Performance Characteristics of a Camera-Based Tangible Input Device for Manipulation of 3D Information

Zeyuan Chen* Christopher G. Healey† Robert St. Amant‡

North Carolina State University

Abstract

This paper describes a prototype tangible six degree of freedom (6 DoF) input device that is inexpensive and intuitive to use: a cube with colored corners of specific shapes, tracked by a single camera, with pose estimated in real time. A tracking and automatic color adjustment system are designed so that the device can work robustly with noisy surroundings and is invariant to changes in lighting and background noise. A system evaluation shows good performance for both refresh (above 60 FPS on average) and accuracy of pose estimation (average angular error of about 1°). A user study of 3D rotation tasks shows that the device outperforms other 6 DoF input devices used in a similar desktop environment. The device has the potential to facilitate interactive applications such as games as well as viewing 3D information.

Index Terms: I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Color H.5.2 [Information Interfaces and Presentation]: User Interfaces—Input Devices and Strategies

1 Introduction

Manipulation of 3D information has become more and more important with the increasing popularity of 3D graphics in various applications and platforms. 3D tasks such as rotation and docking are challenging, however, with conventional 2D input devices. 6 DoF input devices tend to be more effective but are often hard to use or too expensive for consumer applications.

We have developed a prototype tangible 6 DoF input device that is low-cost and intuitive to use to support exploration of 3D information. The system is composed of a physical wireframe cube made of thin rods and colored corners (Figure 1) and a single camera. In use, the user watches live video of his or her hands, captured by the camera. Computer vision algorithms estimate the cube's pose so that when the user moves or turns the cube, virtual 6 DoF information changes to follow.

The system processes video in two phases. First, the system estimates a rough region containing the cube, based on its known shape and by tracking correlation between successive frames. Second, the system detects the vertices of the cube within the region to estimate its pose, using correspondences between locations of the cube vertices in the source video frames and their coordinates in a 3D model, a virtual wireframe cube identical to the physical one (Figure 1).

There are multiple ways to map 6 DoF information to the manipulation of 3D objects: direct mapping and relative mapping [33]. In direct mapping, the virtual object has exactly the same pose as the physical cube (Figure 2) or a pose with an offset (Figure 10). For instance, in order to view a virtual teapot, the system projects it inside the cube (Figure 2) so that the user's rotation or translation of the cube causes the teapot to follow correspondingly. In relative mapping, the pose of the cube is mapped to certain properties of virtual objects. For example, in one of the applications discussed below, the orientation of the cube is mapped to the velocity of a moving virtual object; in effect, the cube functions as a joystick.

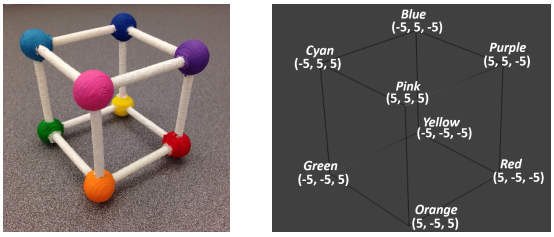

Figure 1: An example of a physical wireframe cube and its mathematical model.

In this paper, we propose novel approaches to track the wireframe cube and estimate its pose accurately and robustly. A system evaluation shows good performance on orientation accuracy, with an average angular error of about one degree, and an average frame rate of 63 frames per second (FPS). We then describe a formal user study of 3D rotation tasks, following past work on 3 DoF and 6 DoF devices [3, 11, 17, 20]. Our device shows better performance than other isomorphic and non-isomorphic techniques used in a similar desktop environment; it even achieves results comparable with some experiments in a surround screen virtual environment (SSVE) [17]. Finally, we describe two applications to demonstrate how our device integrates with other systems in need of 6 DoF input devices and how it facilitates virtual environment design and the control of virtual objects.

2 Related Work

2.1 6 DoF Input Devices

Of the wide range of 6 DoF input devices available, desktop isometric devices are among the most common. For instance, 3Dconnexion's SpaceNavigator is a desktop 3D mouse that allows users to control 3D objects by manipulating its pressure-sensitive handle. However, this type of device often requires significant practice, even for several hours in some cases [32], before the users become comfortable with them.

Another popular type is the free-moving device, which manipulates 3D information through its rotation and translation in free space. A free-moving device is generally easy to learn because it maps the absolute pose of the device directly to the virtual object, but such devices are often much more complex to design. The controller of the HTC Vive contains 24 sensors and has to communicate with two base stations for accurate pose estimation [13]; the result is good performance but a costly system.

HCI has a history of developing free-moving devices based on visible light or IR camera data, such as the videomouse [10], SideSight [1], and HoverFlow [16]. Such systems may perform processing that is much more complex than conventional input devices such as the mouse or trackpad.

*e-mail: zchen23@ncsu.edu
†e-mail: healey@ncsu.edu
‡e-mail: stamant@ncsu.edu

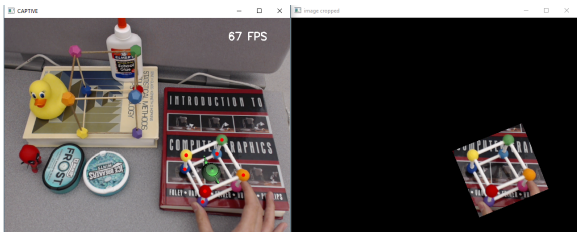

Figure 2: Left: the target smaller cube is detected successfully with heavy noise in the background. Right: the predicted region R_c within which the cube is detected.

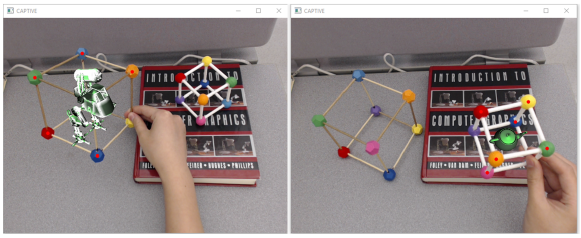

Figure 3: The demonstration of switching between two models of cubes. Left: the user activated the larger cube and a robot was rendered inside. Right: the user activated the smaller cube and a teapot was rendered inside.

