Assignment 1 – Jedi Trainer
CAP6121
Due: 2/14/10 11:59pm

"…help me Obi-Wan Kanobe, you're my only hope…"

In order to help train young Jedi, we require a set of training tools to help them with their lightsaber skills. In this first assignment, you will build a Jedi Training simulator similar to the one shown in the figure. The goals for this first assignment will be to get familiar with Microsoft XNA, the 3DUI XNA framework, stereoscopic viewing, and working with the Wiimote.

Minimum Requirements

The main goal of the Jedi Trainer is to help young Jedi with their lightsaber skills. To do this, your application must have two modes. First, it must contain a training droid, like the one shown in the figure, that moves around randomly (or pseudo randomly) and shoots lasers at you at varying times. The user's job is to deflect these lasers with the lightsaber so he/she does not get hit. Second, the simulator will have attack droids that come after you. The user's job in this case is to fend off these droids by using different lightsaber attacks. You will use a Wiimote as your proxy for the virtual lightsaber you render on the screen. Note that in this game you will not have to travel in the environment, all attack droids will come to you. The game must work in 3D stereo and you should utilize it to assist in gameplay. How can 3D stereo improve the player's overall experience? This question needs to be addressed in your implementation.

You must develop two strategies for controlling your lightsaber. In the first strategy, since you know the orientation of the Wiimote, you should be able to create an interaction technique that mimics its motion. Utilize the Wii MotionPlus as well as the accelerometer's from the Wiimote. Try using them separately and see if one is better than the other for this task. For the second strategy, you will create a simple gesture recognizer for the Wiimote.

Wiimote Gesture Recognizer

Your gesture recognizer will recognize the following gestures.

- Zorro - user slashing a Z shape in the air in front of them
- Slash - a quick motion from the top right to the bottom left in front of the user
- Stab - a quick forward thrust with the Wiimote
- Parry - hold the Wiimote facing upwards, and flip it over while making a semi-circle motion to the left.

To do this you will utilize Rubine's gesture recognition algorithm. This algorithm is a simple feature-based linear discriminator. For more info see

Rubine, D. "Specifying Gestures by Example". In Computer Graphics, Volume 25, Number 4, July 1991; pp 329-337.

The basic approach is to first collect a set of training samples from the Wiimote for each gesturer (10-20). Then you can compute features from these samples and train the learning algorithm.

The features you should use are

- the total duration of the gesture
- the maximum x, y, and z values
- the minimum x, y, and z values
- the mean x, y, and z, values
- the median x, y, and z values
- the length of the diagonal of the bounding volume
- the sine of the starting angle in the XY plane
- the cosine of the starting angle in the XY plane
- the sine of the starting angle in the XZ plane
- the sine of the angle from the first to last points in the XY plane
- the cosine of the angle from the first to last points in the XY plane
- the sine of the angle from the first to last points in the XZ plane
- the total angle traversed in the XY plane
- the total angle traversed in the XZ plane
- the absolute value of the total angle traversed in the XY plane
- the absolute value of the total angle traversed in the XZ plane
- the squared value of the total angle traversed in the XY plane
- the squared value of the total angle traversed in the XZ plane
- the Euclidian distance between the first and last points
- the total distance traveled by the gesture
- the maximum acceleration squared

The classifier itself is straightforward. First, the mean value of each feature for each gesture is computed. A covariance matrix for each gesture is then constructed from the

feature vectors of the training samples and their means. These covariance matrices are then averaged together to create a single common covariance matrix. This matrix is inverted and multiplied by the mean features for each gesture, which yields a weight value for each feature per gesture. To classify a gesture, its features are computed and a linear sum of those features and the weights for each gesture are found, the gesture with the largest result is the one which is considered most likely by the classifier. There is a final rejection criterion which tests if the likelihood of each gesture is relatively similar. In this case, assume the gesture made was ambiguous, and discard its results.

The mathematics is described below.

The classifier is defined as

$$v_{\hat{c}} = w_{\hat{c}0} + \sum_{i=1}^{F} w_{\hat{c}i} f_i \quad 0 \le c < C$$

where F is the number of features,

$w_{\hat{c}}$ is the weights, and the classification

of symbol g is the c that maximizes $v_{\hat{c}}$

For each symbol,

$$\overline{f_{\hat{c}i}} = \frac{1}{E_{\hat{c}}} \sum_{e=0}^{E_{\hat{c}}-1} f_{\hat{c}ei} \quad \text{where } 0 \le e < E_{\hat{c}}$$

and $E_{\hat{c}}$ is the number of training samples per class

To compute the weights, first we need the common covariance matrix for each class

$$\Sigma_{\hat{c}ij} = \frac{1}{E_{\hat{c}} - 1} \sum_{e=0}^{E_{\hat{c}}-1} \left(f_{\hat{c}ei} - \overline{f_{\hat{c}i}} \right) \left(f_{\hat{c}ej} - \overline{f_{\hat{c}j}} \right)$$

Then the overall covariance matrix can be found

$$\Sigma_{ij} = \frac{\sum_{c=0}^{C-1} \Sigma_{\hat{c}ij}}{-C + \sum_{c=0}^{C-1} E_{\hat{c}}}$$

The weights can finally be calculated using the common covariance matrix and the mean feature vectors from each class

$$w_{\hat{c}j} = \sum_{i=1}^{F} \left(\Sigma^{-1}\right)_{ij} \overline{f_{\hat{c}i}}\ ,\ 1 \le j \le F$$

$$w_{\hat{c}0} = -\frac{1}{2} \sum_{i=1}^{F} w_{\hat{c}i}\ \overline{f_{\hat{c}i}}$$

Note a matrix library will be provided to assist in taking inverses.

Collision Detection

You will need to know when a laser hits the lightsaber and when your lightsaber hits an attack droid, so some form of collision detection is needed. You should try to get PhysX working with your game using the contents of the directory provided on the isuelab drive. If all else fails you can make use of XNA's collision detection system.

The Virtual World

The world you create for the Jedi Trainer is completely up to you and your imagination. The models you use for the training droid, the attack droids and the lightsaber are completely up to you. The only requirement that I have is that you try to incorporate sounds in the game and try to add special effects for the lightsaber. You can also make the scoring for the game anyway you wish.

Deliverables

You must submit a zip file containing your source and any relevant files needed to compile and run your application. Also include a README file describing what works

and what does not in your application, any known bugs, and any problems you encountered. To submit, you can email me your zip file. Please not that both team members must submit an individual README describing what parts of the assignment you worked on and what parts your partner worked on.

Grading (based loosely based on the following):
60% correct functionality
30% creativity
10% documentation