Advanced Animation Techniques in a Dismounted Soldier System

Scott M. Johnson, John Carswell
Intelligent Decisions, Inc.
Orlando, FL
sjohnson@intelligent.net, jcarswell@intelligent.net

Pat Garrity
U.S. Army Simulation and Training Technology Center
Orlando, FL
pat.garrity@us.army.mil

ABSTRACT

The Dismounted Soldier Training System (DSTS) is a program of record with systems fielded by PEO STRI throughout the US Army. The system provides a hardware platform that instruments each Soldier trainee with eight worn Inertial Measurement Unit (IMU) based motion tracking sensors and a motion tracked, instrumented weapon. The U.S. Army Research Laboratory, Human Research and Engineering Directorate, Simulation and Training Technology Center (ARL-HRED-STTC) is performing research and development to leverage the motion tracking capabilities of the DSTS system as well as emerging motion tracking technologies to develop a more seamless and natural fusion of soldiers' physical movements with their body movement within the virtual environment and interactions with objects in it. Achieving this objective requires the injection of real-time data from the motion tracking system into the animation system of the underlying game engine in order to control the virtual avatar. Game engine frameworks provide mechanisms that support injection through features such as forward and inverse kinematic solvers and animation blending. Individually, these features are adequate to support simple representations of the soldiers' actions, but more complex actions require a fusion of techniques. This paper describes our approach to solving the challenges in fusing many animation techniques together towards the goal of suspension of disbelief that the virtual avatar's motion is entirely the motion of a single Soldier.

ABOUT THE AUTHORS

Scott M. Johnson is a Principal Engineer for Intelligent Decisions. He was the Software Technical Lead for the U.S. Army's Dismounted Soldier Training System and was recognized with the PEO STRI 2012 Modeling and Simulation Award for his work. He has nine years of experience in simulation and training and ten years of experience as a video game developer. He has a Master's Degree in Computer Science and Electrical Engineering from the University of Michigan in 1994 and a B.S. degree in Electrical Engineering from Purdue University in 1992.

John Carswell is a Chief Engineer for Intelligent Decisions. He is responsible for research and development activities and new technology integration with ID's Simulation and Training business unit. He has over twenty-six years engineering experience in simulation and training domain both in commercial product development and military research. He has been involved in dismounted soldier and immersive training technology for the last twelve years. He earned his B.A. in Computer Science from Stetson University in 1989.

Pat Garrity is a Chief Engineer at the U.S. Army Research Laboratory, Human Research and Engineering Directorate, Simulation and Training Technology Center (ARL-HRED-STTC). He currently works in Dismounted Soldier Simulation Technologies conducting research and development in the area of dismounted Soldier training & simulation where he was the Army's Science & Technology Manager for the Embedded Training for Dismounted Soldiers program. His current interests include Human-In-The-Loop (HITL) networked simulators, virtual and augmented reality, and immersive dismounted training applications. He earned his B.S. in Computer Engineering from the University of South Florida in 1985 and his M.S. in Simulation Systems from the University of Central Florida in 1994.

Advanced Animation Techniques in a Dismounted Soldier System

Scott M. Johnson, John Carswell
Intelligent Decisions, Inc.
Orlando, FL
sjohnson@intelligent.net, jcarswell@intelligent.net

Pat Garrity
U.S. Army Simulation and Training Technology Center
Orlando, FL
pat.garrity@us.army.mil

INTRODUCTION

The fundamental problem of a virtual dismounted Soldier simulator is measuring the motion of the Soldiers in real time and displaying a representation of those Soldiers in the virtual environment. The Dismounted Soldier Training System (DSTS) is a fielded simulator that uses Inertial Measurement Unit (IMU) motions trackers to measure the Soldier's real time pose and then display that pose in a virtual environment using a game animation system. There is ongoing research to improve that process from the currently fielded system by using the Unreal 3 game engine and keeping the existing man worn hardware as is. The solution should enable the training participants to move and shoot in a natural way. This includes allowing the Soldier to aim and fire his weapon with a familiar sight picture. And finally, the captured pose must be replicated across a network so that all the training participants can see the same representation of the Soldier. This paper explores the many animation techniques that were fused in order to represent the Soldier virtually.