Previous works [27, 30] have demonstrated that there is no strong dependency between the shape of the input device being haptically manipulated and the virtual object being visually perceived. This suggests it is acceptable to use a free-moving device with a particular shape (e.g. a cube) to manipulate virtual objects with different shapes.

2.2 Cube-Based Systems

Solid cubes, in some cases instrumented with sensors, have been used for a variety of purposes in augmented reality (AR), virtual reality (VR), and tangible user interfaces (e.g. [2, 5, 7, 15, 18, 23, 25, 26]). In Fish Tank VR, the flat surfaces of a cube can act as displays. In AR systems, the surfaces can hold markers for visual detection. In AR and tangible UIs, handheld cubes can be aligned in a stable way with flat surfaces in the environment; further, they support easy recognition of the cardinal directions in three dimensions. The wireframe cube described in this paper is similar to pCubee [25] (a larger, tethered cube), in that it allows for inspection of virtual objects in its interior. The possibility of interacting with those virtual objects (e.g., using a stylus) is a current area of our research.

2.3 Pose Estimation with a Single Camera

Input systems with a single camera are much simpler and cheaper than other sensor-based or multi-camera systems, but the design of such systems can be very challenging. MagicMouse [31] and ARToolKit [14] estimate pose by tracking 2D markers. They are simple and fast, but have several limitations. First, they are sensitive to occlusions. The pose estimation fails if part of the maker is covered or the borders are moved outside of the view. Second, the pose estimation becomes more unreliable if the markers are more "horizontal" to the camera and the rotation range is limited to 180° because a 2D marker cannot be tracked after it flips over. Even with multi-marker tracking (e.g. the ARToolKit Cube), when the markers are rotated, the marker-to-marker transition is not smooth and thus the variance of estimation errors is large. Third, MagicMouse and ARToolKit are sensitive to the changes in lighting environments.

Color provides useful information to object tracking. Wang and Popovic [28] track hand pose by detecting a color glove. However, their system can only estimate the approximate pose which makes it less useful in applications requiring high accuracy, such as architecture modeling or sculpting in a VR environment. Additionally, its rotation range is limited due to constraints on joint movement.

Another standard approach is to use RGBD (RGB plus depth) images from a device like the Kinect. However, such systems [12, 19, 22, 24, 27] are often too complex to set up or unable to achieve high refresh rate.

3 Cube Tracking

The location of the cube in current video frame is predicted based on the shape of the cube and correlation between successive frames. Let the minimal region that covers the cube in the previous frame be R_p, and the minimal region that covers every parts of current cube in motion be R_m. The predicted region R_c that covers the current cube is the union of R_p and R_m since the cube could either stay stationary or be moved to a new location.

A cube is considered successfully detected if more than three 3D-to-2D correspondences pass the Random Sample Consensus (RANSAC) test in pose estimation (described below). The 3D vertices are back-projected to the 2D screen using the estimated pose. The minimal bounding box that covers them is R_p.

The motion region R_m is detected based on the cube's overall shape: a cluster of rods. Adaptive background modeling [6] is adopted to track pixels in motion, but we use gray-scale images for simplicity. We propose a "bounding box of bounding boxes" approach to track the rods of cube. The contours of pixels in motion are extracted and the minimal oriented bounding boxes are applied to approximate their shapes. Since the bounding boxes of cube rods are thin, long rectangles, a bounding box is considered valid only when its width, height and height-width ratio are all within certain ranges. R_m is the region of a minimal bounding box that covers all of the corner points of valid bounding boxes.

In practice, if there are many objects in motion in the background, the detection of R_m will not decrease the overall accuracy, but it will increase the computational cost. Therefore, users are allowed to turn on and off the tracking functions.

The vertices of the cube are detected within region R_c so that noise in the background is excluded (Figure 2). If the system fails to detect the vertices, R_c will be reset to the entire frame so that the system can always recover from tracking failures.

The within-region detection makes it possible for users to switch among several cubes (Figure 3). If there are n cubes in view of the camera, a user can activate one of them by moving it. The system will keep tracking the active cube until the user picks up another one to use.

The time to activate a cube is short. Assume all cubes have the same probability $\frac{1}{n}$ to be detected by the system and there are no overlaps among them. The target cube is always in R_m during the activation process. Once the target is detected, which means R_p only covers the target, all other cubes will be excluded because they will neither be in R_p nor R_m. The probability of detecting the target within k frames (denoted as P_k) and the activation time $T_{activation}$ are computed as:

$$P_k = 1 - (1 - \frac{1}{n})^k$$
$$T_{activation} = (\left\lceil \frac{\ln(1-p_t)}{\ln(1-\frac{1}{n})} \right\rceil - 1) \times \frac{1}{F} \quad (1)$$

where p_t is the threshold of probability and F is the frame rate. For instance, if there are two cubes in view, p_t is 0.99 and F is 60 FPS, $T_{activation}$ will be about 0.1 seconds, which means the activation time is no more than 0.1 seconds with probability of 0.99.

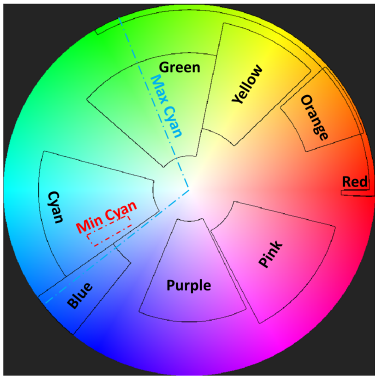

Figure 4: Black contours: optimal color ranges of the eight cube vertices in Hue-Saturation space. Red and cyan dash contours: the minimum and maximum ranges of cyan vertex on the color wheel.

4 Vertex Detection

The cube's vertices are detected based on their colors and shapes. The Hue, Saturation and Value (HSV) color space is used to separate the eight vertex colors. We use HSV rather than RGB because hue and saturation are invariant to changes in illumination.

The colors of surface points of a vertex look similar to each other, but not identical. First, the points are captured from slightly different perspectives by camera. Second, it is infeasible to paint each point on a surface evenly in practice. Therefore, each vertex has a hue and a saturation range on the color wheel. An example of optimal ranges of the eight cube vertices that are generated by our automatic color adjustment system (described below) is shown in Figure 4. Note that the range of red is the combination of two ranges because it is a border case near the hue angle of 0°. For each color, the source frame is converted to a binary image by applying thresholding with the corresponding ranges.

