Using Inertial Data for Locomotion in Virtual Environments

Zachariah W. Kohn

Abstract

This technical report describes a simple method for using inertial data to facilitate locomotion in virtual environments. The means of locomotion for virtual environments contributes greatly to the users' sense of presence in the environment. Although walking-in-place is not so realistic as real walking, it is a very cost-effective alternative. Using a simple version of this algorithm both Type I and Type II errors were decreased compared to a previous walking-in-place algorithm. Type I errors occur when the algorithm detects a step when none occurred. Type II errors occur when the algorithm does not detect a step when one did occur. Type I errors decreased from 3% to 1% and Type II errors decreased from 32% to 11%. The more complex version of this algorithm promises even better results.

Introduction

The means of locomotion in virtual environments plays an important role in the user's sense of presence [Slater 1994]. Additionally, walking-in-place (virtual walking) as a means of locomotion in virtual environments is a cost-effective alternative to other forms of locomotion. The cost for equipment and lab space is much less than other techniques such as wide-area tracking. Unfortunately, the innaccuracy of many virtual walking methods is problematic. Previous algorithms based on a neural network had Type I and Type II errors as 3% and 32% respectively [Usoh, 1999]. However, the use of inertial information to detect footsteps presents a potentially cheap, accurate alternative to current means of locomotion. The technique described in this report can provide good results at a fraction of the cost of other techniques.

In this case inertial data is gathered by a Crossbow Accelerometer and a National Data Acquisition board. The accelerometer is fastened to the head mounted display (HMD). This location was chosen because of the existing attachment points and cables that are used for the HMD. Other locations such as the foot or back could also be used, but these possibilities have not been studied.

Inertial Data

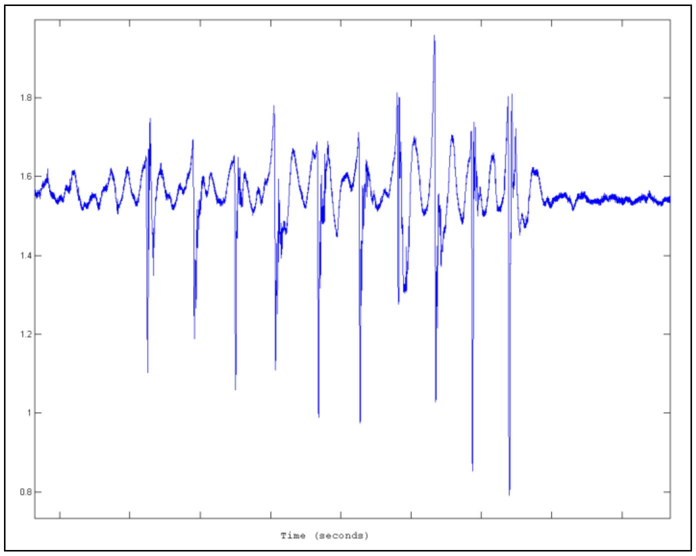

Figure 1. Inertial data from 10 footsteps (Time v Voltage)

As the body moves and interacts with the environment (feet hitting the floor as a step is taken) shock waves vibrate through the body. These vibrations can be captured by the accelerometer, which measures inertial information that can be converted into digital data.

Figure 1 shows ten steps occurring in approximately nine seconds. Each spike represents a footstep. The inherent noise in the signal can be seen in the first and last seconds of the graph. This is data for the accelerometer's Z dimension. The Y-axis is raw voltage. The accelerometer gives a reading of approximately 1.5 volts (in Z) while at rest. Acceleration due to gravity accounts largely for this value. Since most of the interesting vibrations occur in the vertical dimension, this is the portion of data used to detect steps in this algorithm.

