Augmented Reality Markerless World Alignment

Rishivandhan Musuvathi, Nate Kite, Edward Liu, Mihir Joshi

Advised by Dr. Kyle Johnsen

Sponsored by Gulfstream Aerospace

Disclaimer

The assumptions, findings, calculations, and conclusions expressed and described in this report and its exhibits were developed by undergraduate engineering students who are not licensed professional engineers. This report was prepared as an academic exercise as a partial fulfillment of the College of Engineering Capstone Senior Design course. No part of this report should be used for planning, budgeting, construction, or fiscal-related decisions without a complete review and written endorsement from an independent, qualified, and licensed engineer who is willing and able to become the engineer of record for all aspects of the study, calculations, findings, recommendations, and the project. A complete copy of this report was provided to the client without any financial reimbursement to its authors or the University of Georgia. The client may keep one copy of the report and is hereby given permission to copy and share the report as their needs dictate; however, a copy of this disclaimer shall accompany all copies made. By the acceptance of and/or use of this report and the exhibits hereto, the client and all reviewers of the content included herein shall indemnify and hold harmless the University of Georgia, College of Engineering, University employees, and the authors of this report from any and all liability, of whatsoever nature, that may result from such review, acceptance, or use.

Table of Contents

1. Introduction

1.1 Gulfstream Aerospace

Gulfstream Aerospace is a subsidiary of General Dynamics, one of the largest defense contractors in the world. Gulfstream is responsible for the design, development, and manufacturing of some of the world's most technologically advanced business-jet aircraft.

The company was founded in 1958 and is headquartered in Savannah, Georgia. Over the past sixty years, its luxury private jets have become highly well-regarded, and are often used by corporations, governments, and high-net-worth individuals for business and personal travel.

1.2 Project Background

As a massive aerospace manufacturer, Gulfstream must maintain an extremely high standard of safety and efficiency. To meet this need, Gulfstream uses AR systems (running on the Microsoft Hololens) during the process of constructing a plane. These technologies allow users to track the millions of components which go into their aircraft, and provide an intuitive way for workers to visualize the plane they are building.

In order for these systems to function, Gulfstream must map the real-world plane to a 3D model at a very high level of precision, and track the user's movement within the real world. Currently, this localization process is done using QR codes, which are placed within the aircraft during development. Scanning the QR codes is currently a source of frustration, and our goal is to find a replacement localization system.

1.3 Project Goal

Gulfstream asked us to develop an augmented reality (AR) application that does not rely on cumbersome QR codes or the discontinued Hololens. We have decided to develop a simple, easy to use iPad Unity AR app that will perform the alignment. The app is very straightforward, in that the user only needs to move the iPad around to scan the area with LiDAR, before the app localizes the pre-prepared 3D model in AR such that it aligns with the real world. This means that our app has applications at almost every stage of Gulfstream's construction process, as they have very detailed models of every stage of their plane construction, and can therefore be used to localize any model on corresponding real plane in AR regardless of what stage of construction it is in.

Our app can also have applications outside of plane construction, if Gulfstream has a need to compare 3D models with real world counterparts of other objects such as objects that are not yet part of a plane but may be added to one in the future. The 3D model does not have to be a plane in order to be used with our app, which means the applications of our app are not dependent on the model being a plane.

1.4 Design Team

Rishivandhan Musuvathi

- Developed the bulk of the of the Unity application, including the LiDAR scanner, UI, project setup, networking, and Unity-side alignment
- Helped test and implement point cloud registration algorithms
- Acted as team lead and coordinated meetings with clients and mentors, and work sessions among team members

Nate Kite

- Led development of solutions, especially research into the various algorithms used for localization
- Implemented GeDi and the control panel, and aided with alignment and UI in Unity
- Spearheaded troubleshooting across the entire application
- Primarily responsible for presenting project information to clients and mentors

Edward Liu

- Built a Flask server to accept a point cloud in JSON, return a transformation matrix.
- Conducted background research on a variety of topics and possible solutions
- Assisted with testing and debugging the prototype
- Contributed most work for milestones and managed their deadlines and submissions

Mihir Joshi

- Took measurements and created test models in CAD
- Assisted in testing of application

1.5 Project Management Tools

We used the following technology to coordinate our development:

- **Discord:** Communication through text and voice chat meetings, sharing screens during live calls or sending screenshots/images, etc.
- **GitHub:** Used for storing and backing up software.
- **Google Drive:** Used for storing non-software files related to the project, such as photos, videos, and milestone submissions.