The vertices are located using a blob detector for robustness. Each vertex is a blob in the binary images and the center of the blob is the location of the vertex. The binary images are blurred using a Gaussian filter with a 9 by 9 kernel before applying the blob detector so that the "holes" within the blobs are filled and noise in the background is filtered out. A blob is considered valid if its size is large enough and it is approximately circular. If there is more than one valid blob in a single binary image, the blob with the largest area is chosen to represent the vertex.

The eight vertices are detected separately, so this step can run in parallel. Eight threads are used to detect the vertices, which is about three times faster than a single-threaded approach.

5 Pose Estimation and Model Matching

The pose of the camera is estimated from the 3D-to-2D correspondences by solving the Perspective-n-Point (PnP) problem. Let X be a set of 3D coordinates of vertices, C be a set of colors and x be a set of coordinates of 2D points on the screen. A function f_k is used to map each color in C to the 3D coordinates X:

$$X_j = f_k(C_i) \tag{2}$$

where i and j are the indices of the elements in their sets. A model cube is defined with a list of $\langle f_k(C_i), C_i \rangle$ pairs. Figure 1 demonstrates an example of a model and its corresponding physical cube. After the vertex detection process, the 2D coordinates x_i of color C_i are known. A list of color-to-2D correspondences $\langle C_i, x_i \rangle$ is built accordingly. Those two lists are joined together and a list of 3D-to-2D correspondences can be constructed from the joined list.

The PnP problem is to estimate the location and orientation of a camera with respect to an object in the scene from the list of $\langle f_k(C_i), x_i \rangle$ correspondences. The approach of Gao et al. [8], which is a fast solution with high accuracy, is adopted to solve this problem. At least three 3D-to-2D point correspondences are required to solve the PnP problem. The cube has eight vertices. Therefore, the pose can be estimated accurately even if some vertices are occluded, which improves robustness.

RANSAC is applied to eliminate outliers after solving the PnP problem. The number of inliers is a good indicator of how well the detected 2D points fit the model. Suppose there are n models defined by $f_1, \ldots f_k, \ldots f_n$. The detected 2D points x_i are checked against each of the models by constructing $\langle f_k(C_i), x_i \rangle$ for each k. The model with the highest number of inliers is chosen as the matched model.

Note that the activation process described in previous sections does not require that the cubes are built from different models. That being said, even identical physical cubes can be activated individually. However, if the users want the system to recognize different cubes and treat them differently, the cubes have to be built from different models. For example, Figure 3 shows switching between two models of two cubes. The f_k for the larger and smaller cubes are different since they have different layouts of colored corners. After activation, different 3D objects are rendered inside to demonstrate they are recognized respectively by the system.

6 Automatic Color Adjustment System

In practice, the changes in lighting environment or background may add noise to the algorithm and affect the accuracy of the device. An automatic color adjustment system is developed to dynamically generate the optimal Hue and Saturation (HS) ranges to minimize noise. The Value ranges are set to the maximum possible for robustness.

The system consists of two major parts: a monitoring and an adjustment thread. The monitoring thread runs for every frame to check if the current noise is within a tolerable range (Eq.3). The adjustment thread is triggered if Eq.3 does not hold.

After pose estimation, the 3D model of each corner is back-projected to the 2D screen, resulting in a circle-like area c. In the thresholded binary image of each corner, the pixels in c are the "correct" ones (denoted by S_{in}) while the rest are "incorrect" ones, or noise (denoted by S_{out}). The initial value of noise (denoted by S_{init}) is set to one at first, but updated to S_{out} after each run of the adjustment thread. t_0 and t_1 are lower and upper thresholds. If the background noise does not change, $\frac{S_{out}}{S_{init}}$ is one. If the noise increases significantly such that $\frac{S_{out}}{S_{init}} > t_1$, the adjustment thread is run to obtain smaller HS ranges, so noise can be filtered out (Figure 5). If the noise decreases such that $\frac{S_{out}}{S_{init}} < t_0$, the adjustment thread is run to obtain larger HS ranges for improved robustness. Note that the changes in the lighting environment can increase noise (S_{out}), but can not decrease S_{in} because the optimal ranges are much larger than the minimum ones.

$$t_0 \leq \frac{S_{out}}{S_{init}} \leq t_1 \tag{3}$$

The adjustment process has two phases: shrink and expand. In phase one, the HS ranges are shrunk to the minimum allowable values while Eq.4 holds. S_c is the area of circle c. t_s is a threshold for the shrinking phase.

$$\frac{S_{in}}{S_c} > t_s \tag{4}$$

The first phase minimizes the background noise while maintaining a good shape of the target vertex to ensure that it is detectable. The second phase is designed to expand the HS ranges for better recognition robustness without sacrificing accuracy. In phase two, starting from the minimum ranges from phase one, the HS ranges are expanded to the maximum allowable values as long as Eq.5 holds. t_e is a threshold for the expanding phase.

Figure 5: An example of the automatic color adjustment system. Left: some noise (the green box) is added to the scene. The the monitoring thread triggers the adjustment thread automatically. Middle: the thresholded image of green channel before running the adjustment system. Right: the image of green channel after running the adjustment system. The noise is filtered out with the newly generated optimal ranges.

$$\frac{S_{in}}{S_{in}+S_{out}} > t_e \tag{5}$$

The average values of the maximum and minimum HS ranges are the optimal ranges so that the optimal line to separate two colors is one that represents the largest margin. As an illustration, a group of maximum, minimum and optimal HS ranges of the cyan vertex are shown in Figure 4.

7 Performance Evaluation

Input devices can be difficult to evaluate independent of specific tasks, but it is worthwhile, if possible, to establish their performance characteristics independent of their use in practice; this allows for the identification of performance bounds that do not depend on human capabilities (e.g. [9]). We are particularly interested in the frame rate of the system and estimation error of orientation in 3D rotation tasks of the input device. In our performance evaluation, the camera is a Logitech HD Pro Webcam C920; processing is on a Windows laptop with a 2.50GHz CPU and 8GB RAM.

