AUTOMATIC FLYTHROUGH AUTHORING IN VR

ROBIN SCHMIDIGER[1], ROMANA RUST[2] and ROI PORANNE[3]
[1,2,3]*ETH Zurich*
[1]*schmrobi@student.ethz.ch, 0000-0002-3110-1061*
[2]*rust@arch.ethz.ch, 0000-0003-3722-8132*
[3]*roi.poranne@gmail.com, 0000-0002-1749-2596*

Abstract. The presentation of architectural models in virtual reality (VR) enables architects to evaluate their designs and present them to stakeholders, improving fidelity of architectural communication in architectural design. Virtual walkthroughs are a powerful tool to convey architectural concepts and unique features of the planned building. However, creating walkthroughs is a rather tedious process involving manual setting of keypoints at points of interest and in between, which are then interpolated to create the trajectory. This often results in awkward camera poses, sharp turns, intersection with geometry and clipping problems, all of which must be resolved by tweaking or adding new keypoints. Furthermore, the process requires constantly switching between an inside view, for local refinements, and an outside overview, for more global adjustments. To address this challenge, we present an interactive design tool for walkthroughs in VR. Our tool automatically provides a set of ideal keypoints, based on a customizable set of objectives, and instantly generates a complete and smooth trajectory that passes through them. It allows the user to tweak positions and update the walkthrough in real time, in VR. In addition, it allows the designer to have both an inside view and an overview simultaneously.

Keywords. Automatic Walkthroughs; Virtual Reality; Optimal Trajectory; User Interface; SDG 11.

1. Introduction

Walkthroughs, flythroughs, and virtual tours--also known as trajectories--are some of the most effective techniques for showcasing and highlighting a building and its unique features. Advances in real-time 3D rendering allow walkthroughs to reach new levels of realism, and have become necessary in many practices. As VR devices become more ubiquitous, so will immersive walkthroughs. There is a twist however: users are no longer limited to walking and looking in the direction of the path prescribed by the designer but are free to look around and stray from the path. Walkthrough designers should take this into consideration and carefully examine their walkthroughs in VR, using a system that facilitates tweaking and fine-tuning.

POST-CARBON, Proceedings of the 27th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2022, Volume 1, 49-58.

Designing a walkthrough can be a rather tedious process. It involves manually specifying a path via key points, which are then interpolated in various ways. Standard interpolation methods, e.g. splines, often result in paths that intersect with the geometry, leading to distracting clipping problems. These must be resolved by tweaking the existing key points, or by adding new ones. With this iterative process, the designer inevitably ends up with many redundant keypoints that are difficult to modify. Automatic trajectory generation algorithms have been proposed and are discussed in Section 2, but were not intended for an interactive VR interface.

Our goal is to develop a computer-assisted, immersive, and interactive tool for walkthrough design. The tool provides an initial walkthrough, optimized for a variety of metrics, and allows the user to tweak and simultaneously optimize it in an intuitive fashion in VR. The challenge lies in the optimization itself, which involves finding an optimal set of optimal keypoints based on metrics such as spatial coverage and visibility. Next, we address the question of how to specify the best visiting order, and finally, how to create a smooth and intersection-free trajectory. To enable interactivity, these computations must be performed in real time. Our contributions in this paper are threefold:

- An optimal keypoint placement method.
- A collision-free optimal trajectory that optimizes keypoint ordering.
- A VR design interface.

The early design phase has the highest impact on a building project and hence greatly influences material consumption and environmental concerns. Therefore, it is paramount that we include both implicit (aesthetic, cultural or emotional) as well as explicit (e.g. functional, environmental, economic) criteria early on to mitigate design problems. With our project we respond to goal 11 of SGS - (Make cities and human settlements inclusive, safe, resilient and sustainable), improving the communication of design ideas to stakeholders, that are involved at later stages of the design-to-construction processes, and therefore foster better informed design decisions.

2. Related work

Many applications provide tools to manually generate walkthroughs. A common approach is to prescribe a sequence of keypoints, which are then interpolated to create a path. Keypoint specification can be done by placement in the plan or the model of the building, and is the traditional approach when working with CAD/BIM/3d modelers. Alternatively, game engines allow the user to walk the scene in a first- or third-person perspective, and to assign keypoints at their avatar's position. Previous work on automatic walkthrough generation in the context of architecture is rather limited. To the best of our knowledge, the only academic work that directly addresses this problem is by Graf and Yan (2008). They proposed to create a graph where each node represents a room and each edge represents a doorway. This graph is then used to create a traversal of the building.

