A New Input Device for 3D Sketching

Mark A. Schroering, Cindy M. Grimm, and Robert Pless
Washington University in St. Louis
{mas10,cmg,pless}@cse.wustl.edu

Abstract

We present a 3D input device consisting of a stiff piece of paper which is tracked by a digital video camera. The user can also draw on the paper using a pen-like device. The user moves the paper to specify the location of a virtual plane. By drawing on the paper, the user can specify points in 3D space.

The primary technical contribution of this paper is a new pose estimation algorithm suitable for a hand-held, moving pattern. To demonstrate the usefulness of the device we developed a sketching application for simple characters. The characters are constructed by sketching and joining together 3D ellipses, much as traditional cartoon characters are created in 2D using 2D ellipses.

Keywords: 3D User interface, sketching, ellipse, tracking

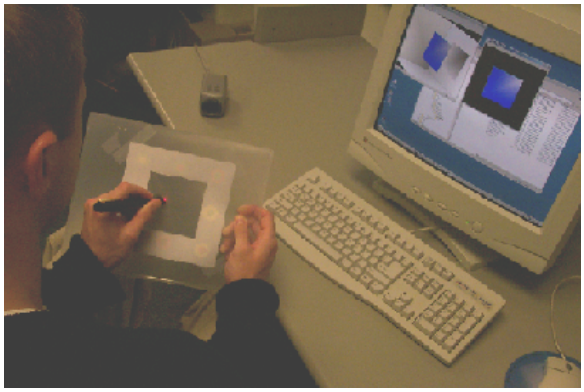

Figure 1: The user holds the tracking pattern and draws with a laser pointer. The screen shows a virtual representation of both the plane in space (left) and the input on the plane (right).

1 Introduction

In this paper we present a 3D input device built from a commodity digital video camera, a piece of cardboard, and a laser pen (see Figure 1). The user uses the cardboard to specify a location and orientation in space. They can then "draw" using the laser pen to specify 3D points on the plane of the cardboard. The advantage of this device is that it can be built using items commonly found in the home or office, making it accessible to a wide variety of users.

The primary technical contribution of this paper is a new pose estimation technique that is robust to low-quality cameras, changing lighting positions, occlusions, and partial visibility. The calibration pattern is a set of color-coded ellipses with a transparent region in the middle for drawing. We compare the accuracy of this pattern to the standard position estimation using a checkerboard [20].

We had several goals for this project. First, the device must be robust to different working conditions, including poor-quality cameras and changes in lighting. Second, the calibration pattern must work when partially occluded or when part of the pattern is out of the camera's field of view. The device must also use inexpensive equipment that is attainable by most people.

We developed a sketching application to test the usefulness of the device. The user sketches 3D ellipses which can be joined together to create simple cartoon characters.

Section 2 discusses previous calibration work and research on spatially-aware user interface designs. Section 3 describes the tracking pattern, the virtual camera model, and the tracking process. Section 5 discusses the technique used for tracking the laser pointer. Section 6 describes the 3D shape sketching application. Section 7 lists the results and analysis from accuracy tests that were performed on the system. We close with future work.

2 Previous work

Tracking a plane: The problem requires that we track a planar object in the scene. This exact problem and several variants have been addressed in prior literature, most commonly, in the context of camera calibration. A calibration method implemented in the OpenCV library [13] uses many images of a black and white checkerboard, finds the corner points

for each image, and calculates both the intrinsic parameters of the camera (the calibration matrix), and the extrinsic camera parameters (the relative orientation of the camera and the plane). The VISUAL PANEL [21] system tracks a flat cardboard pattern without using any explicit calibration pattern. Because a pattern is not used, a more complicated computational problem needs to be solved to determine the position and orientation of the piece of cardboard.

In our case we have a camera whose intrinsic parameters are static, and we are primarily interested in capturing the position and orientation of a plane with a known pattern. Many techniques have been proposed for this problem, including a differential method which tracks continuous motions of a plane using spatio-temporal image derivatives [2], and simultaneous estimation of both the position of a plane and the texture pattern of that plane [4]. Other work that explicitly uses ellipses as part of planar patterns for calibration and pose estimation include [17] which calculates a set of specific points using tangent lines and intersection, Song [18], who uses an iterative approach and requires a corresponding pair of conics [16], and Kaminski [12], who reconstructs the conic and the calibration parameters. Most related to our solution is the single image direct methods of computing planar pose. Much of this work is summarized in [11], which integrates previous work into a unified geometric framework which uses combinations of point and line features.