Related Work

(Zhu 2004) used IMUs to capture a real-time pose by fusing the data from multiple on board sensors into a single orientation. (YEI 2012) demonstrates their commercial motion capture suit with seventeen sensors on a character in Unreal 3. Their work is to show the fidelity of the captured data using their sensors. (RNI 2011) used a seventeen sensor MVN motion tracking suit (Roetenberg 2009) to capture a real-time pose, then they demonstrate it in Unreal 3 and show some interactions with weaponry. They, however, are not generating a first person view with a sight picture that could be used for aiming.

The DSTS fielded with real-time motion capture capability using Virtual Battle Space 2 (VBS2). The virtual character's head and the rifle are controlled by an IMU head sensor and an IMU rifle sensor. Arm sensors were used for gesture recognition to perform actions such as opening doors. Thigh sensors were used for posture detection such as standing, crouching and going prone. The problem faced in this paper is very similar to the problem in VBS, but this work extends the VBS based capability to using arm sensors with more free movement of the arms. In this work, the Soldiers can release the left hand from the weapon and freely move their arms. They can give arm signals and shoot around corners. The work from this paper separates the task of getting the functionality to work in Unreal 3 and then getting it to work in VBS. This work is directly influencing the work done on the DSTS program to bring the extra arm motion and improved interaction with the rifle to the DSTS program.

DSTS Hardware

Since the problem is restricted to the DSTS hardware, it is instructive to first learn some details of the DSTS. Figure 1 shows a Soldier in the DSTS wearable hardware. He is wearing a Helmet Mounted Display (HMD) that is driven by a backpack computer. He is instrumented with motion tracking sensors on his head, arms, forearms, and thighs as shown in Figure 2. He carries a simulated M4 that is instrumented with its own motion tracking sensor. The M4 has a joystick on the front handle that allows the Soldier to remain stationary in his own safe space while his avatar moves around the virtual world. The M4 has an analog pressure sensor that measures the applied pressure to the butt stock.

Figure 1. A Soldier wearing the DSTS hardware

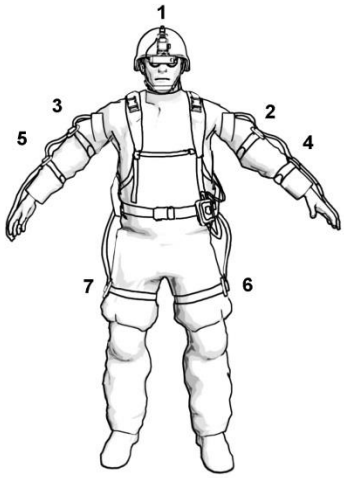

Figure 2. Placement of the sensors on each Soldier

Capturing a Real-Time Pose

The goal is to construct an accurate real-time pose of the Soldier using the DSTS hardware. From a practical point of view though there are many sources of error. First, it is very important to note that the sensors used on the DSTS system measure three degrees of freedom (3DOF) and provide the system with only their current rotation in space. They do not directly know their positions in space. It is only after applying the rotations of the sensors to the bones of an animated character that the sensor rotations become associated with a position. Figure 3 shows the Unreal 3 character that was used in this research. He is six foot two in the virtual world. One can imagine that the rotations from a human Soldier that is five foot two applied to an Unreal character that is six foot two will create some error.

Figure 3. Soldier Model in Unreal 3

Another source of error is misalignment between an ideal pose and the Soldiers actual pose during the calibration process. Calibration is the process where the rotation of the physical sensors is measured to establish the alignment of each sensor relative to corresponding bone on the Unreal character. At the time of calibration, the Soldiers are all asked to stand momentarily in an attention pose. At that moment the system has an internal ideal representation of the attention pose and each bone in that pose is compared against the actual rotation of the bone's sensor. While the system must assume that the Soldiers actual pose matches the ideal, in practice, the Soldiers take a pose that is *approximately* the ideal calibration pose that the system is expecting. Calibration is a mandatory portion of real-time motion capture systems, but typically it is only done on one to three actors at a time. These actors are trained or coached in the calibration process, and calibration is repeated until good correlation is achieved. However, the DSTS system regularly calibrates nine to eighteen Soldiers at a time and the Soldiers are not trained as motion capture actors. There is no time to visually inspect the results and iteratively make corrections for every Soldier so the error remains. The real-time animation software must be resilient to these alignment errors in order to be effective in a deployed system.

