A graph generation pipeline for critical infrastructures based on heuristics, images and depth data

Mike Diessner [1,*] **and Yannick Tarant** [1]

[1] *German Aerospace Center, Institute for the Protection of Terrestrial Infrastructures, Sankt Augustin, Germany*

Correspondence*:
Mike Diessner, German Aerospace Center, Institute for the Protection of Terrestrial Infrastructures, Rathausallee 12, 53757 Sankt Augustin, Germany
mike.diessner@dlr.de

ABSTRACT

Virtual representations of physical critical infrastructures, such as water or energy plants, are used for simulations and digital twins to ensure resilience and continuity of their services. These models usually require 3D point clouds from laser scanners that are expensive to acquire and require specialist knowledge to use. In this article, we present a prototypical graph generation pipeline based on photogrammetry. The pipeline detects relevant objects and predicts their relation using RGB images and depth data generated by a stereo camera. This more cost-effective approach uses deep learning for object detection and instance segmentation of the objects, and employs user-defined heuristics or rules to infer their relations. Results of two hydraulic systems show that this strategy can produce graphs close to the ground truth. While this study focuses on hydraulic systems, the general process can be used to tailor the method to other types of infrastructures and applications. The user-defined rules create transparency qualifying the pipeline to be used in the high stakes decision-making that is required for critical infrastructures.

Keywords: Graph generation, relational graph, critical infrastructure, photogrammetry, depth data, image data, scene understanding, digital twin

1 INTRODUCTION

Critical infrastructures are assets, organizations or facilities that are integral to the survival and functioning of a nation and its society. Disturbances in their services can have major adverse impacts, such as supply shortages, increased risks to public health and national security, or disruptions of financial services and the broader economy. Critical infrastructures can be physical or virtual assets and appear in sectors such as energy, healthcare, food, water, transportation, communication, nuclear systems and more (Osei-Kyei et al., 2021; Alcaraz and Zeadally, 2015; Yusta et al., 2011).

Methods such as simulations and digital twins are used to monitor, maintain and optimize infrastructures, and to prepare and train for worst-case scenarios by simulating extreme and often rare events. Thus, these methods have the potential to increase resilience, security, performance and continuity of infrastructures and are an invaluable tool to ensure national security and well-being (Lampropoulos et al., 2024; Sousa et al., 2021). In many cases, a digital twin of a physical asset, such as a nuclear facility or a water treatment plant, requires a virtual copy of the objects (e.g., pipes, pumps and valves) and their relation to each other to be used for simulations or training in virtual reality. The process of generating these models requires a large amount of manual work and is thus expensive and time-consuming (Franke et al., 2023). Most existing methods use laser scanners to gather 3D point clouds of the facilities that are then processed and analyzed. However, these scanners are expensive, creating demand for methods using more affordable and accessible equipment, such as stereo cameras.

The main objective of this article is the development of an algorithm that uses cost-effective photogrammetry (Moon et al., 2019), i.e., RGB images, depth data and camera positions, to automatically detect relevant objects within a building and predict the relations between these objects. The algorithm should yield a graph network, where nodes represent objects sand edges represent physical connections between objects. As the main use of this pipeline is for high stakes decision making for critical infrastructures, the algorithm design will focus on explainability and interpretability besides accessibility and cost-effectiveness, where possible. While we were able to validate the pipeline on synthetic scenes that mimic the real world, it remains a prototype that requires validation on actual real-world scenes.

This article is structured as follows. Section 2 places our proposed method into context by giving an overview of the related works. Section 3 details the developed graph generation pipeline and the data of the hydraulic systems used as the test environments for validation. Section 4 presents the results of applying the proposed pipeline to two hydraulic systems. Section 5 conducts a sensitivity and runtime analysis to investigate the robustness and practical feasibility of the pipeline. Lastly, Section 6 outlines the strength and limitations of the pipeline and Section 7 draws a brief conclusion.

2 RELATED WORKS

Most of the time, a virtual geometric representation of a building or a facility is the starting point of a digital twin. This representation is then enriched with attributes and information from individual objects, sensors and real-time data (Jones et al., 2020; Grieves, 2014). In the last decade, frameworks such as building information modelling (BIM) have been used increasingly in the planning and building of new facilities and can provide the required geometric and attribute data (Ghaffarianhoseini et al., 2017; Borrmann et al., 2018; Lu et al., 2022). While the adoption of building information modelling has grown rapidly in the last decade, it was likely not used for most older existing buildings. In these cases, the first step in building a digital twin is the collection of geometric data, the identification of relevant objects and the extraction of key attributes (Borkowski, 2023).

A popular method for collecting geometric data is the use of 3D laser scans. Laser scanners create a three-dimensional point cloud with a high resolution providing a detailed virtual representation of reality. However, these scanners are expensive and can cost many tens to hundreds of thousands of dollars. They are also heavy and require expensive software for processing the acquired data. Furthermore, specialized training is needed for operation and they struggle to represent reflective surfaces correctly (Moon et al., 2019). An alternative approach is photogrammetry that aims to generate precise 3D models from photographs and usually involve the computation of depth data. Multi-view stereo, for example, uses multiple perspective of a stereo camera or a RGB and a thermal camera to compute the depth information and a 3D scene (Goesele et al., 2006; Vidas et al., 2013), while Structure-from-Motion uses a series of 2D images to estimate the depth data and camera positions (Schönberger and Frahm, 2016). Overall, systems used for photogrammetry cannot achieve the same level of detail and resolution as 3D laser scanners but are cheaper, more accessible and more mobile.

3D laser scanning and photogrammetry provide two different outputs: point clouds and images. Both present distinct strength and shortcomings as inputs for object detection. Albeit that point clouds have a high resolution and are very accurate they are also sparse and the point densities can be highly variable leading to high computational requirements and costs (Zhou and Tuzel, 2018). Furthermore, there are a limited number of labeled data sets for point clouds available as labelling in three-dimensional space is challenging (Zimmer et al., 2022). This also makes collecting and labelling custom data sets for training difficult and time-intensive. Despite of this, point clouds were used in object detection for hydraulic systems in various articles (Qiu et al., 2014; Kawashima et al., 2014; Cheng et al., 2020; Alex and Stoppe, 2025), while the use of photogrammetry is limited (Hart et al., 2023; Zhao et al., 2025). However, as object detection on images is computationally cheaper and more training data is available, easier to collect and label, it lends itself for object detection as part of a graph generation pipeline that aims to be widely accessible, applicable to different use cases, and time- and cost-effective. Furthermore, models for object detection on images are generally more researched and thus more mature than their counterparts using point clouds. This is also reflected in the many models that are available, many of which even offer good out-of-the-box performance (Redmon et al., 2016; Kirillov et al., 2023; He et al., 2017; Yuan et al., 2021). In practice, we found that object detection on the many different perspectives of the images was reliable and results from different view points had the advantage of enabling validation of detections across images. The latter was especially valuable for featureless objects and instances in which lights and reflections made object detection challenging as there were multiple opportunities to detect the same object. This decreased the possibility of erroneous detections. Thus, we identified photogrammetry in combination with object detection on images as the superior methods for the pipeline presented in this article. While the presented pipeline is closely linked to multi-view stereo it is not just a geometric 3D representation of a scene but also aims to encode relational information between objects of the scene in a graph.

The prediction of a relational graph of the objects contained an image, also known as scene graph generation, through graph neural networks has seen increased interest by the deep learning community in recent years (Shit et al., 2022; Yang et al., 2018; Li et al., 2024; Cong et al., 2023). These approaches combine the detection of relevant objects in an image with the prediction of relations between the detected objects and encode both in a graph. Similar to other deep learning methods, these models are black boxes lacking explainability and interpretability (Buhrmester et al., 2021; Şahin et al., 2025) and although there are efforts to remedy this in the form of explainable AI (Xu et al., 2019; Zhang et al., 2022; Dwivedi et al., 2023; Peng et al., 2024), it is questionable if these efforts suffice when it comes to high stakes decision-making (Rudin, 2019). Furthermore, scene graph generation networks require data and additional labeling and cannot easily be transferred to other use cases and scenes involving different objects without collecting new data. For example, many graph generation networks are based on the 3DSSG data set that contains 1,482 scene graphs with 48k objects and 544k relations (Wald et al., 2019, 2020; Lv et al., 2024; Yeo et al., 2025). Collecting and annotating a data set of this size for a specific use case is, in practice, at least impracticable if not infeasible. An alternative to these models is an approach using heuristics and rules based on which relations between objects are inferred. Letting the user define a custom set of rules gives them full control and enables them to tailor the method to a specific application or transfer it to another without the need of collecting additional training data. While there is some initial manual work involved in setting up the rules, it is likely negligible compared to the effort of collecting and labelling data for the graph neural networks. This article chooses the latter approach due to its flexibility and cost-effectiveness but mainly because the rules make the inner workings of the algorithm transparent and explainable which

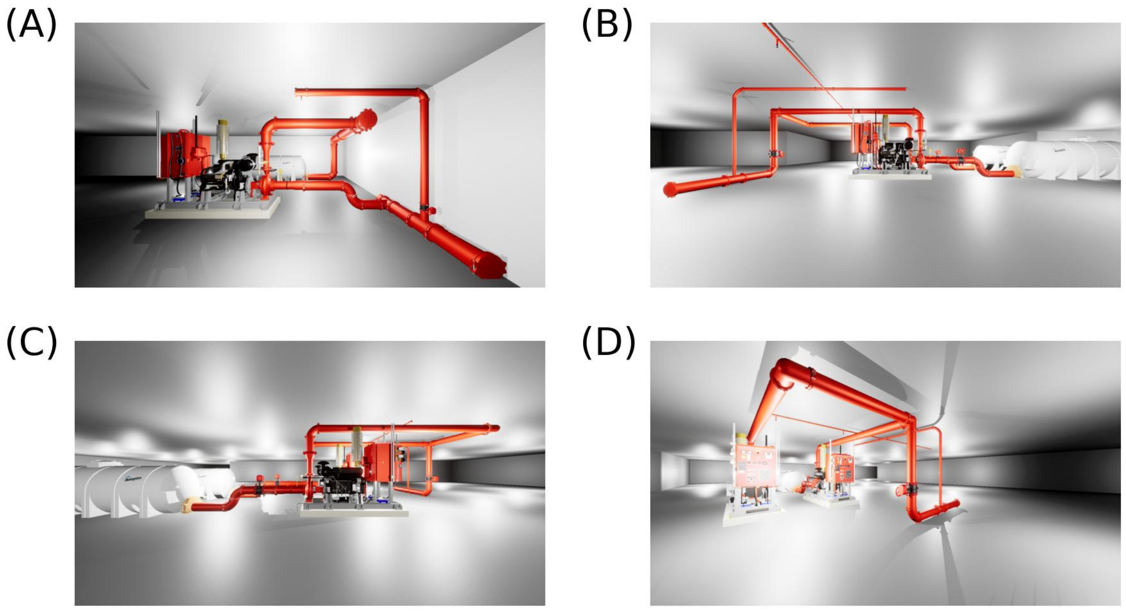

Figure 1. Synthetic hydraulic systems. (**A**) shows system 1 consisting of pipes, one pump, one tank, one valve and one sprinkler. (**B**), (**C**) and (**D**) show system 2 consisting of pipes, two pumps, two tanks, three valves and four sprinklers.

is essential for high stakes decision-making as it is required in critical infrastructure (Rudin, 2019). The process of defining these rules and their requirements are discussed in Section 3.2.3.

