When Anchoring Fails: Interactive Alignment of Large Virtual Objects in Occasionally Failing AR Systems

Anil Ufuk Batmaz[1] and Wolfgang Stuerzlinger[2]

[1] Kadir Has University, Istanbul, Turkey
aubatmaz@khas.edu.tr,
[2] Simon Fraser University, Vancouver BC, Canada
w.s@sfu.ca,
http://vvise.iat.sfu.ca

Abstract. Augmented reality systems show virtual object models overlaid over real ones, which is helpful in many contexts, e.g., during maintenance. Assuming all geometry is known, misalignments in 3D poses will still occur without perfectly robust viewer and object 3D tracking. Such misalignments can impact the user experience and reduce the potential benefits associated with AR systems. In this paper, we implemented several interaction algorithms to make manual virtual object alignment easier, based on previously presented methods, such as HoverCam, SHOCam, and a Signed Distance Field. Our approach also simplifies the user interface for manual 3D pose alignment in 2D input systems. The results of our work indicate that our approach can reduce the time needed for interactive 3D pose alignment, which improves the user experience.

Keywords: 3D User Interfaces, 3D Pose Alignment, Augmented Reality Interaction, Orbiting, 3D Navigation

1 Introduction

Augmented Reality (AR) systems show the viewer virtual geometry overlaid over the real world. There are many applications of this technology, including entertainment, medicine, education, training, commerce, and maintenance [1]. In many of these areas using an AR application can decrease the cost, time, and effort required to perform tasks. On the other hand, there are still many limitations of AR technologies, which can affect the performance of the application and also negatively impact the user experience. In this work, we present several methods that improve the user experience in situations when the tracking in AR systems fails. To accurately and robustly visualize virtual objects superimposed over real ones, AR systems depend on knowing the 3D poses of both the viewer and the objects in the environment. Many different 3D tracking technologies exist, yet, none of the contact-less options for 3D tracking is known to be 100% robust.

Most 3D tracking systems that use cameras or optical sensors can be affected by light sources, varying lighting levels, resolution, and/or occlusion issues. Inertial measurement units (IMUs) can measure the motion of the viewer (or the object), but are prone to drift. Combining multiple tracking systems can increase robustness, as demonstrated by many current commercial systems. Yet, even those systems that use combinations of tracking technologies are not 100% robust in practical application scenarios, especially in outdoor environments. This manifests as occasional "glitches" in tracking, where the overlay does not match the real world. If tracking does not recover on its own, the only option is then for the user to manually correct the pose of the overlay, which requires at least specifying the pose of the virtual content by control of the 3D position and 3D orientation. In this case, the user has to manually adjust the six degrees of freedom (6DoF) of the pose (3D position + 3D orientation). While naïve approaches for this task afford direct manual control over all 6DoF, this also leads to a complex user interface that requires training and strong spatial cognition skills.

Many AR applications, including maintenance, focus on a restricted scenario where the user looks at a *single* object with known geometry, with information or a virtual model overlaid on said object [2].

However, the target real-world object might not always be completely visible to the tracking camera, especially if the user is close to the object. In this case, the tracking camera on the AR system can be used to recognize the (relative) pose of the object through a variety of methods, including attaching one or more markers to the target object or detecting features of the object. Then, the system first detects specific features in the images, and by using these features, derives the pose of the real object.

With feature-based algorithms, specific features of the object are detected in the image, and the pose of the object is computed based on these features. One approach to this is to *anchor* the features of the real world object to the virtual one to derive the pose of the real object relative to the camera, an approach used in many computer vision systems [3, 4]. In general, marker-less approaches are preferred, as the object does not have to be instrumented with markers, which can interfere with real-world usage.

Regardless if markers are used or not, current computer vision algorithms are not always robust and can fail to detect the real object pose accurately. This applies to all current AR systems, regardless if tracking is outside-in or inside-out, especially in outdoor scenarios. Changes in the environmental conditions, such as lightning, the noise in the system, e.g., due to resolution or timing limits, or visibility/occlusions issues, can affect the performance of the pose tracking algorithm in terms of accuracy and can cause the tracking to temporarily fail in unpredictable ways [5–7]. Sometimes the tracking system is able to quickly recover on its own, but this is also not always guaranteed [8–10].

In scenarios where the the object that the user is interested in is large enough so that it does not fit into the camera view, tracking algorithms can fail even more easily, since the computer vision method may not be able to detect anchoring

points and then match them to the virtual model. Or the computer vision system might not have enough information to recognize all relevant poses [11].

