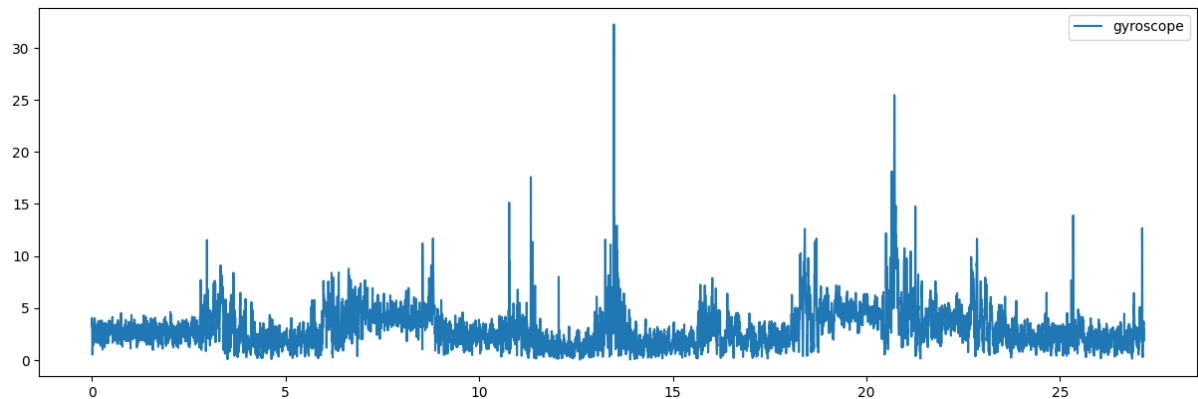


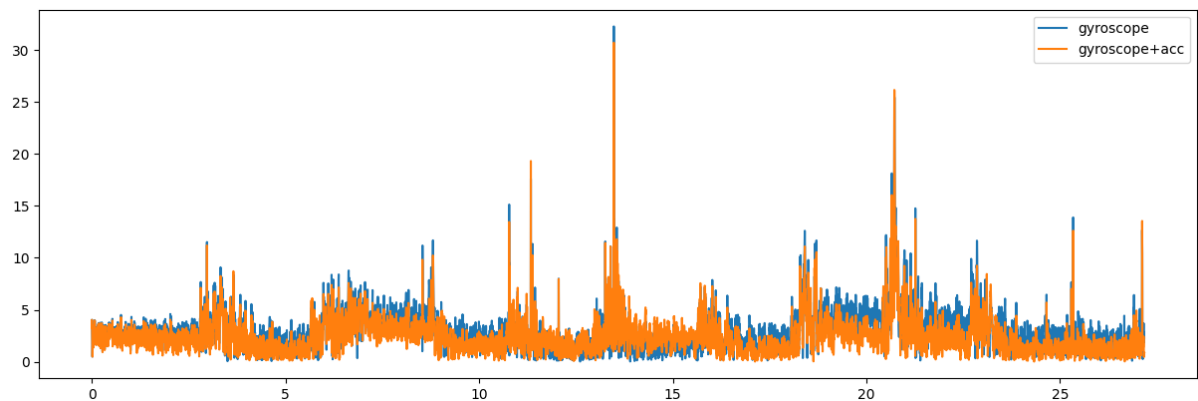
**Problem 3**

Experimentation with different values of alpha for tilt correction (tilt error in degrees shown on y axis and the time in seconds in the x axis):