3 MATERIALS AND METHODS

3.1 Data

The nature of critical infrastructures prohibits the use of images and data collected at the actual real-world sites due to security concerns. Thus, we implement virtual representations mimicking the most important characteristics of their real-world counterparts in Unreal Game Engine 5 (Epic Games, 2022c). We chose the Unreal Engine as it allows for rapid generation of realistically rendered scenes and contains tools to provide global dynamic illumination (Epic Games, 2022a), rendering of meshes with arbitrary resolution (Epic Games, 2022b) and scripted capturing of realistic images in combination with segmentation masks. We use Colosseum (Codex Labs, 2022), a fork of AirSim (Shah et al., 2017), to simulate cameras in the Unreal Engine allowing us to produce RGB images and depth data along with the intrinsic and extrinsic camera parameters, such as the position and orientation of the camera at the time of taking the images. While real-world testbeds are preferable, the realistic results of the Unreal Engine strike a good balance between accessibility and an accurate representation of the real world.

This paper focuses on the graph generation of hydraulic systems by investigating two test environments. Hydraulic system 1 consists of a pump, a tank, a valve, a sprinkler and pipes as shown in Figure 1 (**A**). Hydraulic system 2 increases the complexity by adding another pump and tank, two valves and three sprinklers to the design. This system is shown from multiple perspectives in Figure 1 (**B**), (**C**) and (**D**). Pipes, pumps, tanks, valves and sprinklers are the only objects in these test environments. It should be noted that the distinct color of the pipes compared to the floor, walls and ceiling is chosen for illustrative purposes only. The components of the pipeline, such as the models used for object detection, are not reliant

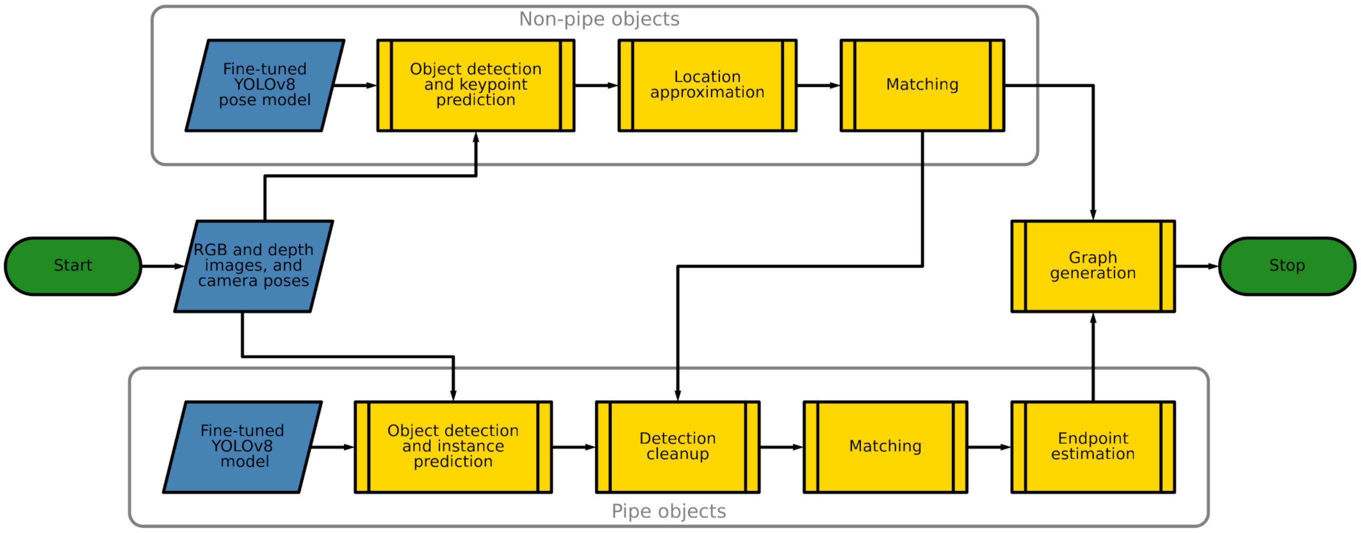

Figure 2. Flowchart of the graph generation pipeline. Pipe and non-pipe objects are treated differently and are combined in the 'Graph generation' module.

on the color of the pipes as they were trained and tested on a wide range of different color and texture compositions. We use domain randomization to randomly generate different color and texture combinations for the pipe objects and use them to produce a diverse set of training images via the Unreal Engine for fine-tuning YOLOv8 as outlined in more detail in Schreiber et al. (2024).

Data for hydraulic system 1 comprises 16 RGB and depth image pairs with a resolution of 1920×1080 pixels and a field-of-view of 114 degrees. For the more complex hydraulic system 2, we stock up the number of RGB and depth image pairs to 29 with identical resolution and field-of-view.

The training data for YOLOv8 consists of photos of hydraulic systems taken with a Canon EOS 7D SLR camera at a resolution of 6000×4000 pixels and synthetic images generated with Unreal Engine 5 as non-interlaced 8-bit RGBA images with a resolution of 1920×1006 pixels.

3.2 Pipeline

The graph prediction pipeline is structured as shown in Figure 2. The pipeline differentiates between pipe and non-pipe objects and processes them with two distinct approaches. Both use RGB images, depth data and camera poses including the camera position and camera orientation of the individual images and a fine-tuned YOLOv8 model (Redmon et al., 2016). While we chose YOLOv8 for object detection, the pipeline in this study does not require the use of YOLOv8. The detection model is solely an input of the pipeline and can be replaced with any preferred model. We decided to use YOLOv8 as it is an established model that does not require a large data set for fine-tuning the pre-trained model to a new detection task as it is the case for vision transformers (Dosovitskiy et al., 2021). It also provides versions for instance segmentation and keypoint prediction that are used as described in the following sections.

3.2.1 Non-pipe objects

Non-pipe objects are processed using a fine-tuned YOLOv8 pose model (Redmon et al., 2016) that detects the pumps, tanks and valves within the images and predicts one keypoint for tanks and two keypoints for pumps and valves. These keypoints represent the connection points of the tanks, valves and pumps with other objects, such as pipes. The largest YOLOv8 architecture pre-trained on the COCO data set (Lin et al.,

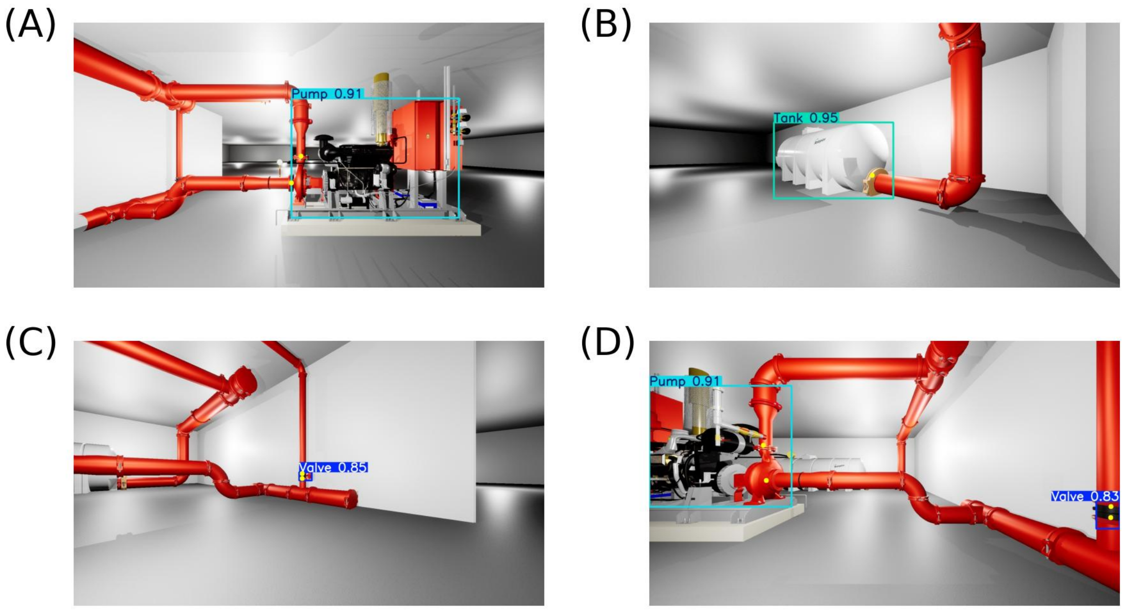

Figure 3. Object detection of pumps, tanks and valves including one keypoint for tanks and two keypoints for pumps and valves. Bounding boxes indicate detected objects with class label and class confidence score. Keypoints are indicated by yellow dots.

2014) was chosen as a foundation. This model was fine-tuned using a data set consisting of 526 images, with 280 being real-world RGB images and the remaining 246 being synthetically created using Unreal Engine 5 (Epic Games, 2022c). The training was conducted for 150 epochs using the Adam optimizer and a learning rate of 10^{-4}. Figure 3 shows predictions of the model for four images. Detections for an object consist of a bounding box with a class label and a class confidence score, and keypoints plotted in yellow. For example, Figure 3 (**A**) shows a pump detected with a confidence of 91 % and two keypoints indicating the connection points with the neighboring pipes. YOLOv8 uses Non-Maximum-Suppression (NMS) to discard redundant detections by only using bounding boxes with the highest confidence score and dismissing other overlapping bounding boxes with lower confidence scores.

The second module in the ’Non-pipe objects’ branch of the pipeline aims to approximate the location of the objects within the bounding boxes. While it would be possible to use an instance segmentation method to get exact masks of the objects (similar to the segmentation used for the pipes in Section 3.2.2), exact masks are not required for the pipeline to achieve satisfying results. Hence, we opt for an approximation of object masks sparing us time-consuming instance labelling of the more intricate non-pipe objects. The main objective of the location approximation is to delete any background pixels within the predicted bounding boxes, leaving us with only pixels belonging to the object. Experimentation showed that we can achieve this objective by chaining three methods together: (i) We disregard any pixels with depth values larger than the median of all depth values within a bounding box. This eliminates the majority of the background. (ii) We refine the mask by projecting the remaining pixels to a point cloud down-sampled to one voxel per cm^3 and removing any statistical outliers (voxels that deviate more than one standard deviation from a neighborhood consisting of the closest 25 % of all other voxels). (iii) Finally, we employ Density-Based Spatial Clustering of Application with Noise (DBSCAN) to approximate the location of the object (Ester et al., 1996; Schubert et al., 2017). We setup DBSCAN such that points that are less than 2 cm apart are assigned to a cluster. Finally, the largest cluster is taken as the approximated location.

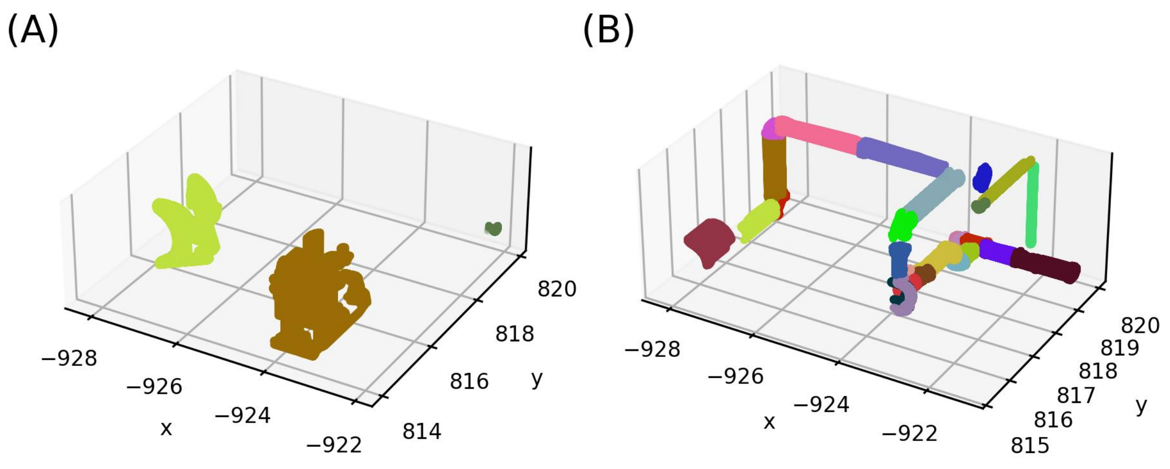

Figure 4. Object matching across individual images. (**A**) shows the matched objects of a pump, a tank and a valve and (**B**) shows the matched pipe objects. Each color indicates a distinct object.