A final source of error in capturing the pose is that there are not enough sensors in DSTS to capture the full range of human motion. For example, the movement of the clavicles and the rotation of the shoulders compared to the hips are missing. To fill in information that is not measured by sensors, our animator created a base pose for the sensor

data to be applied to in real time. The base pose is a left foot forward combat ready stance holding the M4 with two hands. The sensor rotations are applied to this pose. This has proved to be an adequate visual approximation, but still is not *exactly* accurate to the Soldier's true pose.

Figure 4. The Aim Pose

Strategy for Aiming

Now those accumulated sources of error will become a driving force in the design of the solution. Consider the problem of having a Soldier aim his rifle as in Figure 4. His hands are precisely on the rifle. The rifle butt stock is against his shoulder. His view is directly down the sights. In terms of an animation system with a skeleton of bones, this means that there are four constrained chains of bones: the head, left hand, right hand, and rifle. Each chain ends with an effector that must be exactly placed relative to each other. Now consider again that the capture of the real time pose using the sensors is only approximate. Simply applying the real-time sensor data to the skeleton is not enough to satisfy the four constraints. When the Soldier is in the pose of Figure 4, the left hand in the virtual world can be tens of centimeters away from the rifle handle. Without some assistance, the accuracy of the capture process is not adequate to meet the constraints of the aim pose.

But the aim pose has fewer degrees of freedom. When aiming, the Soldier's entire upper torso from his chest upwards is fixed and the rotation comes from his torso and the rifle. So the arm sensors can be ignored when in this pose and the rotation of the upper torso can be driven from the rotation of the rifle.

A strategy was developed to create three modes of operation. In Mode 1, all the sensors are used and no constraints are maintained. The arms are free to move and give arm signals to other trainees in the simulator and the head can look anywhere. The rifle is attached to the avatar's right hand and moves with it. In Mode 2, the rifle is up against the Soldier's shoulder and the hands are on the rifle. The rifle rotates about the point where it touches the Soldier's shoulder. The head is free to move around independent of the rifle. Mode 3 is the full aim pose and is the same as Mode 2 except that the Soldier is aiming using a sight picture. The Soldier can always freely move his head but he will only get a sight picture when his head is aligned to the rifle.

The transitions between the modes are driven by the stock pressure sensor on the rifle. Mode 1 is active when there is no stock pressure. This means that the rifle is not rooted to the shoulder. Mode 2 starts as soon as there is pressure on the stock signifying that the Soldier has rooted the rifle to his shoulder. Additional pressure on the stock leads to mode 3. A smooth transition was created between modes 2 and 3 based on the analog stock pressure reading from the rifle.

Visual Results

It helps to see the visual results before going into the details of the animation techniques. Figure 5 shows Mode 1 where the Soldier is free to move his arms without constraints. In the picture, the Soldier is seeing his virtual arms and the rifle for the first time and is examining the detail of the M4 model. Figure 6 demonstrates Mode 2 where the Soldier has shouldered the weapon and is looking around for targets. Figure 7 is Mode 3 and the Soldier has taken an aim pose and applied more stock pressure. The result is that he sees the sight picture and is able to aim. Of course the Soldier sees his first person view and the other participants in the simulation see him in the third person view.

Figure 5. Mode 1: Solider (Left), Unreal 3rd Person View (Center), Unreal 1st Person View (Right)

Figure 6. Mode 2: Solider (Left), Unreal 3rd Person View (Center), Unreal 1st Person View (Right)

Figure 7. Mode 3: Solider (Left), Unreal 3rd Person View (Center), Unreal 1st Person View (Right)

ADVANCED ANIMATION TECHNIQUES

The results seen above are the synthesis of many animation techniques. Those techniques will now be explained in much more detail. The techniques for Mode 1 will be covered followed by the techniques for Modes 2 and 3.