Algorithm

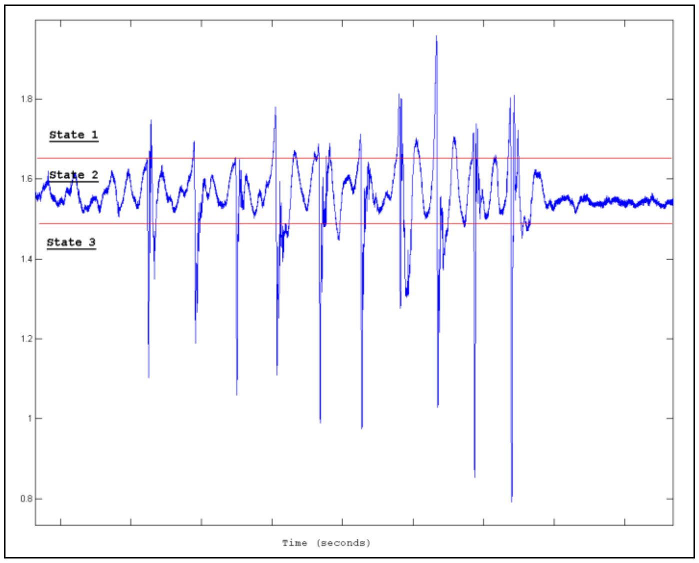

Figure 2. Static thresholding (Time v Voltage)

The general idea behind this algorithm is to look for large, fast changes in the inertial data. This is accomplished by using a three-state thresholding system. This system, shown in Figure 2 above, breaks the voltage readings into three ranges. States 1 and 3 are for extreme readings, while the purpose of State 2 is to provide a noise region. The algorithm looks for changes in the data that go from State 1 to State 3 in a short period of time. It searches for the spikes that can be seen in Figure 2.

Specifically, when the system detects a change from State 1 to State 2 to State 3 and back to State 2 again, it records a step. The reason to detect state changes from 1 to 3 is to look for the downward spike. The downward spike represents a change in acceleration from positive to negative with State 2 representing no acceleration. The reason to wait until State 2 is entered again before announcing a step has occurred is to ease some of the bookkeeping. This does introduce a small amount of latency but this small considering other delays in the virtual environment display system.

The figure above shows the two threshold values between states held constant. Although this makes the system simple it still provides enhancements over previous walk-in-place algorithms. Accuracy can be further increased by allowing for the threshold values to be changed dynamically (called adaptive thresholding). With adaptive thresholding the values for the thresholds do not need to be invented. Different accelerometers may give slightly different readings in different conditions. Additionally, adaptive thresholding can adapt to a user's step. The techniques for adaptive thresholding are described in the following section.

Definitions

The following is a list of definitions used in the description of the algorithm:

Baseline

The baseline is the average voltage reading over time.

Basenoise

Basenoise is a measure of the standard deviation of the voltage while the accelerometer is held in a fixed position. This is a measure of the error inherent to the system.

Lower/Upper Threshold

These are defined as the boundaries between the three states. They can be dynamic or fixed.

Lower Threshold Bound

The Lower Threshold Bound is defined as a bound above which the Lower Threshold can not move. The Bound is specified as being just below the Basenoise range from the Baseline.

MINTIME

A constant value that constrains how far apart steps (and thus spikes) should be. A value of 1/4 of a second is conservative and corresponds to four steps in a second.

MAXTIME

A constant value that constrains how long a step should take. This number indicates how long it should take for the voltage to move from State 1 to State 3. A value of 2/3 of a second gives good results.

Upper Threshold Bound

The Upper Threshold Bound is defined as a bound below which the Upper Threshold can not move. The Bound is specified as being just above the Basenoise range from the Baseline.

Baseline

The baseline is important because it serves as a means to estimate where and how the thresholds should be set. Using an estimate instead of an actual average keeps the bookkeeping costs down. To keep track of the baseline an adaptive estimator (based on Van Jacobson's algorithm) is used. The basic form is shown below:

```
baseline =  baseline + γ × (current_value - baseline)
```

This is changed slightly to decrease the value of **γ** over time. This is necessary because **baseline** is initialized to some value that may be far away from where the average should be. The value of **γ** should be small to keep the **baseline** smooth. However, if **γ** is too small initially it takes a long time for **baseline** to reach a good approximation of the actual average value. In this algorithm **γ** starts out at 1 and is changed on each of the first 20 or 50 successive passes though the algorithm. **γ** is changed in the following manner:

```
γ = γ + (.0001 - γ) / 2
```

The value .0001 was chosen because it gives a smooth average for the inertial data.