Spatially-aware devices: Non-traditional user interfaces, such as Graspable [6], Tangible [10], or Manipulable [8] interfaces, have been shown to be better than the traditional mouse and keyboard for many applications. In these alternative interfaces, the user picks up and manipulates a real 3D object. Other examples of this type of interaction are interface props used for neurosurgical visualization [9] and BRICKS [7], a software and hardware framework for quickly prototyping graspable user interfaces.

Most position-aware devices have been implemented using specialized hardware, such as the Polhemus FASTRAK six degree-of-freedom trackers [14]. Our device is not as general as the ones listed above, but it also does not require specialized hardware.

3 Drawing surface tracking

The core technical challenge of this project is the computer vision algorithm for tracking the drawing surface. Tracking is theoretically well understood in the computer vision community. First, a tracking pattern is built with easily recognizable features. Second, these features are found in the 2D video image stream. Third, the perspective transformation that takes the 3D pattern features to the 2D image features is calculated.

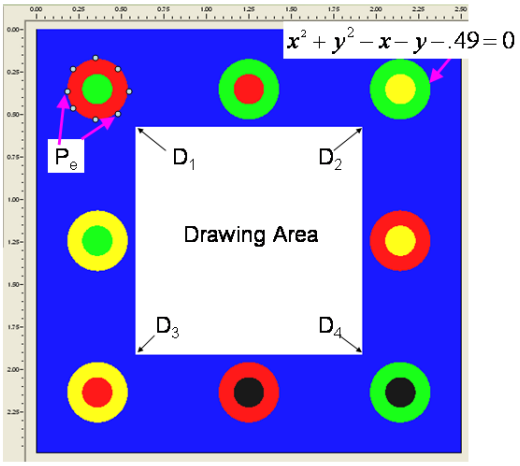

Figure 2: Tracking pattern with chromaglyphs

We designed our tracking pattern to be robust to lighting conditions, poor-quality cameras, and partial occlusions. The pattern can also be built by a naive user with low-cost materials. The pattern is built using ellipses that are detectable under different illumination conditions. Our application requires a minimum of three ellipses to be detected.

The input to this process is a video stream of the tracking pattern on the back of the drawing surface. Each image is scanned to find the pixels that lie on the ellipse borders. The equations of the ellipses are then calculated using these boundary pixel points. The ellipse equations are used in a minimization search to find the rotation and translation of the digital camera relative to the drawing surface. The translation and rotation of the drawing surface can then be determined.

3.1 Tracking pattern

We considered several properties when designing the pattern. First, the pattern features must be easily distinguishable under different lighting conditions. Second, the pattern must be asymmetric, or different poses will have the same appearance. Third, the tracking pattern must work when partially occluded. Lastly, there must be sufficient space left on the drawing surface for sketching.

To satisfy these constraints, the decision was

Table 1: Camera parameters

	Parameter	Description
Intrinsic	(u_0, v_o)	Center of the image projection
	(α, β)	Combination of the focal length, the aspect ratio, and the scale up to image coordinates.
	γ	The skew in the camera
Extrinsic	R	Camera's rotation matrix
	T	Translation of the camera.

made to use eight chromaglyphs [19] arranged around a drawing surface (see Figure 2). The points D_{1-4} represent the corners of the drawing space. A chromaglyph is composed of N discs, each with a unique color, chosen from a set of M prototype colors [19]. For this application $N = 2$, and $M = 4$. The permutations of the disc color uniquely identify the glyph. The number of chromaglyphs was chosen so that only 1/4 of the pattern needs to be visible. The location of the chromaglyphs on the pattern allows for a transparent drawing region in the center of the pattern.

3.2 Camera model

We use the standard virtual camera model, whose parameters are summarized in Table 1. The intrinsic parameters are those that specify the camera itself. For our application, calculating exact intrinsic parameters is less important because we are interested in the relative movement of the plane. The extrinsic parameters describe the spatial relationship between the camera and the world. These parameters change as the camera's position and orientation change. The virtual camera model does not include distortion parameters because it does not affect relative changes in orientation and position. For example, the change in position may not be exactly correct, but moving the paper a little more moves the virtual paper a little more.