2. Project Requirements and Constraints

After speaking with Gulfstream, our team identified four key requirements:

1. The application must localize the user with high precision (< 1 cm in error).
2. The localization process should be fast and intuitive.
3. The localization process should not require physical installations.
4. The application should run on modern, accessible technology.

2.1 Localization

The chief purpose of this application is to properly align a 3D model to the real world. Gulfstream's models are extremely precise, as their planes consist of hundreds of thousands of small components, and AR applications become disorienting to use if the environment is misaligned. As a result, Gulfstream asked for an extremely precise alignment: preferably within 1 mm of error, and no higher than 1 cm. While this is a tight requirement, it makes for the best possible experience for Gulfstream's engineers, and allows for maximum productivity.

2.2 Alignment Strategy

As mentioned above, Gulfstream currently uses QR codes to align the real world to the virtual one. However, these QR codes are cumbersome and take a long time to scan, and their installation and maintenance is a drain on resources. Developing an alignment strategy which does not rely on fiducial markers is the primary goal of this project, and any solution we come up with must be faster and less frustrating to use.

2.4 Hardware

Gulfstream currently uses the Microsoft Hololens, a mixed reality/augmented reality headset. However, the Hololens is old, frustrating, and has since been discontinued by Microsoft. Gulfstream asked us to develop a solution which runs on modern software, and specifically called out the iPad Pro as the best option. This simplifies development for ourselves and Gulfstream, and makes things easier on users.

3. Background Research

3.1 Brainstorming

In our brainstorming phase, we considered a range of possible approaches. However, early approaches were ruled as ineffective or infeasible.

Idea	**Reason for exclusion**
3D object recognition using AR Foundation	Requires a 3D marker.
Radio sensors to broadcast location	Less convenient than QR codes.
Image recognition	Likely imprecise, and infeasible due to lack of team expertise.
Manual localization (user stands in predetermined spot)	Too imprecise, though this idea may be combined with other solutions.

Figure 3.1.1 Early ideas and reason for exclusion.

From this, our team concluded that in order to be more convenient than QR codes, an approach must require no real-world components. This led us away from more “creative” solutions and towards sensor data. The iPad Pro supports a reasonably precise LiDAR scanner, [1] which is useful for operating in 3D space.

3.2 LiDAR-Based 3D Localization

Our initial research focused on localization algorithms which are used in various robotics applications. We never implemented these approaches, though they helped demonstrate that LiDAR was an appropriate foundation for a localization scheme.

3.2.1 Simultaneous Localization and Mapping (SLaM): Literature on LiDAR-based localization is overwhelmingly targeted towards SLaM. However, these techniques aren’t useful to us, since SLaM techniques can’t localize the user in known space, and Apple’s ARKit already implements a sophisticated SLaM solution.

3.2.2 Adaptive Monte Carlo Localization (AMCL): AMCL localizes the user by sampling possible perspectives and comparing those perspectives to the user’s sensor data. Early on, we considered this to be a promising candidate. However, we couldn’t find a usable implementation of AMCL in 3D, and were forced to discard the idea. [2][3]

3.2.3 Neo Localization: This localization algorithm is used by Neobotix robots, and is described as “a simple but effective replacement for the standard AMCL localization.” [4] While this algorithm was appealing, it would have required the use of ROS2 (which we wanted to avoid), and extensive reconfiguration to force it to work with the iPad’s sensors.

3.3 Point Cloud Registration

Point cloud registration is the task of aligning two point clouds. By converting a 3D model to a point cloud, we can use these algorithms to align the model to the LiDAR scan. These algorithms are well-supported and designed for use in a 3D context, making them attractive options.

There are many point cloud registration algorithms, [5] so only the algorithms which we actually implemented and tested will be shown.