One scenario where this can occur very easily is when the user stands close to a large object, such as a 300 meter long container ship, which features large, featureless surfaces, e.g., along the side of the ship [12, 13]. Consider here the case when the user is (say) 1-2 meters away from the ship's side, looking directly at it to inspect some new damage. In this situation, any camera facing in the viewing direction will see very few features.

Also, because the user is so close to the side of the ship, even the silhouette may not be distinct enough to detect the relative pose, if said silhouette is even visible in the field of view of the camera (and it is typically not). This naturally limits any computer vision-based approach. While one can *temporarily* rely on IMUs to compensate for a loss of camera-based tracking, IMU-based approaches drift too much to make this a practical solution for more than a few seconds. In the ship inspection scenario accurate, robust pose tracking is thus a very challenging problem that has no known solution.

When the anchoring algorithm of the tracking system fails in such scenarios, one approach that can help here is if the user can interactively adjust the pose of the virtual (overlaid) model in the AR system to match reality better. One interesting application scenario is to use a mobile device, such as a tablet, as the AR viewing device, i.e., as a "window" to see the virtual content. Still, a tablet provides only a 2D input surface, to afford control over all 6DoF. In such cases, the AR system can benefit from advanced user interface methods that simplify the interaction by reducing the need to specify all 6DoF directly for matching the alignment of the virtual model to the real world, which then also improves the user experience.

The goal of this paper is to present and test algorithms that improve the user experience when vision-based algorithms, such as anchoring-based ones, fail to track the target object. In the current maintenance applications, when the anchoring algorithm fails, the user then has to manually match the virtual and real world object to be able to use the AR system. Moreover, such pose matching needs to happen through the 2D tablet screen, and thus, 2D interaction methods, even though adjustments in more than 2DoF are typically required. Our objective is to simplify the user interface of the AR system to make it easier for a user to quickly match the poses of the a virtual and real object.

In this paper, we focus on a scenario where a computer vision-based tracking method fails in some way in an tablet-based AR context and where the user then has to interactively adjust the pose of the virtual model. The rest of the paper is organized as following: in the second section, we describe a use case for the given problem, followed by an explanation how we stabilized the tracking of the virtual objects in the third section. The fourth section describes the degrees of freedom restriction methods we added to simplify the user interface. In the fifth section we discuss the presented methods, followed by the conclusion.

2 Use case

In this paper, and as exemplar large object, we focus on maintenance scenarios for a commercial passenger airplane [14], more than 30 meters long, 30 meters wide, and 12 meters high. We also ignore the particulars of the computer vision-based tracking method and consider only the design of the user interface for aligning the virtual model in the occasional situations when the computer vision method fails to accurately detect the pose of the object. We implemented our system in Unity for an AR tablet.

3 Stabilizing the tracking

As airplanes are usually in a known orientation on the flat ground surface when they undergo maintenance, it is reasonable to assume that the aircraft does not have any rotation along the roll (side-to-side axis) or pitch (front-to-back axis) axes [15]. Since we also know already that the pose of the object is currently not automatically detectable, the user can only rely on the 6DoF IMU tracking.

Since state-of-art AR tablets have an embedded 6 DoF IMU, it is possible to use the rotation data of the tablet to stabilize the virtual object pose. However, the 6 DoF IMU data on tablets is typically noisy, and thus requires filtering before the data can be used for overlaying the virtual content in the AR system. As we know that the user is holding the tablet in their hands, we use the One-Euro filter to stabilize the virtual object rotation, which is appropriate for interactive applications.

3.1 One-Euro Filter

The One-Euro (1€) filter is a user-centred, low-pass filter designed to work efficiently with human movement speeds [16]. The One-Euro filter is also a fast algorithm that can predict a virtual objects' position with less delay compared to other methods, such as a Kalman filter. Compared to other algorithms, it also yields better performance in terms of mean error distance for tracking applications and significantly reduces battery consumption. These features of the One-Euro filter motivated us to implement this algorithm for our scenario, see Figure 1.

Our AR system uses the 9 DoF data received by the tablet's IMU and compass. This data is subject to jitter, which can have detrimental effects on the user experience and tracking accuracy [17–21]. We filter the tablet pose with the One-Euro filter, as this method also features minimal lag. The implementation of this algorithm requires fine-tuning of the filter coefficients. Each tablet and application scenario typically requires their own calibration coefficients based on the performance of the AR system, but this is usually fairly easy to do.