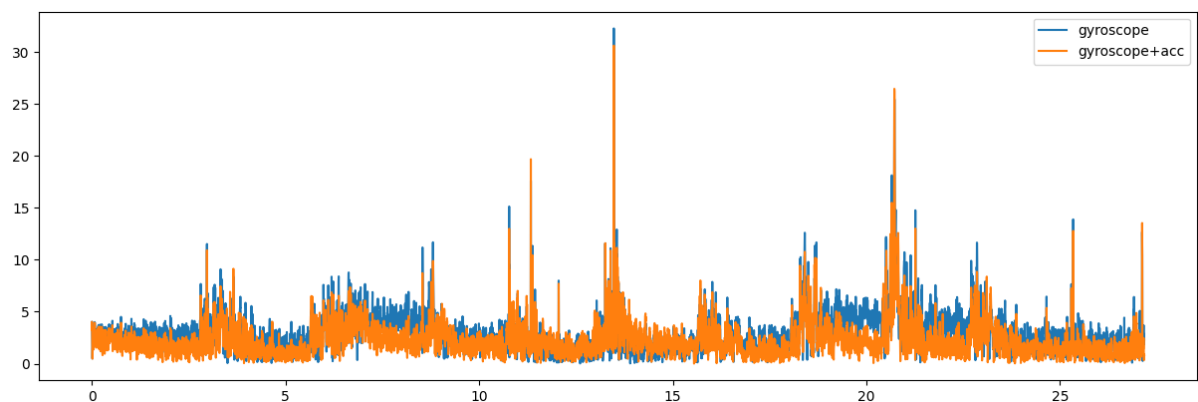


The first figure shows the tilt error without any tilt correction (i.e., just the gyroscope). Although we do not see a severe case of drift accumulation (possibly because the movement was only recorded for ~27 seconds), we see that there are areas in which the tilt error begins to accumulate for a short period of time – particularly during the roll and pitch motion.