3.3.1 Iterative Closest Point (ICP): ICP is a simple point cloud registration algorithm, and is easily implemented in Python via libpointmatcher. [6] However, when we used ICP for localization, the effects were far too small to be noticeable.

3.3.2 Coherent Point Drift (CPD): CPD is another point registration algorithm, which models point clouds as “Gaussian Model Mixtures.” We implemented this using pycpd, [7] but suffered from the same issues as ICP; the alignment was only a very small transformation.

3.3.3 GeDi [8]: GeDi is a deep-learning based neural network. It performs a coarse-to-fine algorithm, which means it first performs rough alignment, then fine-tunes that initial alignment for a more precise result. This addresses the issues we faced with ICP and CPD, where the algorithm could not produce results if the initial alignment was not close. GeDi is also extremely robust to noise, making it ideal for a real-world application. According to PapersWithCode, it is a state-of-the-art algorithm, performing the best on several important benchmarks. [5]

4. Design

As discussed in the previous section, we researched and evaluated different alignment approaches. Our final solution uses GeDi, a state-of-the-art neural network used for point cloud registration. [8] This section discusses its implementation in detail.

4.1 Overview

Our application works in three stages: frontend (Unity), the external server (Flask), and the point cloud registration algorithm (GeDi). The frontend is responsible for collecting LiDAR data from the environment, while GeDi is responsible for the actual localization. Flask manages networking between the iPad and the server. See Figure 4.1.1 for a high-level overview.

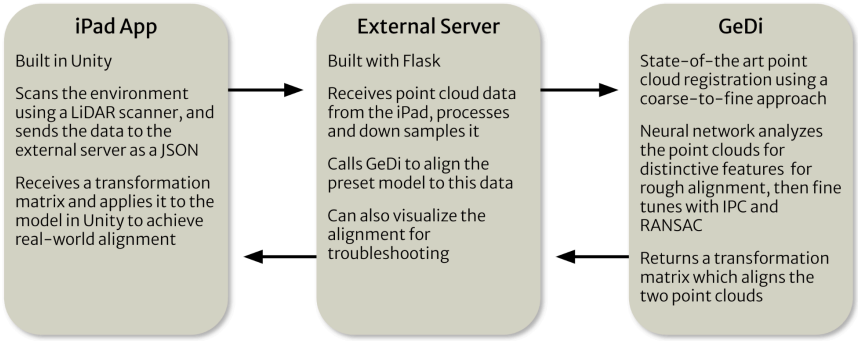

Figure 4.1.1 - Information flow through the application.

4.2 Scanning in Unity

The alignment process begins in the Unity app, which we built using Unity's AR Foundation and Apple's ARKit. The app is responsible for collecting environmental information using the iPad's LiDAR sensor, and ultimately for rendering the model[1] once aligned.

To collect the data, we found an existing implementation on GitHub which uses an ARDensePointCloudManager to collect and visualize the depth images which are collected through the sensor. [9] That data is stored as a point cloud, which can then be serialized as a JSON and sent to the external server through an HTTP request.