After we fine-tuned the filter algorithm, we used it to rotate the virtual object in the inverse direction of the filtered tablet IMU data. For instance, if the tablet is rotated 30 ° along an axis, we rotate the virtual object image -30 °, which stabilized the virtual object.

Fig. 1. Using the IMU data provided by the tablet, our AR system stabilizes the rotation of the virtual plane.

4 Restricting the Degrees of Freedom

Restricting the degrees of freedom makes it easier for users to interact with an object in a virtual environment, e.g., by locking distinct position or orientation axes, or by limiting the speed of change [22–24]. Various approaches have been used in 3D environments where the user only needs to control two or fewer DoFs at a time with the current input mapping.

Such methods allow users to interact with a 3D virtual object with 2 DoF input, e.g., on the 2D screen of a AR tablet. In our use case, we considered the situation where the user looks at the real world object from a previously known distance. For simplicity, we used the final distance between the tablet and the target object from the vision-based anchoring algorithm, just before it failed.

Thus, in this paper, we restrict the DoFs by allowing users to orbit around a virtual object. In contrast to naïve orbiting approaches that simply keep the user at a given distance from the *center* of the object, we use approaches that keep the distance between the user and the *surface* of the object constant. This class of approaches has been shown to improve the user experience, as the view roughly corresponds to how a user would move around an object to inspect it. It also avoids that the user accidentally orbits into an object, e.g., when the user was standing close to the body of the aircraft, but after orbiting can find themselves inside the fuselage, an aircraft engine, or the wing, which is undesirable.

4.1 HoverCam

HoverCam is a navigation method for orbiting the camera around an object, while keeping the distance between the camera and the virtual object constant [25], which enables the viewer to maintain a reasonable view of the object (Figure 2).

This is particularly relevant for airplanes, which are non-convex objects with long protrusions (such as the fuselage and the wings). For such objects, naïve, i.e., object-center-based orbiting methods require the user to constantly adapt the distance, as they may otherwise orbit through a wing or may need to move closer to the side of the aircraft to be able to see sufficient detail. HoverCam avoids this issue by maintaining a constant distance from the object [25].

During orbiting at a constant distance to the object surface with HoverCam, the user only has to focus on movement along two directions (Figure 2), horizontal and vertical. We map horizontal and vertical finger drag movements to the corresponding orbiting motions for the virtual object. For simplicity, we typically use the distance to the surface based on the outcome of the computer vision-based tracking algorithm before the tracking was lost, i.e., the last valid distance value. However, this requires that the tracking algorithm reliably reports when it looses tracking, which is not always the case.

To address this issue, the user can simply adjust the distance to the virtual object with the traditional two-finger pinch-to-zoom in gesture in our 2D AR tablet system, which is mapped to moving the camera directly towards (and away) from the object surface that the camera faces.

In essence, our implementation of the HoverCam method enables the user to orbit at user-defined distances, while still maintaining a constant viewing distance and avoiding collisions with the object. Given that there are only two interaction methods (dragging and pinch-to-zoom), this greatly simplifies the user interface for aligning the virtual object in the AR view.

4.2 SHOCam

While HoverCam generates overall a reasonable orbiting path around the object, this method suffers from motion artifacts at corners and concavities of the virtual object, which can reduce the quality of the visual experience (Figure 2).

Thus, we also implemented SHOCam [26], an improved version of the HoverCam method, which still maintains all properties of the HoverCam method. Yet, for rotation around corners or within concavities, SHOCam provides a smoother result than HoverCam (Figure 2), which is more visually pleasing. Given that SHOCam provides a smoother camera motion, this results in a better visual and user experience during interaction compared to HoverCam. These features also make SHOCam less error-prone.

4.3 Implementation Issues with HoverCam and SHOCam

Both HoverCam and SHOCam require distance computations for the virtual model. We initially tried different physics libraries for Unity, including HAVOK,