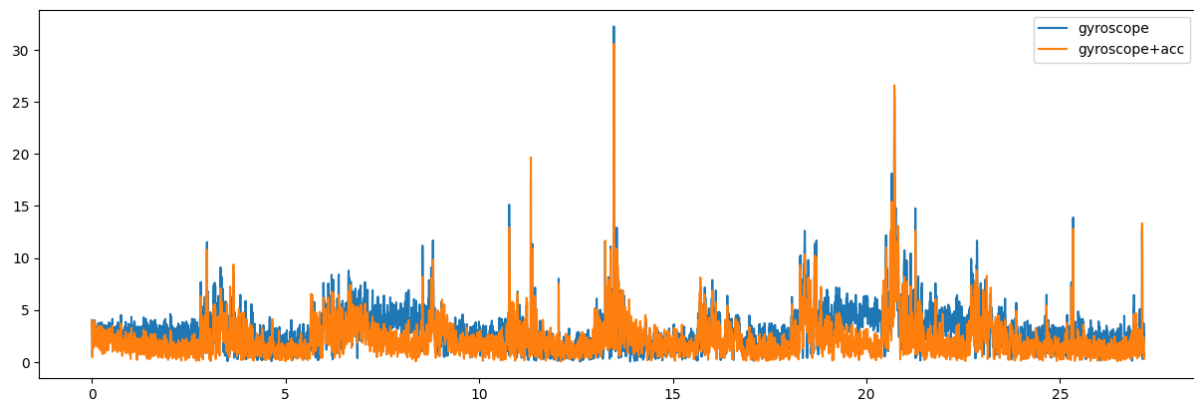
Alpha = 0.01



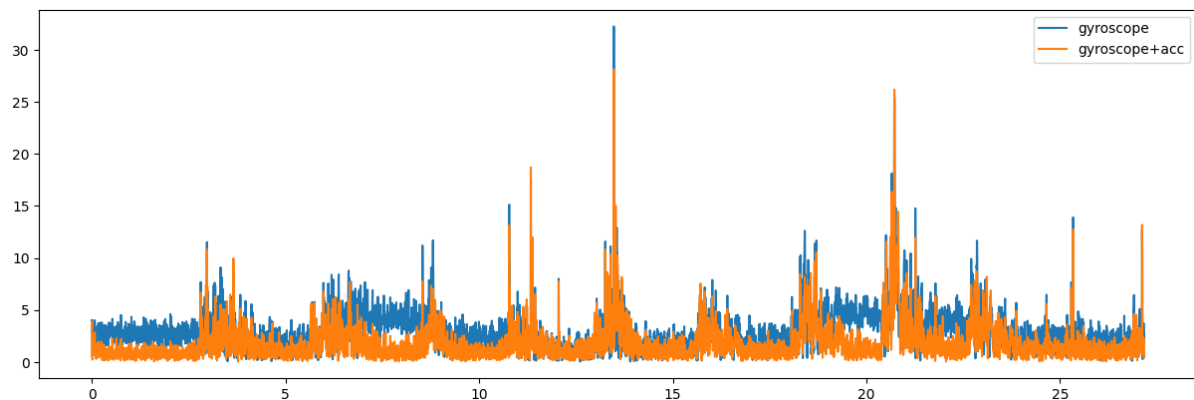
Alpha = 0.04



Alpha = 0.50



Alpha = 0.9



It is clear that as the value for alpha is increased, the severity of correction is increased (i.e. the tilt error is brought down to 0 much faster). This was evident in the output video where the overpowering effect of tilt correction caused the camera to snap back to 0 degrees. Setting alpha to be a high value resulted in the movement being more jittery. Thus, a value of 0.04 for alpha was chosen as it seemed to balance the trade-off between accuracy in tilt correction and stability in the movement.

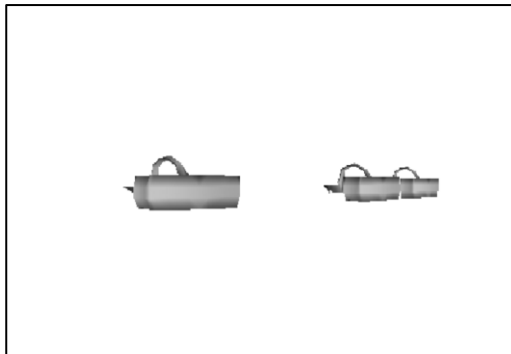
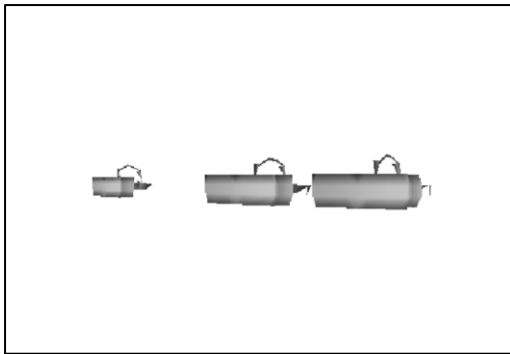
Although not implemented in this project, the accelerometer could be further processed by removing sample outliers that significantly differ from the previous measurements. This is because sudden changes in the combined acceleration are more likely to happen due to movement rather than drift as suggested by LaVelle [1]. This can improve the performance of the tilt correction.

### **Problem 6**

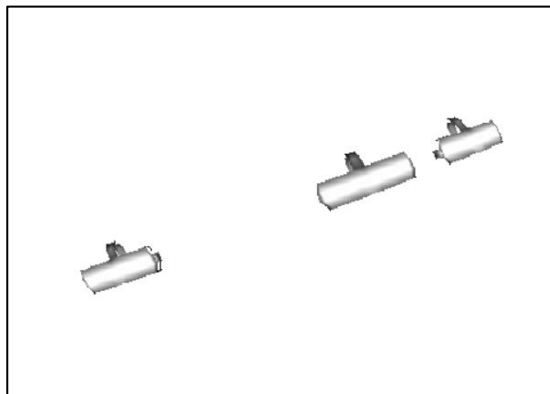
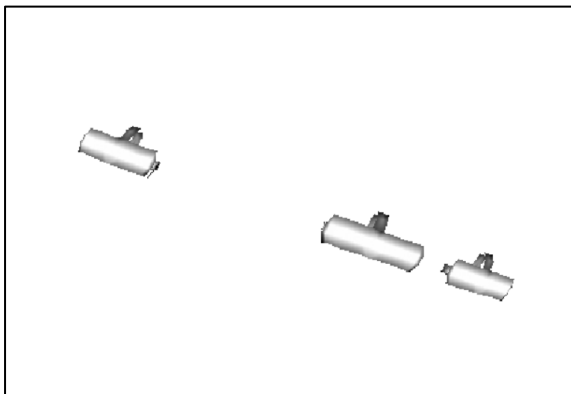
- 1. Produce static renders for the report, for different orientations of the camera, demonstrating that your tracking and transformation matrices work.**

