Dynamic Scene Reconstruction for Planning Environments

Albaraa Ammar Othman[1,2], Prabhneet Singh[1,2], Emanuele De Pellegrin[2], Maria Koskinopoulou[3], Ronald P. A. Petrick[1]

[1]Department of Computer Science, Heriot-Watt University, Edinburgh, Scotland, UK
[2]School of Informatics, University of Edinburgh, Edinburgh, Scotland, UK
[3]School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh, Scotland, UK
ao2000@hw.ac.uk, ps115@hw.ac.uk, edepell@ed.ac.uk, M.Koskinopoulou@hw.ac.uk, R.Petrick@hw.ac.uk

Abstract

Dynamic Scene Reconstruction is the complex task of creating a digital counterpart to a real-world environment. However, recent advancements in Vision Language Models (VLMs) have demonstrated their emerging ability to understand and recognise aspects of an environment. While the deployment of VLMs in real-world domains remains an open challenge, they provide a method for generating semantic descriptors of elements in a scene. This paper introduces a framework which segments scenes using recent foundation models and then transforms the visual representation into symbolic attributes populated by RGB-D camera data and VLM-assigned descriptors. These attributes are suitable for constructing a partial PDDL representation for planning and plan visualisation. We describe the approach and evaluate the framework using a series of construction tasks, demonstrating the pipeline from initial scenes to plan visualisation.

Introduction

Dynamic Scene Reconstruction is the process of creating a simulated version of a real world domain. Such environments can be used for a range of tasks involving autonomous agents such as decision making, training, teleoperation, navigation, and acting as a digital twin (Bavelos et al. 2025; Gu et al. 2023; Ni et al. 2025). Reconstructing a scene while attaining a strong semantic understanding of its properties and dynamics is a challenging task.

The task of identifying objects and object properties has a long history in areas like vision, sensing and robotics, with both manual and automated approaches being proposed (Redmon et al. 2016; Chen et al. 2019; Schneider et al. 2009; Vidal et al. 2023). With the recent establishment of Generative AI tools, new methods have been introduced to aid in this process. Furthermore with game engines becoming more prevalent in the field of robotics as a simulation tool, the need for a process that reconstructs scenes for such systems is seen as an important area of research.

In this paper, we consider the problem of automatically identifying objects and their spatial properties in a given scene in order to build a symbolic representation that supports automated planning and plan visualisation. Automated planning focusses on finding a plan to reach a required goal state from a prespecified initial state. Automated planning has previously been deployed for a variety of different tasks such as robotics, logistics and manufacturing, although the success of these techniques is often limited by the correctness of the underlying domain model.

(a) (b)

Figure 1: Example environment.

In this work, an RGB-D camera is used to capture an environment and segmentation model (SAM 2) (Ravi et al. 2024) is used to extract objects from the video. Spatial properties for each object are determined from the camera data and are supplemented element descriptions using a Vision Language Model (VLM). This representation is then converted into a PDDL domain and problem model (McDermott et al. 1998), capable for use with classical planning, providing a pipeline for converting continuous visual data into discrete symbolic data supporting planning model concepts. Given the induced planning model we reconstruct the scene digitally in the Unity Game Engine (Unity Technologies 2025) using the Planning Domain Simulation (PDSim) system (Pellegrin and Petrick 2024), an open source planning domain simulation extension for Unity. To test the validity of the reconstruction, plans are generated and executed in the virtual environment. We mimic a real-world manufacturing environment using LEGO Bricks (Figure 1a) and use these examples to test the approach.

The rest of this paper is organised as follows. First, we review related work for scene generation, automated planning and plan visualisation. We then present our pipeline for extracting objects and properties from an RGB-D video, which is used to induce a partial PDDL planning model. This model is then used to dynamically reconstruct scenes in PDSim, which we then evaluate with a set of experiments. Lastly, we discuss our results and future work.