In this work, we rely on path-planning and trajectory-optimization techniques commonly employed in robotics. Path-planning refers to task of finding any path between two points of an environment, while avoiding obstacles. See Yang et al.

(2016) for a recent review. The path does not have to be optimal in any sense, and hence, path-planning is often followed by a trajectory-optimization phase (Kelly 2017). Lin et al. (2013) proposed a path-planning approach for 3D spaces using Fast Marching Methods. They provide insights on how, in the context of walkthrough planning, common 2D path-planning can be used instead of 3D path-planning, effectively reducing the complexity of the problem.

Creating a walkthrough of an architectural model is in particular similar to generating paths for drone cinematography (Galvane et al. 2018). Although virtual cameras are not constrained by the laws of physics, drone and virtual camera trajectory optimization share many goals. These include, but are not limited to, collision avoidance, cost optimization, smoothness of the trajectory, and coverage planning. Gebhardt et al. (2016) for example, developed a computational design tool that allows its users to easily create quad-rotor trajectories. They used an optimization-based method to ensure that the trajectories are feasible in the real world. Another method proposed by Nägeli et al. (2017) also tackles automated aerial videography. This time, online optimization is utilized, and the system is extended to work with multiple drones. Interactively-defined aesthetic framing objectives are also incorporated.

3. Automatic walkthrough generation

The computational aspect of our approach consists of two steps. First, a set of keypoints is determined. These keypoints are then sequenced by (approximately) solving the Travelling Salesmen Problem, and a path is created. In the second step, the path is further optimized to create a smooth and collision-free trajectory, while optionally considering other objectives as well. This optimization process can continuously run in the background while the user edits the keypoints, constantly improving the trajectory.

3.1. DETERMINING KEYPOINTS

To determine keypoint locations, we first construct an importance map, where higher values signify greater importance. The map is based on metrics such as distance to walls and visibility (Section 3.1.1). The keypoints are iteratively extracted based on their importance (Section 3.1.2).

3.1.1. Computing the importance map

The importance map is sampled on a voxel grid and consists of three components: obstacle, distance, and visibility (Fig. 1). The obstacle map is a binary map where a voxel contains '1' if it intersects with the structure's geometry. Voxels of the distance map contain the distance to the closest obstacle (floors and ceilings excluded) and is computed as a distance transform of the obstacle map. The goal of the visibility map is

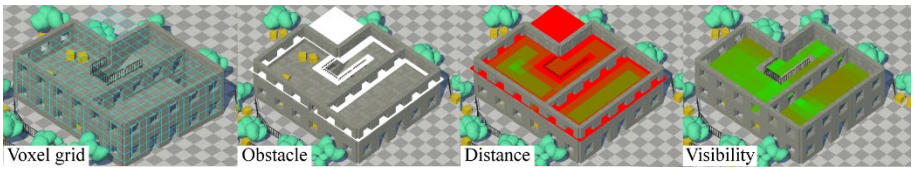

Figure 1. The different maps used for locating the keypoints

to measure how much of the structure is seen from each voxel. To compute the visibility at a certain voxel, we cast rays from that voxel to all other voxels and count the number of successful rays. The result is divided by the total number of cells and stored in the visibility map. Once the maps are computed, they are combined together as a weighted sum

$$\text{Importance}(P) = \lambda_{\text{obs}}(1 - M_{\text{obs}}) + \lambda_{\text{dist}} M_{\text{dist}} + \lambda_{\text{vis}} M_{\text{vis}},$$

Where $M_{\text{obs}}, M_{\text{dist}}, M_{\text{vis}}$ are the obstacle, distance and visibility maps respectively, and the λ's are user defined weights. Here, λ_{obs} is chosen as an arbitrarily high number, so voxels that collide with obstacles have low importance value. As can be inferred, voxels that are distant from obstacles have higher importance values, as well as voxels that are visible from many other voxels.