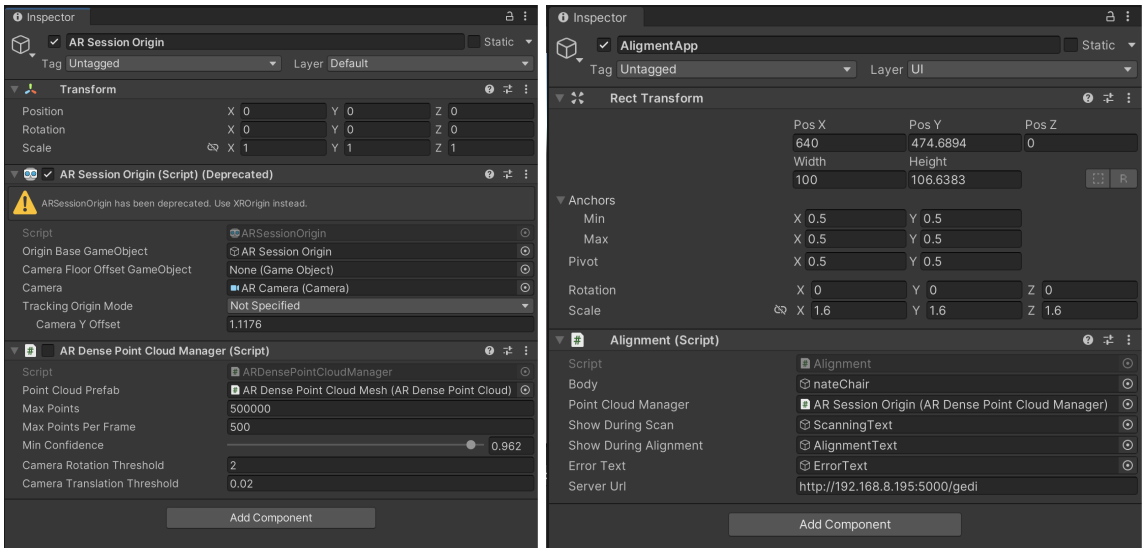

Figure 4.2.1 - The ARDensePointCloudManager attached to the AR Session Origin, and the Alignment GameObject.

[1] Note that the model must be loaded ahead of time. However, it is not used until the alignment process is complete.

4.3 External Server

The external server, built in Flask, acts as a middleman between the iPad and GeDi. It can be configured using a control panel, where the user must input the model for alignment.[2] This model is converted to a point cloud, which is necessary for GeDi to function properly. Our control panel also supports visualization, which can aid in debugging.

When the external server receives the LiDAR data, it immediately mirrors it. This is necessary because Unity is a left-handed coordinate system, whereas Open3D, which GeDi uses, is a right-handed coordinate system. This causes other issues, which are resolved later (See section 4.5 - Alignment in Unity). The data is then downsampled (to improve runtime) and sent to GeDi.

There are many requirements to installing and running the external server. See our GitHub (github.com/rishivandhan/CapstoneARTool) for full documentation.

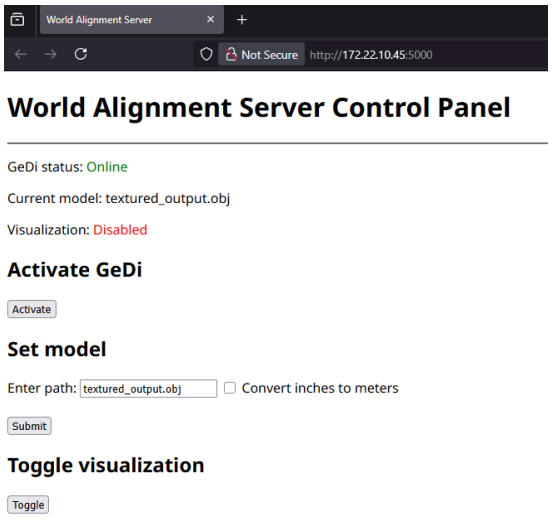

Figure 4.3.1 - Server control panel.

4.4 GeDi

Once processed by the external server, the data is sent to GeDi. GeDi is a deep-learning-based neural network which aligns the 3D model in the server (which has, at this point, been converted to a point cloud) to the LiDAR scan.

GeDi returns a transformation matrix, which is then returned to the external server, and sent directly to Unity.

4.5 Alignment in Unity

The external server returns a 4x4 transformation matrix to Unity. However, this matrix cannot be applied as is. Unity uses a left-handed coordinate system, whereas Open3D and the OBJ file format are right-handed. To account for the differences in coordinate systems, we multiplied the x component of the

[2] The "Convert inches to meters" checkbox was used in development when CAD models were described in units other than meters. For ease of use, all models should face Y-up and be in meters. The model in the server must be the same model as the one in the iPad.

transformation by -1, and we took the inverse of the rotation. See Figure 4.5.1 for a visual depiction of the alignment process as it relates to the coordinate systems.