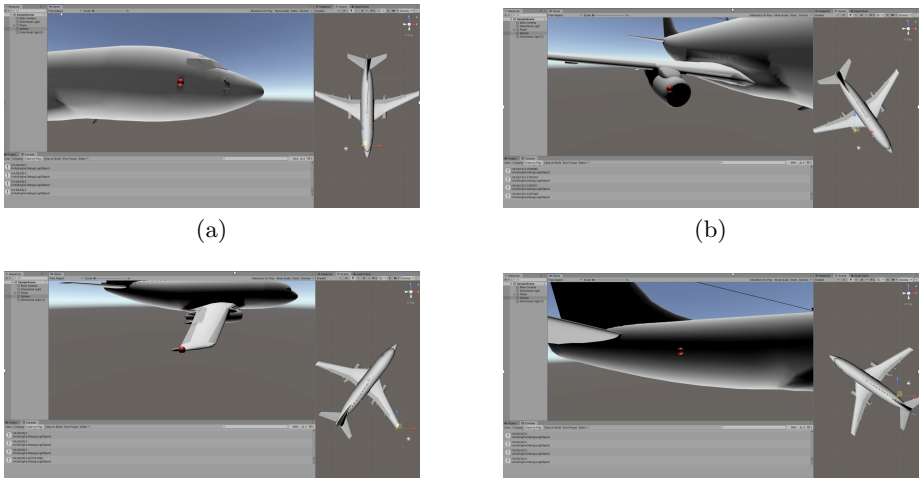


Fig. 2. Results for an example scene results with the SHOCam and HoverCam algorithms. In this scene, the distance between the user and the plane is fixed at 2 meters. When the camera moves around the plane, the algorithm maintains this distance between the camera and the closest point of the virtual model.

which was too slow, the Unity Physics Package, which was not sufficiently stable, and Unity's Data-Oriented Technology Stack (DOTS) library, which exhibited lag during the interaction. We settled on the the Unity Physics Library, as it performed best overall. Yet, this library supports only convex meshes with up to 255 triangles, far below the complexity of our virtual aircraft model.

The limitation of DOTS to convex meshes (such as the convex bounding polyhedron for the wing shown in Figure 3-1), but not the true non-convex 3D model mesh (Figure 3-2), creates a position-recognition problem for both HoverCam and SHOCam.

Both methods send a ray from the camera to the virtual object, and detect the collision point with the convex mesh, but not the object itself (Figure 3-3). This creates a bias in the virtual object position for the non-convex object (Figure 3-4). This could lead to false positives, which increases frustration and thus decrease the quality of the user experience.

To address this problem, the virtual aircraft's 3D model could be split into convex sub-objects, each with less than 255 triangles (Figure 3-5). This would enable the user to follow the surface of the virtual object smoothly with HoverCam or SHOCam. The necessary splitting could either be done manually, which is laborious, our automatically. Yet, algorithms to compose complex meshes into convex subparts have their own challenges, including the fact that they can generate (too) many subparts for non-convex curved portions of a model.This leads to a more complicated model hierarchy in Unity, and makes the system more error prone during the developing.

