applications*, 24(3), 2004.

[27] S. You and U. Neumann. Fusion of vision and gyro tracking for robust augmented reality registration. In *Proceedings of Virtual Reality*, 2001.

[28] Z. Zhang. A flexible new technique for camera calibration. *Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 2000.