The camera model provides a function, $V(p)$, that converts points in the world frame to points in the image frame. The function takes a point in the world frame, p, as input and returns a 2D image point, q. The function uses a perspective transform, C, which is composed of the intrinsic parameters. The extrinsic parameters form a rotation plus translation matrix, $[R|T]$. The following equations show how $V(p)$ transforms p to q:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = P[R|T]p \tag{1}$$

$$q = (u/w, v/w) \tag{2}$$

3.3 2D feature detection

In this section, we describe how to find the features in the 2D image. The projection of the pattern causes the 3D circles to project as ellipses. The image is scanned to find the pixels that lie on the ellipse boundaries. The color of the pixels along the ellipse borders are used to determine which ellipse was found. The border points are then passed to an ellipse-fitting routine.

3.4 Image Scanning

Image scanning is used to find the points in the image that fall on the borders of the chromaglyphs. The scanning classifies features in the image by color ratios. The colors chosen for the chromoglyphs are located in different corners of the color space [19]. Each color has a maximum or minimum concentration of red, green, or blue. Because of this, the colors are still distinguishable under different illumination conditions.

The scanning software acquires a buffer of pixel color values from the digital camera. This buffer is a snapshot of what the camera currently sees in its field of view. Each pixel in the buffer is then assigned one of the five prototype colors (red, green, yellow, black, or blue). If the pixel color cannot be determined, it is labelled as 'unknown'.

The scanning software picks a pixel that is one of the four prototype colors as the starting point for a flood-fill search. The search returns a connected region of pixels that lie on the border of the ellipse. For example, when a red pixel is found, the flood-fill search will look for all of its neighboring red pixels that border blue pixels (see Figure 3).

Since low-resolution cameras are being used, blurring can occur along the ellipse borders. This causes some of the pixel colors to be unidentifiable and are marked as being 'unknown'. When one of these unknown pixels is found during the flood-fill search, a color check of the unknown pixel's neighbors is performed. If one of the unknown pixel's neighbors is a different color, then it can be assumed that this pixel lies on the border. This technique allows for a thin line of blurring along the ellipse borders.

We first search for the outer border points. When the flood-fill search is complete, the center of the

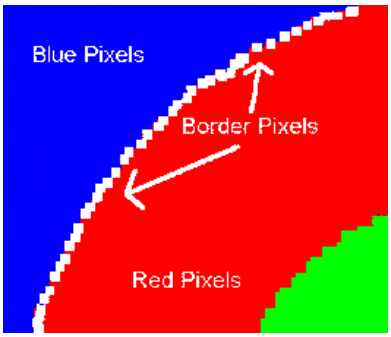

Figure 3: The red pixels that border the blue pixels are detected.

ellipse is calculated using the outer border points. This is also the center of the inner ellipse. The color of the inner ellipse is then identified by the center point's color. The color of the outer and inner discs provide the identity of the chromaglyph. A flood-fill search is then performed on the inner circle to obtain the inner ellipse border points.

The entire image is scanned in this fashion to identify all of the chromaglyphs that are in the field of view. When the scanning is finished, the border points for the visible ellipses are passed to the ellipse fitting algorithm.

3.4.1 Ellipse fitting

Once an ellipse's border points have been identified, we can fit an ellipse to them. There are many approaches to ellipse fitting. The method used here is the one developed by Fitzgibbon, Pilu and Fisher [5], which treats the ellipse fitting as an eigenvalue problem. A general conic has the following form:

$$F_e(x, y) = ax^2 + bxy + cy^2 + dx + ey + f \qquad (3)$$

Fitzgibbon's method adds restrictions to force the conic to be an ellipse. The fitting is performed on each set of border points, which produces up to 16 sets of ellipse equation coefficients. These equations are used in the camera parameter search stage.

3.5 Camera parameter search

The final step is to search for the camera pose that maps the 3D circles ($z = 0$) to the ellipses found in the image. We pick a set of points, P_e, that lie on the 3D circles to use for the search (see Figure 2). We used eight points per ellipse.

As the drawing surface is oriented in front of the camera, the tracking pattern that the camera sees gets warped. The point P_e on the circle is mapped to a 2D image point, q_e using the camera model function, V, described in section 3.2. If T and R represent the camera's true orientation, then the points should lie close to the ellipses found in the image scanning step. This means that the ellipse equation, $F_e(x, y)$, should evaluate to a value that is close to zero when using these points.