Camera orientations after barrel distortion (i.e. pincushion distortion pre-correction):

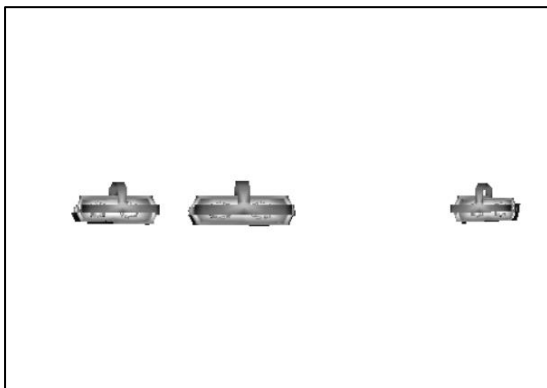
Yaw  $\pi/4$  radians:



Roll  $\pi/8$  radians:



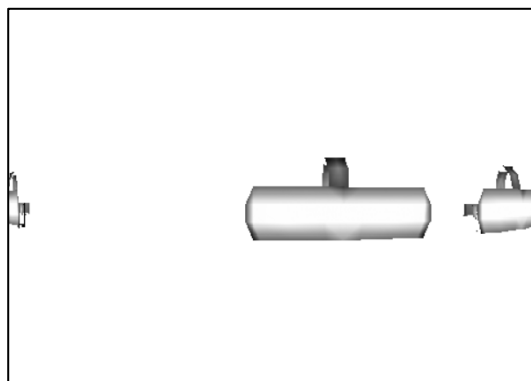
Pitch  $\pi$  radians:



Camera placed at  $z = 20$ :



Camera placed at  $z=0$  and  $y=2$ :



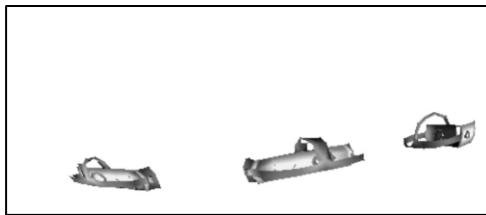
**2. What are the advantages and disadvantages of using simple dead reckoning vs including the Accelerometer? Support your claims with data from your simulation.**

Advantages of using just simple dead reckoning:

- It provides reasonably accurate orientation estimations over a short period of time.
- Simple to implement without any complex calculations. As seen in the implementation, the only operations required are linear operations, conversion from axis-angle representation and calculating the quaternion product.

Disadvantages of using just simple dead reckoning:

- When used to track for longer periods of time, errors such as drift errors accumulate over time, thus leading to inaccuracies in position. This was observed in the video where tilt was most noticeable during the final pitch motion as seen below:

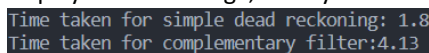


- Gyroscope can have a bias due to temperature, packaging stress and noise [from lecture].

Advantages of including accelerometer:

- It can compensate for the drift error resulting from the dead reckoning filter.
- Better accuracy for orientation estimates.
- The subtlety of the drift error correction can be manually controlled with the alpha parameter. As stated in the 'problem 3' section of this report, the alpha parameter can control the trade-off between orientation accuracy and stability/intensity in camera correction motion.
- Accelerometer output can produce a good estimate for the direction of gravity if averaged over a long period of time [1].