Background and Related Work

Scene Reconstruction Both manual and automated approaches to scene reconstruction exist. For example, manual 3D modelling may be practical for a scene with few items although as scenes become busier, manual approaches become unsustainable. Automated approaches exist which employ the use of high-end expensive 3D scanners or cheaper but less accurate phone-based applications. Typically these applications use algorithms which involve a blend of Structure from Motion (SfM) and Multi-View Stereo (MVS), which takes overlapping images and converts them into dense points clouds, i.e., an array of points each with a coordinate to recreate a scene (Verykokou and Ioannidis 2023). These approaches create a singular high visual fidelity mesh to represent a scene where all objects act as a singular background (Downs et al. 2022). An emerging technology, 3D Gaussian Splatting (3DGS) is a high-quality and real time rendering technique used to build 3D representations of objects and environments (Kerbl et al. 2023). Scenes are modelled as collections of 3D Gaussian ellipsoids, with each ellipsoid's colour, shape, position, and opacity optimised to reconstruct the scene across multiple camera viewpoints. Environments typically take a graphical structure and are commonly represented with Scene Graphs (SG) consisting of nodes and edges, where each node represents an object and each edge a relationship between nodes (Rosinol et al. 2020). These graphs allow for rich semantic descriptions as they can represent object properties, descriptor values (e.g.,colour, material), and parent-child relationships (e.g., spatial relationships between objects) (Gu et al. 2023). In dynamic or unknown environments where scenes change in real time, the scene graphs can be updated with network or foundation models (Rosinol et al. 2020; Gu et al. 2023) that give rise to Dynamic Scene Graphs (DSGs).

Object Identification Object identification is the process of providing labels to objects in a scene, a task which has a long history in vision and sensing research (Dalal and Triggs 2005; Girshick 2015; Redmon et al. 2016). In small datasets, approaches that assign similarity scores between objects can be used, such as feature matching which identifies matching points between different images and can be undertaken using techniques like Scale-Invariant Feature Transform (SIFT), Speed up Robust Feature (SURF), and Oriented Fast and Rotated Brief (ORB) (Karami, Prasad, and Shehata 2017). For larger datasets, Machine learning (ML) techniques such as neural networks or clustering algorithms can be used to sort objects into distinct classes. ML approaches are trained on labelled data which allows them to estimate classes for new data but are generally successful in environments where all objects are previously known.

The vast landscape of generative tools such as Large Language Models (LLMs) and their visual counterpart, Vision Language Models (VLMs), introduce a new method for labelling objects in unknown environments. These models are trained on vast amounts of unlabelled, unsupervised data (Naik, Naik, and Naik 2024; Ferrara 2023; Zhiheng, Rui, and Tao 2023) offering the possibility of exceptional performance on the labelling task as the model can generalise and infer details about previously unknown or obscured objects (Raschka 2024). LLMs and VLMs also have the inherent ability to interpret and generate human-coherent content (Berrios et al. 2023), with LLMs specialising in text and VLMs able to perceive and generate images or video frames.

In busy scenes, the labelling tool can fail to identify all objects, so a method to extract meaningful regions, such as segmentation can simplify the labelling task (Othman, De Pellegrin, and Petrick 2026). In simple cases, segmentation can be completed by extracting colour and depth regions, or areas within a detected edge. For a more sophisticated approach in unknown environments, foundation models which specialise in segmentation such as Meta's Segment Anything Model (SAM) (Ravi et al. 2024) can be used.

Game Engines Simulators are extensively used in the field of robotics as they allow for safe, cost effective and scalable environments for robot development (Collins et al. 2021; Kargar et al. 2024). Recently tools such as game engines have become a promising approach due to their support for high-quality physics, rendering and integrability (Collins et al. 2021; Kargar et al. 2024; Wang, Han, and Tiwari 2021). For example, tools like Unity (Unity Technologies 2025) and NVIDIA's PhysX (NVIDIA Corporation 2025) provide realistic graphics and accurate physics properties within a robust framework. These tools are also used as digital twins—digital recreations of real-world environments and systems (e.g., robots)—which allow for digital counterparts of physical systems to be tested in realistic environments, for instance in tasks such as autonomous decision making and action execution (Verdouw et al. 2021).

Automated Planning Automated planning research addresses the problem of reasoning about action sequences to transform an initial state into a new state that achieves a set of goal conditions (Ghallab, Nau, and Traverso 2016). A planning problem can be defined as a tuple $\Pi = \langle P, A, I, G \rangle$, where P is a set of properties that define a state space (including a set of objects), A is a set of actions, I is a set of initial state properties, and G is the set of goal conditions. This definition can be considered in the context of a state transition system, where a state captures all the properties of P that are true, and actions in A transform states to new states. A solution to the planning problem is a sequence of actions (a plan) that when applied to the initial state I transforms it to a state where the properties of G are true.