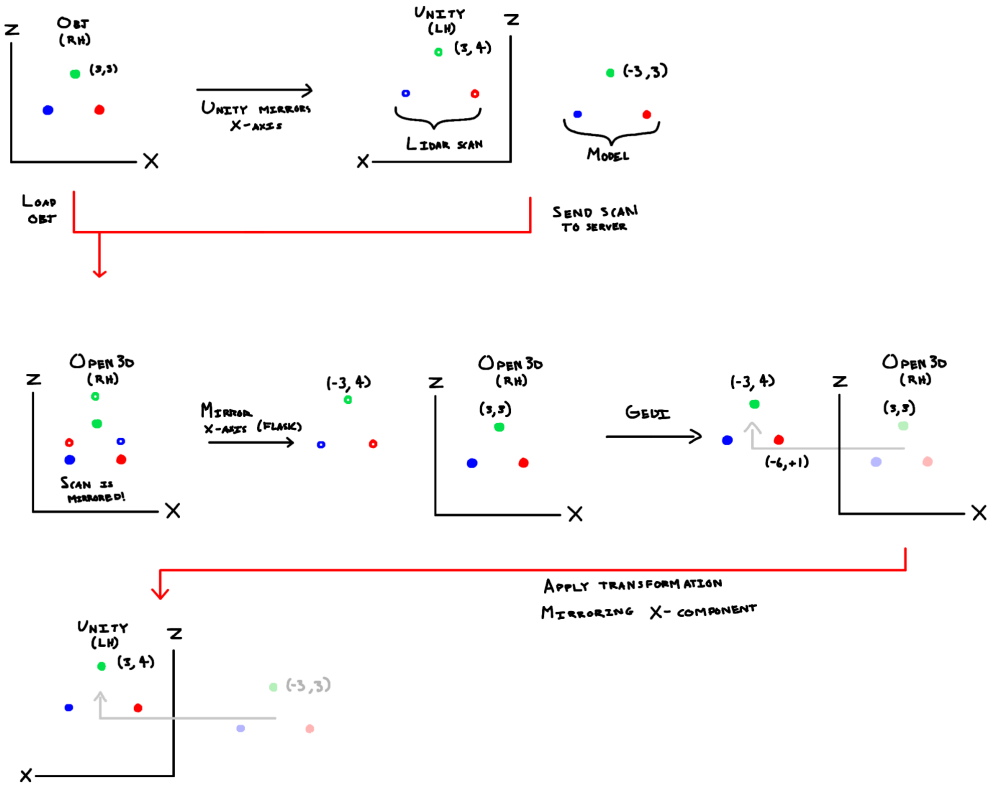

Figure 4.5.1 - Coordinate system conversion

Once the alignment process is complete, we display the GameObject with a transparent blue material, which allows the user to see the GameObject as an overlay on the real world. See Figure 4.5.2 for an example of the result post-alignment.

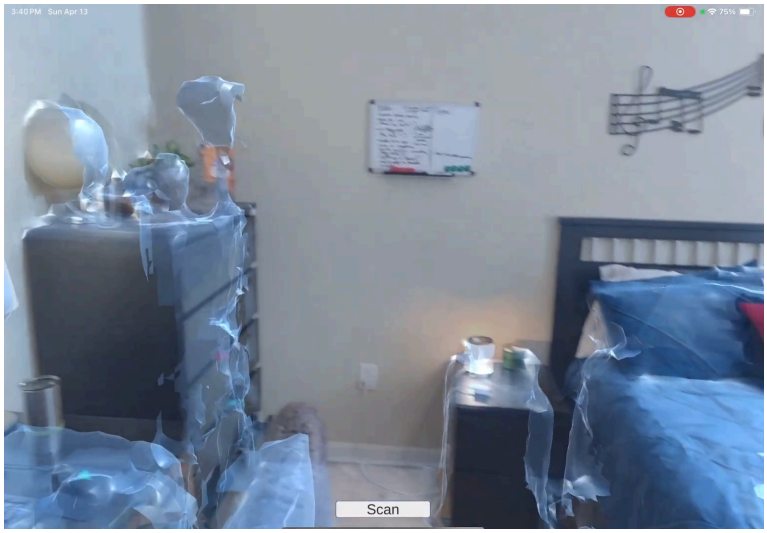

Figure 4.5.2 - Final rendering

5. Evaluation

5.1 Alignment Accuracy and Error Evaluation

As seen in Figure 5.1, GeDi does return an accurate transformation matrix, aligning the model to the real world as seen though the LiDAR scan. Blue is the 3D model in its initial location. Red is the point cloud seen by LiDAR. Green is where the model is moved after transformation. The localization shown in Open3D, on the computer running the Flask server, is very accurate, and has an error less than 1 cm.