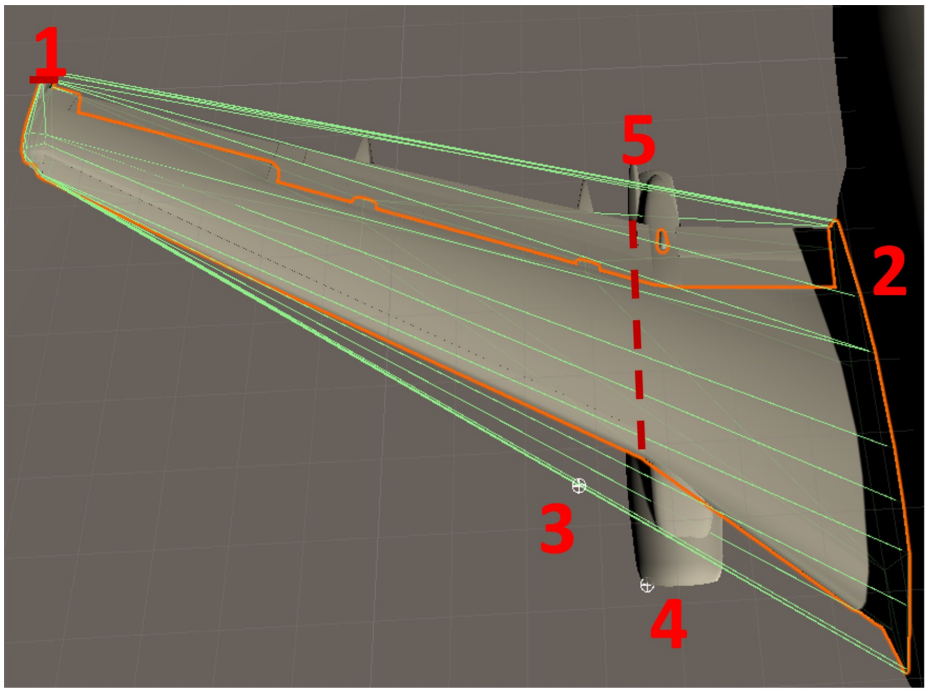


Fig. 3. Issues with the polyon mesh that restrict implantation of HoverCam and SHOCam in Unity. The 3D model of the plane wing is shown with orange lines and the convex bounding collider with green lines. In location #1 the 3D model and convex mesh collider both match. At #2, the mesh collider and virtual object occupy the same position, but since the mesh collider is convex, the green line and orange mesh do not match. In this case, HoverCam and SHOCam cannot compute the correct distance. Another example is shown at #3, where a white sphere is over the green line, while it supposed to be attached to the orange one. In location #4, it is possible to see a correct distance calculation, i.e., the 3D model and mesh collider occupy the same position, so that sphere is over both 3D model and mesh collider. As a proposed solution, a cut through the model along the red line between #4 and #5 could be used to divide the wing into two different virtual objects, both of which are (roughly) convex). This would allow to create (approximately) convex mesh colliders so that algorithm is less error prone. At show through this principle, modeling either only convex 3D objects or subdividing them into smaller objects can reduce the errors in distance calculations.

Additionally, SHOCam and HoverCam's accuracy has recently been challenged in studies that use these methods for Virtual and Augmented navigation [27]. Thus, and to increase the accuracy of our AR system, we decided to implement another method based on signed distance fields.

4.4 Signed Distance Field

The limitation to convex meshes for our initial HoverCam and SHOCam implementations forced us to transition to an algorithm that is independent of Unity's technical limitations. Thus, we decided to adopt a method based on a *Signed Distance Field* [28]. In Computer Graphics, a signed distance field specifies the distance to the nearest surface at every point in space and often used to improve the shading around a virtual object, as the impact of the light reflected from the virtual object can be approximated based by the distance between the virtual object and the space around it [29].

For this paper, we used a signed distance field to calculate distance values and gradients around the virtual aircraft object, similar to the computation in SHOCam Figure 3. In the bottom right of Figure 3, each line represents a constant distance from the virtual object's surface.

To integrate the signed distance field (Figure 4) into our system, we used a Python script with the igl library [30] on a desktop computer, which precomputes the distance field and then communicates individual distance measurements at the current viewpoint location over UDP to the software on the tablet. We also tried the Fast Winding Numbers method, which uses a similar approach but was designed to also support point clouds [31]. However, the Fast Winding Number method was not as fast as we expected and the corresponding lag reduced system responsiveness. Thus, we did not include this method in our system.

On the positive side, compared to the original HoverCam and SHOCam methods, the signed distance field yielded more precise and accurate distance values, especially in areas where the mesh is more complex, such as around the engine Figure 5-a or small protrusions on the fuselage's surface Figure 5-b.

Since this methods provides a more accurate value for the distance between the tablet and the virtual object, it also allows the user to work more accurately with the virtual model, i.e., make annotations on the virtual model that correspond accurately to the damage to the real object. This improves the user experience and also enables a better maintenance-related workflow.

5 Discussion

In this paper, we presented an AR system that improves the user experience in situations where a computer vision-based anchoring/tracking algorithm fails. In this situation, our user interface enables the user to quickly adjust the position of the virtual model to match the real one. This is particularly relevant for scenarios that involve the maintenance of large objects supported by AR systems [32]. We