Traditionally, languages such as the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) are used to model planning problems, providing a standard, planner-independent representation language supported by a wide variety of planning engines. The Unified Planning Library (UPL) (Micheli et al. 2025) provides an alternative to standard PDDL specification through an API that enables external planners to be integrated in a Python-based wrapper. Tools like UPL also attempt to address problems of accessibility, by offering a traditional programming environment as an attempt to encourage the wider adoption of planning tools in research and industrial applications.

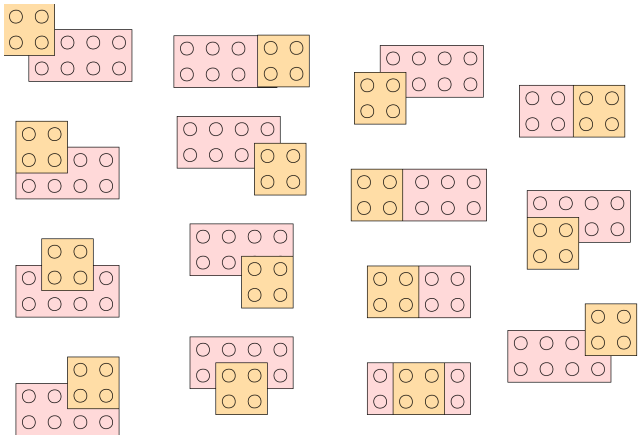


Figure 2: LEGO brick configurations.

Plan Visualisation Plan visualisation is an important area of research exploring the development of tools for tasks like plan simulation and planning domain debugging. While still an under-explored area, tools like LPS (Tapia, San Segundo, and Artieda 2015), Planimation (Chen et al. 2020), and vPlanSim (Roberts et al. 2021) aid in the interpretation of planner outputs, allowing interaction and online inspection of objects and states during plan execution for debugging, analysis, and refinement of planning models (Gerevini and Saetti 2020). Our work extends the Planning Domain Simulation (PDSim) system (De Pellegrin and Petrick 2024), an open source Unity-based extension for visualising planning domains. PDSim is an external plug-in for the Unity game engine, providing a user interface to 2D/3D graphics and animations, and a connection to external planners and PDDL language parsers using UPL. PDSim leverages Unity's underlying physics and animation system to create animations for PDDL-defined objects and properties. These animations can be used to set up a scene in Unity, allowing plans to be executed in the simulated virtual environment.

Scene Reconstruction Pipeline

We now present our pipeline for dynamically reconstructing scenes for planning-related tasks. As a running example, we use a LEGO environment to mimic a simple manufacturing domain, due to the vast catalogue of LEGO elements available with different shapes, colours and interaction methods.

Interacting with LEGO Bricks is relatively straightforward, though replicating their behaviours in a planning domain is far more complex. The anatomy of a LEGO brick involves studs (the connectors on top of a brick) or anti-studs (the area studs snap into). In traditional LEGO building, the relative position of bricks for stacking also matters. For instance, Figure 2 shows 15 different possible combinations for stacking two bricks. Therefore, to simplify our model we treat stacking as one brick on top of another, similar to Blocks World. We now present our pipeline for scene reconstruction, as illustrated in Figure 3.

Data Capture A mounted Intel RealSense D435 depth camera was used to capture data at a resolutions of

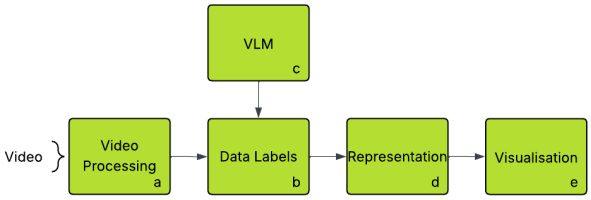


Figure 3: Scene reconstruction pipeline.

Figure 4: The identified objects in reference to the original image.

1920x1080, 1280x720, 848x480, 640x360 and 424x240 at 15 frames per second over a 10 second recording window. The camera was positioned directly above the LEGO pieces and as perpendicular to the table surface as possible to minimise distortion.

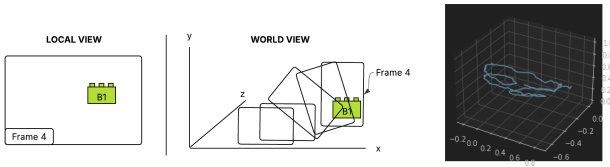
Video Processing The pipeline begins (Figure 3a) by extracting colour and depth frames from the recordings. Next, SAM2 (Ravi et al. 2024) is used to segment and create masks for all elements in each frame. The system then automatically corrects duplicate and fragmented masks. The final masks can be seen in Figure 4. The depth frames which were previously extracted from the video and the masks are now used to determine a physical footprint and height for each LEGO element. Each element is also assigned a 3D Cartesian location in their respective frame.