The goal of the search is to find the values of T and R that minimize $(F_e)^2$. We created an error function, $E(T, R)$, that we are trying to minimize. For each ellipse that was visible in the image, the function converts its corresponding border points in the world frame, P_e, to 2D points in the image frame. The corresponding ellipse function $F_e(x, y)$ is then evaluated for each of these converted 2D points. The results are squared and summed for all eight points. The equation for $E(T, R)$ is as follows:

$$\min E(T, R) = \min \sum_{e=1}^{n} \sum_{i=1}^{8} [F_e(V(P_{ei}))]^2 \qquad (4)$$

where $n \leq 16$ is the number of ellipses found in the image scan. We require a minimum of three and use a maximum of eight to reduce computation time. The search begins with a guess at the values of T and R (either the previous rotation and translation, or a default camera location with no rotation). If $E(T, R)$ does not evaluate to a value that is close to zero, then T and R need to be modified in some way. A Simplex Search [15] is used to minimize $E(T, R)$. We represent R as three consecutive rotations around the x, y, and z axes.

The number of iterations needed to find the minimum in the initial search is approximately one hundred. Subsequent searches are started using the previous values for T and R; they converge with about thirty iterations. If the display surface's orientation does not change much, the search takes about ten iterations.

4 Drawing surface's position

In the previous sections, the drawing surface is assumed to be stationary and located at $z = 0$. T and R represent the transformation that moves the camera relative to the drawing surface. In the sample drawing application, the camera is assumed to be stationary and the drawing surface moves relative to the camera. Here we describe how to find the transformation for the drawing surface.

The rotation of the drawing surface is simply R^{-1}. The correct translation of the drawing sur-

face is not T^{-1}; we want the relative movement of the plane, not the absolute position. We take the camera's view axis and intersect it with the drawing surface to find the focal point of the camera. As the plane is moved, the focal point will change as well. We store the original focal point, f_{orig}, is stored at the beginning of the program execution. The simplex search is performed and the new focal point, f_{new}, is retrieved from the camera model. The drawing surface's translation, T_v, is then equal to $-(f_{new} - f_{orig})$. The final transformation to orient the drawing surface relative to the camera can be expressed in matrix form as $[R^{-1}|T_v]$.

5 The laser pen

We use a laser pointer to sketch on the display surface. The inner region of the display surface is transparent, which allows a laser pointer to shine through and be detected by the camera. When the pen is on it produces a bright red spot in the transparent region. The image scanning routine finds the location of this red dot in the 2D image. This location is then converted to a 3D world coordinate using the position and orientation of the drawing surface.

After the camera's position is calculated we can map the 3D corner points of the drawing area, D_{1-4}, to 2D image points using the camera model's conversion function, V. All red pixels are tested to see if they fall within the quadrilateral defined by the corner points. The centroid of these pixels, P_{pen}, is used as the 2D pen point. Once P_{pen} is found, we calculate the corresponding 3D point by taking a ray L from the camera's position E through the point P_{pen} (see Figure 4). The value for E is obtained from the camera model. Because the plane lies at $z = 0$ we know that:

$$P_{3D} = E + (-E_z/L_z)L \tag{5}$$

This 3D point, P_{3D}, corresponds to the point when the drawing surface is at $z = 0$ in the world frame. The point needs to be transformed to the current location of the drawing surface in the World Frame. To do this, we multiply P_{3D} by the transformation matrix, $[R^{-1}|T_v]$, that we found in Section 4. Now we can track the laser pointer as it moves in the World Frame.

6 Drawing application

This section describes a drawing application that uses the 3D input device. The drawing application allows the user to make characters by sketching ellipses. A virtual rendering of the drawing surface and the 3D scene are shown on the monitor.

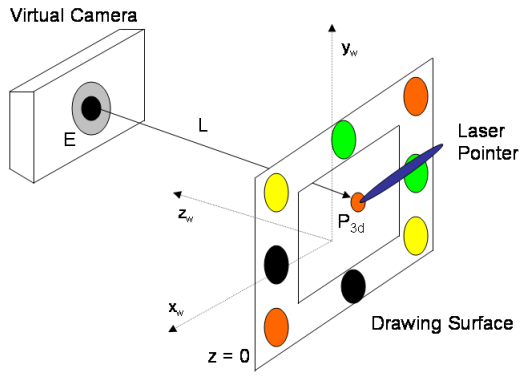

Figure 4: Tracking the laser pointer in 3D space.