The last module in processing the non-pipe objects matches identical detections across images and combines information in one final object. In this step, the approximated locations of merged objects are combined and their endpoints (predicted via the keypoints) are averaged. Before averaging, possible false predictions are disregarded via statistical outlier removal of keypoints that deviate more than two standard deviations from all keypoints. Two objects from different images are merged if the following rule for the pairwise distances **d** between all their voxels is true:

$$\frac{\sum_{i=1}^{n} \mathbb{1}_{\{d_i < np\_max\_distance\}}}{n} > np\_min\_percentage,$$

where n is the number of elements of vector **d**, and $np\_max\_distance \in (0, \infty)$ and $np\_min\_percentage \in [0,1]$ specify the maximal distance in meters between two object points to be considered a match and the minimal percentage of matched points required for two objects to be matched, respectively. $\mathbb{1}$ is the indicator function that returns 1 if a distance d_i is smaller than $np\_max\_distance$ and 0 otherwise. Thus, two objects are merged if the fraction of the number of pairwise distances smaller than $np\_max\_distance$ and the number of all pairwise distances is larger than $np\_min\_percentage$. The pipeline sets $np\_min\_percentage$ to 0.1 requiring two objects to have at least 10 % of points within $np\_max\_distance$ meters. Thus, whether two objects are merged is mainly controlled by setting $np\_max\_distance$. Figure 4 (**A**) shows three matched non-pipe objects from a total of 16 images using the approach outlined in this section.

3.2.2 Pipe objects

The pipeline uses the YOLOv8 model (Redmon et al., 2016) to predict instance segmentations for individual pipe elements. This model is pre-trained on the instance segmentation images of the COCO data set (Lin et al., 2014). Since the instance segmentation task is more complex compared to the pose estimation task, the largest architecture of YOLOv8 and a data set consisting of 1,030 images are applied. The pre-trained model is fine-tuned with 280 real-world RGB and 750 Unreal Engine images for 200 epochs and optimized using Adam with a learning rate of 10^{-4}. Figure 5 shows predictions for four images.

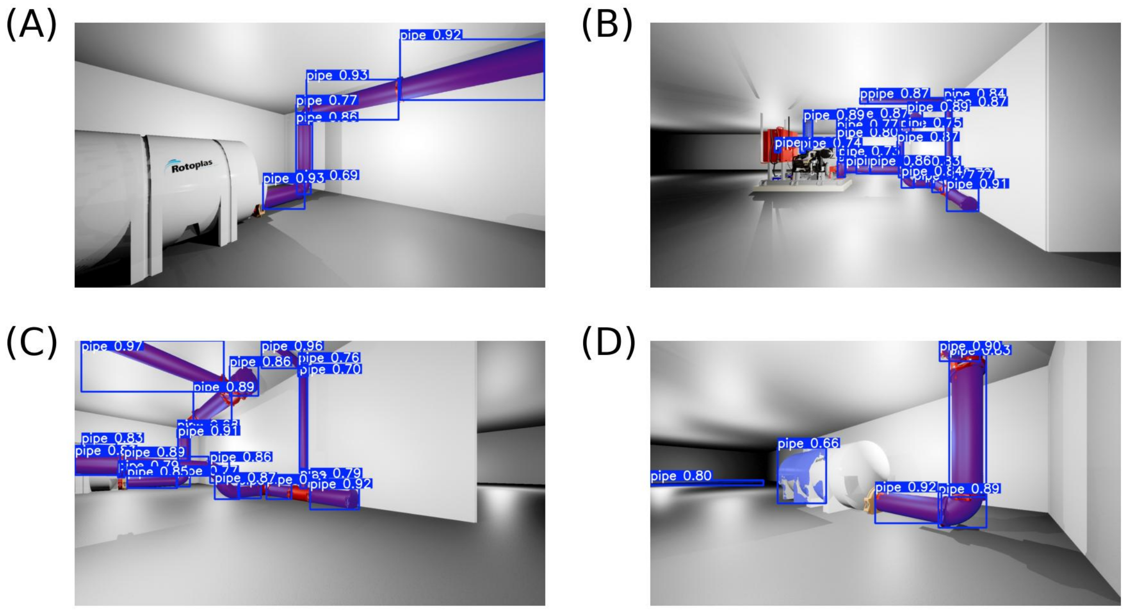

Figure 5. Instance segmentation of pipe objects. Bounding boxes indicate detected objects with class label and class confidence score, while segmented objects are shaded blue.

Similar to the model used in Section 3.2.1, each prediction consists of a bounding box with class label and class confidence score. However, instead of keypoints the model predicts masks for the pipe elements (shaded in blue). Predictions are not perfect (nor are they expected to be) and are characterised by some false negatives, where pipe elements are not detected, e.g., in Figure 5 (**C**), and some false positives, where parts of the background or other objects are falsely predicted as a pipe, e.g., in Figure 5 (**D**). There are also cases for which only parts of the pipe are detected or single pipe elements are split into two or more instances. The aim of the pipeline modules downstream from the instance segmentation are to fill in the missing pipe elements and to discard false detections in the generation of the graph.

The detections of YOLOv8 are cleaned up by removing the outer two pixels of each mask as they oftentimes contain pixels from the background. The masks are then projected into the three-dimensional world view, where they are further cleaned with a combination of DBSCAN (Ester et al., 1996; Schubert et al., 2017) and statistical outlier removal to discard any voxels that belong to the background or objects other than the specific pipe contained within the mask (similar to the process described in Section 3.2.1). In this step, the projection to three-dimensional space through the use of the depth data is instrumental to enable the differentiation between the pipe object and the background/other objects. The pinhole camera model is used for this projection (Hartley and Zisserman, 2003). Finally, other non-pipe objects, such as the pump, also contain pipes that are detected by the model. However, in this case we do not require these pipes for our final relational graph as they are part of the non-pipe object. Thus, the final step of the 'Detection cleanup' module removes masks that are part of the previously matched non-pipe objects as the arrow from the 'Matching' module of the 'Non-pipe objects' branch to the 'Detection cleanup' module of the 'Pipe objects' branch in Figure 2 indicates. Pipes that overlap with any of the non-pipe objects are eliminated from the data set. Figure 6 shows the cleanup process from its starting point in (**A**) to its finished product in (**B**), where (**C**) and (**D**) show are second view zoomed in on the pipe network.

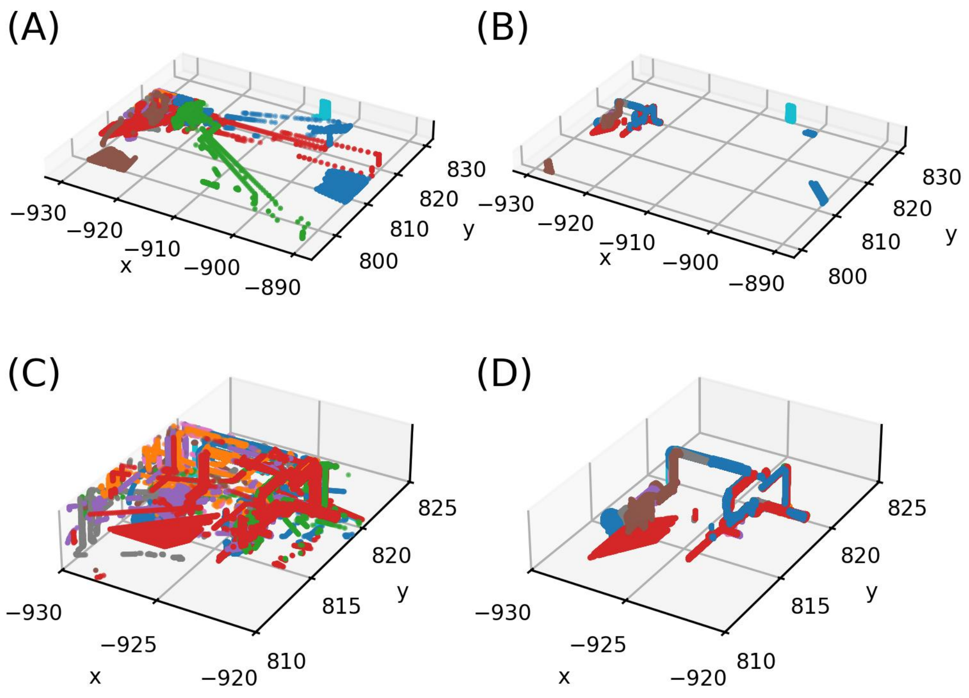

Figure 6. Cleanup of pipe segmentations projected to world view. (**A**) shows the initial segmentations, while (**B**) shows segmentations after cleanup. (**C**) and (**D**) are a zoomed-in version of (**A**) and (**B**). Each color indicates segmentations from an individual image.

After cleaning the detections, the remaining pipe masks are matched across the images. The matching process consists of two steps. First, each mask is projected onto all other images and the intersection with the masks of these images is computed. This is done for all masks and all images. This process is depicted for one image pair in Figure 7, where (**A**) shows the target image on which masks of the source image (**B**) are projected. (**C**) shows this projection with pipe elements of the target image now displayed in gray. (**C**) shows that at least three pipe elements of the source image are projected onto pipe elements of the target image. This will result in a positive intersection area for these pipe element pairs indicating possible matches. Projections are based on the pinhole camera model (Hartley and Zisserman, 2003). Second, to validate that the overlapping masks are actual matches, they are projected into three-dimensional world view, voxels are down-sampled to one cm^3 and DBSCAN (Ester et al., 1996; Schubert et al., 2017) is used to find distinct clusters that are at least 10 cm apart. Each of these clusters is subsequently considered an individual object.

While the endpoints of the non-pipe objects are predicted by YOLO as outlined in Section 3.2.1, the YOLOv8 model used for instance segmentation does not predict keypoints. Thus, an approximation of the endpoints for the pipe objects is required. Pipe objects can be classed into two categories: straight pipes and non-straight pipes, such as bent pipes and T-fittings. We differentiate between these classes by computing a rotated bounding box around the objects and comparing its longest side against its shorter sides. If the ratios of $\frac{\text{length of long side}}{\text{length of short side}}$ are larger than hyperparameter $p\_threshold \in [0, \infty)$, the pipe is classed as straight. If the ratios are equal or smaller than this hyperparameter, the pipe is classed as non-straight. Voxels are binned along the longest side of the bounding box and the mean of all voxels within the first and the last bin are used as the two endpoints of the pipe. Figure 8 (**A**) and (**B**) show wireframes of the hull of two straight

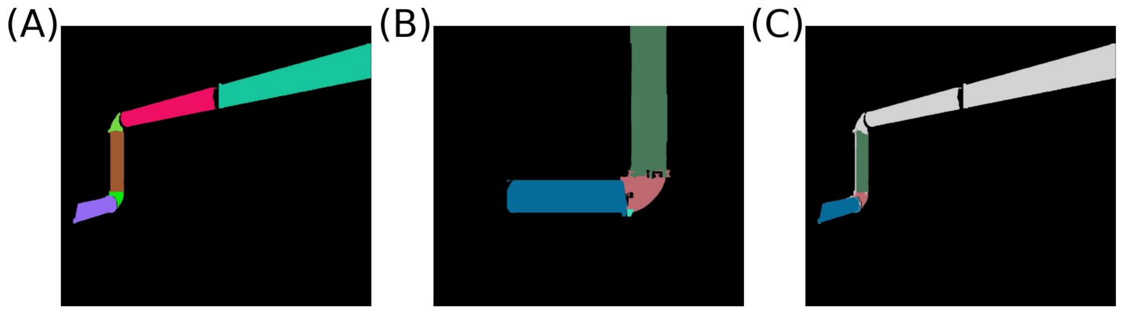

Figure 7. Projection of segmented pipes. (**A**) is the target image on which segmented pipes of the source image (**B**) are projected. (**C**) shows the projected segmentations from (**B**) in color on the segmentations of (**A**) in gray.

pipes in blue and the approximated endpoints as red spheres. For the non-straight pipes, it is generally more challenging to find an appropriate heuristic that gives satisfying results. Thus, we begin by computing the centroid of the object and place it towards the most relevant neighboring object. In detail, we find the object that has the highest number of voxels within $p\_max\_distance \in (0, \infty)$ meters of the hull of the pipe object and use the linear combination $(1 - p\_w) \times centroid + p\_w \times neighbor$ to find the final position of the endpoint. $p\_max\_distance$ is the maximal distance in meters around the pipe objects hull in which other objects are considered and $p\_w \in [0, 1]$ controls how far a centroid is pulled towards the other objects. The pipeline sets $p\_w = 0.3$ which slightly nudges the endpoints towards the objects neighboring in close proximity. This process is executed twice for each centroid yielding two endpoints for each pipe object. Figure 8 (**C**) and (**D**) show two examples of two 90 degree bents. While this method provides satisfying results for most objects, it struggles with pipe objects that have an ambiguous shape (Figure 8 (**E**) and (**F**)) as discussed in detail as part of the limitations in Section 6.