Mode 1 Techniques

Mode 1 Technique : Real-time Motion Capture

The first advanced technique is Real-time Motion Capture. As already discussed, this is the technique of taking rotations from sensors and applying them to the bones of an avatar. It is done by first calibrating the rotations of the sensors to the bones of an animated skeleton. Then every frame a new rotation for the avatar's bone is computed and applied.

Change of Basis for a Rotation

A rotation from a sensor starts off as a quaternion defined in the reference frame of the sensor vendor. It quickly becomes apparent that the quaternion needs to be converted to another frame in order to use it. For instance, the sensors on the Soldier report rotations as quaternions in the frame defined by the X axis is to a character's right, the Y axis is forward, and Z is up. The frame where you want to use the quaternion is Unreal which defines its axes as the X axis is forward, the Y axis is right and the Z axis is up. The answer is that if there are two reference frames A and B, and there is a rotation defined in frame A as M_A, and there is a transformation M_{AtoB} that transforms points from frame A to frame B, then the transform of the rotation M_A to the frame B is:

$$M_B = M_{AtoB} * M_A * (M_{AtoB})^T \qquad (1)$$

Equation 1 is called a Change of Basis for a rotation. (It is written in matrix notation but it still applies to quaternions if they are first converted to matrices). Some sensor vendors provide this operation as a part of their Software Development Kit (SDK), but none of the three that are supported in our implementation made the math of the underlying operation explicit and none had a name for the operation. Our implementation can use any sensor vendor for any sensor across the whole system so it was important to have a generic way to convert rotations without being tied to a single vendor's SDK. Now no matter what frame the vendor reports the quaternion in it can be changed to a common frame across all the sensors.

Real-time Motion Capture Math

Now the sensor rotation is ready to be a part of calibration. The calibration equation and subsequent real-time rotations were derived for the project first for the Dismounted Soldier program by our sensor vendor (Strootz 2012). The notation from the original derivation has been changed to be consistent across this paper and (Johnson 2002). In this formulation, the head sensor is considered a reference sensor that defines a single reference frame on the Soldier. It happens to be a vendor's frame (X = Right, Y = Forward, Z = Up) and it is called the calibration frame. The sensors all report natively in a frame where the yaw is the magnetic north of the earth. The heading of the reference sensor is used to give each sensor that reports in the magnetic frame a way to transform to the calibration frame. Thus it puts all the sensors in the same frame.

> Let t_c be the time at the instant the Soldiers is in the calibration pose.
> For each Unreal bone with a sensor:
> Let MSensorToMagnetic(tc) be the rotation reported from the sensor at the time of calibration
> Let MCalibrationFrameToMagnetic(tc) be the rotation from the yaw of the head sensor to the magnetic frame at the time of calibration
>
> $$MSensorToCalibrationFrame(t_c) = MSensorToMagnetic(t_c) * (MCalibrationToMagnetic(t_c))^T \qquad (2)$$

Then each frame the rotation to apply to each bone on the Unreal skeleton is calculated:

$$
\begin{aligned}
&MBoneToActorFrame \\
&\quad = MBoneToActor(tc) * MSensorToMagnetic * (MCalibrationToMagnetic)^T * \\
&\qquad (MSensorToCalibrationFrame(tc))^T \\
&\quad = MBoneToActor(tc) * MSensorToCalibration * MSensorToCalibrationFrame(tc)^T
\end{aligned} \tag{3}
$$

This formulation has the property that the Soldiers can wear the sensors (except the head sensor) on each bone anywhere they want as long as they are on the correct bone. While this is against official doctrine of the DSTS program, the math is more flexible than the program.

Equation 3 calculated the required rotation of the bone in Unreal. It just needs to be applied to the character. Unreal 3 provides a mechanism for applying rotations to bones called the bone controller. During the real-time process of creating the rotations for the bones, Unreal checks to see there is a bone controller attached to that bone. The game code can provide an external rotation for the Unreal animation system at that time. We chose to calculate and apply the bone rotations in actor space so that the real-time motion would be independent of how the actor is placed in the level or whether the actor is in a moving vehicle.