Before passing the object's coordinates to the representation stage, they are converted from local space mapped relative to the camera frame to world space coordinates where each camera frame will occur at a different point and likely a different rotation (Figure 5).

Camera motion between frames is estimated using Open-3D (Zhou, Park, and Koltun 2018). This estimation allows for the camera's coordinates to be mapped in the world space. The camera's trajectory for the LEGO example is shown in Figure 5b. Next, we project each estimated local coordinate into the world space using the equation:

$$\mathbf{p}_{\text{world}} = \mathbf{p}_{\text{cam}} + \mathbf{R} \cdot \mathbf{p}_{\text{local}},$$

where $\mathbf{p}_{\text{cam}}$ is the camera coordinate in the world space, $\mathbf{R}$ is the rotation matrix of the camera, and $\mathbf{p}_{\text{local}}$ is the coordinate of the element in the local frame. To finish, the physical



(a) (b)

Figure 5: Local to world coordinate conversion and estimated camera trajectory.

Predicate	Description
`(Brick ?b)`	`?b` is a brick
`(Location ?l)`	`?l` is a location
`(Dimension ?d)`	`?d` is a dimension.
`(Colour ?c)`	`?c` is an RGB colour
`(Label ?la)`	`?la` is a text label.
`(brickAt ?b ?l)`	Brick `?b` is at location `?l`
`(on ?b ?base)`	Brick `?b` is on brick `?base`
`(clear ?b)`	No brick is on brick `?b`
`(stacked ?b)`	Brick is stacked on top of another Brick.
`(brickDimension ?b ?d)`	Brick `?b` is scaled to dimension `?d`.
`(brickColour ?b ?c)`	Brick `?b` is colour `?c`
`(brickLabel ?b ?la)`	Brick `?b` has label `?la`

Table 1: PDDL scene predicates.

footprint (3D scale), height above table, local coordinate and an RGB colour (extracted from a segmented mask) populate a CSV file for use in later stages.

Data Labels The next stage of the pipeline (Figure 3b) uses an external VLM (Figure 3c) to provide descriptor values for the calculated segments. In this paper, we make use of the **qwen3-vl:235b-cloud** model through Ollama. The VLM is iteratively provided with each mask and the prompt: `Label this {hint} using a maximum of 3 words. Do not label the colour.` The hint is user specified and in this case is '`LEGO Element`'. The first three words of the response serve as the item label.

Representation This stage of the pipeline (Figure 3d) converts the VLM labels and calculated object properties into a partial automated planning representation. The data is converted into its PDDL equivalent and then into Python for use with the Unified Planning Library. We consider each element of a planning problem $\Pi = \langle P, A, I, G \rangle$, below.

Properties (P): Initially each PDDL property is manually defined using the properties in Table 1, including `Brick`, `Location`, `Colour`, `Dimension` and `Label` type predicates. Each `Brick` is a unique element in the scene, each `Location` a 3D Cartesian coordinate of the form `loc_x_y_z`, each `Colour` is an RGB colour of the form `col_r_g_b`, each `Dimension` is the calculated 3D scale of the form `dim_width_height_depth` and `Label` is the VLM-induced description. The `Colour` and `Dimension` properties are necessary for the upcoming visualisation stage as it allows the user to distinguish between the different elements.

We define a domain-specific set of predicates for our experimental environment. The `brickAt` predicate maps a `Brick` to a specific `Location`. The `on` relationship is used to model a `Brick` being stacked on another `Brick`. The `clear` property indicates that a brick can be placed on top of it, i.e., no brick is on top of a particular brick. The `stacked` property tracks Bricks which are currently in a stack. The `brickDimension`, `brickColour` and `brickLabel` predicates assigns a `Dimension`, `Colour` and `Label` to a particular `Brick`, respectively.

Actions (A): Domain actions are also prespecifed as shown in Table 2. The `move` action allows `Bricks` to move between `Locations`. The `place` and `remove` actions introduce construction by allowing `Bricks` to be placed on top of and removed from the top of each other.