3.1.2. Extracting keypoints based on importance

With the importance function computed, our next goal is to place keypoints. The desiderata are to place them at locations with high importance values, but evenly distributed. A naïve approach would be to place a point in each voxel that exceeds a certain importance threshold. However, this has the risk of clumping many points closely together in regions of high importance, or neglecting peaks of high importance that are below the threshold. Another alternative is to pick local maxima of the importance map, but we found them to be too few, in general. Our approach is a mix of the two approaches. We iteratively pick the center of the voxel with the highest importance value as the next keypoint. Then, we reduce the importance of this voxel and all the voxels around it by a value proportional to the distance to the voxel. This creates a "cone" of low values around the selected voxel, preventing any close voxels from being selected in subsequent iterations. The radius and proportion of this cone can be adjusted by the user. Smaller radii will have higher chances of generating denser clusters of keypoints, while larger radii might produce too few keypoints. The process continues until the desired number of keypoints has been reached.

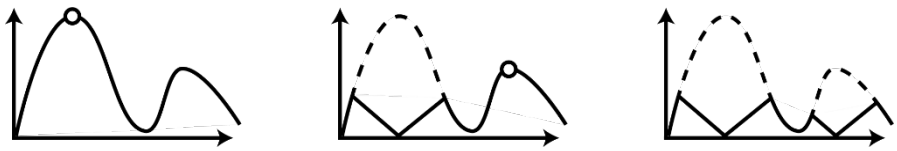

Figure 2. An illustration of the keypoint extraction process. From the left, the point with the maximal importance value is selected. It is then reduced to zero, with its vicinity reducing proportionally, creating a cone. The next maximum is the selected as the next keypoint, and the process repeats.

3.2. COMPUTING A TRAJECTORY

Once the points of interest have been determined, the next task is to compute an optimal trajectory that passes through them. We divide the task into two stages too. First, we determine the optimal sequence in which these points should be visited. We then create an initial rough trajectory and follow-up by a full trajectory-optimization routine.

3.2.1. Sequencing the keypoints

The optimal sequence is, arguably, the one that produces the shortest path that visits all of the keypoints. The problem of finding such a path is known as the Traveling Salesmen Problem. We begin by computing all shortest paths between every pair of keypoints. To do so, we leverage Unity's Navigation System, which works by automatically creating Navigation Meshes, and use them for path-finding. Ideally, we would then be able to solve the Traveling Salesmen Problem, but this problem is known to be intractable for more than a few points, since it scales factorially. Instead, we use a common approximation and compute the minimal spanning tree first, then proceeding with a depth first search. This creates a path that is guaranteed to be shorter than twice the length of the Traveling Salesmen path, and can be executed in real time. In any case, we also allow the user to modify the sequence, in order to address subjective considerations.

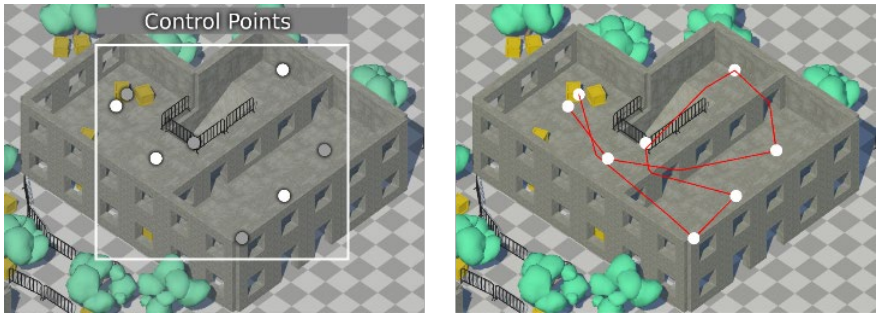

Figure 3. An image showing the keypoints on the left, and the optimal sequence shown on the right. The greyed out keypoints are points on the lower level that would normally be occluded.

3.2.2. Optimizing the trajectory

As can be seen in Figure 3, the path created is a polyline that has jagged edges and sharp corners, which might be visually disturbing. A common way to overcome that is to replace the straight edges by a smooth spline, e.g. a cubic b-spline. However, as mentioned, splines will sometime overshoot and collide with the structure geometry. As a remedy, users can add more keypoints, but this quickly becomes unmanageable. Instead, we opt to use a trajectory optimization approach. We discretize the polyline using a user defined number of points n (typically $n = 100$ points) and represent them by a vector $(\mathbf{x}_1, \dots, \mathbf{x}_n)$, where $\mathbf{x}_i = (x_i, y_i, z_i)$. Trajectory optimization is then formulated as a minimization problem

$$\min_{\mathbf{x}_1,\dots,\mathbf{x}_n} \mathcal{O}(\mathbf{x}_1, \dots, \mathbf{x}_n),$$

Where $\mathcal{O}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ is an objective that measures the quality of the trajectory. We solve this minimization problem using the quasi-Newton method L-BFGS (Nocedal and Wright 2006). The objective itself is a weighted sum of different subobjectives we specify in the following.

