MagicMouse: an Inexpensive 6-Degree-of-Freedom Mouse

Eric Woods
HIT Lab NZ
eric.woods@hitlabnz.org

Paul Mason
Lincoln University, New Zealand
masonp3@lincoln.ac.nz

Mark Billinghurst
HIT Lab NZ
mark.billinghurst@hitlabnz.org

Abstract

An inexpensive computer input device was developed that allows the user to operate within both 2D and 3D environments by simply moving and rotating their fist. Position and rotation around the X, Y and Z-axes are supported, allowing full six degree of freedom input. This is achieved by having the user wear a glove, to which is attached a square marker. Translation and rotation of the hand is tracked by a camera attached to the computer, using the ARToolKit software library. Extraction, calibration, normalisation and mapping of the data converts hand motion into meaningful operations within 2D and 3D environments. Four input scenarios are described, showing that the mapping of the position and rotation data to 2D or 3D operations depends heavily on the desired task.

Keywords: Input Device, six degrees of freedom, camera, 2D, 3D

1 Introduction

With the constant increase in availability and performance of 3D graphics cards, there comes a parallel need to be able to intuitively perform operations in 3D environments such as navigation, and control. These operations benefit from the ability to translate along and rotate about the X, Y and Z axes, defined here as Xpos, Ypos and Zpos, and Xrot, Yrot and Zrot respectively. Previous research has shown that manipulation operations are most effective with the ability to operate in six degrees of freedom (DOF) simultaneously (WARE 1990).

Three-dimensional input devices are available in many forms, but generally they are either limited or very expensive, or both. The keyboard can be used to navigate in three dimensions, but it is not designed for this, so it lacks both intuitive and analogue control. Joysticks are the cheapest dedicated tools available, and can be analogue; however, they only offer more than 3 DOF by adding "hat" buttons (mini joysticks) and sliders, which are not always very intuitive. A variety of ultrasound, magnetic, mechanical and radio frequency devices can be attached to a hand to track it, but most only determine position, require the user to be tethered to the machine and are expensive.

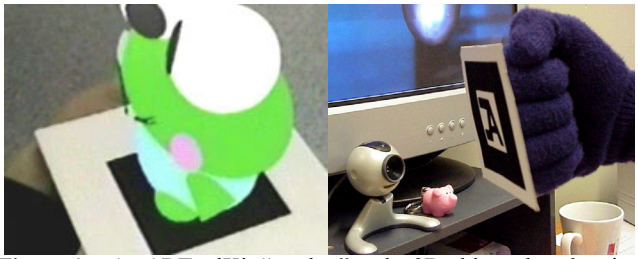

Figure 1a: An ARToolKit "marker" and a 3D object placed on it.
Figure 1b: The MagicMouse glove, marker and webcam.

This paper offers a cheap, intuitive, 6 DOF alternative using a black cardboard square, a standard USB video camera and some computer vision software (ARTOOLKIT 2002).

2 The MagicMouse

The MagicMouse has two key software components: ARToolKit which generates a transformation matrix representing the hand; and functions that extract Xpos, Ypos, Zpos, Xrot, Yrot and Zrot from the matrix, normalise them to range from –1 to 1 (using calibration data) and map them to a variety of operations.

ARToolKit is a free, open-source C software library that uses computer vision techniques to calculate camera pose (position and orientation) relative to a black square marker. It is fast enough to perform this calculation at 30 frames per second on a normal desktop PC (800Mhz Pentium III). ARToolKit is typically used for applications that augment video of the real world with computer- generated objects (**Figure 1a)**.

2.1 Extraction of Data from ARToolKit

The user wears a glove with a marker attached (**Figure 1b**). The ARToolKit library analyses each video input frame and creates an OpenGL transformation matrix that describes the position and rotation of the marker relative to the camera. Code was written that converts the transformation matrix into Xpos, Ypos, Zpos, Xrot, Yrot and Zrot. Once these 6 values are normalised they are very easily mapped to operations in 2D or 3D environments.

2.2 Calibration and Normalisation

The system requires calibration so that incoming data can be normalised to a value ranging from –1 to 1 for each DOF.

The position values are determined by calculating the position of the marker within the *viewing volume* (**Figure 2)**. Objects must be within this volume to be visible, so a calibration procedure was created that determined the maximum and minimum X and Y extents at which the marker could be seen.

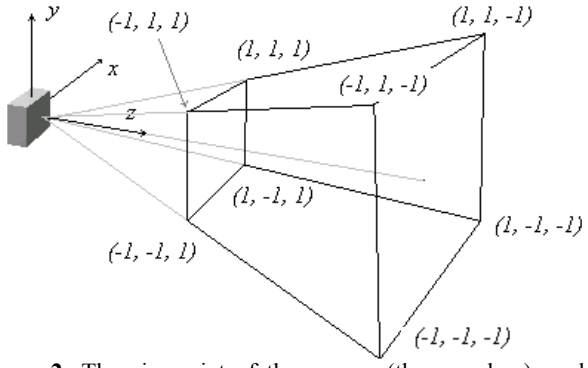

Figure 2: The viewpoint of the camera (the grey box), and the viewing volume, normalised using calibration data.

Users can set the front and back Z-depth of the volume. To ensure the extents of –1 and 1 could be reached under all

conditions, the measured maxima and minima were reduced by 10%. The resultant values were used to assemble linear equations for the boundary planes as a function of the raw *z*-value. For a given z-value the maximum and minimum x- and y-values, can be calculated. Knowing these values correspond to 1.0 and –1.0, we can then normalise the raw x- and y-values obtained from the transformation matrix. The raw *z*-values are normalised using the maximum and minimum values collected from the front and back.