Initial State (I): The calculated object properties and the VLM-induced labels are used to automatically build the initial state. The domain is initialised using Algorithm 1. The Cartesian locations, RGB colour and 3D scale must be encoded as strings to be a valid representation. For the "wedge plate" element in Figure 1a, the created objects and assigned predicates are seen in Table 3.

Goal (G): In this example our goal state is to stack all bricks. To do this we iterate over each pair of bricks and apply the necessary `on` predicates.

Visualisation The domain is visualised (Figure 3e) in Unity using PDSim. To do this, the PDSim extension is first installed in Unity and the PDSim back-end server launched in the Python code. Once executed, the server will automatically be identified by PDSim and the domain and objects will be created. Upon creation, all objects are cube bodies placed at the same coordinate. To visualise movements the animations for predicates must be manually defined using

Algorithm 1: Initial State Initialisation

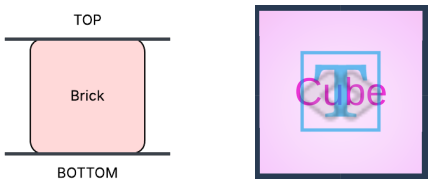
Input: VLM Labels and Property CSV
Output: Initialised objects
for each row in CSV **do**
    create a `Brick` Object.
    restructure x,y,z coordinate into a single string
    create a `Location` Object.
    restructure width, height, depth scale into a string
    create a `Dimension` Object.
    restructure RGB colour into a single string
    create a `Colour` Object.
    create a `Label` Object.
    assign `Brick` a `Location` using `brickAt`.
    assign `Brick` a `Dimension` using `brickDimension`.
    assign `Brick` a `Colour` using `brickColour`.
    assign `Brick` a `Label` using `brickLabel`.
    assign `Brick` as clear.
end for

Action	Description	Preconditions	Postcondition
(move ?b ?l1 ?l2)	Move brick ?b from location ?l1 to location ?l2	(brickAt ?b ?l1), (clear?b), (not (stacked ?b)	(not (brickAt ?b ?l1)), (brickAt ?b ?l2)
(place ?b1 ?b2 ?l)	Place a brick ?b1 on top of brick ?b2 in location ?l	(brickAt ?b1 ?l), (brickAt ?b2 ?l), (clear ?b1), (clear ?b2), (not (stacked ?b1)	(not (clear ?b2), (stacked ?b1)), (on ?b1 ?b2).
(remove ?b1 ?b2 ?l)	Remove brick ?b1 from the top of brick ?b2 in location	(brickAt ?b1 ?l), (brickAt ?b2 ?l), (on ?b1 ?b2), (clear ?b1), (stacked ?b1)	(not on(?b1 ?b2)), (clear ?b2), (not (stacked ?b1)) ?l

Table 2: Example environment actions.

Created Object	Assigned Predicate
(Brick obj_001)	(clear obj_001)
(Location loc_n33_359_1053)	(brickAt obj_001 loc_n33_359_1053)
(Dimension dim_32_1_31.72)	(brickDimension obj_001 dim_32_1_31.72)
(Colour rgb_47_187_237)	(brickColour obj_001 rgb_47_187_237)
(Label Wedge Plate)	(brickLabel obj_001 Wedge Plate)

Table 3: Created objects and assigned predicates.



(a) (b)

Figure 6: Environment preparation.

Unity Script Graphs, where each node is an operation within Unity that can be customised using the Unity or PDSim libraries. These graphs are then executed whenever a predicate is referenced in the plan.

PDSim automatically creates prefabs (an object template in unity) for each type of object in our scene. To prepare the environment for scene reconstruction these prefabs should be improved to allow for the correct placement of objects. First, the top and bottom of the Brick prefab is marked as seen in Figure 6a. These labels help to align Bricks during placement to avoid Bricks overlapping. To improve scene explanation we add a text object to the prefab which will show the VLM induced label (Figure 6b) during reconstruction.

