Dead Reckoning

Jump to bottom

mcgredonps edited this page on 13 Jul 2018 · 23 revisions

Dead reckoning is a method for the estimation of the position and orientation of an entity, based on a previously known position and orientation and estimates of the passage of simulation time and motion. We know the entity position and orientation was previously reported by the sender, but sometimes can't send an dedicated PDU sending data updates quickly enough. The purpose of dead reckonings use in simulation is to limit the issue rate of ESPDUs.

See IEEE 1278.1 Annex E for specifics on the dead reckoning model and governing formulas.

Estimating the positions and orientations of other entities, it is not necessary to receive a report about every change in position and orientation more often than it can be sent. Perhaps we sent an ESPDU for a tank's position once every two seconds, and the tank is driving on a straight line. We'd like to update the position every 1/30th of a second that occurs in the entity's course, even though we receive an update ESPDU every 2 seconds.

This issuance of dead reckoning information that has crossed threshold values is managed both by issuing and receiving simulations. The receiving simulation might not care about getting a position update every two seconds, or it may care deeply about the matter, and as a result run an estimated position algorithms. On the other hand the PDU position and speed generation software knows something about what the entity plans on doing. If the tank is driving in an unaccelerated straight line it's the simulator generating that has the best information. This allows us to not sent a PDU very often. It also allows the generating simulation, by running it's own dead reckoning thread, to discover when receivers are likely to have bad simulation positions, and resend.

The sender knows it's own position, speed acceleration, and orientation, and includes this in the information sent with the entity state PDU. If it wants the receiving simulation to make use of dead reckoning (DR) it can fill out the expected fields. It can locally run its own speed estimate, and immediately send an ESPDU update when it expects to.

The IEEE-1278 Annex E says

> A DRM notation shall consist of three elements. The first element shall indicate whether the model specifies rotation as either fixed (F) or rotating (R). The second element shall specify dead reckoning rates to be held constant as either rate of position (P) or rate of velocity (V). The third element shall specify the coordinate system to be used with the dead reckoning algorithm as either world coordinates (W) or body axis coordinates (B). The second element identifies the algorithm as being first (P) or second (V) order with respect to position. If the first element specifies rotation (R), the algorithm is first order with respect to orientation.

> This notation may be summarized as follows:
>
> DRM (F or R, P or V, W or B)
>
> For example, a DRM that considers orientation fixed and a constant rate of position in world coordinates would be expressed as:
>
> DRM (FPW)

Some fields, such as the entity location and entity orientation, are to be kept updated in the ESPDU object. In addition in the dead reckoning parameters record are 8 bits that specify what is held by the ESPDU. The fields below use the Java API--what's more, the API using open-dis. Other languages or implementations can use different method calls if the designer has made different decisions. It must be carried by the implementor. The important values for dead reckoning are the what algorithm is specified by the sender, entityLocation, and entityLinearVelocity, and 11 bytes of room for other data.

The algorithm used for dead reckoning values as follows:

Field Value	Dead-Reckoning Algorithm
0	Other
1	Static (Entity does not move.) DRM(F, P, W)
2	DRM(R, P, W)
3	DRM(R, V, W)
4	DRM(R, V, W)
5	DRM(F, V, W)
6	DRM(F, P, B)
7	DRM(R, P, B)
8	DRM(R, V, B)
9	DRM(F, V, B)

The deadReckoningParameters are additional space taken up by every ESPDU. This includes angularVelocity and linearAcceleration, triple values each, though only 4 bytes per number (a float).

Finally a 8 bit field can be filled with what parameters you want to use.

The DeadReckoningParameter values are kept in a class:

```
public class DeadReckoningParameters extends Object implements Serializable
{
   /** Algorithm to use in computing dead reckoning. See EBV doc. */
   protected short  deadReckoningAlgorithm;

   /** Dead reckoning parameters. Contents depends on algorithm. */
   protected short[]  parameters = new short[15];

   /** Linear acceleration of the entity */
   protected Vector3Float  entityLinearAcceleration = new Vector3Float();

   /** Angular velocity of the entity */
   protected Vector3Float  entityAngularVelocity = new Vector3Float();
```

Example

The animation examples seem to use linear velocity at most, though they can use others, or non at all. The table of algorithms described above is as below:

Field	Model	Formula	Comments
1	STATIC	N/A	Static entities
2	DRM (FPW)	$P = P_0 + V_0 \Delta t$	Constant velocity (or low acceleration)
3	DRM (RPW)	1) $P = P_0 + V_0 \Delta t$ 2) $\left[R\right]_{w \to b} = \left[DR\right]\left[R_0\right]_{w \to b}$	Similar to DRM 2 but where orientation is required (e.g., visual simulation)
4	DRM (RVW)	1) $P = P_0 + V_0 \Delta t + \frac{1}{2} A_0 \Delta t^2$ 2) $\left[R\right]_{w \to b} = \left[DR\right]\left[R_0\right]_{w \to b}$	Similar to DRM 5 but where orientation is required (e.g., visual simulation)
5	DRM (FVW)	1) $P = P_0 + V_0 \Delta t + \frac{1}{2} A_0 \Delta t^2$	High speed (e.g., missile) or maneuvering at any speed
6	DRM (FPB)	1) $P = P_0 + \left[R_0\right]^{-1}_{w \to b}(\left[R1\right] V_b)$	Similar to DRM 2 but when body-centered calculation is preferred

Field	Model	Formula	Comments
7	DRM (RPB)	$1) \quad P = P_0 + \left[R_0\right]_{w \to b}^{-1}(\left[R1\right]V_b)$ $2) \left[R\right]_{w \to b} = \left[DR\right]\left[R_0\right]_{w \to b}$	Similar to DRM 3 but when body-centered calculation is preferred
8	DRM (RVB)	$1) \ P = P_0 + \left[R_0\right]_{w \to b}^{-1}(\left[R1\right]V_b + \left[R2\right]A_b)$ $2) \left[R\right]_{w \to b} = \left[DR\right]\left[R_0\right]_{w \to b}$	Similar to DRM 4 but when body-centered calculation is preferred
9	DRM (FVB)	$1) \quad P = P_0 + \left[R_0\right]_{w \to b}^{-1}(\left[R1\right]V_b + \left[R2\right]A_b)$	Similar to DRM 5 but when body-centered calculation is preferred

Algorithms 4 to 8 are described in the IEEE-1278.1 document.

Dead reckoning is not always used. If you don't mind an update every five seconds you can leave it empty.

Top

▸ Pages 97

Tutorial Contents

Clone this wiki locally