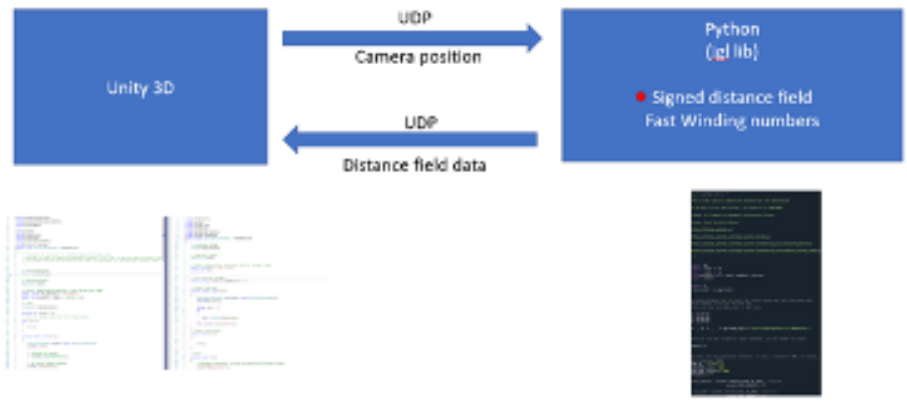
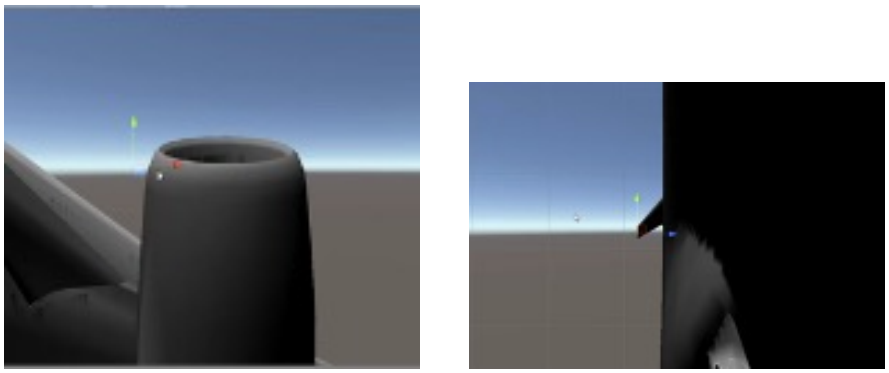


Fig. 4. Software architecture of the AR system for integrating a signed distance field. Since we used a Python library on a desktop for the signed distance field computation and lookup and Unity on the AR tablet to interact with the virtual objects, we used UDP to communicate between two parts of the system. We send the current camera position from the Unity application on the tablet to the Python code, the Python code then retrieves the distance value for that position from the signed distance field, and subsequently sends this result to the Unity application on the AR tablet.



(a) (b)

Fig. 5. Illustration of situations where the signed distance field yields more accurate results. a) The cavity associated with the turbine of an aircraft engine (facing upwards in the image) is highly non-convex. Methods based on convex geometry either "fill in" this cavity or need to decompose the engine into hundreds of individual convex parts. The signed distance field provides accurate results in this region. b) Similarly, a small antenna attached to the fuselage is again a non-convex part. The signed distance field again provides reliably distance values around such a small non-convex part of the aircraft.

tested our approach with a virtual model of an airplane and showed that the proposed methods are useful in a AR context.

Our most significant contribution is this paper was to apply a Computer Graphics algorithm to compute a signed distance field to support an orbiting technique within a 3D user interface on an AR tablet. Signed distance fields are mostly used in real-time rendering [33] and computer vision [34]. We thus also extend the utility of such algorithms to 3D user interfaces, where the results improve the user experience and make the AR system overall more robust.

Moreover, in this paper, we re-implemented two different previously presented algorithms for desktop systems, HoverCam and SHOCam, in Unity 3D and used them in a AR tablet application, where the virtual objects are posed in 3D. The SHOCam algorithm had been previously tested on a large displays and thus this work also supports the use of such Computer Graphics algorithms to improve the user experience during 3D interaction.

Finally, we applied the One-Euro filter, a method to decrease jitter and delays in a interaction software, within our system. We used this algorithm to decrease the jitter generated by the IMU of the AR tablet and stabilize the virtual object position when the tablet is rotated.

Overall we created a mobile AR application that can be used on a AR tablet to the improve user experience when the computer vision-based anchoring/tracking algorithm fails. In such cases, our method enables the user to quickly and simply fix the pose of the virtual model so they can continue using the AR application, e.g., to annotate the virtual model to record some damage to the real object.

6 Conclusion

In this paper, we implemented four different algorithms to improve the user experience in a AR tablet aircraft maintenance application, to better deal with situations when a tracking algorithms fails to identify the correct camera and/or object pose. We first suggest using the One-Euro filter to stabilize the virtual aircraft's pose vertically and horizontally when the tablet cannot acquire positional data from the tracking method.

When no reliable tracking information is available, we also proposed to use our implementation of the HoverCam and SHOCam methods, which allow the user to directly control the virtual object's pose through the surface of the tablet, to limit the movements to orbiting the virtual model along two dimensions. We also verified that SHOCam improves the visual experience during AR inspection over HoverCam.

Finally, we used a signed distance field to compute more accurate distance values and gradients even in the presence of non-convex or complex shapes, e.g., in areas with small protrusions on the plane's fuselage. This made orbiting more robust and addressed the restrictions of Unity's distance calculation methods through an external Python script.