3.2.3 Connection prediction and Graph generation

The processes for non-pipes (Section 3.2.1) and for pipes (Section 3.2.2) yield endpoints for each object that can be used to predict how individual objects are connected and how all objects that form an entire hydraulic system relate to each other. The whole scene can effectively be described as a relational graph in which objects are nodes and object connections are indicated by edges between these nodes. The idea of the graph generation process is to first create a initial graph with probable connections and then refine this graph by defining and enforcing a set of rules. This process is depicted in Figure 9.

The initial graph is generated by computing the pairwise distances between the endpoints of all objects and iteratively connecting objects, i.e., drawing edges between nodes weighted by their respective pairwise distance, starting with the smallest distance. Nodes are iteratively connected until all remaining distances between endpoints are larger than the hyperparameter $graph\_max\_distance \in (0, \infty)$. Two individual objects are restricted to a single connection. An example of an initial graph is shown in Figure 9 (**A**).

The initial graph is then refined by defining a set of rules and enforcing them. This means that edges breaking at least one of the rules are deleted from the graph, beginning with the edge with the largest distance, until no further rule violation exists. For the hydraulic systems considered in this articles, the following rules are enforced:

- **Rule 1**: Pumps and valves are limited to a maximum of two connections, tanks to a maximum of one connection.

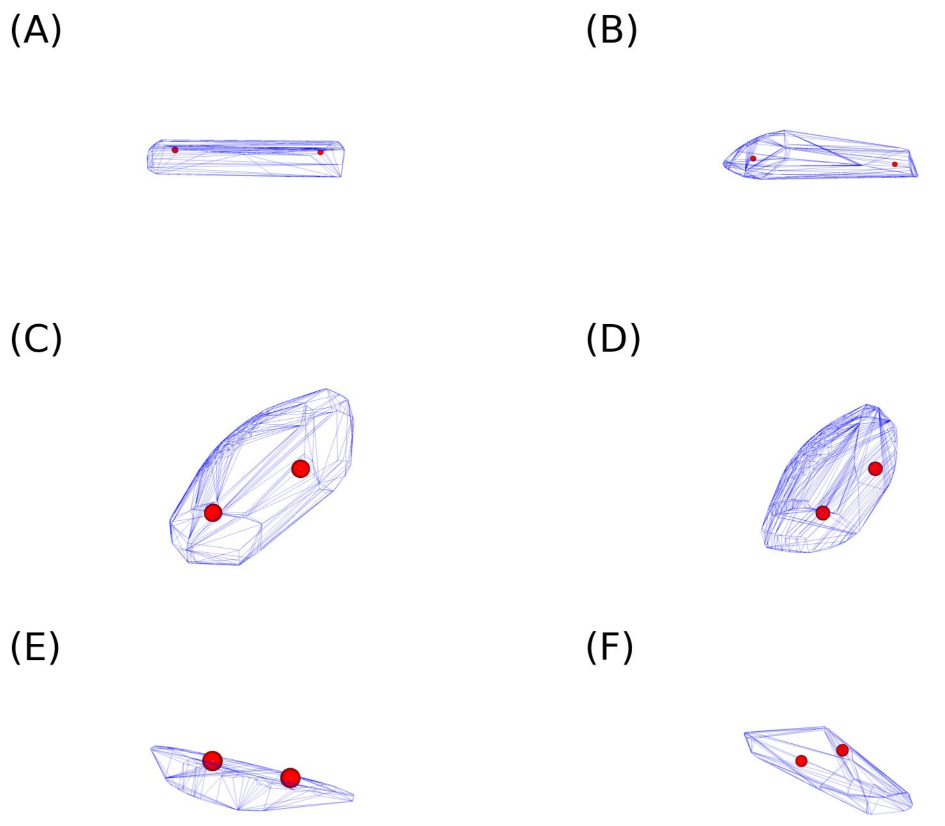

Figure 8. Endpoint approximation of pipe elements. Blue lines are a wireframe representation of the pipe elements while red spheres indicate approximated endpoints. (**A**) and (**B**) show approximations for straight pipe elements, (**C**) and (**D**) for bent pipe elements, and (**E**) and (**F**) for ambiguous pipe elements.

- **Rule 2**: Pipes are limited to a maximum of three connections.
- **Rule 3**: Neighbors of the same non-pipe object cannot be connected.
- **Rule 4**: Single nodes are not allowed.
- **Rule 5**: Cycles are not allowed.
- **Rule 6**: Pipes directly connected to pumps are classed as Reducer/Expander.
- **Rule 7**: Pipes with three neighbors are classed as PipeCrossing.

The graphs in Figure 9 (**B**) through (**H**) display the results after enforcing each of the rules. While the initial graph (**A**) is convoluted and no clear system can be identified, the graph takes a more realistic shape with each step until its final shape is reached at (**F**). After that updates only address the type of the existing nodes. Straight and bend pipes are assigned type 'Pipe', T-fittings are assigned type 'PipeCrossing', and pipes adjacent to pumps are assigned type 'Reducer/Expander'. The diameter of the latter decreases or increase in the direction of the flow to facilitate proper functioning of the connected pump. The final graph is thoroughly discussed in Section 4.1.

While these rules work well for the hydraulic system considered in this paper, they are not appropriate or sufficient for every type of structural critical infrastructure. However, the pipeline makes it straightforward

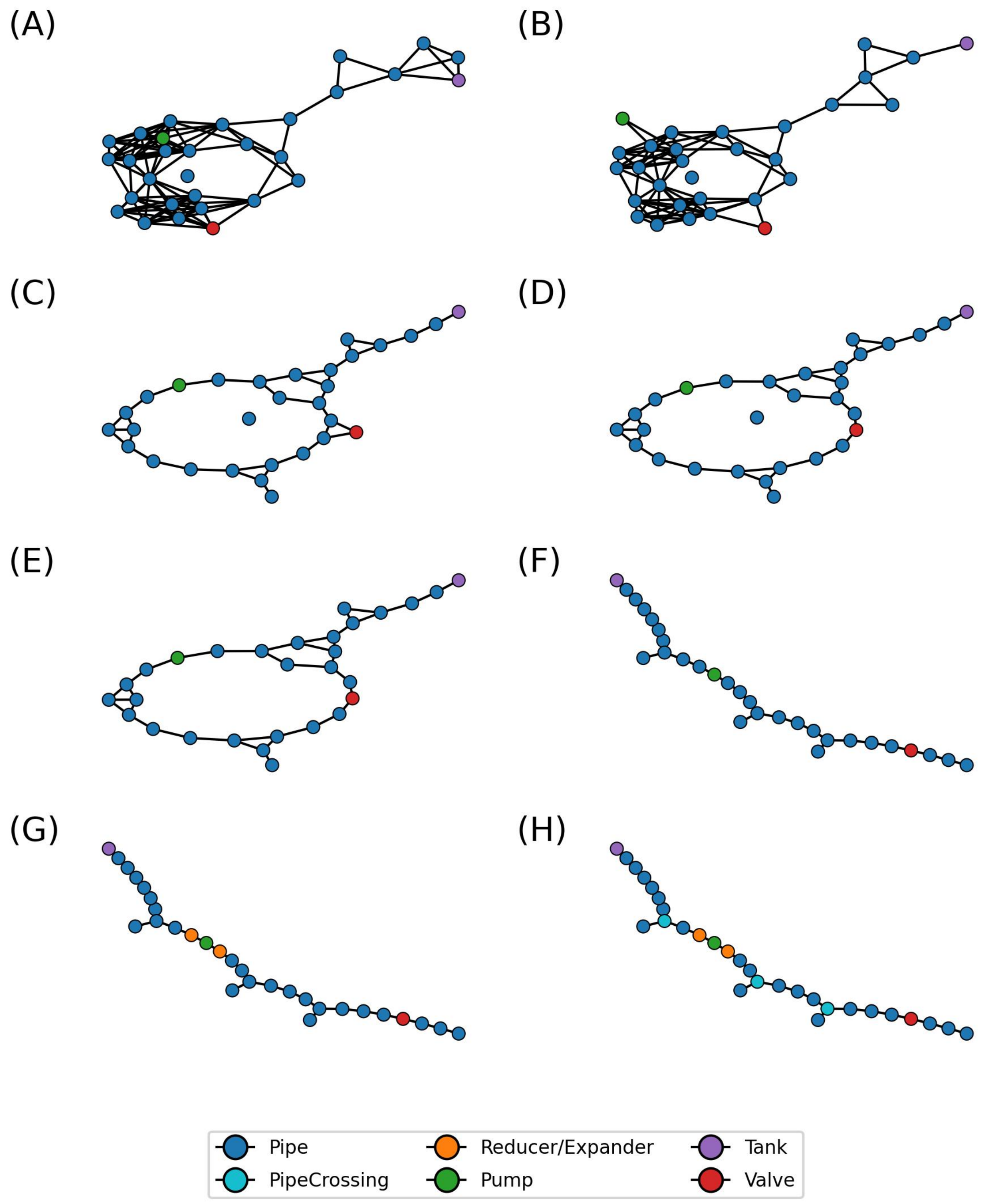

Figure 9. Graph generation. (**A**) shows the initial graph based on distance between endpoints, and (**B**) through (**H**) show graphs enforced by the individual rules discussed in Section 3.2.3.

to add new rules. Each rule is essentially a function that takes the current graph as the input and returns the adjusted graph as the output. The way in which the function alters the graph is defined by the rule itself. For Rule 1, for example, the function gathers all pumps and valves, counts the connections of each and deletes edges beginning with the largest distance for instances where one pump or valve has more than two connections. Adding a new rule requires defining a new function and inserting it at the appropriate position relative to the other rules. An advantage of implementing rules as simple functions is that their behaviour can easily be validated and scrutinized by feeding them a graph that breaks the specific rule and observing

Table 1. Parameters used in the prediction of hydraulic systems 1 and 2.