7.1 Frame Rate

Two factors may significantly affect the frame rate: the cube tracking functions and the number of different cube models. The cube tracking functions allow the system to detect vertices within a region R_c, which decreases the computational cost. However, R_c increases when the system fails to detect the cube (e.g., the whole cube is occluded) or there are many other objects in motion in the background. The worst case is that R_c is always equal to the entire frame. Additionally, checking against models is also computationally expensive.

The test environment is shown in Figure 2. There are multiple colorful objects in the background acting as noise, which adds additional computational cost to the system. The target cube is the smaller one. The user is asked to play with the target cube however they want as long as it does not overlap the larger one. There are three groups of experiments and the data are collected from 9,000 successive frames for each of them. For the first group, the cube tracking functions are turned on and there is only one model of the cube in the database. The area of R_c is about 10% of the whole frame, which is a typical size when users interact with the input device. For the second group, the cube tracking functions are turned off to simulate the worst case where the system always fails to track the cube. For the third group, the cube tracking functions are turned on, but there were are different models of cubes in the database.

The means and standard deviation of frame rates (in FPS) for the three groups are M = 63.75 (σ = 10.89), M = 29.42 (σ = 5.13) and M = 26.80 (σ = 2.83). We ran independent-samples t-tests (without an equal variances assumption) to compare the means. For large sample sizes (e.g. 9000 frames in this experiment), much smaller alpha values are required for significance. The alpha value is set to 0.0005 to show a strong grade of evidence according to the advice

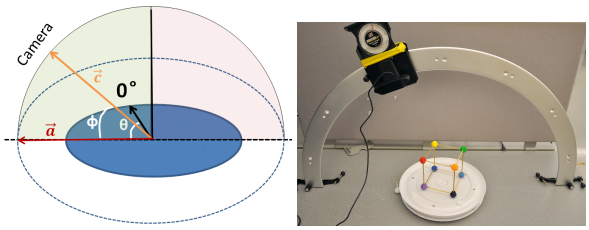

Figure 6: Left: Definition of θ and ϕ in the spherical coordinate system. Right: Setup of the evaluation system.

of Raftery [21]. The t-test that compares group one and two shows that the cube tracking functions significantly increase the frame rate on average: $t_{12800.253} = 270.521 (p < 0.0005)$. The t-test that compares group one and three shows that adding another model to the database significantly slows down the system: $t_{10211.828} = 311.408 (p < 0.0005)$. However, if two cubes in a scene are built from the same model (i.e., they are identical), the frame rate will still be 63.75 FPS on average.

Generally, the maximum capture rate of a consumer video camera like Logitech C920 is 60 FPS. This means an algorithm that works above 60 FPS is utilizing the full potential of the video camera.

7.2 Estimation of Orientation

Past evaluation of comparable input devices evaluated performance using synthesized virtual cube animations generated with Blender [2]. Here we introduce a novel approach to evaluate the accuracy of pose estimation, using a physical cube.

If a camera is abstracted to a viewpoint and the center of the physical cube is defined as the world origin, any of the possible locations of the camera can be denoted by some (r, θ, ϕ) in spherical coordinates (see Figure 6). r is not related to the rotation tasks, so the orientation of the camera is denoted by (θ, ϕ). Note that a real camera can also rotate along its "look-at" vector. But the physical cube is symmetrical regardless of its colors. So the rotation along the "look-at" vector is always equivalent to some combinations of θ and ϕ, which means it is not independent of (θ, ϕ) and the three dimensions of the camera rotation can be reduced to two. In other words, the (θ, ϕ) coordinate system is sufficient to evaluate system performance on rotation tasks. The evaluation system (Figure 6) consists of a half-circle camera track and a turntable with gradation from 0° to 360°.

Suppose $\vec{a}$ is a vector along the intersection line of the camera track plane and horizontal plane, pointing to the left. θ is the angle between $\vec{a}$ and the 0° marker line. ϕ is the angle between $\vec{a}$ and $\vec{c}$ that points to the camera. Because of the symmetry of the evaluation system, only locations that are above the horizontal plane are evaluated. Therefore, the domain of ϕ is [0°, 180°] and the domain of θ is [0°, 360°). In the experiment, we take samples of θ every 10° and eight typical angles of ϕ, resulting in 288 (θ, ϕ) combinations.

The $(\hat{\theta}, \hat{\phi})$ that are measured directly using tools are defined as ground truth. $\hat{\phi}$ is measured using a pitch and slope locater bound to the camera and $\hat{\theta}$ is measured using the gradation on the turntable.

The interval errors between adjacent measurements are chosen as error metrics. Suppose the turntable is rotated and the ground-truth coordinate is changed from $(\hat{\theta}_{i-1}, \hat{\phi})$ to $(\hat{\theta}_i, \hat{\phi})$. The estimated pose is changed from $(\theta_{i-1}, \phi_{i-1})$ to (θ_i, ϕ_i). The interval error $ThetaError_{(\hat{\theta}_i, \hat{\phi}_i)}$ is defined by (6). The evaluated errors of ϕ are computed similarly.

$$ThetaError_{(\hat{\theta}_i, \hat{\phi}_i)} = \left| \left| \hat{\theta}_i - \hat{\theta}_{i-1} \right| - |\theta_i - \theta_{i-1}| \right| \tag{6}$$

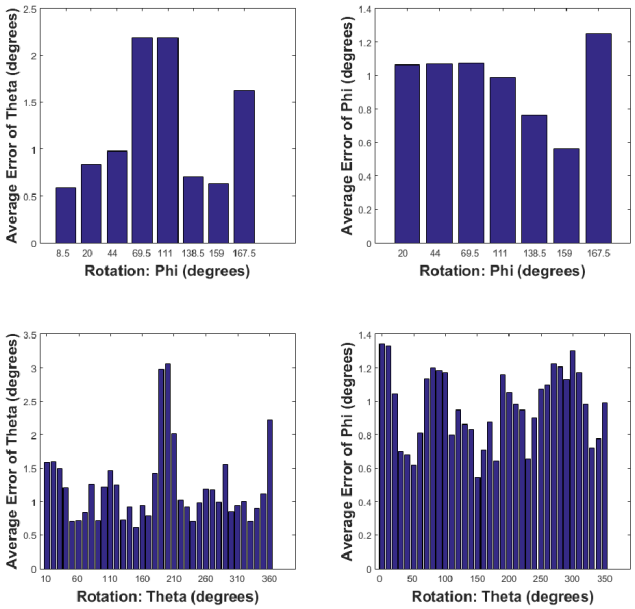

Figure 7: Average interval errors along θ and ϕ respectively with data collapsed.