Adaptive Threshold I

As described above, having the thresholds constant can cause a number of problems. Using a technique called adaptive thresholding, the thresholds can be changed to use the baseline value as an estimate of where they should be. The sensitivity can be increased or decreased by changing variables that dictate how close to the bounds the thresholds are set. With the first technique the thresholds vary a specific amount above and below the baseline of the signal over time:

```
upper_thresh = upper_thresh_bound + (upper_thresh_bound - baseline) × up_amount
lower_thresh = lower_thresh_bound + (lower_thresh_bound - baseline) × down_amount
```

The **up_amount** and **down_amount** are constants that specify how quickly the upper or lower threshold and the upper or lower bound converge. Good values for these constants are .25 and .25. With these values the upper and lower thresholds are set to always be one quarter of the distance between the bound and the baseline from the bound.

Adaptive Threshold II

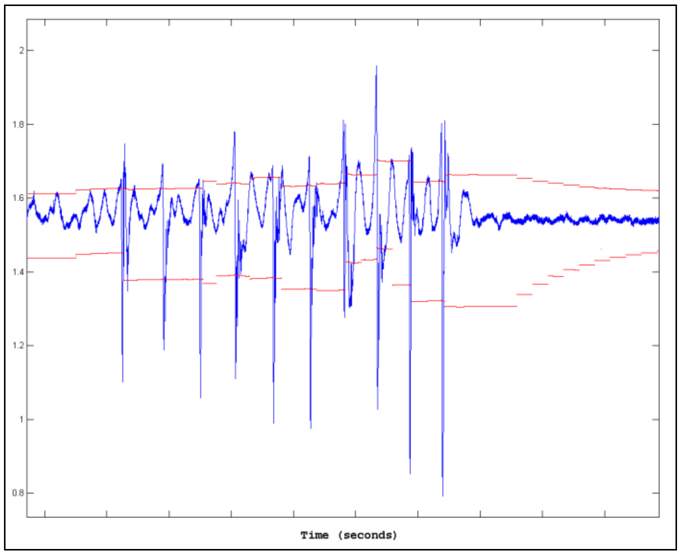

Figure 3. Adaptive thresholding (Time v Voltage)

The second technique incorporates the first technique and adds more complexity. This technique changes the thresholds based on the height of the spikes. Expansion of the thresholds moves them out and away from the baseline (the upper threshold moves up, the lower moves down). The thresholds are compressed if no steps occur over a certain period of time. They compress increasingly until reaching some limit (the Upper/Lower Threshold Bounds).

This expansion and compression is controlled with the **up_amount** and **down_amount** variables. These two variables are changed when a step occurs, or when no step has occurred for some amount of time. They are changed by some percent of their previous value:

```
up_amount = up_amount + up_amount × β
down_amount = down_amount + down_amount × β
```

Another approach changes the amount based on the height of the spikes compared to the baseline value. This technique is shown below:

```
up_amount = (β × (highest - upper_thresh_bound)) / (upper_thresh_bound-baseline)
down_amount = (β × (lowest - lower_thresh_bound)) / (lower_thresh_bound-baseline)
```

With this idea the **up_amount** is set to the value that will solve the following equation (**down_amount** is solved in a similar manner):

```
upper_thresh_bound + β × (highest – upper_thresh_bound) = upper_thresh_bound +
        (upper_thresh_bound – baseline) * up_amount
```

This equation (indirectly) describes that the **upper_thresh** should be set so that it is **β** of the way between the **upper_thresh_bound** and **highest**.

In Figure 3 the latter technique is shown. When the algorithm starts no steps occur for the first second, but the thresholds are compressed to the Upper/Lower Thresh Bounds. As steps occur the thresholds expand. After approximately seven seconds, no more steps occur and the thresholds slowly compress.