Parameter	Module	System 1	System 2
np_max_distance	Non-pipe matching	0.75	0.75
p_matching_min_overlap	Pipe matching	0.70	0.70
p_threshold	Pipe endpoint estimation	2.00	2.00
p_max_distance	Graph generation	1.50	1.50
graph_max_distance	Graph generation	1.50	1.50

if the returned graph no longer includes rule violations. This makes the rules transparent and their behavior predictable. Thus, this approach is easy to interpret, particularly, when compared to methods relying on deep learning black boxes as introduced in Section 2.

4 RESULTS

This section presents the results for two hydraulic systems generated by the Unreal Engine 5 (Epic Games, 2022c), as discussed in Section 3.1. We compare the predicted graph against the ground truth in Figures 10 and 11 and highlight differences through three-dimensional plots depicting the individual objects of the systems in Figure 12. Neighboring connected pipe objects are contracted in these figures to make the comparison, both qualitative and through computing the graph edit distance Gao et al. (2010) as a quantitative metric, more convenient. Furthermore, we maintain the position of the pipes to enable downstream simulation. Hence, only the visual appearance of the graphs changes and no information about the actual 3D system is lost. This is shown in Figure 12 where each individual object is retrieved from the graphs. Values of the hyperparameters used to generate these results are shown in Table 1. Section 3.2 describes all hyperparameters in detail and Section 5 discusses how hyperparameters were selected and conducts a sensitivity analysis. Potential implications of these differences are addressed in Section 6.

4.1 Hydraulic system 1

Hydraulic system 1 consists of one tank, one pump, one valve and one sprinkler as well as three T-fittings as displayed in the ground truth graph in Figure 10 (**A**). Figure 10 (**B**) shows the graph predicted by the pipeline detailed in Section 3.2 with errors indicated by red edges. The overall shape of the graphs is almost identical. The three T-fittings (displayed as 'PipeCrossing') are present in both graphs and the graph starts with a tank on one end, followed by the pump and valve and pipe elements in between each of the non-pipe objects. Solely the sprinkler is not included. However, this is not a shortcoming of the pipeline itself but rather the sprinklers in the images of the specific data set are too small to be detected by the fine-tuned YOLOv8 model (Redmon et al., 2016). The graph edit distance Gao et al. (2010) using a cost of 1 for deletion and insertion and a cost of 2 for substitution equates to 4. This indicates a high similarity between the ground truth and the prediction. Figure 12 (**A**) shows some errors in the prediction of the individual pipe objects. Particularly, there are instances where two subsequent pipe objects are falsely predicted as one pipe object: The gold pipe parallel to the y-axis at ground level at (-924, 819) and the olive pipe parallel to the y-axis at ceiling level at (-922, 820), respectively. Both pipes combine a straight pipe and a 90 degree bent in one pipe object. Moreover, the plot shows that the tank (bright green object at (-928, 817)) is approximated in the correct location but is not detected entirely missing a significant part towards the x-axis. Lastly, a pipe object at ceiling level at (-924, 819) is partly missing. Implications of these error and mitigation are discussed in Section 6.

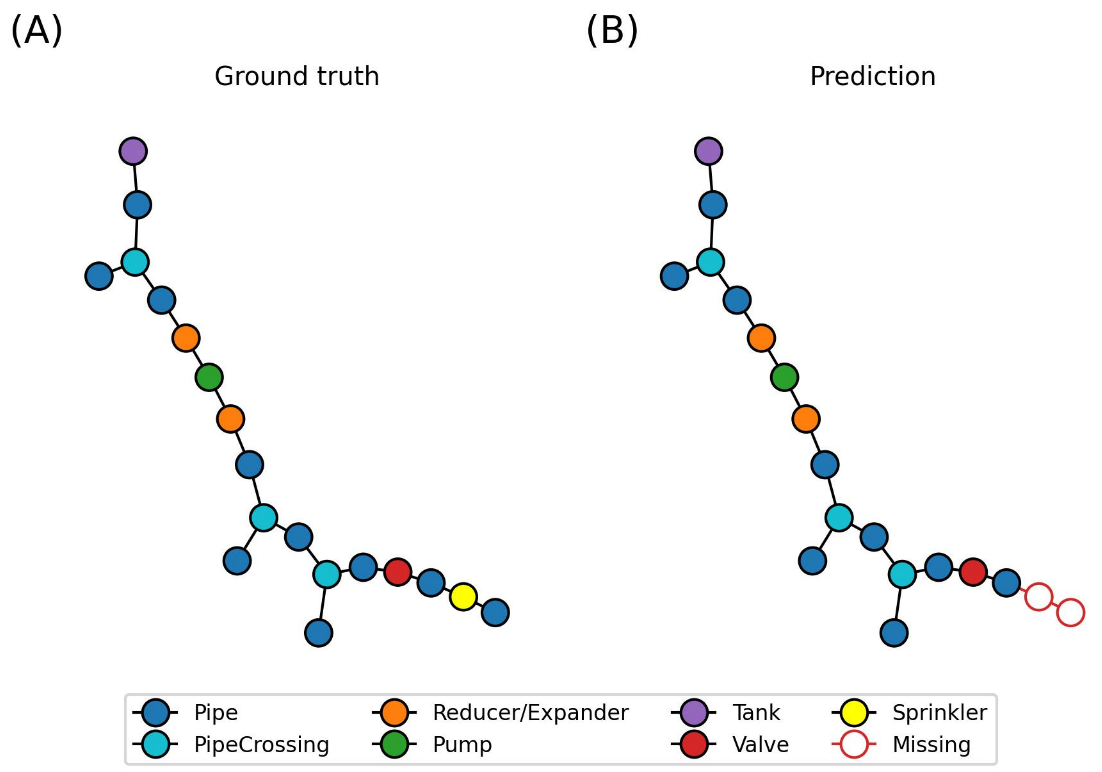

Figure 10. Comparison of ground truth (**A**) to predicted graph (**B**) for hydraulic system 1.

4.2 Hydraulic system 2

Hydraulic system 2 is more complex than system 1. It consists of two tanks, two pumps, three valves and four sprinklers separated by various pipes and T-fittings. The system has an X-shape with the tanks and pumps on one side and the sprinklers on the other as shown in the ground truth in Figure 11 (**A**). The prediction (**B**) of the side with the pumps and tanks only shows a minor difference in the number of predicted pipe elements between the valves and the pumps. This difference is due to the detection of extra pipe elements by YOLOv8 (Redmon et al., 2016), as discussed in Section 3.2.2, and the consequent incorrect matching within the pipeline. In Figure 12, these errors manifest, for example, as the small additional pink pipe element at (0, -6). The opposite side with the sprinkler system of the graph in Figure 11 (**B**) is not connected to the graph due to an undetected thin vertical pipe at (-6, -8) in Figure 12 (**B**). See Figure 1 (**B**) for comparison, where the thin vertical pipe can be seen on the left. Similar to system 1, the sprinklers objects were too small to be detected by YOLOv8 and thus are neither included in the graph nor the three-dimensional representation of the system. The GED Gao et al. (2010) for the entire hydraulic system 2 is 22. Considering that the reason for the missing sprinklers is not the pipeline itself but rather the detection model which is just an input to the pipeline and can be replaced with any prediction model, disregarding the errors caused by the missing sprinklers might be a more accurate metric. In this case the GED drops to 10.

5 SENSITIVITY ANALYSIS AND RUNTIMES

In this section, we want to outline how good values for the hyperparameters listed in Table 1 can be selected before analyzing the sensitivity of the outcome to changes in these hyperparameters.

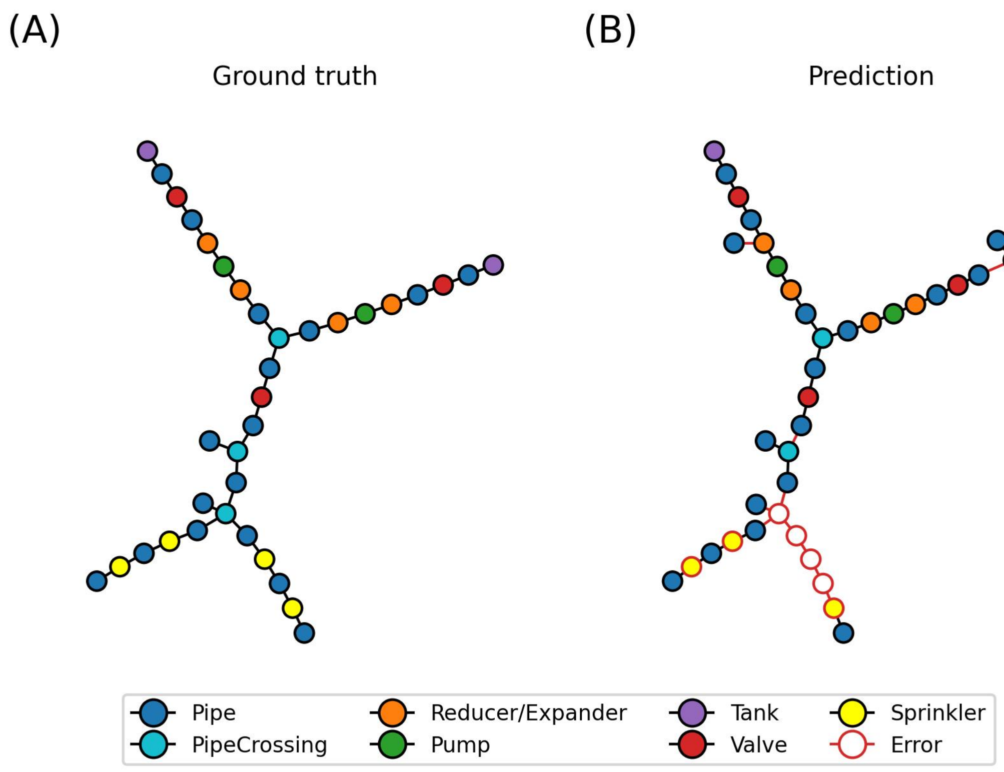

Figure 11. Comparison of ground truth (**A**) to predicted graph (**B**) for hydraulic system 2.

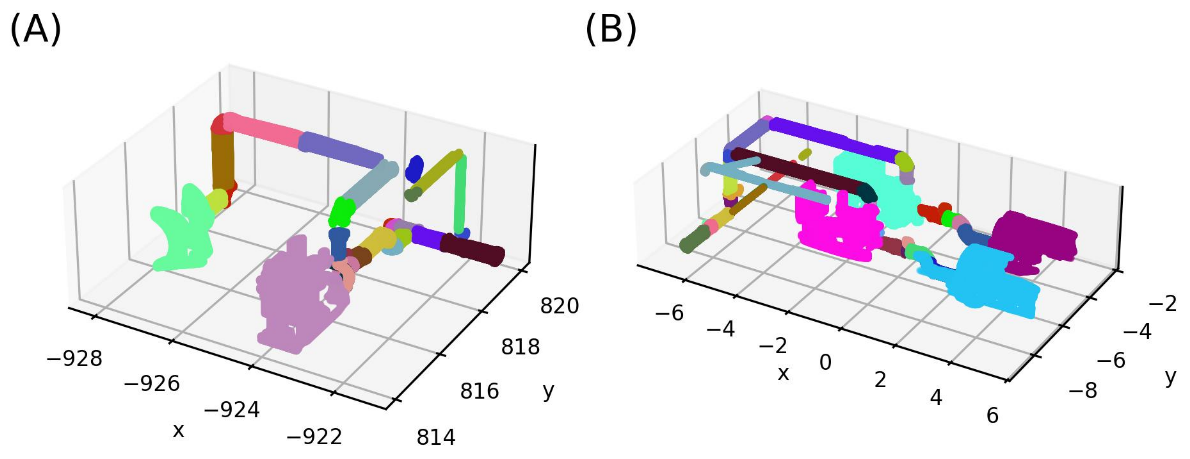

Figure 12. Prediction of individual objects for hydraulic system 1 (**A**) and hydraulic system 2 (**B**).