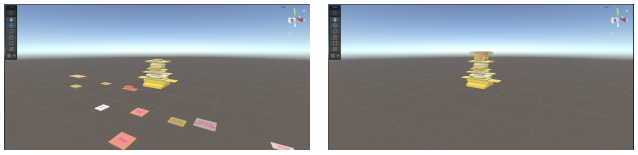
The Unity Script Graphs for each predicate animation can now be created as described:

- **brickAt:** This animation should deal with moving a `Brick` object to its desired `Location`. To do this we make use of PDSim's *Translate to Point* node which takes three inputs: the moving object (the Brick), the moving offset (the labelled bottom of the Brick) and the Cartesian coordinates of the desired `Location`. As our locations are structured as `loc_x_y_z` the script graph extracts the Cartesian coordinate from the string and creates a Unity transform object (a position). The transform is then fed to the Translate to Point node to allow for the brick's movement to the location.
- **on:** This animation will deal with stacking `Bricks`. For instance, to animate `(on ?b1 ?b2)` (`b1` is on `b2`), we use PDSim's *Translate to Object* node which takes four inputs: the moving object, the moving offset, the target object and the target offset. In this case the moving object is `b1`, the moving offset is the annotated bottom edge of `b1`, the target object is `b2` and the target offset is the annotated top edge of `b2`. The offsets ensure that the objects are correctly aligned.
- **brickDimension:** This animation should scale a `Brick` to the `Dimension` extracted from the sensor data. Similar to the `Location` in the `brickAt` animation, we parse the `Dimension`, `dim_width_height_depth` and create a Unity Transform which represents scale rather than a Cartesian coordinate. We then apply the scale to the `Brick` using a built-in Unity node.
- **brickColour:** This improves the visuals of the scene by re-colouring each `Brick` to match it attached `Colour`. The Script Graph parses the `rgb_r_g_b` colour, creates a new Unity Colour property and applies the new `Colour` to the `Brick`'s mesh.
- **brickLabel:** To visually distinguish between elements we attach the VLM-induced label to each object. The Script Graph extracts the label and edits the initial 'Cube' text to match the annotation.

As seen in Figure 1b, at this stage our scenes are reconstructed to visualise the initial state. The resulting scene with the executed plan to stack all bricks is seen in Figure 7. The same technical approach can also be applied to other scenarios, such as the DUPLO example shown in Figure 8.

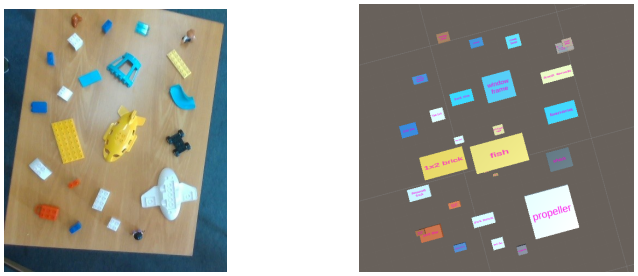
Experiments and Discussion

The pipeline was implemented using Python 3.12 and visualised using PDSim and Unity version 2022.3.62f1. The



(a) (b)

Figure 7: Goal execution.



(a) (b)

Figure 8: DUPLO reconstruction.

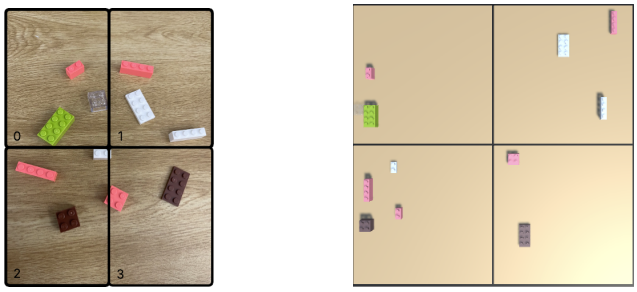
hardware used for stage (a) of the pipeline was a laptop with a 16-core CPU, 3072-core GPU and 8GB of memory, running Ubuntu 24. The remainder of the pipelined used a desktop PC with a 16-core CPU, 10752-core GPU and 64GB of memory, running Windows 11.

Preliminary Experiments Results from preliminary experiments motivated many decisions made during implementation. To select our VLM, multiple scenes were tested with different models. In these tests, Claude performed best and was chosen for the main tasks, although the cost of use and rate-limitation deemed it infeasible. So, the decision was made to use the cloud-based, Ollama-hosted VLM.

For a proof of concept we investigated a grid-based approach with a single top-down image (Othman, De Pellegrin, and Petrick 2026). This approach split an image into an NxN grid and used Anthropic's Claude-Sonnet 4 to identify LEGO elements in a scene and provide a description, dimension estimation and a colour. The VLM was then used to assign **Left** and **Above** relationships to pairs of bricks in each grid square to aid in positioning. The input and resulting simulated scene are shown in Figure 9.