Mode 1 Technique: GunHand Controller

In Mode 1, the rifle is attached to the Unreal character's right hand. The sensor on the rifle could be considered as a sensor attached to the wrist bone and thus it would be just like any other sensor. However, what makes the rifle different is that it is attached to a grip bone placed on the skeleton that is made so that the fingers can wrap around rifle. The weapon actually rotates the wrist bone but it is attached to a different bone. This breaks the normal pattern of calibrating a sensor to a bone.

There is a single constant rotation between the grip bone on the Unreal character and the wrist bone. The GunHand controller is a custom Unreal bone controller that concatenates the chain of rotations from the grip bone to the wrist bone in the skeletal model. It computes the constant rotation between the grip bone and the wrist bone as its calibration rotation. Then every frame it computes MBoneToActorFrame following the same pattern as the basic real-time motion capture from Equation 3.

Combining the Real-time Motion Capture and the GunHand controller is enough to accomplish Mode 1. Again, in Mode 1, the rifle is attached to the right hand and moves the right wrist. The head and arms are free to move without constraints to each other.

Mode 2 Techniques

Mode 2 requires a few more techniques and is very different from Mode 1. Recall that in Mode 2, the rifle rotates as if attached to the shoulder and the hands are on the rifle. Keeping the hands on the rifle requires a complete change of thinking from Mode 1. In Mode 1, the rifle was attached to the right hand. The position of the rifle came from the position of the right hand. In Mode 2, the tables are turned. The rifle is attached to an invisible bone on the shoulder that is placed there by the animator and rotated by the real-time data stream from the rifle. Then the arms are placed onto the rifle using an Inverse Kinematics (IK) technique that is built into Unreal 3.

Mode 2 Technique: Two Bone Arm IK

Unreal 3 has a built in bone controller called a SkeletalLimbController that affects an entire arm. It implements a closed form solution to IK and allows the user to specify a target effector, and a target elbow location. It assumes that the arm is made of an upper arm, a forearm and a wrist with optional bones to control the roll of the skin. The IK is performed on the two bones of the arm. The wrist rotation can be set to the target rotation separately. Note that the target effector can only be a bone on the character's skeleton, not a bone on the rifle. This means that proxy bones for weapons and other attachments must be made to the Unreal character. One cannot simply attach a prop to the character and then have the IK seek locations on the prop. So the animator had to add more bones to the Unreal character's skeleton to represent the hand holding positions for the M4.

The Unreal 3 SkeletalLimbController properly places the arms and hands on the rifle as the rifle rotates. It is limited however in that it only controls the arm bones. It will not pull the upper torso along so that the shoulders are in place enough to allow the arms to reach their target. This means that rotating the torso to allow the arms to reach the rifle is unaccounted for by the IK.

For more depth on Two Bone Arm IK, readers can see (Tolani 2000).

Mode 2 Technique: Real-time Aim Blend

The torso is rotated by the weapon for aiming by an animation blend in the Unreal animation system. Unreal 3 provides a method to blend a set of nine poses together to allow aiming. It takes two blend parameters Aim.X and Aim.Y that vary from -1.0 to 1.0 and blends together the relevant poses to make the character aim in different directions. Our animator used a rig in Autodesk MotionBuilder to provide the nine blend poses for this controller. The controller works perfectly when controlled by a user with a mouse. It needs to be augmented though to track the yaw and pitch of the rifle because there is no way to directly calculate the required Aim.X and Aim.Y that correspond to the yaw and pitch of the rifle. This becomes its own Inverse Kinematics problem.

Feedback loops were created to augment the existing aim blend. The inputs to the feedback loops were the desired pitch and desired yaw of the rifle bone where the rifle attaches to the shoulder. That data comes from the sensor on the rifle. The outputs were the Aim.X and Aim.Y necessary to feed the Unreal aim blend. The feedback loops implemented simple first order controllers:

$$Aim.X[n] = K_{yaw} * (DesiredRifleYaw[n-1] - ActualRifleYaw[n-1])$$
$$Aim.Y[n] = K_{pitch} * (DesiredRiflePitch[n-1] - ActualRiflePitch[n-1])$$

where K_{yaw} and K_{pitch} are constant gains set by tuning. Values of $K_{yaw} = K_{pitch} = 0.005$ worked very well and were stable at real-time framerates of 30Hz and better.

