This document works as follows: There are sections relating to potential work that needs to be done for specific concepts. For most suggested improvements, some pros and cons of enacting that change will be listed

*(This list is in no particular order)*

**IMU Integration:**

In the existing system, only the Camera and Sound subsystems have been integrated together. The IMU could be added in order to further augment the user experience. This particularly comes in handy when the user does not see an aruco marker, but has not yet moved from the last position where they saw an aruco marker.

**Note:** The IMU was built out and set up for use with a PI, however the final project code does not include it (primarily due to some risks stated below, with little time to mitigate them). Simply utilizing the existing code base inside of the final product would be sufficient to add it

**Pros:**

* Potential for faster response time (if integrated in a multi-threaded system - high complexity though)
* Less of a need for high Aruco coverage around the room
  + (Arucos will still be needed in sets of ~4 in order to mitigate errors)

**Cons:**

* System may become less intuitive to use (it becomes unclear whether or not the system is updating your position, or strictly your rotation)
  + Suggestion from team: Add different sounds for when operating on strictly IMU data versus when operating on Aruco data
* The IMU can be deceptive: it will only provide rotational feedback, however if the user moves at all, and does not see an Aruco marker, the feedback given will no longer be accurate, as the coordinate system will “move with the user”.

**Camera Latency Refinement:**

Currently the camera system runs sluggishly - it operates at roughly 30 frames per second, however those 30 frames seem to lag behind real time by up to 1 to 2 full seconds. There may be configuration options to speed up the communication, but it will likely come at some sort of price.

There is also the possibility that this is a limitation of the system (perhaps because the pi-cam has to communicate the picture information across a thin bus), and thus the only way to improve this would be to use a more powerful interface / camera / processing core, as needed.

**Pros:**

* The Camera is the bottleneck of the system, if it is sped up then the whole system will respond faster and feel more natural

**Cons:**

* Could end up costing something valuable to the project - such as:
  + Field of View
  + Aruco Marker identification capabilities (?)
* *(May require recalibration after changing camera settings)*

**Sound Library Polish:**

The current sound-library implementation utilizes a head-related transfer function to communicate the position of the sound to the user. However, this results in the sound feeling as if it is coming from “inside the head” - it doesn’t really simulate reality, but still has a strong sense of orientation. We (and some other sources, such as UC Davis) believe that adding reverb to the sound effects will create a more realistic experience for the user, making the sounds more externalizable.

**Note:** There was an attempt to implement this inside of sound\_library.h and sound\_library.c based on some of the example code available through OpenAL, however upon trying to compile it the compiler returns errors that we could not afford to debug at the time.

**Pros:**

* Potentially more realistic sound

**Cons:**

* Could actually reduce overall performance of product by making orientational information more vague
* Might not make the sound more natural

Furthermore, it has a **very** primitive implementation that converts an uncompressed .wav file into information for use within OpenAL. This should be refactored to at least work with any type of .wav file, and likely should work for any possible sound format (there are likely libraries out there that will handle that for you if you go looking. It may be necessary to “upgrade” to c++ in order to find code that will accomplish these tasks).

**Pros:**

* Improved user experience when interfacing with more back-end stuff (download any sound and use it!)
* More flexibility with available sounds, and lowered hassle

**Cons:**

* None really - just requires time investment and is strictly a low-priority quality-of-life improvement for both the developers and the customer.

OpenAL also does not currently adapt to any changing in device once the main program has been started. It would be ideal if it will continuously check to see if the primary audio device is still connected, or if perhaps there is a new connection available.

**Pros:**

* Improved user experience when dealing with different bluetooth headphones

**Aruco OpenCV Polish (?):**

This project ended up having a very heavy reliance on filtering in order to get the final product to work. The Aruco library from OpenCV contribution package is quite powerful, and very precise when dealing with translation, however its’ pose estimation can be quite erroneous at times. It may be possible for an upcoming team to dig into the library (if they have any sort of computer-vision based expertise) and investigate the underlying issue that we’re having:

***Pose estimation appears to have several ‘solutions’\* when determining the X, Y, and Z rotation, whereas in the context of our device only one of these solutions is valid.***

*\* We refer to these as solutions, because they often cause large jumps in pose angle, but the jumps are consistent in all three dimensions of rotation (X, Y, Z), and have very small degrees of error when it jumps, thus suggesting that there are two possible data sets that are correct.*

The hope is that the device could figure out with full certainty what the exact rotation of the marker is -- which theoretically should be possible given that the exact dimensions and shape of the marker is known before run-time, however these are simply guesses.