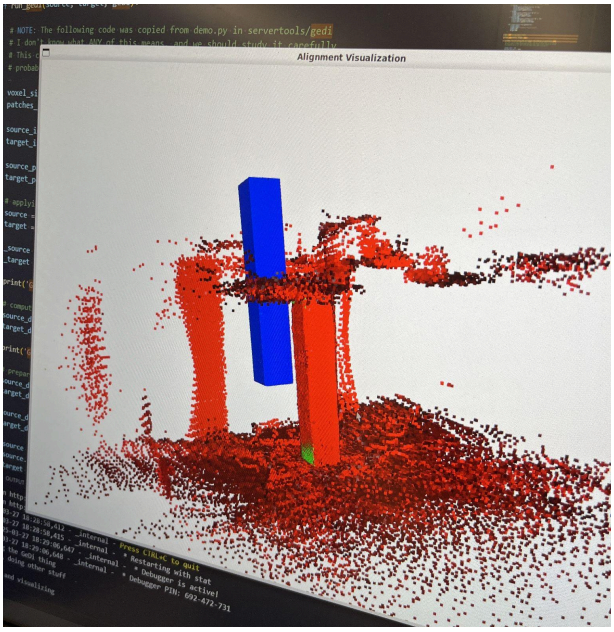

Figure 5.1.1 - Alignment Visualization in Open3D

With that said, this does not always result in perfect accuracy once the iPad tries to visualize this transformation. In Figure 5.2, you can see an attempt to align a model of a table with the real world table, and there is some error of approximately 5 cm. This error mainly results from 3 factors: imperfect models, inaccurate scans, and Unity transformation error.

Figure 5.1.2 - Table Alignment

If the model is imperfect and does not perfectly match the real world object, then it may not be possible to match some parts of the model with the real world. We encountered this problem because we did not make very high detail models. This should not be a problem for Gulfstream, as they have very detailed models of their planes.

Inaccurate scans are partly a result from the hardware limitations on the iPad Pro 11-inch LiDAR, and partly a result of how the iPad moved in differing ways while performing the scan. The iPad Pro's LiDAR is very usable and works for our app, but in terms of accuracy, it does not match up to an industry LiDAR scanner built specifically for 3D. This is something out of our control, but it should not be a significant problem, as we were still able to have the iPad visualize a 3D model of a pillar both in Open3D and in the iPad Unity app's AR with less than 1 cm of error.

Regarding user and iPad movements during the scan, the user should make sure to scan as many salient features as possible. With the pillar, we could easily scan all 4 sides by walking around it. With the table in Figure 5.2, there are more sides and it is harder to scan, in particular, we did not thoroughly scan the 3 interior sides underneath the table as much as we scanned the exterior sides. This may be a problem for Gulfstream in some stages of construction, depending on how easily they can move the iPad around and how many salient features they can scan. There is some room for refinement both the scanning procedure and in processing the point cloud generated by the scan.

When Unity applies a transformation onto an object, there is some error, likely from rounding errors during the calculations related to comparing and transforming points. This should not be a problem for Gulfstream as it had no significant impact on our app's ability to visualize the localization of the 3D model, but it should be noted that most of our test cases were smaller than the size of a plane, and that the design team are not experts in Unity. It is very likely that there is room for improvement in the transformation algorithm and visualization software.

5.2 Client Feedback and Stakeholder Needs Evaluation

We presented the final prototype to our client contacts, the Gulfstream XR team, and received feedback from them. Overall, they are satisfied with the progress of our project. Due to the complexity of this project, their foremost concern was detailed documentation, so that they could replicate and test our prototype effectively, as well as understand our design process and why we made the decisions we did.

Clear documentation is a stakeholder need that we successfully meet with this Final Report as well as the README on our GitHub repository.

Our application is also easy and intuitive to use. After everything has been installed and set up, the user only needs to open the app and begin scanning. Our server control panel and app UI make setting up and debugging easier as well, so it is also easy to use for the software engineers.