Table 2: 3D Input Device Events to Mouse Events

Mouse Event	3D Input Device Event
Mouse Button Down	When the laser pointer is turned on for the first time.
Mouse Move	When the laser pointer is on and the location changes.
Mouse Button Up	When the laster pointer is turned off after being on.

6.1 Virtual display windows

The virtual display consists of two windows that are displayed on the computer monitor. The first window is a re-creation of the 2D drawing surface (see Figure 5). The user sees the cross-section of any shapes that the drawing surface is currently intersecting. While the user is drawing with the laser pointer, a trace of the pointer's movement is rendered on the virtual surface with a series of yellow dots.

The other window shows the 3D virtual scene. In this window, the user can see the virtual drawing surface's orientation in 3D space relative to the world. The virtual drawing surface's position and orientation in the world frame is calculated by applying the transformation matrix, $[R^{-1}|T_d]$, to the virtual drawing surface's starting position (see Section 4).

6.2 Character creation

Because this application does not use a mouse, the typical mouse events have to be mapped to the actions of the 3D Input Device. Table 2 lists the mapping of the typical mouse events to 3D Input Device events.

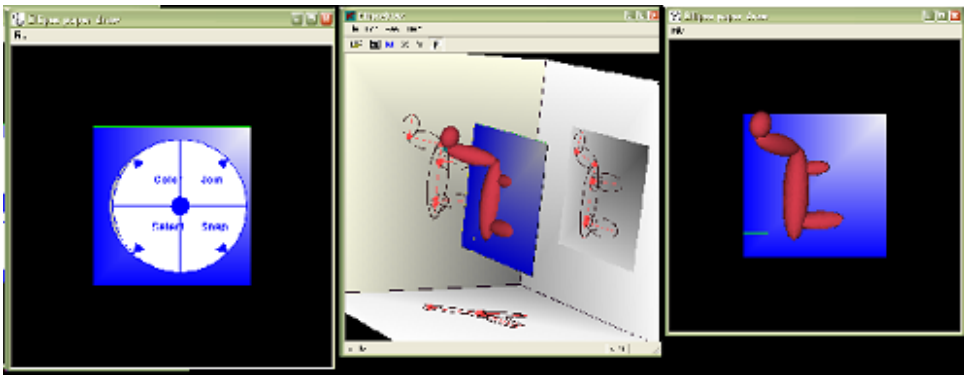

Figure 5: Left: Pie menus. Middle: The virtual plane in the scene. Right: The drawing surface.

The drawing application receives the 3D points from the 3D input device as *Mouse Move* events. These points indicate where the laser pointer is at in the world frame (see Section 4). The drawing application renders these points as small dots that appear in the window with the virtual drawing surface. These points provide feedback that lets the user know what kind of shape is being drawn.

When the application receives the *Mouse Button Up* event from the input device, it fits an ellipsoid to the 3D points. This ellipsoid is then rendered in the virtual scene. The user can join ellipsoids to create 3D characters. The characters can also be saved to a file and opened later for editing.

6.3 Pie menus

Pie menus [3] are used in the drawing application to allow the user to change program settings and perform certain actions (see Figure 5). Each piece of the pie is a single menu action. Some actions bring up sub-menus that appear on top of the parent menu. The sub-menu is simply another pie menu that appears on top of the parent menu. The pie menu is brought up by shining the laser pointer in the lower left corner of the drawing region.

The user can select between the different menu items by turning on the laser pointer and moving it to the location of that item in the pie menu. While the user is moving the active laser pointer around, the 3D Input Device will be sending *Mouse Move* events to the drawing application. The currently selected menu item will be highlighted. When the desired menu item is highlighted, the user turns off the laser pointer. This triggers the *Mouse Button Up* event in the drawing application. The selected menu item is then executed.