If nothing else, the next team should keep a careful watch on the Aruco Library’s repository and ensure that they pull in any updates which may be beneficial to the Virtual Cane project, namely any sort of stability on pose angle estimation.

**Pros:**

* May be able to eliminate the need for filtering on pose angle
  + This will also technically increase the response of our system, as it has less need for large filter buffers.

**Cons:**

* The underlying issue may be impossible to fix
* Investigating and potentially fixing the problem will likely be **very** time consuming

**Filtering Modification / Tweaking / Refinement:**

In order to make the first stage of the Virtual Cane project successful, our team felt the need to implement an outlier detector filter. It will likely be necessary to tweak this filter in order to optimize results, so below is a list of parameters that may need to be tweaked:

* Change buffer size of filtered data points:
  + By decreasing the buffer size, that will reduce the history\* which the buffer uses to detect outliers, thereby likely reducing the overall stability of the filter, while increasing the response time to new input data
  + By increasing the buffer size, the buffer will consider more history\*, slowing the system’s response to new data, but increasing stability
    - \* history is only applicable for time-based filters, not for when the markers vote on an overall position of the chair.
* Changing the error bounds of the filter:
  + The larger the error bounds, the more generous the filter will be with including data points in the consensus. It will converge faster if the error bounds are larger, but at the risk of potentially including “outlier” information.

Furthermore, there are some optimizations that may be worth considering:

* Instead of emptying the buffer completely after calling “empty” multiple times in a row, consider removing the oldest data from the buffer
* Create an clear function that will remove all data from the buffer
* Optimize the efficiency of the detect algorithm by reducing the number of calculations of the absolute error
  + Might potentially reduce processing time. It is not clear that the algorithm is terribly expensive even though it requires a minimum of n^2 operations
* Create a seperate consensus algorithm that looks strictly for agreement between multiple data points, rather than requiring more than half the data to agree. (E.G., if out of four points two points agree, and two are completely different outliers, then just use the two points that agree)
  + This would be good for when markers vote on the position of the chair, and may reduce the # of markers required to reach a reasonable consensus

We don’t know how much these changes will actually benefit the project, but they are things that we considered doing but did not have the time to risk doing it.

**Threading:**

The overall performance of the system *could* theoretically be improved by threading. However, as the primary bottleneck of the system is the camera, this likely will not be useful unless the upcoming team uses threads to process the vision system and the IMU, that way the IMU can “instantaneously” update rotational position, before the vision system has time to respond to the change in head rotation. Threads could also be used to speed up the processing of the Outlier Detector, but will likely not produce any meaningful speedup in the project.

**Pros:**

* Capable of instantly responding to IMU data
* Potential for increasing camera response speed? (Unlikely)

**Cons:**

* System currently runs at about 30% when running the main code. Threading will likely increase system utilization - which means the following things may occur:
  + Increased temperature of the on-board arm chip
  + Increased battery usage
    - These may or may not have any impact on the overall project
* If threading is used to combine the IMU with the camera, care must be taken so that the camera properly zeros the IMU whenever it sees an Aruco marker
  + There may be some complexities involved here due to the fact that the camera subsystem lags up to two seconds behind real time.

**Full System Automatic Building:**

There should be makefiles for the building of all code within the system, and a build script that will set up a fresh Raspberry Pi (Or any linux system, ideally) with all of the appropriate libraries and packages. Our team worked towards this goal but could not fully automate all of the building of the Pi.

**System Refinement / Project Scalability:**

Currently the system is only compatible with one reference object. This means that at any point in time, the device can only be calibrated to guide the user to exactly one point, and no other point. In order for the product to be useful, this **must** be scaled up such that aruco markers (within range, anyway) are capable of guiding users to multiple objects, and which object the user is being guided to can be changed at run-time. (These objects should probably play different sounds, too).

This will require some design effort, and should likely be the main focus of the team.