These controllers are smooth, responsive, and are very simple to implement. But they lag the desired input and have steady state error which means that they do not perfectly track the rotation of the rifle. A key point though is that they are only used to rotate the upper torso. Even though the aim blend will aim the rifle as it blends the aim poses, it is just a blend and it is not accurate to where the physical weapon's sensor is reporting. In a later phase of the animation system, the real-time bone controller for the rifle will overwrite the final orientation of the rifle bone with the data reported from the rifle's sensor. Then the hands will be placed by the Two Bone Arm IK. So the feedback loop only needs to get the upper torso in an approximate position and the other techniques will make pointing the weapon and placing the hands precise.

Mode 3 Technique

Mode 3 is the remaining mode for discussion and it is for creating a sight picture for the Soldier to aim. All the animation techniques are in place for Mode 3 with the exception of placing the final game camera down the sights.

Mode 3 Technique: Moving the Game Camera to Produce the Site Picture

The desired effect of looking down the iron sights of the M4 is accomplished by moving the game camera to a location on top of the weapon. The game camera leaves its normal spot between the eyes of the Unreal character and does a linear blend to a location on the rifle. The camera orientation blends from the view from the head smoothly to an orientation that is the combination of the rifle orientation and the head orientation. The actual calculation for the game camera is expressed as pseudo code using Euler angles.

$$CameraRotation.Yaw = rifleRotation.Yaw$$
$$CameraRotation.Pitch = rifleRotation.Pitch$$
$$CameraRotation.Roll = (headRotation * (rifleRotation)^T).Roll$$

The effect is the natural sight picture that a Soldier would expect. The rifle rolls left and right as expected but the overall view is never rolled by the weapon. If you roll the view by the weapon and not the head rotation it will make

the Soldier's sick. The Soldier is always allowed to turn his head to view in any mode. If the Soldier turns his head away from the direction of the weapon then the sight picture will blend away allowing him to see more of the scene. When he realigns his view with the rifle, the sight picture blends back in.

SYNCHRONIZING THE POSE ACROSS THE NETWORK

The final animation pose of all these techniques needs to be replicated across the network so all the simulation participants see the same pose. This problem has been studied at length including (Capin 1998) which recommends using a Kalman filter (see Brown 1997). But the complexity of the Kalman filter did not match the complexity of this problem. (Capin 1998) is sending 74 bones over the network. The DSTS hardware requires a synchronization of at most six bones. We chose first order DIS dead reckoning (IEEE1278.1) with quaternions representing positions and angular velocities taking the place of linear velocities. This was combined with Unreal's built in networking abstraction called "replication". Network packets with quaternion and angular velocities were sent unreliably from the client to the server as built in remote procedure calls in Unreal. The Unreal 3 server adjudicates which actors receive those update packets based on proximity and other factors. Then each client renders the character based on the base pose animation set by the mode and the real-time data from the network packet.

It would not be right to use DIS dead reckoning and not include the classic diagram that shows it working. Figure 8 shows the data arriving at a receiving client. It differs from the linear version of DIS dead reckoning because it is plotting quaternions on a unit circle. The blue arrows are angular velocities, not linear velocities so they are perpendicular to the path of motion across the unit circle.