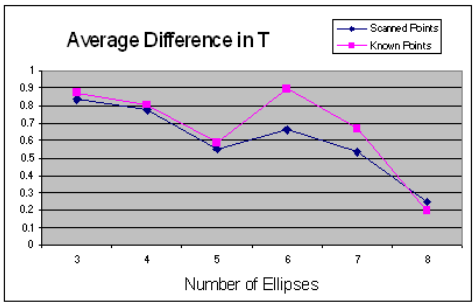

Figure 6: Results from Accuracy Test

7 Accuracy tests

7.1 Accuracy

The accuracy of the image scanning, ellipse fitting, and camera parameter searching was tested using a simulated version of the calibration pattern. A virtual camera is positioned at different orientations to view the display surface. The actual values for T and R are known in this case. The camera takes a snapshot of the scene and the ellipse search is performed. The starting point for the search is always $T = (0, 0, 5), R = (0, 0, 0)$. The rotation angles were varied from 0 to 5 degrees, the x, y translation from -1 to 1, and the z translation from 4 to 6. The tests were run on a set of 1,000 camera positions. The following error measurements were calculated:

- $T_{x,y,z}, R_{x,y,z}$ Error - Difference between the known and the converged camera parameters. The rotation values are given in degrees.
- $E(T, R)$ - Value of $E(T, R)$ after the search.

Figure 6 show the average values for $T_{x,y,z}$ Error per the number of ellipses that were detected in

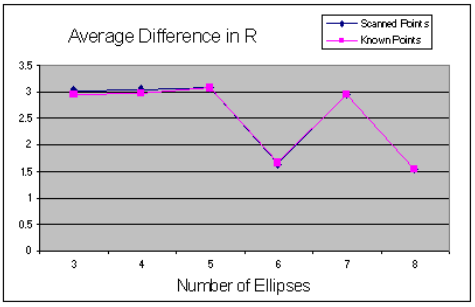

Figure 7: Results from Accuracy Test

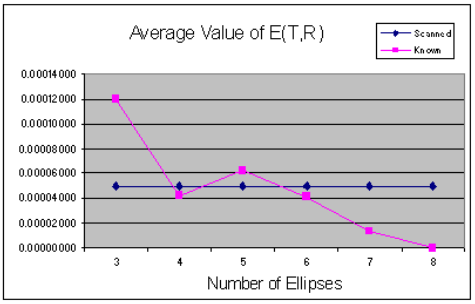

Figure 8: Results from Accuracy Test

the image for that camera position. Figure 7 shows average values of $R_{x,y,z}$ in the same format as the first graph. The last graph, Figure 8, shows the results when evaluating $E(T, R)$ using both sets of converged values for T and R.

One can see from the graphs that the average error gradually decreases as the number of ellipses increases. This would be expected since an increase in the number of ellipses provides the calibration method with more data to use in the minimization of the error function.

7.2 Real world tracking

The accuracy of the ellipse search calibration method was compared to the OpenCV method [13]. A 5×7 checkerboard pattern was placed in the center of the calibration pattern. The ellipse search and the OpenCV calibration methods were then performed on a set of 10 images of size 640×480 pixels. The 10 images were snapshots from the digital camera of the drawing surface at different orientations (see Figure 9).

$E(T, R)$ was evaluated with the values of T and R found from both the ellipse search and the OpenCV method. The 3D coordinates of the 24 interior checkerboard corner points were measured. These points were then converted to 2D points in the Image

Table 3: Real world tracking results

	Avg	**Min**	**Max**	**Dev**
E - Ell	$8.0e^{-6}$	$4.5e^{-6}$	$1.1e^{-5}$	$2.4e^{-6}$
E - OCV	$8.0e^{-5}$	$3.1e^{-5}$	$1.6e^{-4}$	$5.2e^{-5}$
Px Err Ell	8.19	6.69	9.68	2.11
Px Err OCV	0.60	0.59	0.60	0.001

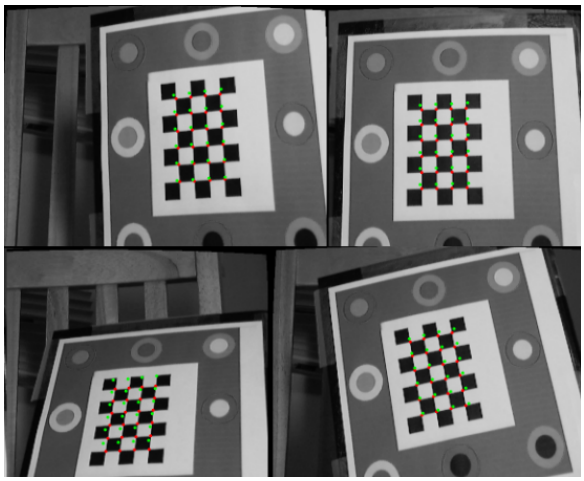

Figure 9: Real world tracking test images.