There are two reasons to use interval errors instead of the absolute differences of angles as error metrics. First, it reduces systematic errors. Second, in a 3D rotation task what really matters is how accurately the object is rotated from its previous orientation.

For each $(\hat{\theta}_i, \hat{\phi}_i)$, 50 successive frames are used to evaluate the errors. The results are shown in Figure 7. The average error of θ is 1.2 degrees and the average error of ϕ is 1.0 degree. The overall estimation error is low no matter how θ and ϕ change. In practice, a user can rotate the cube bimanually and freely, which ensures that our device is an accurate 6 DoF device with an unlimited rotation range.

The pose estimation is more accurate when the object is viewed from "oblique" angles (e.g. when $\hat{\theta}$ is about 45°). However, when $\hat{\phi}$ is close to 90°, the accuracy decreases. This is because the projection of the center of the camera on the horizontal plane is very close to the origin of the cube in such cases. A small change may lead to relatively large estimation errors.

8 Experimental Study

Orientation Matching Experiments evaluate how 6 DoF devices perform in rotation tasks. The experiments conducted by Chen et al. [3] and Hinckley et al. [11] require users to rotate a 3D house from a starting orientation to a new orientation, with a threshold of 6 degrees for a match. Performance results vary widely for such an apparently simple task, depending on input device and interaction technique. In Hinckley's experiments [11] with a 2 DoF mouse, the Arcball and Virtual Sphere techniques, users took more than 25 seconds to complete the task. Poupyrev et al. [20] found that users could accomplish the task in 5.15 seconds with a Polhemus SpaceBall and isomorphic rotation (i.e., an absolute mapping between the orientation of the input device and the virtual house). LaViola and Katzourin [17] showed that duration could be reduced below 2 seconds with a 6 DoF Polhemus FASTRAK magnetic sensor in a SSVE with head tracking and a stereoscopic display. The latter result matches the performance observed by Ware and Rose [30] on the rotation of instrumented physical objects, with improvements to accuracy. Our experimental study follows the norms of previous works [3, 11, 17, 20]: test the device/system using accepted experimental designs, then report the results.

The goal of the experimental study is to further investigate how well our input device works for rotating virtual objects in a desktop environment. We followed the design of LaViola et al. [17] and Poupyrev et al. [20], based on the original experiment designed by Hinkley [11]. LaViola et al. attribute their order-of-magnitude improvements to their SSVE and the non-isomorphic rotation technique with a proper scaling factor. However, SSVE is often complicated to set up and too expensive for general use. We want to show that our system, while working with an isomorphic mapping, outperforms other 6 DoF input devices in a similar desktop environment (e.g., a Polhemous SpaceBall [20]), and has performance comparable to LaViola et al.

In pilot experiments with a same interface as Poupyrev [20] (the leftmost image in Figure 8), some participants reported that they had difficulties in telling the angular differences between the model and target houses. Therefore, we designed three modes for the interface, as shown in Figure 8.

8.1 Participants and Apparatus

Fifteen unpaid participants, twelve male and three female, were recruited from a computer science graduate student population, with ages from 21 to 27. Three of the participants had some experience with formal 6 DoF input devices while the rest of them had none. Because the participants' ability to construct the correspondence between the input device and the virtual objects also matters, we asked whether they played video games. Thirteen of them said yes and two of them reported they were not familiar with video games, though they had played them a few times.

The experiments were conducted in the desktop environment with a 17" 1920×1080 pixel monitor. The 6 DoF input device described above was used, along with a foot pedal for confirmation.

8.2 Experimental Task

Following LaViola et al. [17], and Poupyrev et al. [20], the participants were instructed to rotate a solid shaded 3D house from a random orientation such that it matched a target orientation showing the front of the house, which was denoted by a door. The participants were told to finish the task as quickly and accurately as possible. Similar to Poupyrev et al., the house was designed to provide clues for participants to understand its orientation from any angle, such as the asymmetric assignment of colors to each wall, the roof, and the unique locations of windows.

The interface modes M1 and M2 are shown on the left in Figure 8. The upper-left window displays the model house, while the upper-right window shows the target. The bottom-left window is the video view of the physical cube. The orientation of the model house is always the same as the orientation in video view. The bottom-right window displays text information, e.g. current threshold, the completion time and orientation error of each trial. Interface mode M3, on the right in Figure 8, shows the same views in a different layout, minus the target.

Users pressed the left button of the pedal to begin a trial. A randomly generated orientation of the model house was displayed for three seconds, during which users were allowed to grasp the physical cube but not to rotate it. After the delay, text information appeared and an audio beep indicated that the rotation task was started. Users could rotate the cube with both of their hands and were allowed to re-adjust their hands however they wanted. When the users believed that the orientation error was below the threshold, they pressed the middle or the right button of the pedal to confirm. If the error was actually below the threshold, the model house disappeared and the text information window displayed the completion time and orientation error of the trial. Otherwise, the trial would not stop and the text would display the current error and indicate it was

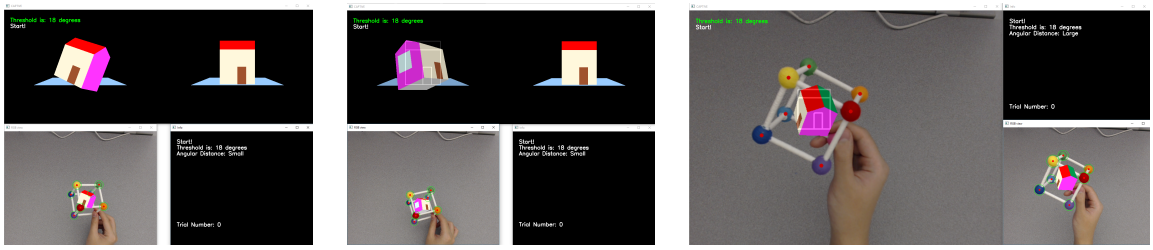

Figure 8: Three modes of interfaces in the experiments. Left: mode one (denoted by M1) is the same as the interface of Poupyrev that the target house lies over a flat plane which serves as a reference of the target orientation. Middle: in mode two (denoted by M2) adds a translucent contour of the target house to the location of the model house in order to help participants confirm the target orientation. Right: in mode three (denoted by M3), the model house is augmented to the video view and the target house is removed. A translucent contour of the front of the house is added to the location of the model house as well, to indicate the target orientation.

larger than the threshold. The participants were allowed to press the confirmation button as many times as they wanted during the trial.