6. Conclusion

Our tool represents a working prototype that satisfies major design requirements and resolves the concerns present in Gulfstream's current localization scheme. With refinement, this prototype could be effectively deployed for use in aircraft construction. Our code is available at github.com/rishivandhan/CapstoneARTool.

6.1 Requirement Satisfaction

Gulfstream had four primary requirements:

1. The application must localize the user with high precision (< 1 cm in error).
2. The localization process should be fast and intuitive.
3. The localization process should not require physical installations.
4. The application should run on modern, accessible technology.

Our application does not totally satisfy these requirements, but it is a major step towards a complete solution. The third and fourth requirements are satisfied completely, and the second only is easily accomplished with user testing and a UI overhaul. The first requirement was mostly satisfied, but our prototype falls short of the extremely high standard of precision afforded by the existing solution.

6.2 Recommendations

When we began this project, Gulfstream asked us to investigate approaches they hadn't considered. Our prototype represents a promising path forwards, and could be developed into a full application. We recommend several paths for improvement on our prototype.

6.2.1 Testing: Our team did not have access to a proper testbed, so our error metrics are only guesswork. Gulfstream should carry out extensive testing to verify that our prototype is reliable, precise, and worthy of further investment.

6.2.2 Alignment: Our prototype does not meet Gulfstream's standard for alignment (though testing will reveal how far we are from this goal). This might be fixed by retraining GeDi on a dataset after correctly orienting the vertical axis. Restricting degrees of freedom dramatically decreases the potential for error in alignment.

6.2.3 User Experience: Due to time constraints, we could not carry out user testing. However, this is critical to minimizing user frustration, and user input seems to have a major impact on alignment precision, so it's important that users are consulted when designing the final form of the application. A technologically superior app may fail if its interface is more clumsily designed.

We have identified three major UI questions which can be solved with user testing: How will the user select the model to align to? How should the scan be completed? (By a timer? Or manually by the user?) And, how should the user be instructed to use the application?

6.2.4 Runtime: Our prototype runs slowly. Good UX design will help mitigate user frustration through clear feedback, but improving load times is the best solution. Downsampling the data further may help, but better hardware will also be a major improvement. We ran our application through Windows Subsystem for Linux (WSL) on a 4090 NVIDIA GPU. We hope that if the application is run on a native Linux machine, it will run acceptably quickly.

6.3 Acknowledgements

This project was an excellent learning experience, and our team benefited enormously from the mentorship provided both by the Gulfstream XR team and Dr. Johnsen. Many thanks to UGA and Gulfstream for providing an excellent experience and learning opportunity!

7. References

[1] *Evaluating the accuracy and quality of an iPad Pro's built-in lidar for 3D indoor mapping.* https://www.sciencedirect.com/science/article/pii/S2666165923000510

[2] *Multi-sensor three-dimensional Monte Carlo localization for long-term aerial robot navigation.* https://journals.sagepub.com/doi/pdf/10.1177/1729881417732757

[3] amcl_3d by rsasaki0109. https://github.com/rsasaki0109/amcl_3d

[4] Neo localization. https://neobotix-docs.de/ros/packages/neo_localization.html

[5] PapersWithCode - point cloud registration. https://paperswithcode.com/task/point-cloud-registration

[6] libpointmatcher. https://github.com/norlab-ulaval/libpointmatcher

[7] pycpd. https://pypi.org/project/pycpd/

[8] Learning General and Distinctive 3D Local Deep Descriptors for Point Cloud Registration. https://ieeexplore.ieee.org/document/9775606

[9] AR Foundation - DensePointCloud. https://github.com/cdmvision/arfoundation-densepointcloud

8. Appendix

```python
@app.route('/gedi', methods=['POST'])
def localize_gedi():
    if not model_pcd:
        print("Model has not been loaded!")
        return "Model has not been loaded!", 500
    if not gedi:
        print("GeDi has not been loaded!")
        return "GeDi has not been loaded!", 500

    try:
        print("Beginning alignment with GeDi...")
        scan_pcd = tools.build_pcd(request)
        transformation = tools.run_gedi(model_pcd, scan_pcd, gedi)

        print("Transformation matrix acquired: ")
        print(transformation)

        # Visualize output
        if visualize:
            print("Visualizing output...")
            aligned = copy.deepcopy(model_pcd)
            aligned.transform(transformation)
            tools.visualize(source=model_pcd, target=scan_pcd, transformed=aligned)

        print("Sending response to iPad.")

        # Return transformation data as json
        return jsonify({"transformation": transformation.tolist()})
    except Exception as e:
        print(str(e))
        return jsonify({"error": str(e)})
```