From our initial evaluations we find that our proposed methods reduce the time needed for interaction with an AR tablet maintenance system when the anchoring algorithm for large objects fails, which improves the user experience .

7 Future work

In the future, we are planning to merge our proposed solution with a real-world aircraft AR tablet application and to tune the algorithms based on the real world experience. For instance, the current One-Euro filter coefficients are tuned only for stabilizing the virtual object, without considering the data from the DoF restriction algorithm. A future improvement is to integrate both solutions better by changing the corresponding One Euro Filter coefficients.

Moreover, we did not run a vision-based tracking algorithm on the tablet we used in our work, which means that we did not account for any restrictions in terms of computation due to the presence of the tracking algorithm. Depending how a tracking algorithm performs, one may have to re-consider the chosen user interface algorithm. For instance, even though the SHOcam method improves the visual experience, using HoverCam could lead to better interactive performance due to limitation in terms of computational power of the used tablet.

We also plan to implement our approach for AR headsets, such as Hololens 2, to make it easier to interact with a virtual object that needs to match a real one within the context of the AR scenario. AR headsets would also allow users to interact with the objects in mid-air, which would provide the interaction with virtual objects in the third dimension, i.e., visual depth. In this case, we will likely need additional interaction methods to also allow the corresponding adjustments.

References

1. Alkhamisi, A.O., Arabia, S., Monowar, M.M., et al.: Rise of augmented reality: Current and future application areas. International journal of internet and distributed systems **1**(04), 25 (2013)
2. Wang, W., Lai, Q., Fu, H., Shen, J., Ling, H., Yang, R.: Salient object detection in the deep learning era: An in-depth survey. IEEE Transactions on Pattern Analysis and Machine Intelligence (2021)
3. Yu, Y., Guan, H., Li, D., Gu, T., Tang, E., Li, A.: Orientation guided anchoring for geospatial object detection from remote sensing imagery. ISPRS Journal of Photogrammetry and Remote Sensing **160**, 67–82 (2020)
4. Dou, Z., Gao, K., Zhang, X., Wang, H., Wang, J.: Improving performance and adaptivity of anchor-based detector using differentiable anchoring with efficient target generation. IEEE Transactions on Image Processing **30**, 712–724 (2020)
5. Park, M.W., Makhmalbaf, A., Brilakis, I.: Comparative study of vision tracking methods for tracking of construction site resources. Automation in Construction **20**(7), 905–915 (2011)
6. Chi, S., Caldas, C.H.: Automated object identification using optical video cameras on construction sites. Computer-Aided Civil and Infrastructure Engineering **26**(5), 368–380 (2011)