8.3 Experimental Design and Procedure

The experiment was a $3 \times 2 \times 2$ within-subjects design. The independent variables were the interface modes (as described in the previous sections), the amplitude of rotation, and the orientation error threshold. The amplitude of rotation, also known as the angular range, was the angular distance from the starting orientation of the model house to the target orientation. For consistency with experiments by others, two levels were used: a *small* random angle from 20 to 60 degrees or a *large* random angle from 70 to 180 degrees. The two levels of orientation error thresholds were six and 18 degrees.

The dependent variables were the completion time and error of orientation. Completion time was the time from the start, three seconds after the users pressed the start button, to the end of a trial when the users successfully confirmed an orientation error below the threshold. Error of orientation was the angular distance between the model and target house at the end of each trial.

The experiments began with a description of the experimental tasks and procedure, the devices, and the input system. As with Poupyrev et al. [20], there was a training session that was no more than 20 minutes to stabilize the participants' performances, during which the participants could practice trials under the 12 conditions; they were also instructed on how to properly use the physical cube. After the training session, they were asked if more practice was needed. All of the participants in our experiments said no. The experimental session consisted of three groups of trials, one for each of the three interface modes. In each group, the participants were given four sets of ten trials. Each set was under one of the four conditions (two rotation amplitude levels and two orientation error threshold levels). The order of the four sets was randomly chosen to eliminate order effects.

8.4 Results

A repeated measures three-way ANOVA was used for the analysis of completion time and error of orientation, with the three factors described in the previous section: mode (M), threshold (T) and angular range (A).

Some data cleaning was necessary. In most systems, especially in the desktop environments, the input device is always separated spatially from the virtual objects. The mismatch between different frames of reference (e.g., the misalignment between the device and the screen) has a significant negative impact on the performance of users in rotation tasks [4, 29, 30]. Users' ability to map orientations from one frame of reference to another varies from individual to individual. In our experiments, 12 participants had no problems with the manipulation of the virtual object, but three participants (two male and one female) reported that they were often unable to rotate the virtual object towards the orientation they expected. In particular, one participant was unable to rotate the virtual house around the z-axis (perpendicular to the screen) no matter how long she tried. Thus the data collected from this participant was incomplete. In the analysis below, only the data from the 12 participants who successfully completed the tasks were used.

Table 1: Mean completion time (in seconds, denoted by s) and mean orientation error (in degrees, denoted by symbol $^\circ$) per condition.

	T = 6°		T = 18°	
	A= Small	A= Large	A= Small	A= Large
M = M1	2.71s	4.29s	1.35s	2.80s
M = M2	2.20s	3.62s	1.30s	2.69s
M = M3	2.49s	3.92s	1.51s	2.52s
M = M1	3.86°	3.73°	8.54°	8.84°
M = M2	3.49°	3.59°	8.26°	8.24°
M = M3	3.90°	3.78°	8.64°	8.83°

Means for completion time and orientation error are shown in Table 1, for the three modes, two threshold values and two rotation amplitude levels. The summary of the main effects and interaction of factors for completion time and orientation error is given in Table 2. We also conducted a post-hoc analysis on threshold and angular range for both completion time and error. The pairwise comparisons were performed using the Bonferroni adjustment at α = 0.05. For completion time, the participants completed the task significantly faster when the threshold was larger (t_{11} = -11.138, p < 0.01) or when the angular range was smaller (t_{11} = -12.072, p $<$ 0.01). For error, there were no significant differences between small and large angular ranges (t_{11} = -0.594, p = 0.565), but the error was significantly higher for the larger threshold (t_{11} = 10.395, p < 0.01). These results are intuitive to explain. For completion time, with a smaller threshold, the participants had to match the two houses more carefully and thus it took more time. The participants had to rotate the house for a longer distance and often needed to re-adjust their hands during the trial if they started from a larger angular range, which also required more time. For error, participants tended to finish the tasks less accurately with the larger, less-strict threshold.

Additionally, the completion time was significantly affected by the interaction between the interface mode and threshold. The results showed that the differences of completion time among the three modes were larger when the threshold was smaller, and the mean completion time of M2 and M3 were both smaller than M1 (Figure 9). This indicated when the participants worked on more accurate tasks, the translucent contour of the target or the augmented reality scene helped them to evaluate the angular differences faster in the desktop environment.

Table 2: The main effects and interaction for time and error. M: mode; T: Threshold; A: Angular range.

Effect	Time	Error
M	F(2,10) = 2.49 p = 0.132	F(2,10) = 1.74 p = 0.225
T	F(1,11) = 124.06 p $<$ 0.05	F(1,11) = 108.06 p $<$ 0.05
A	F(1,11) = 145.73 p $<$0.05	F(1,11) = 0.353 p = 0.565
M×T	F(2,10) = 10.82 p $<$ 0.05	F(2,10) = 0.20 p = 0.825
M×A	F(2,10) = 2.48 p = 0.134	F(2,10) = 0.015 p = 0.985
T×A	F(1,11) = 2.67 p = 0.130	F(1,11) = 5.31 p $<$ 0.05
M×T×A	F(2,10) = 1.3 p = 0.315	F(2,10) = 0.28 p = 0.761

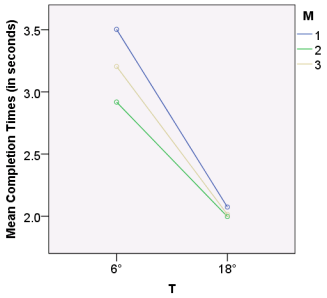

Figure 9: The interaction between mode and threshold had a significant effect on the completion time.