In our formulation, we used five different subobjectives. The first subobjective penalizes deviation from the initial polyline trajectory. Letting $(\mathbf{x}_1^0, \dots, \mathbf{x}_n^0)$ represent

that trajectory, this subobjective is expressed by

$$\mathcal{O}_d = \sum \|\mathbf{x}_i - \mathbf{x}_i^0\|^2$$

The next two subobjectives aim to minimize the velocity and acceleration of the trajectory. Minimizing the velocity serves to reduce the length of the trajectory, at the cost of slightly straying away from the keypoints. Minimizing the acceleration encourages a smooth trajectory, as well as constant velocity. They are expressed using first-order finite differencing, i.e.,

$$\mathcal{O}_v = \sum \|\mathbf{x}_{i+1} - \mathbf{x}_i\|^2, \qquad \mathcal{O}_a = \sum \|\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}\|^2.$$

These three subobjectives result in a smooth trajectory, but one that might pass through walls and obstacles. We encode obstacle avoidance as an additional objective which penalizes proximity to obstacles. We let the user define the desired distance from obstacles D. Then we find, for each $\mathbf{x}_i$, the closest point on an obstacle $\mathbf{x}_i^c$. If the distance between $\mathbf{x}_i$ and $\mathbf{x}_i^c$ is greater than D, the objective contributes 0. Otherwise we define

$$\mathcal{O}_c = \sum (\|\mathbf{x}_i - \mathbf{x}_i^c\|^2 - D)^2,$$

to encourage $\mathbf{x}_i$ to move to a distance D away from $\mathbf{x}_i^c$. Note that $\mathbf{x}_i^c$ is updated in each iteration and scenes crowded with obstacles can slow down the procedure. We note that we have also experimented using the distance map as a means to formulate this objective, but we found the results to be inferior, since the resolution of this map is limited. Finally, we included a flying height subobjective that ensures that the elevation of the trajectory over the floor remains as constant as possible. To this end, for each $\mathbf{x}_i$ we compute the distance to the floor $f(\mathbf{x}_i)$ and let

$$\mathcal{O}_h = \sum (f(\mathbf{x}_i)^2 - H)^2$$

be the height objective, where H is a user-specified desired height. This objective is useful in treating issues that occur around slopes, as can be seen in Figure 4. These are related to inconsistent height and lack of smoothness occurring between the keypoints at the bottom and top of the slope.

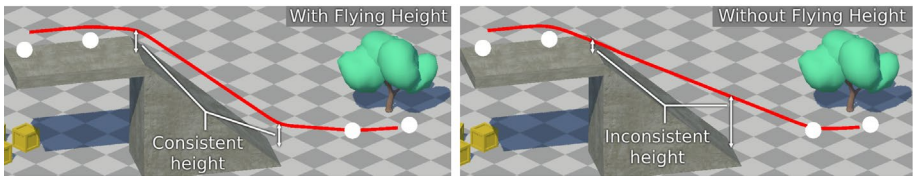

Figure 5. Results with and without the flying height objective. As can be seen, without this objective the height of the trajectory above the slope is only determined by the top and bottom key points, and thus is inconsistent along the slope. The issue is not present with this objective included.

The subobjectives are combined to form the total objective

$$\mathcal{O} = \lambda_d \mathcal{O}_d + \lambda_v \mathcal{O}_v + \lambda_a \mathcal{O}_a + \lambda_c \mathcal{O}_c + \lambda_h \mathcal{O}_h$$

Where λ are user-defined weights. The result of optimizing the path (shown in Figure 3) is shown in Figure 5. The significance of each subobjective is shown in Figure 6.

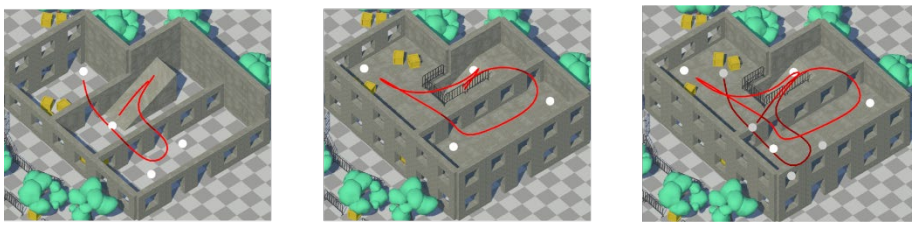

Figure 5. an optimal walkthrough trajectory of a two story building. Left and middle images show the trajectory in the first and second floor, and the right image shows the entire trajectory