7. Bae, H., Golparvar-Fard, M., White, J.: Image-based localization and content authoring in structure-from-motion point cloud models for real-time field reporting applications. Journal of Computing in Civil Engineering **29**(4), B4014,008 (2015)
8. Tomsett, R., Widdicombe, A., Xing, T., Chakraborty, S., Julier, S., Gurram, P., Rao, R., Srivastava, M.: Why the failure? how adversarial examples can provide insights for interpretable machine learning. In: 2018 21st International Conference on Information Fusion (FUSION), pp. 838–845. IEEE (2018)
9. Athalye, A., Engstrom, L., Ilyas, A., Kwok, K.: Synthesizing robust adversarial examples. In: International conference on machine learning, pp. 284–293. PMLR (2018)
10. O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G.V., Krpalkova, L., Riordan, D., Walsh, J.: Deep learning vs. traditional computer vision. In: Science and Information Conference, pp. 128–144. Springer (2019)
11. Brilakis, I., Park, M.W., Jog, G.: Automated vision tracking of project related entities. Advanced Engineering Informatics **25**(4), 713–724 (2011)
12. Li, Q., Mou, L., Liu, Q., Wang, Y., Zhu, X.X.: Hsf-net: Multiscale deep feature embedding for ship detection in optical remote sensing imagery. IEEE Transactions on Geoscience and Remote Sensing **56**(12), 7147–7161 (2018)
13. Schwegmann, C.P., Kleynhans, W., Salmon, B.P.: Synthetic aperture radar ship detection using haar-like features. IEEE Geoscience and Remote Sensing Letters **14**(2), 154–158 (2016)
14. Palmarini, R., Erkoyuncu, J.A., Roy, R., Torabmostaedi, H.: A systematic review of augmented reality applications in maintenance. Robotics and Computer-Integrated Manufacturing **49**, 215–228 (2018)
15. Eschen, H., Kötter, T., Rodeck, R., Harnisch, M., Schüppstuhl, T.: Augmented and virtual reality for inspection and maintenance processes in the aviation industry. Procedia manufacturing **19**, 156–163 (2018)
16. Casiez, G., Roussel, N., Vogel, D.: 1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2527–2530 (2012)
17. Teather, R.J., Pavlovych, A., Stuerzlinger, W., MacKenzie, I.S.: Effects of tracking technology, latency, and spatial jitter on object movement. In: 3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on, pp. 43–50. IEEE (2009)
18. Batmaz, A.U., Stuerzlinger, W.: The effect of rotational jitter on 3d pointing tasks. In: Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, CHI EA '19, pp. LBW2112:1–LBW2112:6. ACM, New York, NY, USA (2019). DOI 10.1145/3290607.3312752. URL http://doi.acm.org/10.1145/3290607.3312752
19. Batmaz, A.U., Stuerzlinger, W.: Effects of 3D rotational jitter and selection methods on 3D pointing tasks. In: Workshop on Novel Input Devices and Interaction Techniques (NIDIT) at (IEEE) (VR) 2019) (2019)
20. Batmaz, A.U., Rajabi Seraji, M., Kneifel, J., Stuerzlinger, W.: No jitter please: Effects of rotational and positional jitter on 3d mid-air interaction. In: Future Technologies Conference, *FTC '20*, vol. AISC 1289 (2020). DOI https://doi.org/10.1007/978-3-030-63089-8_52
21. Tumanov, A., Allison, R., Stuerzlinger, W.: Variability-aware latency amelioration in distributed environments. In: Virtual Reality Conference, VR '07, pp. 123–130 (2007). DOI https://doi.org/10.1109/VR.2007.352472
22. Mendes, D., Relvas, F., Ferreira, A., Jorge, J.: The benefits of dof separation in mid-air 3d object manipulation. In: Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology, pp. 261–268 (2016)

23. Mendes, D., Sousa, M., Lorena, R., Ferreira, A., Jorge, J.: Using custom transformation axes for mid-air manipulation of 3d virtual objects. In: Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology, p. 27. ACM (2017)
24. Caputo, F.M., Emporio, M., Giachetti, A.: The smart pin: An effective tool for object manipulation in immersive virtual reality environments. Computers & Graphics **74**, 225–233 (2018)
25. Khan, A., Komalo, B., Stam, J., Fitzmaurice, G., Kurtenbach, G.: Hovercam: interactive 3d navigation for proximal object inspection. In: Proceedings of the 2005 symposium on Interactive 3D graphics and games, pp. 73–80 (2005)
26. Ortega, M., Stuerzlinger, W., Scheurich, D.: Shocam: a 3d orbiting algorithm. In: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, pp. 119–128 (2015)
27. Pai, Y.S., Chen, Z., Chan, L., Isogai, M., Kimata, H., Kunze, K.: Pinchmove: improved accuracy of user mobility for near-field navigation in virtual environments. In: Proceedings of the 20th international conference on human-computer interaction with mobile devices and services, pp. 1–11 (2018)
28. Frisken, S.F., Perry, R.N., Rockwood, A.P., Jones, T.R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 249–254 (2000)
29. Xu, H., Barbič, J.: Signed distance fields for polygon soup meshes. In: Graphics Interface 2014, pp. 35–41. AK Peters/CRC Press (2020)
30. Jacobson, A., Panozzo, D., et al.: libigl: A simple C++ geometry processing library (2018). Https://libigl.github.io/
31. Barill, G., Dickson, N.G., Schmidt, R., Levin, D.I., Jacobson, A.: Fast winding numbers for soups and clouds. ACM Transactions on Graphics (TOG) **37**(4), 1–12 (2018)
32. De Marchi, L., Ceruti, A., Testoni, N., Marzani, A., Liverani, A.: Use of augmented reality in aircraft maintenance operations. In: Health Monitoring of Structural and Biological Systems 2014, vol. 9064, p. 906412. International Society for Optics and Photonics (2014)
33. Haines, E., Hoffman, N., et al.: Real-time rendering. CRC Press (2018)
34. Perera, S., Barnes, N., He, X., Izadi, S., Kohli, P., Glocker, B.: Motion segmentation of truncated signed distance function based volumetric surfaces. In: 2015 IEEE Winter Conference on Applications of Computer Vision, pp. 1046–1053. IEEE (2015)