An advantage of the modular approach presented in Figure 2 is that we can tune the hyperparameters of each module individually. For the none-pipe objects, the only hyperparameter to set is $np\_max\_distance$ which controls the merging of objects detected in multiple images. Particularly, it provides the maximal distance in meters that detections can be apart in three-dimensional space to be combined into one object. Intuitively, this parameter can be set to just below the minimal pairwise distance between all non-pipe objects. Figure 13 shows the final none-pipe objects for different values of $np\_max\_distance$. Evidently, if the parameter is too small, one object might be falsely split into multiple objects as is the case for

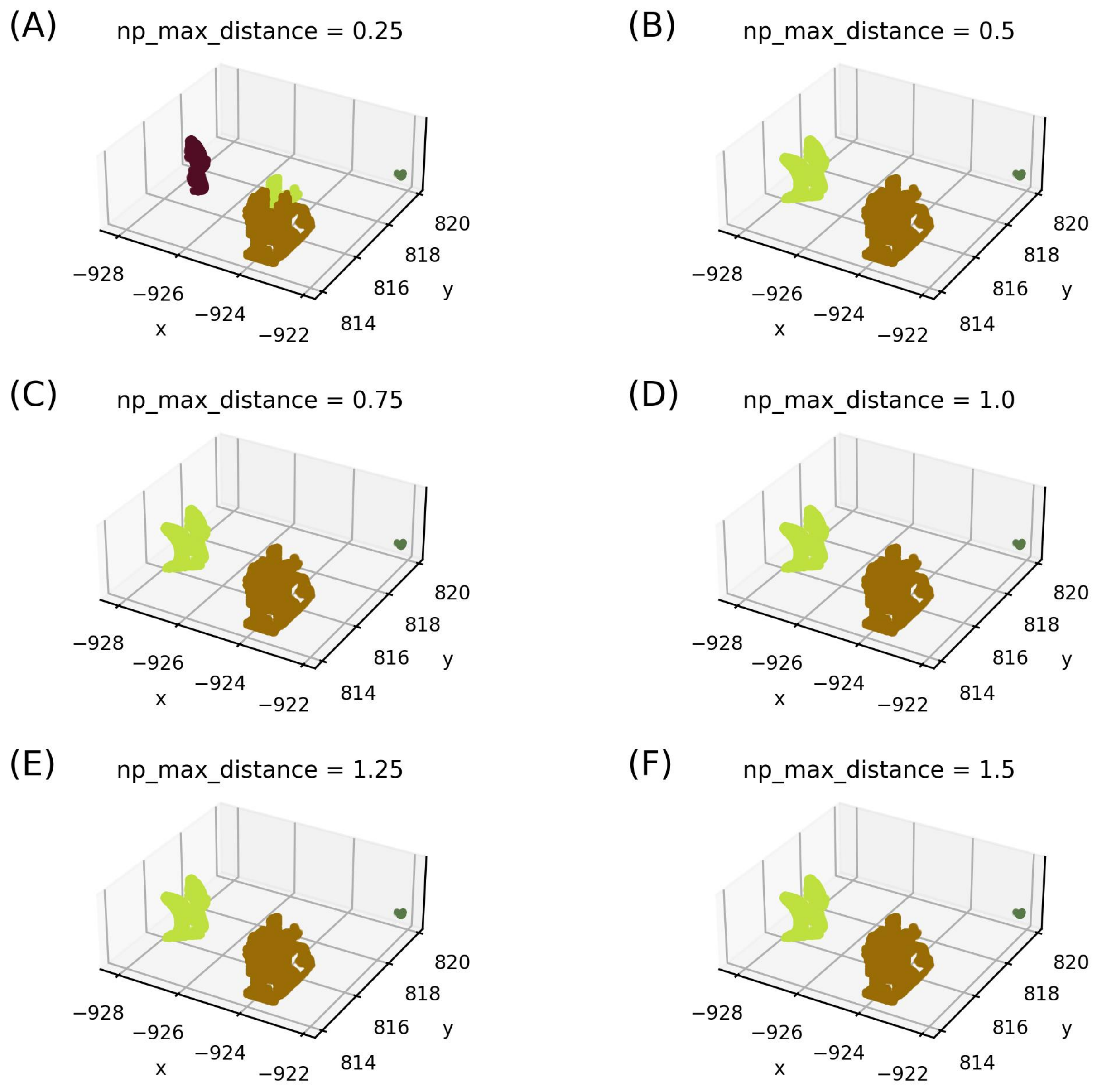

Figure 13. Tuning of hyperparameter $np\_max\_distance$. Orange points indicate hyperparameters used for hydraulic system 1.

$np\_max\_distance = 0.25$ in Figure 13 (**A**). If the parameter is set too large, multiple distinct object might be combined into a single object, however, this is not the case up to the maximum considered distance of 1.5 meters in Figure 13 (**F**). Thus, setting the value to a distance that is just below the minimal pairwise distance between all non-pipe objects is advisable.

The only hyperparameter that requires tuning for processing and matching the pipe objects is $p\_matching\_overlap$ which controls the minimal required overlap of two masks to be matched together as shown in Figure 7. Figure 14 shows results for a range of $p\_matching\_overlap$ values, where each color indicates one pipe object. If the parameter is set too low, many pipe objects will erroneously be matched together. If the parameter is set too high, individual pipe elements might be split into multiple objects. When setting this parameter, it is easiest to plot a range of different values and select the highest value for

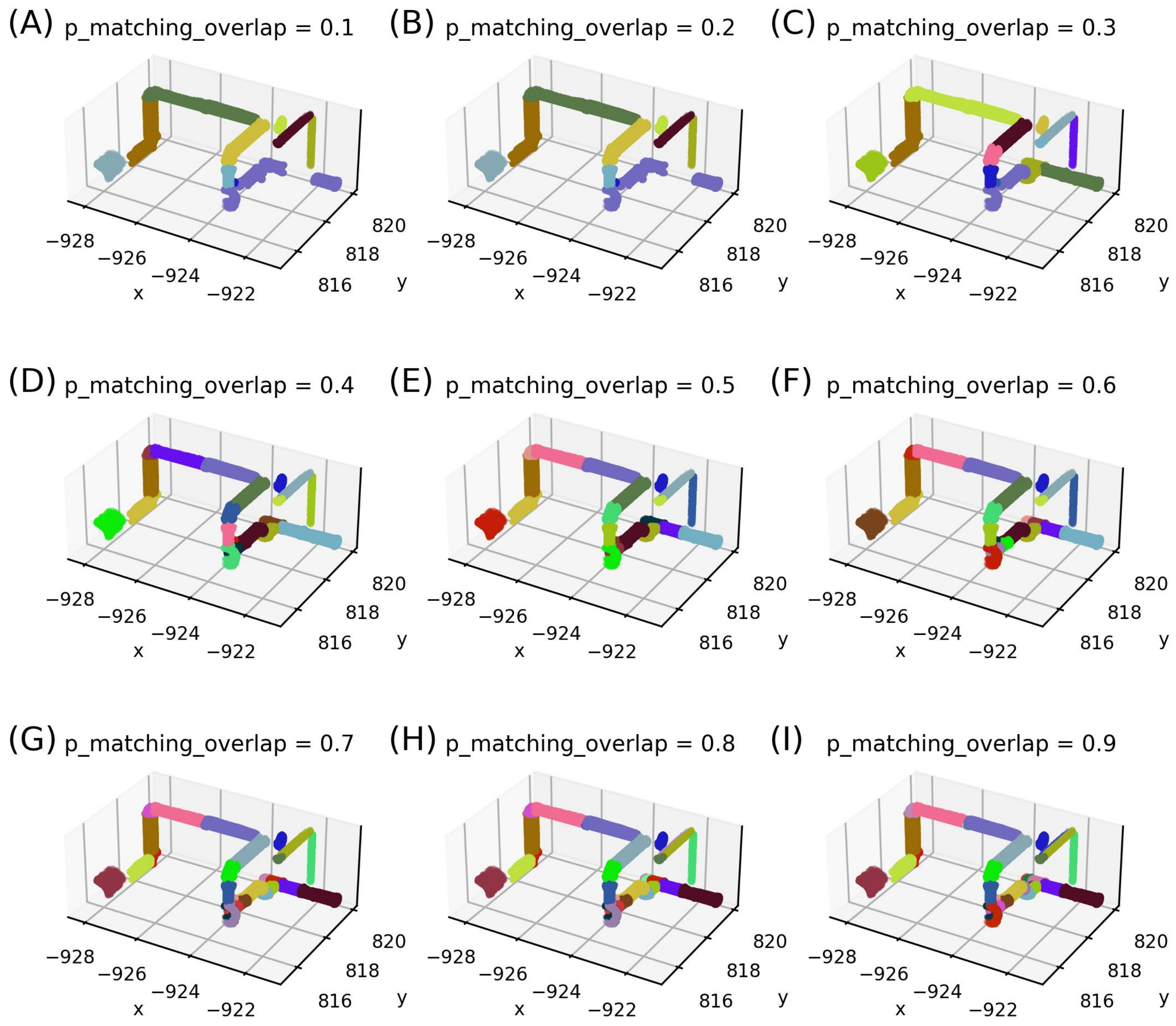

Figure 14. Tuning of hyperparameter $p\_matching\_overlap$. Orange points indicate hyperparameters used for hydraulic system 1.

which individual pipe elements are not split into multiple objects. In this case $p\_matching\_overlap = 0.7$ meets this criteria. This process is feasible as the pipe matching process takes an average of 14.55 seconds to complete (as discussed later in this section) and thus enables rapid experimentation.

When setting the remaining three hyperparameters $p\_threshold$, $p\_max\_distance$ and $graph\_max\_distance$ the same process as before can be applied as approximating the endpoints and generating the graph only takes 1.64 seconds. We chose the graph in Figure 10 as the best performing solution which uses $p\_threshold = 2.0$, $p\_max\_distance = 1.5$ and $graph\_max\_distance = 1.5$. To make this approach of setting the hyperparameters feasible for large scenes, hyperparameter tuning can either be performed on a small area of the scene and then used for the full scene, or it can be performed on one scene and then