The rotation values range from –1.0 to 1.0 about each axis relative to the minimum and maximum defined extents. Hand rotation is limited by the flexibility of the wrist, so the rotation extents ranged from –70° to 70°, with 0 being the resting pose.

2.3 Mapping of Data to Operations

The normalised values of X, Y and Z position and rotation could be mapped to operations in many different applications like robot control and 2D and 3D navigation. The way the data is mapped to operations is critical in ensuring an intuitive interface.

We tested simple 2D and 3D operations using both absolute and relative input. **Absolute** navigation moves the "cursor" in direct response to the position of the controller (in this case, the marker). **Relative** navigation constantly moves the "cursor" in a *direction* equivalent to the position of the marker *relative* to the centre of the control space, and in a *magnitude* that is equivalent to the distance from the marker to the centre of the control space. A standard mouse is absolute, while a standard joystick is relative.

2D navigation was implemented as mouse motion and button clicks, which were synthesised using the Windows API function `mouse_event`. This provides control of any standard windows application that responds to standard mouse events.

3D navigation was implemented as keyboard presses, which were synthesised using the Windows API function `keybd_event`. While using keyboard events limits the response and real world integration of this device, it provides sufficient functionality to trial the device in an application.

Where relative-mode movement was used, a threshold was implemented around the resting pose so that minor movement within the threshold would not make the cursor move.

There was also experimentation with having positive Zrot mapped to a Left Mouse Click and negative Zrot mapped to a Right Mouse Click. Zrot was given a rotational threshold: beyond which it synthesised a "button down" event, and below which it synthesised a "button up" event. This was not very practical, as the X and Y positions often changed as the marker was rotated, causing confusion (accidental drags etc). An alternative would be to do such actions with the spare hand or a small wireless device.

2.4 Resultant Operations and their Usefulness

Four different scenarios were tested as described below:

2D Absolute-Mode Mouse

This uses Xpos and Ypos to synthesise standard mouse movements and Xrot to synthesise the scroll button. It is particularly good for painting programs where normal mouse operation can be clumsy. If the camera resolution is low, the cursor may have an error of up to 4 pixels. The ability to use Xrot for scrolling in text documents was found to be very intuitive.

2D Relative-Mode Mouse (like a 2D Joystick)

Either Xpos and Ypos or Xrot and Yrot are used to "push" the cursor across the screen, and Xrot is used to synthesise the scroll button. This is very similar to the 2D Absolute-Mode Mouse; however, the Relative-Mode offers finer control over the cursor. Due to the incremental nature of this scenario, natural hand jitter is minimised, producing surprisingly steady control. Use of the `mouse_event` function makes analogue control possible. For example, a large Xpos makes the cursor move rapidly right.

3D Absolute-Mode Mouse

Xpos, Ypos, Zpos, Xrot, Yrot and Zrot are mapped directly to the respective properties of a 3D cursor in a fixed volume. This scenario is best suited to fixed-volume operations, e.g. viewing a 3D object from any position or direction. It is not well suited to navigation within an "infinite" volume like a flight simulator. It also has restricted rotation if the Xrot, Yrot and Zrot values of –1 and 1 are mapped directly to -70° and 70°. This is easily overcome if values of –1 and 1 are mapped to -180° and 180°.

3D Relative-Mode Mouse (like a 3D Joystick)

Similar to the 3D Absolute-Mode Mouse, except movements and rotations "push" the 3D cursor in that direction or rotation. This scenario is closest to conventional joysticks (which work in Relative-Mode) except it has an amazing 6 DOF. This makes it well suited to navigation within an "infinite" volume like a flight simulator or a "run and shoot" game, where the 3D cursor controls the position, rotation and subsequent point of view of the aeroplane or person etc. It is completely intuitive and surprisingly accurate as the incremental nature of the scenario smooths out natural hand jitters. Complex navigation such as rotating to the left while moving to the right is much easier than comparative operations using mouse and keyboard combinations.

3 Conclusions and Future Work

This paper has demonstrated that for the cost of a USB camera it is possible to create an intuitive, 6-degrees-of-freedom mouse. Considerable effort was invested in calibrating and normalising the values for Xpos, Ypos, Zpos, Xrot, Yrot and Zrot so that they could be easily mapped to various operations. Use of four mapping scenarios showed that the mapping of the position and rotation data to 2D or 3D operations depends heavily on the task.

This initial version of the MagicMouse did not synthesize Windows joystick input. Creating an implementation of this device that acts as a true joystick driver would maximise the potential of this device, making it both analogue and far easier to integrate into third party applications.

Relative-mode thresholds were implemented as a square area (+-20 units in the X and Y axis, making a 40x40 square). Circular areas should be investigated in case they are more intuitive.

As the resolution of the camera is lower than that of the computer display, undersampling can affect the accuracy of the absolute-mode scenarios. Combined with natural hand instability, jitter of a few pixels was commonplace. While this is of little consequence for some applications, noise reduction algorithms could be applied to this data to reduce jitter. As higher resolution cameras become commonplace, higher frequency sampling should also be possible, making the system more responsive.

The ARToolKit has the ability to associate different markers with different actions. With a variety of scenarios possible, it would be attractive if the user could simply switch between them by simply switching markers, e.g. the user could pick up a marker with the pattern of a musical note to play music.

A variety of other limitations have been recognised with the current implementation. The necessity to keep a fist raised for extended periods of time may cause fatigue; however, the current dangers of repetitive micro movements may be combated by the ability to transform the computer interface experience into a more physical one. This may require alternatives to the desk mounted screen and chair be investigated. The ability to click buttons with the hand that is wearing the marker would be quite attractive.

References

WARE, C. 1990. Using Hand Position for Virtual Object Placement. *Visual Computer* 6 (5): 245-253.

ARTOOLKIT 2002 HTTP://WWW.WASHINGTON.EDU/ARTOOLKIT/