Frame using $V()$ with the values of T and R from the ellipse search and the OpenCV methods. These 2D points were then compared to the 2D corner points that were found in a scan of the image.

Table 3 provides a summary of the results from this test. The average value of $E(T, R)$ was lower for the ellipse search method than the OpenCV method. While the average pixel distance for the checkerboard corners is lower for the OpenCV method than for the ellipse search method.

Figure 9 shows four of the input images used in the test. The green dots on the image represent the corner points that were converted to 2D image points using the ellipse search values for T and R. The red dots are the points that were converted using the OpenCV values for T and R.

7.3 Performance

The current system runs at approximately 10 fps on a 356MHZ Pentium, using 120×160-sized images. The current bottleneck is the image scanning routine; hence the small image sizes.

8 Future work

The primary bottleneck is the image-scanning routine; a faster algorithm would allow us to use higher-resolution images. We would also like to incorporate prediction [1], which would speed-up both the 2D image scanning and the camera parameter search. More advanced prediction would also result in smoother tracking.

References

[1] Ronald Tadao Azuma. Predictive tracking for augmented reality. Technical Report TR95-007, Univ. North Carolina, Chapel Hill, 22, 1995.

[2] Jose Miguel Buenaposada and Luis Baumela. Real-time tracking and estimation of plane pose. In *ICPR*, pages 697–700, 2002.

[3] J. Callahan, D. Hopkins, M. Weiser, and B. Schneiderman. An empircal comparison of pie menus versus linear menus. *Proceedings of ACM Chi'88 Conference on human factors in computing systems*, pages 95–100, 1988.

[4] F. Dellaert, C. Thorpe, and S. Thrun. Super-resolved texture tracking of planar surface patches, 1998.

[5] Andrew W. Fitzgibbon, Maurizio Pilu, and Robert B. Fisher. Direct least square fitting of ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):476–480, 1999.

[6] George W Ftizmaurice and William Buxton. An empirical evaluation of graspable user interfaces: Towards specialized, space-multiplexed input. In *Computer Human Interaction*, pages 43–50, 1997.

[7] George W Ftizmaurice, Hiroshi Ishii, and William Buxton. Bricks: Laying the foundations for graspable user interfaces. In *Computer Human Interaction*, pages 1–8, 1995.

[8] Beverly L Harrison, Kenneth P Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In *Computer Human Interaction*, pages 17–24, 1998.

[9] Ken Hinckley, Randy Pausch, John C Goble, and Neal F Kassell. Passive real-world interface props for neurosurgical visualization. In *Computer Human Interaction*, pages 452–458, 1994.

[10] H. Ishii and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits, and atoms. In *Computer Human Interaction*, pages 234–241, 1997.

[11] Q. JI, M. COSTA, R. HARALICK, and L. SHAPIRO. An integrated linear technique for pose estimation from different features, 1999.

[12] Jeremy Yermiyahou Kaminski and Amnon Shashua. On calibration and reconstruction from planar curves. In *ECCV (1)*, pages 678–694, 2000.

[13] Opencv: http://www.intel.com/research/mrl/research/opencv/.

[14] Polhemus: http://www.polhemus.com/home.htm.

[15] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Downhill simplex method in multidimensions. In *Numerical Receipes in C*, pages 408–412, 1992.

[16] Long Quan. Conic reconstruction and correspondence from two views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):151–160, 1996.

[17] C A Rothwell, A Zisserman, C I Marinos, D Forsyth, and J L Mundy. Relative motion and pose from arbitrary plan curves. *Image and Vision Computing*, 10(4):251–262, 1992.

[18] D M Song. Conics-based stereo, motion estimation, and pose determination. *International Journal of Computer Vision*, 10(1):7–25, 1993.

[19] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. Superior augmented reality registration by integrating landmark tracking and magnetic tracking. *Computer Graphics*, 30(Annual Conference Series):429–438, 1996.

[20] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *ICCV*, pages 666–673, 1999.

[21] Zhengyou Zhang, Ying Wu, Ying Shan, and Steven Shafer. Visual panel: Virtual mouse, keyboard and 3d controller with an ordinary piece of paper. In *Perceptive User Intefaces*, 2001.