Arshia Firouzi

6DOF Virtualization – Orientation & Position

The software associated with this document ("Not-Visualized_3D-6DOF_1-20-2019.py", and "Visualized_3D-6DOF_1-20-2019.py") is meant to simulate the motion of an object in three dimensions with six degrees of freedom. This is in preparation of implementation of such a system as it will include the software necessary to translate processed sensor data into visualization and real-time tracking. This software does not perform anything outside of taking simple instructions and executing them. No control system, hardware level communications, wireless communications, etc. exist here.

The reason there is a version of the code which visualizes the data and one that does not is twofold. Firstly, to demonstrate the simulation is accurate, and secondly, to time how long graphing the state of the system will take.

The following pages dive into the software design, decision making, and relevant information yielded from various tests.

Arshia Firouzi

The 3D-6DOF Simulation Software

<u>The Rectangular Prism Class – “RectPrism” (from RectPrismVSD.py)</u>

The rectangular prism class assumes that the object being tracked (in the future) will have the shape of a rectangular prism. As a result, it contains all the necessary methods to track and transform the system with six degrees of freedom. It also provides information necessary for visualizing the object (it’s vertices, sides, and displacement). The width, length, and height of the object are passed to it (as visible in “__init__”) and remain constant throughout the instance. These variables are input by the user upon running the software.

<u>The Rectangular Prism Method – “initialize”</u>

The “initialize” method is very straightforward. It takes the class’s width, length, and height, and creates a rectangular prism centered at the origin. This returns the initial vertices, sides, and displacement-vector of the object. One should note, the sides are not necessary unless one intends to visualize the object, otherwise – simply the vertices and displacement-vector will do. This comes up in implementation of this code in future conjunction with hardware as the software can run faster without these calculations.

<u>The Rectangular Prism Method – “move”</u>

The “move” method simply calculates the new vertices (or position) of the object based on input displacement values (in the directions of the X, Y, and Z axes). This method is the only method that returns a new displacement vector (the object’s center has now been displaced). Once again, the vertices, sides, and new displacement-vector are recalculated and returned – but the sides do not need to be so if the code is not visualizing the object. This method will deal with accelerometer data in the future.

<u>The Rectangular Prism Method – “rotate”</u>

The “rotate” method takes in rotational data (about the X, Y, and Z axes) in radians and adjusts the vertices of the prism as necessary. First, it returns the entire object to the origin (via the displacement-vector) and performs trigonometric calculations to rotate the vertices of the prism. Afterwards, it returns the vertices with their new orientation to the original position of the object’s center with the displacement-vector. The displacement-vector is not adjusted and so only the calculated vertices and sides are returned. Again, the sides are not necessary unless visualization is to occur.

<u>The Rectangular Prism Method – “refresh”</u>

The “refresh” method constantly reads the “data.txt” file and returns vertices, sides, and displacement-vector based on the file’s commands. The text file explains in its first line how the data is written and interpreted. Rotation is given in degrees as the future hardware will be using degrees.

<u>“main”</u>

“main” takes in user input, initializes the “RectPrism” class, and continuously runs “refresh”. In the visualization version of the code, it also graphically displays the object. For visualization, a 40ms pause is placed after visualization to avoid runtime errors (see line 230). Tests between this and the non-visualized case show that visualization takes up the great majority of the programs cycle time (“refresh” method time). In fact, each cycle of the while loop in “main takes 5ms without visualization and 110ms with it (excluding the 40ms pause). This is important as it means the system which visualizes the code may need to be separate from the system measuring and otherwise interacting with sensors in order to track the object optimally. Any filtration will also need to be done outside the visualization program if possible (Kalman Filtering expected).