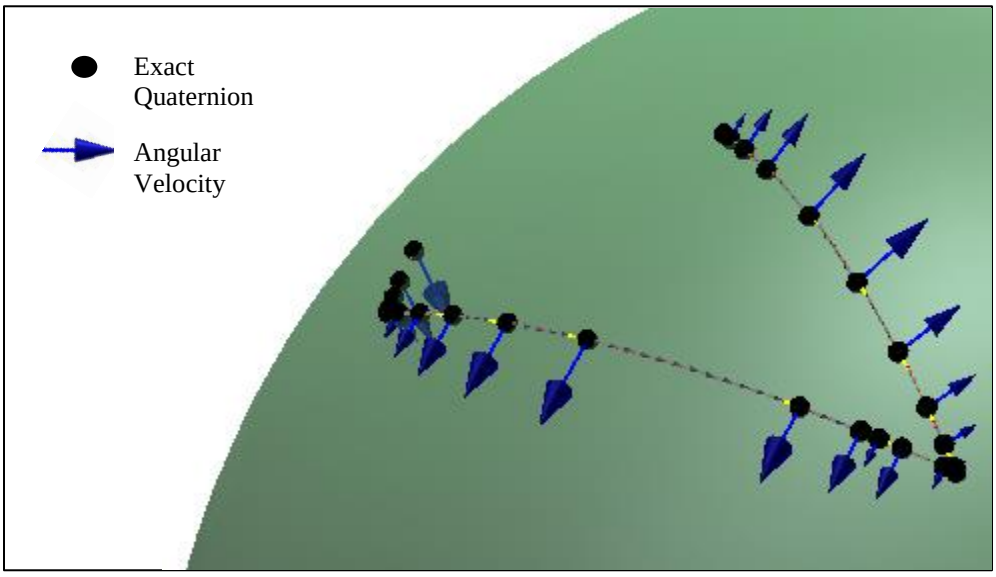

Figure 8. Dead Reckoning of a quaternion representing the rotation of the rifle over time plotted on a unit circle.

The visual results of the dead reckoning were good when using a time threshold of 50ms and an angle threshold of 3 degrees. Those values of course can be tuned to trade off network bandwidth for the quality of the animation. In a test setup with nine motion captured participants, the total network bandwidth utilization as reported by Windows 7 was less than 5% on a Wireless N network.

Locomotion

The DSTS hardware has a joystick on the rifle handle that allows the Soldier to move in the virtual world. The sensors on his thighs are for posture detection and to determine his yaw. They are not used for locomotion. So the walking motions of the lower torso must come from animations. Unreal 3 has blends made for walking so that base

walk loops can be blended together to get walking in multiple directions. The built in blends will take forward, left, right, and reverse walk loops and allow your code to set the direction of the walk from the joystick. We used the same walk loops for each mode and masked off the animation of the upper torso so that the real-time motion and the aim blend would drive the upper torso. Unfortunately this technique leads to what we call the "River Dance" effect where the upper and lower torso's motion do not seem to match. It is mostly seen in Mode 1 where the upper torso is unconstrained. Making the walk loops more natural will have to be left to future work.

CONCLUSION

Despite the inherent inaccuracies in capturing the pose of the Soldier, we were able to solve the problem of letting the Soldier move freely and then attain a highly constrained aim pose. It took a fusion of many advanced animation techniques as well as synchronization over the network to make it useful. With the exception of the walk loops, the goal of suspension of disbelief was met for all the modes of operation.

ACKNOWLEDGEMENTS

We would like to acknowledge the work of our artists Steve Komrowski, Mike Bakerman, and Andrew Catron for supporting this effort.
We would also like to thank the Edge program for supplying Unreal 3 and environmental art assets to show off our work.

REFERENCES

Brown, R. G. and P. Y. C. Hwang. 1997. Introduction to Random Signals and Applied Kalman Filtering, *Third Edition, John Wiley & Sons, Inc*.

Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Standard 1278-1995, Standard for Information Technology, Protocols for Distributed Interactive Simulation, 1995

Johnson S., Complex Matrix Transformations (2002) *Gamasutra*, *May 2002* from http://www.gamasutra.com/view/feature/131399/complex_matrix_transformations.php

RNI – The Research Network, Multimodal Interfaces for Simulation and Training from http://www.resrchnet.com/products/mmist

Roetenberg D., H Luinge, and Slycke P. (2009) Xsens; MVN: Full 6DOF Human Motion Tracking Using Miniature Inertial Sensors. *XsensMotion Technologies BV, Tech. Rep*.

Strootz A., personal communication, April 2012

Tolani D., A Goswami, Badler N., Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs, *Graphical Models, Volume 62, Issue 5*

Zhu, R. and Zhu Z. 2004. A Real Time Articulated Human Motion Tracking Using Tri-Axis Inertial/Magnetic Sensors Package, *IEEE Transactions on Neural Systems and Rehabilitation Engineering, Vol 12 No 2*

YEI Technologies Experimental Projects from https://www.yeitechnology.com/experimental-projects