Disadvantages of including accelerometer:

- Additional hardware cost of including an accelerometer component (although most modern IMU chips can provide both gyroscope and accelerometer measurements).
- Yaw drift correction is still not accounted for – this requires use of the magnetometer readings.
- Increased time complexity as the accelerometer measurements also have to be processed.
  - o The time taken (s) to generate orientation quaternions using the simple dead reckoning filter compared to using the complementary filter that includes the accelerometer integration is displayed in the image, clearly demonstrating the increased processing time required:  

- Both gyroscope and accelerometer need to be calibrated.
- Addition of accelerometer in IMU results in more moving parts, thus more components that could potentially fail.
- Accelerometer data cannot be trusted in the short term in the presence of movement [1].

**3. Comment on the performance of the methods in point 2 above, i.e., comment on the expected number of calculations for each method.**

Although most operations in both functions take linear time, and the computational complexity of both are of the same (Big O) order, the tilt correction provided through the accelerometer requires an additional 2 quaternion multiplications to bring the acceleration component into the global frame (and one additional multiplication to calculate the complementary filter by combining the tilt correction with the dead reckoning output). This, along with the need to compute the angle between two vectors for each IMU reading, makes the

computation of tilt correction in the complementary filter more computationally expensive than the simple dead reckoning process.

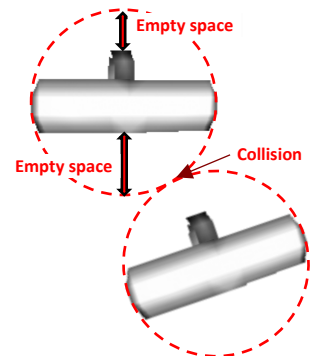
**4. Could positional tracking be implemented from the data provided? If so, how? Comment on the expected accuracy of positional tracking based on the IMU data. What are some potential limitations of this approach?**

As noted by LaVelle [1], although one might think it's possible to remove the acceleration due to gravity and doubly integrate the remaining linear acceleration component from the accelerometer to obtain position, a major issue arises with the drift error. Due to the double integration, a calibration error leads to a quadratically increasing drift error, whereas for the gyroscope reading the drift error would only increase linearly as integration is only performed once. The positional drift error cannot be corrected using an IMU alone, as IMUs cannot detect motion that occurs at constant velocity, which includes no motion [1]. Nevertheless, if this approach was selected, this can be combined with the use of kinematic head model to approximate eye locations as the user rotates their head, but there can be further issues with mismatch between real and virtual world head models if the torso moves, or if the user stands up but is sitting down in the virtual world. A better, more accurate approach would be to combine the IMU readings with visibility information from a camera or from a laser-based tracking approach as implemented in the HTC Vive headset [1] using a complementary filter approach. When using visibility information, the distance from the base station to the features on the headset can be determined by solving the PnP problem (e.g., by having multiple features to detect), and using multiple base stations to infer depth (like a stereo vision scenario). It should be noted however that the position estimates from visual sources such as a camera report reading at a much lower frequency than IMUs.

**5. Comment on collision detection using spheres. How could you improve collision detection and what would the effects of those improvements be on performance? Support your arguments with data for the particular 3D model of the headset that you were provided with.**

Collision detection using spheres was achieved by representing each of the objects as spheres and measuring the distance between their centers, i.e. the objects were considered to be colliding if the distance between their centers was less than or equal to the sum of their radii (which was equal for all the objects in the scene as all objects were the same size).

A big limitation of this technique is its lack of accuracy for complex, non-spherical shapes such as the VR headset. This is largely due to the challenge of selecting a suitable radius for the spheres, which must prevent the objects from colliding before their pixels touch and avoid complex parts of the object overlapping (e.g. the straps of the headset). Setting a large enough radius which covered the width of the headset meant that a lot of empty space, e.g. at the top and bottom of the headset as well as on the left and right of the side-view, was tracked for collision, leading to inaccurate collision being detected even before the objects made contact. Additionally, the computational challenge is also an issue, because as the number of objects in the scene increase, the number of calculations required increase as the number of pairwise distance checks increase.