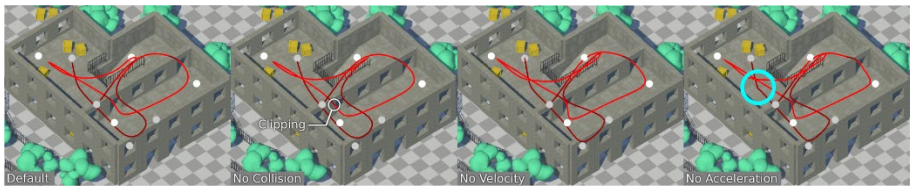

Figure 6. Demonstrating the effects of dropping any of the subobjectives

4. User Interface

While the trajectory generated with our method is optimal in a sense, users may wish to customize and manually modify keypoint locations, as well as certain characteristics of the optimal trajectory. Our algorithm enables this with little-to-no waiting time, which allows for real-time interaction. Our interface allows users to explore the interior of the building in VR in a familiar way, but also allows them to carry a miniature version of the building and observe the exterior. This creates a unique experience, whereby users can be inside and outside of the building simultaneously. A few images demonstrating this interface with the DFAB house (Graser et al. 2020) are shown in Figure 7.

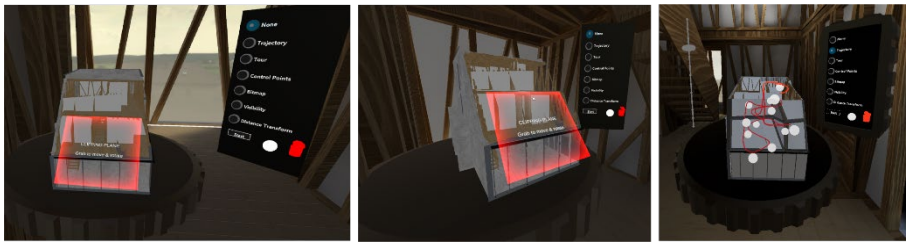

Figure 7. Images showing the miniature inside the building. The user can see the insides by manipulating a clipping plane (shown in the left and middle images). The right image shows the keypoints and trajectory as well.

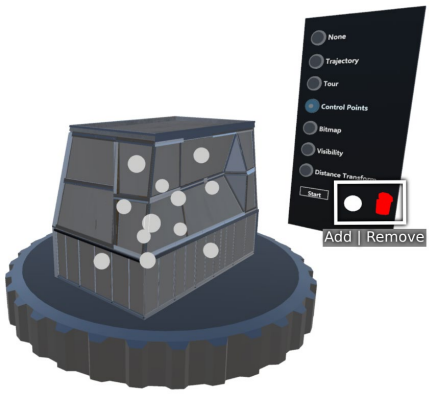

Figure 8. The miniature with its side panel. The user can easily move keypoints around by grabbing them, or add and remove point using the menu. Different visualization modes are shown in the menu as well

The miniature and its side panel are shown (without texture for improved clarity in the paper) in Figure 8. A specific keypoint can be grabbed and relocated using the controller. New keypoints can easily be added by grabbing them from the Add-section on the side panel and dragging them into the miniature. In order to remove a control point, it has to be dragged and released into the red recycling bin. The menu can be used to switch between visualization modes, as can be seen in Figure 9, and to start and stop playing the walkthrough using a button.