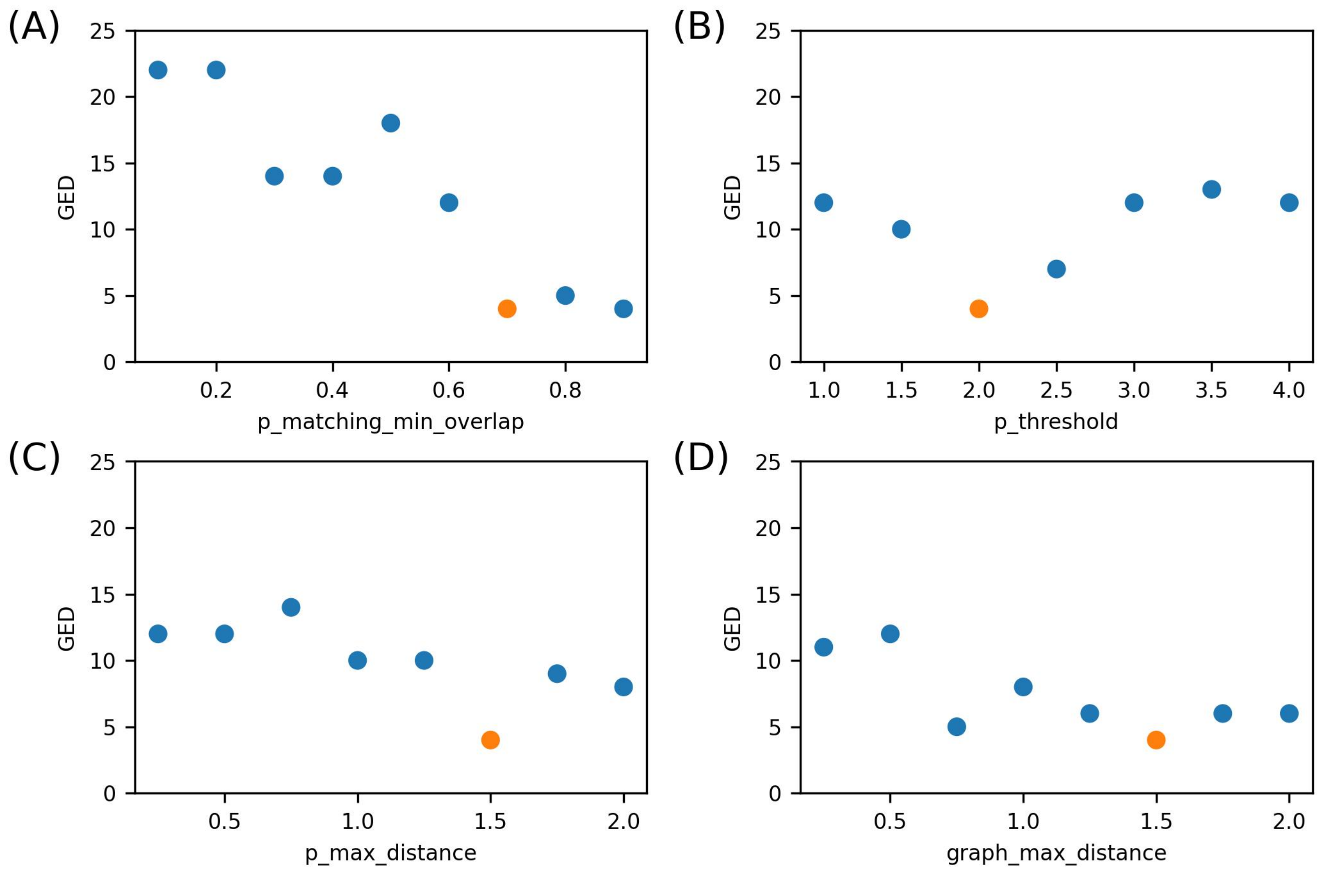

Figure 15. Sensitivity analysis of pipeline hyperparameters. Orange points indicate hyperparameters used for hydraulic system 1.

transferred to another. We opted for the latter option and tuned the hyperparameters on system 1 and applied the pipeline using the same hyperparameters to system 2.

Furthermore, we conducted a sensitivity analysis of the hyperparameter to give further guidance on how changes affect the results. Figure 15 provides the graph edit distance (GED) (Gao et al., 2010) for four hyperparameters over a range of different values, where the orange circles display the hyperparameters used for the results presented in Section 4. The GED is computed with a insertion and deletion cost of 1 and a node substitution cost of 2. We only adjust the value for the displayed hyperparameter while the others remain as given in Table 1. The analysis shows that values above $p\_matching\_overlap = 0.6$ and $graph\_max\_distance = 1.0$ result in similar GEDs, while $p\_threshold$ and $p\_max\_distance$ are more sensitive to changes displaying optimal values at 2.0 and 1.5, respectively. This indicates that it is advisable to set $p\_matching\_overlap$ and $graph\_max\_distance$ towards the upper end of the parameter range while results profit from a more precise tuning of parameters $p\_threshold$ and $p\_max\_distance$ following the process detailed above.

Finally, we present the runtimes of the individual modules of Figure 2 in Table 2. These runtimes correspond to the 29 runs of the sensitivity analysis. The mean time it takes to run the entire pipeline once is 785.68 seconds with a standard deviation of 3.85 seconds. However, results of earlier modules can be saved and reused for subsequent module such that for each module previous modules do not have to be run again. For the hyperparameter tuning of the parameters in the modules Pipes endpoints and

Table 2. Runtime statistics in seconds from applying the pipeline 29 times with different hyperparameters to hydraulic system 1.

Process	n	Median	Mean	Std. deviation	Minimum	Maximum
Preprocessing	29	67.02	67.10	0.52	66.16	68.61
None-pipes object detection	29	3.06	3.09	0.11	3.01	3.60
None-pipes location approximation	29	90.34	90.25	1.55	86.93	92.51
None-pipes matching	29	15.49	15.52	0.14	15.33	15.90
Pipes object detection	29	16.17	16.18	0.13	15.93	16.54
Pipes cleanup	29	14.56	14.54	0.20	14.04	14.84
Pipes matching preprocess	29	553.24	552.02	3.58	544.70	557.84
Pipes matching	29	14.36	14.55	0.79	13.68	16.74
Pipes endpoints	29	1.14	1.05	0.27	0.42	1.48
Graph generation	29	0.58	0.58	0.09	0.41	0.99
Total	29	759.20	758.68	3.85	751.64	766.27

Graph generation this means that previous modules do not have to rerun again and various hyperparameter combinations for $p\_threshold$, $p\_max\_distance$ and $graph\_max\_distance$ can be tested as they only take 1.63 seconds on average to evaluate. This enables rapid experimentation and makes the process of setting the hyperparameters as described above feasible and practical. Experiments were run in parallel on an Intel Xeon Platinum 8260 CPU and 192 GB RAM.

6 DISCUSSION

The results presented in Section 4 show that a combination of data acquisition via photogrammetry, object detection on images and graph generation with an user-defined set of rules is able to generate graphs that are close to the ground truth for two hydraulic systems. While the detection and matching of relevant objects and the prediction of their relations and connections works well, the following points should be noted. Firstly, pipe elbows and T-fittings are treated identical and for both exactly two endpoints are computed, although T-fittings have three connection points in reality. Rule 2, described as part of the graph generation process in Section 3.2.3, limits the number of connections of each pipe to a maximum of three. This enables the method to identify the T-fittings although they are not detected as such initially. Secondly, the location approximation of non-pipe objects as described in Section 3.2.1 uses some crude heuristics to prevent the need of time-consuming labelling required for instance segmentation of the objects. Albeit that the heuristics are crude, they are able to approximate the location of the non-pipe objects sufficiently as shown in Figure 12. The pumps are the most complex objects and are clearly identifiable in both images. Only the tank in Figure 12 (**A**) is hard to identify. However, this is because the side of the tank towards the x-axis is not included in any of the images and thus cannot be detected. Thirdly, we use a simple approximation for the pipe endpoints as presented in Section 3.2.2. The results show however that this approximation is sufficient for connecting most pipe elements. Only the approximation of many small pipe elements in close proximity as, for example, around the valve between the pumps and tanks in system 2 is challenging. For the use in digital twins and simulations, it is likely that the individual pipe elements between two non-pipe objects will be contracted into a single pipe object as it is the case for the graphs in Figure 11 and 12. Hence, the pipe objects between two non-pipe objects are more relevant than how they are connected. While this means that the pipe predicted by our method will be correctly represented in the simulation model, it would be preferable to have a more accurate way of approximating endpoints for smaller pipes that will make the prediction of connections more robust. Section 3.2.2 showed that the approximation performs poorly on pipe elements that are of ambiguous shape, i.e., they are not identifiable as straight or bent pipe elements. A more precise approximation could alleviate this issue. Fourthly, Section 4 showed that some subsequent pipe elements were mistakenly matched together resulting in fewer pipe objects than actually exist. Although technically incorrect, this makes no difference for the functioning of the digital twin if we apply the same logic as for the previous point assuming that the matched pipe elements are actually connected in reality. Consider the case where two subsequent pipe elements are detected as two distinct objects, and the case where the same two pipe elements are falsely matched into a single object by the pipeline. When the 3D locations of the pipes that are stored in the graph are transferred into three-dimensional representation, the model will be the same. The only difference is that for the first case the pipes are connected when transferring the objects into the 3D representation while for the second case the pipes are already connected before the transfer. The model on which the downstream simulations are based will be identical and so should the simulation accuracy. Lastly, we want to point out that the performance of the proposed pipeline of this article is heavily influenced by the quality of the images and the models used for object detection and instance segmentation. Low-resolution images might not be accurate enough to accurately predict all relevant objects and segmentation masks might be ambiguous resulting in the inclusion of other objects or portions of the background. It can also make training more challenging for the model and prevent it from learning the most important features. Errors made by the prediction model, e.g., due to shortcomings in the model itself or poor image quality, are challenging to be reversed downstream in the pipeline. While the multiple perspectives of the images provide multiple opportunities to detect each object, some objects, such as the sprinklers or the thin vertical pipe in system 2, were not detected by the fine-tuned YOLOv8 model (Redmon et al., 2016) and thus were not included in the final graphs. To mitigate this issue, (a) the detection model could be improved with more training data focusing on areas with poor performance, (b) a different model could be explored, or (c) more images of the hydraulic system could be included that show the challenging objects more clearly.

Considering the limitations of the approach presented in this article, future work should address the approximation of the location of the non-pipe objects and of the pipe endpoints. One possible solution could be the skeletonization of the pipe elements as discussed in Alex and Stoppe (2025) and Meyer et al. (2023). Furthermore, although the object detection achieved good results overall, there were issues in detecting smaller objects, such as sprinklers and thin pipes. It could be investigated if this can be improved by collecting more images where small objects are depicted prominently, increasing image resolution, or improving the model, e.g., by exploring the use of vision transformers (Yuan et al., 2021). Updating the model would not affect the pipeline in any way: The detection models are inputs to the pipeline and can be replaced with any preferred model. While the differentiation between straight pipes, elbows and T-fittings worked well in the example test environments, adding them as different object classes in the 'Object detection and and instance prediction' module could make the results more robust as it would make the use of crude heuristics redundant. The models could also be extended to detect more objects than pipes, pumps, tanks, and valves. This would pave the way for applying the pipeline to environments other than hydraulic systems. The latter would at the same time help to validate the method further. While the pipeline shows promising results on generated scenes that mimic real infrastructure, it remains a prototype until it can be validated on a real-world example. Thus, the main objective for the future remains testing and validating the pipeline on an actual critical infrastructure.

7 CONCLUSION

This article proposes a prototypical graph generation pipeline that shows the relations between relevant predetermined objects that are instrumental to the type of critical infrastructure in question. For example, this study investigates hydraulic systems for which pipes, reducers, expanders, tanks, valves and pumps were identified as relevant. The pipeline is based on a combination of photogrammetry, deep learning for object detection and instance segmentation, and heuristics for inferring relations between objects. The use of these methods has the advantages of being cost-efficient (both hardware for data collection and computation) and accessible. The user-defined set of rules for the 'Graph generation' module makes it easy to tailor the pipeline to specific use cases and transfer it from one problem to the next, while its transparency and explainability are vital for the high stakes decision-making required for critical infrastructure.

CONFLICT OF INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

AUTHOR CONTRIBUTIONS

MD: Conceptualization, Formal analysis, Investigation, Methodology, Visualization, Writing – original draft, Writing – review & editing; YT: Conceptualization, Data curation, Formal analysis, Writing – original draft, Writing – review & editing.

FUNDING

This research was funded by the project Automated Model Generation (AMG) within the German Aerospace Center (DLR).

ACKNOWLEDGMENTS

We would like to thank our colleagues Sebastian Sporrer, Lena Schreiber and Norman Wilhelms for their generous assistance, the valuable discussions and their helpful feedback. We further would like to extend our gratitude to our colleagues Marius Stürmer and Kai Franke for the conceptualization and implementation of the test environments in the Unreal Engine. This research would not have been possible without their meticulous work.