8.5 Discussion

Our experiments have shown improvements over other isomorphic and non-isomorphic rotation techniques in a desktop environment, with performance results comparable to a SSVE [17]. For completion time in a desktop environment, Hinckley et al. [11] reported an average of 17.8 seconds for isomorphic rotation. Poupyrev et al. [20] reported an average of 5.15 seconds for isomorphic rotation (4.75 seconds for non-isomorphic rotation). Our average completion time was 2.79 seconds, using a similar configuration (M1). This was still higher than LaViola et al. [17] using SSVE, who reported an average of 2.2 seconds for isomorhpic rotation and 1.96 for non-isomorphic rotation. For orientation error, Poupyrev et al. reported an average of 6.8 degrees and Hinckley et al. reported 6.7 degrees. Our average error over all modes was 6.2 degrees. LaViola et al. had better accuracy results, an average of 3.9 degrees of error.

We attribute our improvements in the experiment to the use of a physical cube that is a tangible free-moving device, which allows users to naturally manipulate virtual objects by moving or turning the physical cube in a familiar way. Our goal is to extend previous design principles [32, 33] to design a rotation task that is direct and increases operational speed without reducing accuracy.

Additionally, we are interested in why three of the participants were unable to cognitively map rotation of the tangible input device (the cube) to changes in orientation of its on-screen representation. Previous work [4, 29, 30] studied the negative influence of mismatches between a haptic frame of reference (i.e. the device) and a visual frame of reference (i.e. the screen) in VR environments. The tasks in our user study were more challenging than previous experiments [4, 29, 30] for two reasons. First, users had to address two additional frames of reference: the egocentric frame (i.e. the head) and the camera frame. In a desktop environment, it is impossible for a user to always look at the screen in a perpendicular direction, which leads to a mismatch between the egocentric reference frame and the visual reference frame. In addition, the orientation of the camera is fixed in our system, but the user's head was allowed to move, which resulted in a misalignment between the camera reference frame and the egocentric reference frame. Second, the angular mismatch was changing due to movements of users' heads and hands in our system while the angles were fixed in the previous controlled experiments [4, 29, 30]. The reference frame mapping issue may not only hinder the performance of users, but also prevent some of them from completing our tasks. In spite of this, we were encouraged to see most participants were able to successfully handle a complex mapping among four frames of reference and achieve good performance with our device. More training of the participants or reducing the mismatch using head tracking, SSVE, or a stereo display similar to [17, 29, 30] may help participants feel more comfortable with our device. We are also designing future experiments to study the influence of mismatches among more than two reference frames, to identify additional useful design principles for desktop environments.

9 Applications

We demonstrate our system as a 6 DoF input device with two applications. The first is the integration of our device with a stereo display system to view virtual objects. The second is to show how our device facilitates augmented reality (AR) game development and how the device can be used as a joystick-like game controller.

9.1 Stereo Object Viewer

zSpace is a Fish Tank VR device. It renders a separate frame for each of the eyes of a user so that the user will see a stereo display of a virtual object, with depth cues, while wearing polarized glasses with head tracking markers. We integrated our device with the zSpace system to make an AR viewer application. This is done by aligning the coordinate system of our camera to the zSpace coordinate systems. The cube serves as a marker to compute the relative pose between coordinate systems during the integration session. After that, a virtual object (e.g. a virtual cube) is rendered "outside" the screen (see Figure 10) based on the pose of the physical cube. Ideally, the virtual cube should overlap with the cube so that the user will feel like they were holding and viewing the stereo object. However, the user may feel disoriented when the virtual object on screen is partially occluded by the rods of the physical cube. Therefore, the virtual object is moved a little higher to avoid occlusion. Replacing the rods with translucent materials might be able to solve this problem. This is left for future work.

9.2 Cube-aided AR Game Development and Controller

Because the cube supports transformations in 6 DoF, as well as acting as a 3D marker, it is suitable for AR game development. AR games not only require manipulation of 3D objects in a virtual space but also alignment of the virtual space to the physical world, e.g., aligning a battle ground to a physical table (Figure 10). With other 6 DoF input devices, such as the SpaceNavigator or the color glove [28], a user needs to perform the alignment manually. However, with our device, the alignment can be completed simply by placing the cube on the table and recording the pose.

Game environment design often requires massive operations like object rotation and positioning. With our device, this work becomes natural to finish: a user can easily add a virtual object to the game environment with a certain pose, just like manipulating a real object. Additionally, if the environment is already aligned, the pose of the object in the physical world is the same as the one in virtual space, which saves work.

Our device can also be used as a joystick-like game controller. Although the device can provide an unlimited rotation range, its

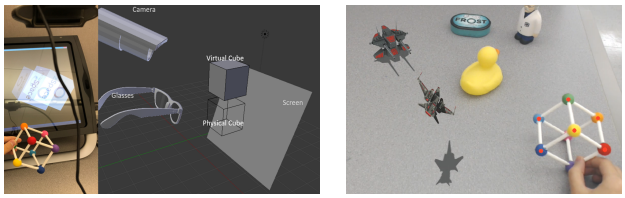

Figure 10: Left: the stereo object viewer. The cube is integrated with a stereo display device. Right: the cube-aided AR game development and controller.

translation range is still limited (as with most free-moving devices). We can map the 6 DoF information onto the velocity of an object so that its angle or position is the integration of the 6 DoF variable over time [33], which yields unlimited rotation and translation ranges. However, unlike elastic input devices that return to a neutral position automatically on release (e.g., a joystick that returns to center), a free-moving device cannot return to a neutral position by itself. We have designed two mechanisms to overcome this problem. First, a user can reset the current pose of the cube to a "neutral pose" by pressing a pedal. Second, the relative pose from the current cube to a neutral pose is used to estimate the 6 DoF information. Therefore, whenever the user presses the pedal, the velocity of the object becomes zero; the pedal is analogous to the brake pedal of a car, which makes the controlling mechanism intuitive to learn. With our controller, a user can manipulate a 3D object simply with one hand and a pedal (or a keyboard).