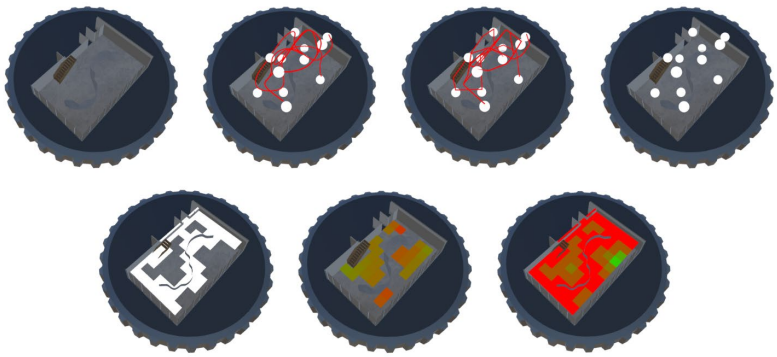

Figure 9. Different visualization modes seen on the miniature

To evaluate the user interface, we ran a small user study with 5 non-expert participants. Users were asked to design a walkthrough by placing keypoints sequentially, using our VR interface, but without the optimization. They were then asked to fill in a System Usability Scale (SUS) questionnaire (Sauro & Lewis 2012). Then, they were asked to design a walkthrough using our optimization and fill in the questionnaire again. In addition, participants were asked to comment on the differences between the two approaches. The SUS score is on a 1-100 scale, where a score above 68 is considered to be above average. The scores for the VR interface alone were 80, 77.5, 100, 87.5, 85 with an average of 86, while the scores for the optimization were

75, 75, 85, 82.5, 82.5 with an average of 80. Both were considered to be within the B range of usability. Participants noted that the initial automatic walkthrough gives them a good starting point and would save a lot of time. Participants also noted that without optimization the process was felt much longer, but at the same time, they felt more in control. Some participants identified boundary cases, which we must address in future work, which likely explain the drop in the score.

5. Conclusion

We presented an approach for automatic walkthrough generation based on detection of important keypoints and trajectory optimization. In addition, we implemented a VR user interface for planning walkthroughs, which leverages our optimization and allows real-time editing. The interface shows favourable results among participants, but further work is required in order to bring it beyond the experimental stage and tighten the experience in the UX/UI sense. While SUS gives initial indication regarding the usability, the fact that it is one-dimensional poses a limitation. Indeed, based on participant comments, it is clear that control and workload, as measured by, e.g. the NASA-TLX, play a critical role in system evaluation. A future user study will examine this with different populations and also include assessment of the resulting walkthroughs.

Acknowledgements

Special thanks to Konrad Graser for providing the DFAB HOUSE 3D model and to Foteini Salveridou for modelling the Unity scene. This work was supported in part by the Swiss National Science Foundation through the National Centre of Competence in Digital Fabrication (NCCR dfab).

References

Delgado, J. M. D., Oyedele, L., Demian, P., & Beach, T. (2020). A research agenda for augmented and virtual reality in architecture, engineering and construction. *Advanced Engineering Informatics, 45*, 101122.

Galvane, Q., Lino, C., Christie. M., Fleureau, J., Servant, F., Tariolle, F.-L., and Guillotel, P. (2018). Directing Cinematographic Drones. *ACM Transactions on Graphics* 37, 3, 18. https://doi.org/10.1145/3181975

Gebhardt, C., Hepp, B., Naegeli, T., Stevšić, S., & Hilliges, O. (2016). Airways: Optimization-Based Planning of Quadrotor Trajectories according to High-Level User Goals. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*.

Graf, R., & Yan, W. (2008). Automatic Walkthrough Utilizing Building Information Modeling to Facilitate Architectural Visualization. *eCAADe* 26 (pp.555). Education and research in Computer Aided Architectural Design in Europe (eCAADe).

Graser, K., Baur , M., Apolinarska , A., Dörfler, K., Hack, N., Jipa, A., Lloret, E., Sandy, T., Pont, D., Hall, D., & Kohler, M. (2020). Dfab house - a comprehensive demonstrator of digital fabrication in architecture. *Fabricate 2020: Making Resilient Architecture*, 04 2020, pp. 130–139.

Kelly, M. (2017). An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4), 849-904.

Lin, Y.-H., Liu, Y.-S., Gao, G., Han, X.-G, Lai, C.-Y, Gu, M. (2013) The IFC-based path planning for 3D indoor spaces. *Advanced Engineering Informatics*, 27(2), 2013, 189-205, https://doi.org/10.1016/j.aei.2012.10.001.

Nägeli, T., Meier, L., Domahidi, A., Alonso-Mora, J.& Hilliges, O. (2017). Real-time planning for automated multi-view drone cinematography. *ACM Transactions on Graphics*, 36, 4, Article 132 (July 2017), 10 pages. https://doi.org/10.1145/3072959.3073712

Nocedal J & Wright S. J. (2006). *Numerical Optimization.* Springer.

Sauro, J., & Lewis, J. R (2012). *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann, Waltham MA, USA.

Yang, L. & Qi, J. & Song, D. & Xiao, J. & Han, J. & Xia, Y. (2016). Survey of Robot 3D Path Planning Algorithms. *Journal of Control Science and Engineering,* 2016(1),1-22. https://doi.org/10.1155/2016/7426913.