REFERENCES

Alcaraz, C. and Zeadally, S. (2015). Critical infrastructure protection: Requirements and challenges for the 21st century. *International Journal of Critical Infrastructure Protection* 8, 53–66. doi:10.1016/j.ijcip.2014.12.002

Alex, A. and Stoppe, J. (2025). Pipe reconstruction from point cloud data. In *Proceedings of the European Workshop on Maritime Systems Resilience and Security (MARESEC)* (University of Rostock), 1–6. doi:10.5281/zenodo.17120162

Borkowski, A. S. (2023). Evolution of BIM: Epistemology, genesis and division into periods. *Journal of Information Technology in Construction* 28. doi:10.36680/j.itcon.2023.034

Borrmann, A., König, M., Koch, C., and Beetz, J. (2018). Building information modeling: Why? What? How? In *Building Information Modeling: Technology Foundations and Industry Practice* (Springer). 1–24. doi:10.1007/978-3-319-92862-3_1

Buhrmester, V., Münch, D., and Arens, M. (2021). Analysis of explainers of black box deep neural networks for computer vision: A survey. *Machine Learning and Knowledge Extraction* 3, 966–989. doi:10.3390/make3040048

Cheng, L., Wei, Z., Sun, M., Xin, S., Sharf, A., Li, Y., et al. (2020). DeepPipes: Learning 3D pipelines reconstruction from point clouds. *Graphical Models* 111, 101079. doi:10.1016/j.gmod.2020.101079

Codex Labs (2022). *Colosseum: A high-fidelity simulator for autonomous vehicles (Fork of AirSim)*. Codex Labs LLC. Version 2.1.0. Available at `https://github.com/CodexLabsLLC/Colosseum`. Accessed: 2025-11-13

Cong, Y., Yang, M. Y., and Rosenhahn, B. (2023). RelTR: Relation transformer for scene graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 11169–11183. doi:10.1109/TPAMI.2023.3268066

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*. 1–22

Dwivedi, R., Dave, D., Naik, H., Singhal, S., Omer, R., Patel, P., et al. (2023). Explainable AI (XAI): Core ideas, techniques, and solutions. *ACM Computing Surveys* 55, 1–33. doi:10.1145/3561048

Epic Games (2022a). *Lumen Global Illumination and Reflections*. Epic Games, Inc. Available at `https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine`. Accessed: 2026-01-23

Epic Games (2022b). *Nanite Virtulaized Geometry*. Epic Games, Inc. Available at `https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-in-unreal-engine`. Accessed: 2026-01-23

Epic Games (2022c). *Unreal Engine 5*. Epic Games, Inc. Version 5.1. Available at `https://www.unrealengine.com/`. Accessed: 2025-11-13

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)* (ACM), 226–231

Franke, K., Stürmer, J. M., and Koch, T. (2023). Automated simulation and virtual reality coupling for interactive digital twins. In *Proceedings of the Winter Simulation Conference (WSC)* (IEEE), 2615–2626. doi:10.1109/WSC60868.2023.10407185

Gao, X., Xiao, B., Tao, D., and Li, X. (2010). A survey of graph edit distance. *Pattern Analysis and applications* 13, 113–129

Ghaffarianhoseini, A., Tookey, J., Ghaffarianhoseini, A., Naismith, N., Azhar, S., Efimova, O., et al. (2017). Building information modelling (BIM) uptake: Clear benefits, understanding its implementation, risks and challenges. *Renewable and Sustainable Energy Reviews* 75, 1046–1053. doi:10.1016/j.rser.2016.11.083

Goesele, M., Curless, B., and Seitz, S. (2006). Multi-View Stereo revisited. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE/CVF), vol. 2, 2402–2409. doi:10.1109/CVPR.2006.199

Grieves, M. (2014). Digital twin: Manufacturing excellence through virtual factory replication. *Whitepaper* 1, 1–7

Hart, L., Knoblach, S., and Möser, M. (2023). Automated pipeline reconstruction using deep learning & instance segmentation. *ISPRS Open Journal of Photogrammetry and Remote Sensing* 9, 100043. doi:10.1016/j.ophoto.2023.100043

Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision* (Cambridge: Cambridge University Press), 2nd edn.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)* (IEEE/CVF), 2961–2969. doi:10.1109/ICCV.2017.322

Jones, D., Snider, C., Nassehi, A., Yon, J., and Hicks, B. (2020). Characterising the digital twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology* 29, 36–52. doi:10.1016/j.cirpj.2020.02.002

Kawashima, K., Kanai, S., and Date, H. (2014). As-built modeling of piping system from terrestrial laser-scanned point clouds using normal-based region growing. *Journal of Computational Design and Engineering* 1, 13–26. doi:10.7315/JCDE.2014.002

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., et al. (2023). Segment anything. In *Proceedings of the International Conference on Computer Vision (ICCV)* (IEEE/CVF), 4015–4026. doi:10.1109/ICCV51070.2023.00371

Lampropoulos, G., Larrucea, X., and Colomo-Palacios, R. (2024). Digital twins in critical infrastructure. *Information* 15, 454. doi:10.3390/info15080454

Li, H., Zhu, G., Zhang, L., Jiang, Y., Dang, Y., Hou, H., et al. (2024). Scene graph generation: A comprehensive survey. *Neurocomputing* 566, 127052. doi:10.1016/j.neucom.2023.127052

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., et al. (2014). Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)* (Springer), 740–755. doi:10.1007/978-3-319-10602-1_48

Lu, Q., Xie, X., Parlikad, A. K., Schooling, J. M., and Konstantinou, E. (2022). Moving from building information models to digital twins for operation and maintenance. *Proceedings of the Institution of Civil Engineers-Smart Infrastructure and Construction* 174, 46–56. doi:10.1680/jsmic.19.00011

Lv, C., Qi, M., Li, X., Yang, Z., and Ma, H. (2024). SGFormer: Semantic graph transformer for point cloud-based 3D scene graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (AAAI), vol. 38, 4035–4043. doi:10.1609/aaai.v38i5.28197

Meyer, L., Gilson, A., Scholz, O., and Stamminger, M. (2023). CherryPicker: Semantic skeletonization and topological reconstruction of cherry trees. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (IEEE/CVF), 6244–6253. doi:10.1109/CVPRW59228.2023.00664

Moon, D., Chung, S., Kwon, S., Seo, J., and Shin, J. (2019). Comparison and utilization of point cloud generated from photogrammetry and laser scanning: 3D world model for smart heavy equipment planning. *Automation in Construction* 98, 322–331. doi:10.1016/j.autcon.2018.07.020

Osei-Kyei, R., Tam, V., Ma, M., and Mashiri, F. (2021). Critical review of the threats affecting the building of critical infrastructure resilience. *International Journal of Disaster Risk Reduction* 60, 102316. doi:10.1016/j.ijdrr.2021.102316

Peng, J., Liu, Q., Yue, L., Zhang, Z., Zhang, K., and Sha, Y. (2024). Towards few-shot self-explaining graph neural networks. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)* (Springer), 109–126. doi:10.1007/978-3-031-70365-2_7

Qiu, R., Zhou, Q.-Y., and Neumann, U. (2014). Pipe-run extraction and reconstruction from point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)* (Springer), 17–30. doi:10.1007/978-3-319-10578-9_2

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE/CVF), 779–788. doi:10.1109/CVPR.2016.91

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 206–215. doi:10.1038/s42256-019-0048-x

Şahin, E., Arslan, N. N., and Özdemir, D. (2025). Unlocking the black box: An in-depth review on interpretability, explainability, and reliability in deep learning. *Neural Computing and Applications* 37, 859–965. doi:10.1007/s00521-024-10437-2

Schönberger, J. L. and Frahm, J.-M. (2016). Structure-from-Motion revisited. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE/CVF), 4104–4113. doi:10.1109/CVPR.2016.445

Schreiber, L. R., Tarant, Y. E., and Franke, K. (2024). Synthetic training data bias in instance segmentation algorithms. In *Artificial Intelligence for Security and Defence Applications II* (SPIE), vol. 13206, 198–208. doi:10.1117/12.3030822

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., and Xu, X. (2017). DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)* 42, 1–21. doi:10.1145/3068335

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2017). AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Proceedings of the Field and Service Robotics (FSR)* (Springer), 621–635. doi:10.1007/978-3-319-67361-5_40

Shit, S., Koner, R., Wittmann, B., Paetzold, J., Ezhov, I., Li, H., et al. (2022). Relationformer: A unified framework for image-to-graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)* (Springer), 422–439. doi:10.1007/978-3-031-19836-6_24

Sousa, B., Arieiro, M., Pereira, V., Correia, J., Lourenço, N., and Cruz, T. (2021). ELEGANT: Security of critical infrastructures with digital twins. *IEEE Access* 9, 107574–107588. doi:10.1109/ACCESS.2021.3100708

Vidas, S., Moghadam, P., and Bosse, M. (2013). 3D thermal mapping of building interiors using an RGB-D and thermal camera. In *Proceedings of the International Conference on Robotics and Automation (ICRA)* (IEEE), 2311–2318. doi:10.1109/ICRA.2013.6630890

Wald, J., Avetisyan, A., Navab, N., Tombari, F., and Nießner, M. (2019). Rio: 3D object instance re-localization in changing indoor environments. In *Proceedings of the International Conference on Computer Vision (ICCV)* (IEEE/CVF), 7658–7667. doi:10.1109/ICCV.2019.00775

Wald, J., Dhamo, H., Navab, N., and Tombari, F. (2020). Learning 3D semantic scene graphs from 3D indoor reconstructions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE/CVF), 3961–3970. doi:10.1109/CVPR42600.2020.00402

Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., and Zhu, J. (2019). Explainable AI: A brief survey on history, research areas, approaches and challenges. In *Proceedings of the CFF International Conference on Natural Language Processing and Chinese Computing (NLPCC)* (Springer), 563–574. doi:10.1007/978-3-030-32236-6_51

Yang, J., Lu, J., Lee, S., Batra, D., and Parikh, D. (2018). Graph R-CNN for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)* (Springer), 670–685. doi:10.1007/978-3-030-01246-5_41

Yeo, Q. X., Li, Y., and Lee, G. H. (2025). Statistical confidence rescoring for robust 3D scene graph generation from multi-view images. In *Proceedings of the International Conference on Computer Vision (ICCV)* (IEEE/CVF), 24999–25008

Yuan, L., Chen, Y., Wang, T., Yu, W., Shi, Y., Jiang, Z.-H., et al. (2021). Tokens-to-token ViT: Training vision transformers from scratch on ImageNet. In *Proceedings of the International Conference on Computer Vision (ICCV)* (IEEE/CVF), 558–567. doi:10.1109/ICCV48922.2021.00060

Yusta, J. M., Correa, G. J., and Lacal-Arántegui, R. (2011). Methodologies and applications for critical infrastructure protection: State-of-the-art. *Energy Policy* 39, 6100–6119. doi:10.1016/j.enpol.2011.07.010

Zhang, Z., Liu, Q., Wang, H., Lu, C., and Lee, C. (2022). ProtGNN: Towards self-explaining graph neural networks. In *Proceedings of the Conference on Artificial Intelligence* (AAAI), vol. 36, 9127–9135. doi:10.1609/aaai.v36i8.20898

Zhao, H., Xia, R., Chen, Y., Zhang, T., Fu, D., and Zhang, T. (2025). A multi-camera vision online measurement method for complex tube parameters based on regional pre-segmentation. *Journal of Manufacturing Processes* 149, 502–517. doi:10.1016/j.jmapro.2025.05.083

Zhou, Y. and Tuzel, O. (2018). VoxelNet: End-to-end learning for point cloud based 3D object detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE/CVF), 4490–4499. doi:10.1109/CVPR.2018.00472

Zimmer, W., Ercelik, E., Zhou, X., Ortiz, X. J. D., and Knoll, A. (2022). A survey of robust 3D object detection methods in point clouds. *arXiv:2204.00106 [Preprint]*. Available at `https://doi.org/10.48550/arXiv.2204.00106` (Accessed November 13, 2025)