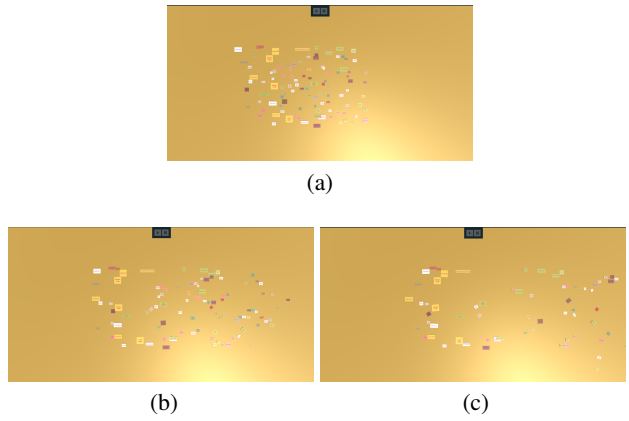
This experiment indicated several challenges: the VLM was unsuccessful in identifying all elements in a scene as the number of elements scaled. The VLM could annotate but struggled with the segmentation task. The VLM also failed to reason geometrically and often was unsuccessful in identifying differences between element heights from the top down image. This led to the use of an RGB-D camera to determine spatial properties and the decision to use the VLM purely for annotation and input segmentation labelling.

To simplify the domain, rather than treating each Cartesian coordinate as a `Location`, this approach treated each grid square as a separate `Location`. This required a subjective input and often resulted in mislabels and multiple identifications of singular elements if a brick was spread across multiple grid squares. The move to a Cartesian representation addressed this problem and allowed for a more precise visualisation.



(a) (b)

Figure 9: Grid-based approach.



(a) (b) (c)

Figure 10: Wind simulation.

Simulation Experiment This project takes the step to visualise our domains within Unity. As mentioned Unity has an powerful inbuilt physics engine which means the scenes we reconstruct can be used to carry out realistic simulations. As a quick experiment to simulate a realistic LEGO environment we assigned a mass of 0.004kg for the base sized brick as well as drag and angular drag at 0.05. Coefficients for friction, static friction and bounciness were set to 0.4, 0.6 and 0.05 respectively. These values were all generated by Anthropic's Claude Sonnet-4. We then simulated a gust of wind in Unity resulting in the pieces scattering as seen in Figure 10.

Labelling Experiment In our implementation, we opted to allow user-defined hints to be given to the VLM, which improved labelling accuracy. This experiment tested the reproducibility of the VLM using the DUPLO scene in Figure 8. The scene is annotated on six occasions, three times with hints and three times without.

Using the hint "`DUPLO elements`" and the prompt described earlier, the VLM produced the results shown in Ta-

with_hint_1	with_hint_2	with_hint_3
propeller	Propeller Turbine	Propeller Turbine
1x2 plate	One Two Tile	One Two Tile
fish	fish	fish
1x1 brick	1x1 brick	1x1 brick
1x1 brick	1x1 brick	1x1 brick
Banana element	banana piece	banana piece
1x1 tile	1x1 tile	1x1 tile
1x2 brick	1x1 brick	1x1 brick
1x2 brick	1x2 plate	1x2 plate
Tile 1x1	1x1 tile	1x1 tile

Table 4: Labels with hints.

without_hint_1	without_hint_2	without_hint_3
cross connector	plus token	model airplane
sponge	sponge	sponge
submarine	submarine	toy submarine
chalk stick	chalk	rice grain
cork	tooth fragment	pill
banana	banana	banana
square	square	square
sugar stick	battery	small bar
block	rectangle	small rectangle
square	square	square

Table 5: Labels without hints.

ble 4. The responses in all three cases were relatively aligned with major differences occurring on elements with sticker patterns or more complex shapes such as the plane parts or the squirrel. In some cases, the VLM produced the same or similar result three times but the labels were still largely incorrect, although LEGO related. For example, the white plane hull was labelled as a propeller, the slide as a banana or the top section of the plane as a fish. As discovered in the preliminary experiments, the VLM struggled to reason geometrically; when it attempted to include sizes (as an NxM stud) they were wrong. Without the hints, the VLM's labels were largely incorrect as seen in Figure 5, although at times it did correctly label the decal on the LEGO element such as "star".