10 Conclusion

We have introduced a novel tangible 6 DoF input device composed of a simple wireframe cube and a single camera. The cube is recognized based on its shape and colors, and its pose is estimated accordingly. The cube tracking system makes the device work well in noisy surroundings. It also allows users to activate the cube they want to manipulate even when there are multiple cubes in the view. The cubes can be built from different models to be recognized individually. An automatic color adjustment system is designed so that the device is adaptive to the changes in lighting environment or background noise.

The performance evaluation shows that the frame rate is 63.75 FPS on average with a one model database. A systematic evaluation shows the orientation estimates of the device produce an average error of only about 1°. An experimental study with the device in use shows improved performance over other isomorphic and non-isomorphic rotation techniques in the desktop environment, approaching the performance of much more complex systems in SSVE.

We have also shown that our device is easy to integrate with other complex systems (e.g., a stereo display system) and can facilitate the development and control of interactive applications.

Currently, we need to run formal experiments to evaluate how our device performs in different lighting environments. This work is left for the future study.

References

[1] A. Butler, S. Izadi, and S. Hodges. Sidesight: multi-touch interaction around small devices. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pp. 201–204. ACM, 2008.

[2] A. Chakraborty, R. Gross, S. McIntee, K. W. Hong, J. Y. Lee, and R. St Amant. Captive: a cube with augmented physical tools. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, pp. 1315–1320. ACM, 2014.

[3] M. Chen, S. J. Mountford, and A. Sellen. A study in interactive 3-d rotation using 2-d control devices. In *ACM SIGGRAPH Computer Graphics*, vol. 22, pp. 121–129. ACM, 1988.

[4] N.-T. Dang, J.-M. Pergandi, F. Crison, J. Ardouin, and D. Mestre. Influence of orientation offset between control and display space on user performance during the rotation of 3d objects. In *EGVE/ICAT/EuroVR*, pp. 129–136, 2009.

[5] J.-B. de la Rivière, C. Kervégant, E. Orvain, and N. Dittlo. CubTile: a multi-touch cubic interface. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pp. 69–72. ACM, 2008.

[6] A. R. François and G. G. Medioni. Adaptive color background modeling for real-time segmentation of video streams. In *Proceedings of the International Conference on Imaging Science, Systems, and Technology*, vol. 1, pp. 227–232, 1999.

[7] B. Fröhlich and J. Plate. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 526–531. ACM, 2000.

[8] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng. Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):930–943, 2003.

[9] J. Guna, G. Jakus, M. Pogačnik, S. Tomažič, and J. Sodnik. An analysis of the precision and reliability of the leap motion sensor and its suitability for static and dynamic tracking. *Sensors*, 14(2):3702–3720, 2014.

[10] K. Hinckley, M. Sinclair, E. Hanson, R. Szeliski, and M. Conway. The videomouse: a camera-based multi-degree-of-freedom input device. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pp. 103–112. ACM, 1999.

[11] K. Hinckley, J. Tullio, R. Pausch, D. Proffitt, and N. Kassell. Usability analysis of 3d rotation techniques. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pp. 1–10. ACM, 1997.

[12] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Computer Vision–ACCV 2012*, pp. 548–562. Springer, 2013.

[13] HTC Vive Teardown. `https://www.ifixit.com/Teardown/HTC+Vive+Teardown/62213`.

[14] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999.(IWAR'99) Proceedings. 2nd IEEE and ACM International Workshop on*, pp. 85–94. IEEE, 1999.

[15] M. Kranz, D. Schmidt, P. Holleis, and A. Schmidt. A display cube as a tangible user interface. In *Proceedings of the International Conference on Ubiquitous Computing (UbiComp)*, 2005.

[16] S. Kratz and M. Rohs. Hoverflow: exploring around-device interaction with ir distance sensors. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, p. 42. ACM, 2009.

[17] J. J. LaViola Jr and M. Katzourin. An exploration of non-isomorphic 3d rotation in surround screen virtual environments. In *3D User Interfaces, 2007. 3DUI'07. IEEE Symposium on*. IEEE, 2007.

[18] J. Lee, S. Teerapittayanon, and H. Ishii. Beyond: collapsible input device for direct 3d manipulation beyond the screen. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 393–394. ACM, 2010.

[19] J. Liebelt and C. Schmid. Multi-view object class detection with a 3d geometric model. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 1688–1695. IEEE, 2010.

[20] I. Poupyrev, S. Weghorst, and S. Fels. Non-isomorphic 3d rotational techniques. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 540–547. ACM, 2000.

[21] A. E. Raftery. Bayesian model selection in social research. *Sociological methodology*, pp. 111–163, 1995.

[22] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 9-13 2011.

[23] B. Salem and H. Peeters. Intercube: A study into merging action and interaction spaces. In *Proceedings of INTERACT*, pp. 57–70. Springer, 2007.

[24] M. Stark, M. Goesele, and B. Schiele. Back to the future: Learning shape models from 3d cad data. In *BMVC*, vol. 2, p. 5, 2010.

[25] I. Stavness, B. Lam, and S. Fels. pcubee: a perspective-corrected handheld cubic display. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1381–1390. ACM, 2010.

[26] A. van Rhijn and J. D. Mulder. Spatial input device structure and bimanual object manipulation in virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pp. 51–60. ACM, 2006.

[27] V. Vuibert, W. Stuerzlinger, and J. R. Cooperstock. Evaluation of docking task performance using mid-air interaction techniques. In *Proceedings of the 3rd ACM Symposium on Spatial User Interaction*, pp. 44–52. ACM, 2015.

[28] R. Y. Wang and J. Popović. Real-time hand-tracking with a color glove. In *ACM transactions on graphics (TOG)*, vol. 28, p. 63. ACM, 2009.

[29] C. Ware and R. Arsenault. Frames of reference in virtual object rotation. In *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pp. 135–141. ACM, 2004.

[30] C. Ware and J. Rose. Rotating virtual objects with real handles. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 6(2):162–180, 1999.

[31] E. Woods, P. Mason, and M. Billinghurst. Magicmouse: an inexpensive 6-degree-of-freedom mouse. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pp. 285–286. ACM, 2003.

[32] S. Zhai. *Human performance in six degree of freedom input control*. PhD thesis, University of Toronto, 1995.

[33] S. Zhai. User performance in relation to 3d input device design. *ACM Siggraph Computer Graphics*, 32(4):50–54, 1998.