Appendix A - The function localize_gedi() in the Flask server receives the scan, aligns it with a model, and returns a transformation matrix. It can optionally visualize the alignment locally on the server computer. For more details, see servertools/pointcloudhelpers.py and https://github.com/fabiopoiesi/gedi (GeDi Repository)

```
[SerializeField]
private string serverUrl = "http://127.0.0.1:5000/gedi";

GameObject startButton = null;

public void beginAlignment()
{
    pointCloudManager.enabled = true;

    // Update UI
    startButton = GameObject.Find("StartButton");
    startButton.SetActive(false);
    showDuringScan.SetActive(true);

    StartCoroutine(align());
}
```

Appendix B - The code on the iPad Unity app invokes the align() function as a coroutine, resulting in the start of the alignment process. See Figure 4.2.1 for a reference on how this appears in the Unity Editor.

```
IEnumerator align()
{
    // Wait for scan
    yield return new WaitForSeconds(15f);

    // Update UI
    showDuringScan.SetActive(false);
    showDuringAlignment.SetActive(true);
    pointCloudManager.DestroyAllPointClouds();
    pointCloudManager.enabled = false;

    // Get point cloud as a list
    NativeArray<Vector3> points = pointCloudManager.pointCloud.points;

    List<float[]> pointList = new List<float[]>();
    foreach (Vector3 point in points)
    {
        pointList.Add(new float[] { point.x, point.y, point.z });
    }

    // Convert to JSON
    string jsonData = JsonConvert.SerializeObject(new { points = pointList }, Formatting.Indented);

    // Send data to Flask server
    UnityWebRequest webRequest = new UnityWebRequest(serverUrl, "POST");
    byte[] bodyRaw = System.Text.Encoding.UTF8.GetBytes(jsonData);
    webRequest.uploadHandler = new UploadHandlerRaw(bodyRaw);
    webRequest.downloadHandler = new DownloadHandlerBuffer();
    webRequest.SetRequestHeader("Content-type", "application/json");

    yield return webRequest.SendWebRequest();
```

Appendix C - The function align() in the iPad Unity app that converts the scan into JSON formatted points, sends it to the backend Flask server, and receives a transformation matrix in response.

```
// Applies the received transformation matrix to body
public void applyTransformationMatrix(float[,] transform)
{
    if (body == null || transform.GetLength(0) != 4 || transform.GetLength(1) != 4)
    {
        Debug.Log("invalid object or transformation matrix");
    }

    // Build matrix out of the transform array
    Matrix4x4 matrix = new Matrix4x4();
    matrix.SetColumn(0, new Vector4(transform[0, 0], transform[1, 0], transform[2, 0], transform[3, 0]));
    matrix.SetColumn(1, new Vector4(transform[0, 1], transform[1, 1], transform[2, 1], transform[3, 1]));
    matrix.SetColumn(2, new Vector4(transform[0, 2], transform[1, 2], transform[2, 2], transform[3, 2]));
    matrix.SetColumn(3, new Vector4(transform[0, 3], transform[1, 3], transform[2, 3], transform[3, 3]));

    // Get translation from the matrix, then invert x
    Vector3 translation = new Vector3(transform[0, 3], transform[1, 3], transform[2, 3]);
    translation.x *= -1;

    // Convert rotation matrix to Quaternion, then invert it
    Quaternion rotation = matrix.rotation;
    rotation = Quaternion.Inverse(rotation);

    // Apply transformation and make body visible
    body.transform.position += translation;
    body.transform.Rotate(rotation.eulerAngles);
    body.SetActive(true);

    Debug.Log("Transformation applied.");
}
```

Appendix D - The function applyTransformationMatrix() in the iPad Unity app that processes the transformation matrix and then transforms the locally stored 3D model GameObject such that it aligns with the real world as seen through AR.