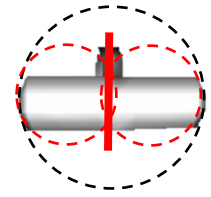


To overcome the concern with precision, more sophisticated hierarchical methods such as bounding volume hierarchies (BVH) can be utilised. In simple terms, hierarchical methods involve a tree being constructed for each object. This bounding region is split into smaller subregions such that their union covers the parent region; these become children of the parent node in the tree. This process is recursively repeated until simple sets are obtained. Using this technique, the collision between two objects can be detected by comparing their bounding regions. If the root vertex of each tree intersects, the check can be repeated with their children until there is no intersection, or the leaves of the tree have been reached. This idea can also extend to splitting the scene into regions so that comparisons are avoided with objects that we know for sure will not collide to

reduce the computational cost (although this is not applicable to the demo scene in this project). A further advantage of this technique is that the granularity in the size of the regions at the leaf of each tree can be controlled based on computational needs. For example, the division of regions into smaller regions can be stopped before individual triangles are reached if there is a limitation in the computational resources available.

Nevertheless, although this improvement can vastly improve the precision of object collision detection, it comes at an extra cost of complexity. For one, implementing a hierarchical method in software is a lot more complex than simple spherical distance-based measurements. Secondly, the recursive nature of the algorithm makes the process computationally expensive as intersection of sub-regions also must be calculated.

To conclude, a greater level of precision can be achieved with hierarchical techniques, although they come at the cost of increased computational demand. This trade-off should be balanced based on business objectives and the context of the application, e.g., integration of collision detection into graphics engines that are used to render surgical demonstrations may favour a higher level of precision.



*A basic example showing how a split can be made along the solid line to create two circular regions which are children of the parent black circle vertex in a hierarchical approach.*

**6. Based on how you implemented distortion correction in this CPU-based software renderer, how could this technique be accelerated when implemented in a GPU-accelerated engine? Discuss all possible options.**

In a GPU-accelerated engine, the distortion pre-correction technique (barrel distortion was used to correct the pincushion distortion for this project) can be accelerated by implementing it as a shader program, i.e., perform distortion shading.

One option is to use a fragment shader to process the texture of the left and right eye each, moving each pixel in relation to the centre of the eye to create a barrel distortion correction [2]. The technique is similar the method implemented for this project on the CPU in the sense that each pixel's destination was calculated. Although this may result in a high-quality pre-distortion correction, a texture lookup for each pixel is very computationally demanding.

An alternative option is to use a mesh-based technique using a vertex shader. In this technique, instead of performing the calculation for every pixel, the computation only needs to be performed for every vertex in a sparse mesh such as the one illustrated in the image below (GPU will do the rest of the interpolation) [2]. This significantly reduces the computational cost, and the complexity vs quality trade-off can be adjusted with the number of vertices in the mesh. For example, a dense mesh with several vertices will produce a better result than a sparser mesh, but at the cost of a greater computational runtime because more vertices will need to be processed.

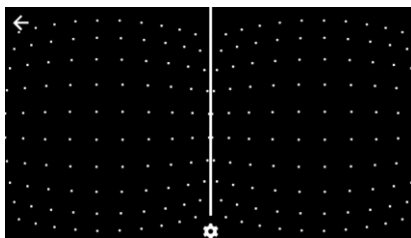


Image Source: <https://smus.com/vr-lens-distortion/>

## References

[1] LaValle, S.M., 2017. Virtual Reality/University of Illinois.[Sl:] Cambridge University Press. 418 p. URL: <http://vr.cs.uiuc.edu/vrbook.pdf>.

[2] Smus, B. (no date) Three approaches to VR lens distortion. Available at: <https://smus.com/vr-lens-distortion/> (Accessed: 03 April 2023).