The following figure shows the algorithm (with adaptive thresholding II). Pink circles indicate spikes that were deemed to be steps and are actual steps. The green cross indicates a spike that the algorithm incorrectly identified as a step, although none actually occurred.

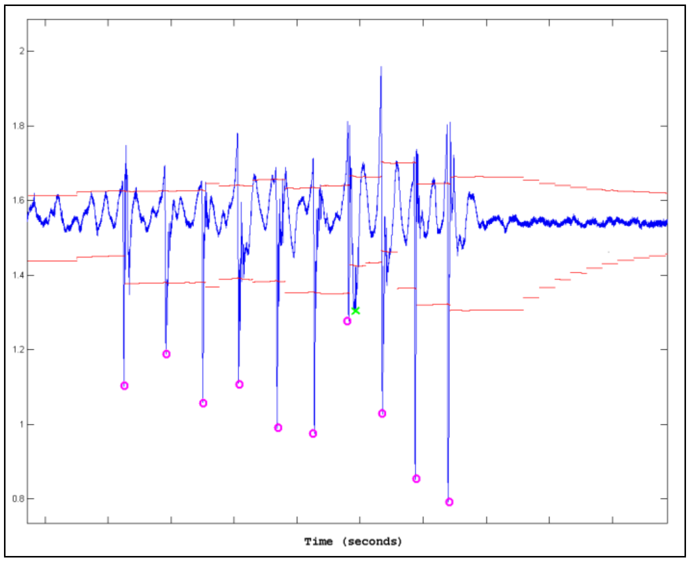

Figure 4 Detecting steps

Results

To measure the performance of this algorithm compared with the previous neural network implementation the Type I and Type II errors that occurred with each method were studied. Type I errors occur when the algorithm detects a step when none occurred. Type II errors occur when the algorithm does not detect a step when one did occur. Type I errors are more serious because they make the user move through the virtual environment without their control.

The error was measured by counting the number of Type I and Type II errors that occurred compared to the overall number of steps taken. These results are from the simplest version of the algorithm that does not use the adaptive thresholding techniques.

	Type I errors (%)	**Type II errors (%)**
Neural Network	3	32
3-state thresholding (static)	**1**	**11**

The table indicates that this algorithm greatly improved the Type II errors and slightly improved the Type I errors. The Type II errors that did occur can be attributed to the users having his/her head in an orientation other than vertical. Since the algorithm only looks at data in the accelerometer's Z dimension, if this is not aligned with the world's Z axis then it does not receive all the information it needs.

The adaptive thresholding techniques improve on the accuracy of the static version of this algorithm. Preliminary tests show that the adaptive technique all but eliminate Type I errors and the dynamic nature of the thresholds also decrease Type II errors.

Another favorable factor is the small lag time relative to the neural network approach. The neural network has lag inherently built in as it uses past data along with present data to detect footfalls. This lag was as much as 1-2 seconds in previous walk-in-place algorithms. The acceleration algorithm is much faster to determine when a footstep has occurred.

Future Work

Many of the Type II errors that still occur using this approach are caused by a non-vertical head orientation. Because the current algorithm only looks for changes in inertial information in the vertical dimension, any rotation around the X or Y dimensions decreases the accuracy of the algorithm. For example, if the accelerometer is rotated by 90° so that the Z axis now lies horizontally then all inertial data is in the accelerometer's X dimension.

One possibility to solve this problem would be to use the orientation of the head to rotate the information from the accelerometer so that it was in world coordinates. Then the vertical component of this information can be used as if the head was perfectly upright.

References

Usoh, M., K. Arthur, M. C. Whitton, R. Bastos, A. Steed, M. Slater, and F. P. Brooks, Jr. "Walking > Walking-in-Place > Flying, in Virtual Environments." *To appear in proceedings of SIGGRAPH 99* (Los Angeles, CA, 1999).

Slater, M., M. Usoh and A. Steed, 1994: “Depth of Presence in Virtual Environments,” *Presence: Teleoperators and Virtual Environments,* MIT Press, 3, 2: 130-144.