Increased Piece Experiment In this experiment, the number of pieces in the scene increased from 10 to 100 in 10 piece increments. For each cluster of pieces, a still image was captured at 5 different resolutions. This experiment was repeated 3 times at 3 different lighting conditions. The test environment allowed for controlled lighting. The lighting levels tested were approximately:

- **0 Lux**: All lights switched off.
- **19 Lux**: A single light switched on offset behind the camera.
- **270 Lux**: A directly overhead light switched on.

The images were captured using the same handheld camera as the core experiment. The camera was positioned perpendicularly to the desk and held at a distance of 42cm using

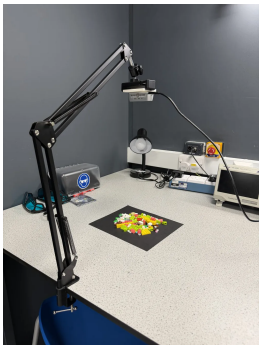


Figure 11: Experiment setup

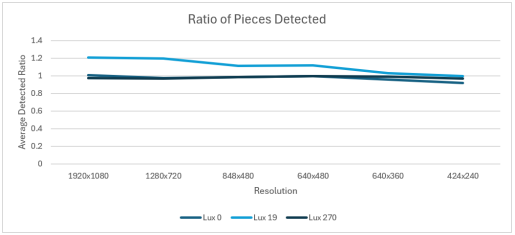


Figure 12: Average ratio of detected pieces.

a tripod. Lux was measured using the *Lux Light Meter Pro*[1] app on an iPhone 17 Pro Max. The experiment setup is seen in Figure 11.

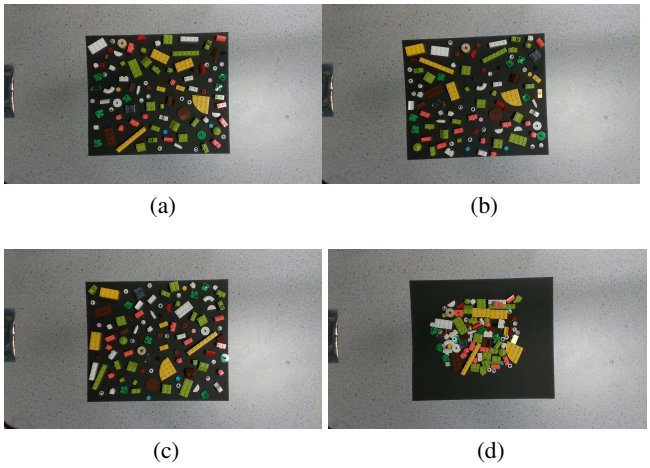
As seen in Figure 12, the average number of detections slightly drops in the lower resolutions. In the 0 lux state the error rate was at most 0.08%. In 270 lux the highest error rate was 0.032%. At 7 lux the error rate significantly increased reaching 0.207% which dropped as the resolution decreased. It was determined that the positioning of the light introduced more shadows in the scene which the segmentation model mistook as more LEGO elements. As the resolution dropped the shadows became less clear and the model picked up the more defined pieces.

Reshuffle Experiment In this experiment, we shuffled the same 100 LEGO elements into three different combinations as seen in Figure 13. We then reshuffled the pieces a fourth time but this time without the white space in between as seen in Figure 13d.

The organisation of the test environment took the same approach as the increased piece experiment. All scenes were captured at a light level of 270 lux.

The piece identification dropped as the resolution dropped. There was only one iteration where more than 100 pieces were identified. This happened in the 1920x1080 resolution where 1 extra piece was identified in one example. Across all of the spaced out scatters we maintained an average identification rate of at least 95% across all resolutions. In the grouped cluster we attained a 94%, 93%, 87%, 82% and 74% element identification rate in the 1920x1080,

[1] https://apps.apple.com/us/app/lux-light-meter-pro/id1292598866



(a) (b)

(c) (d)

Figure 13: Reshuffle experiment.

Figure 14: Reshuffle experiment results.

1280x720, 848x480, 640x360 and 424x240 respectively. Upon closer examination of the masks in the grouped example, it was clear that the segmentation model identified larger pieces but struggled with smaller and partially occluded pieces hidden under other bricks. The model still managed to segment some elements which were in contact with others.

Observations Overall, these experiments highlighted the uncertainty surrounding VLM annotations. Even when attempting to constrain the VLMs with a hint there was still no guarantee of getting the same response for a given element more than once. There was also no guarantee that the VLM would not label out-width the recommended hint (i.e., the "fish" label in the labelling experiment). After the extensive experiments, it became clear the VLM started to label items with LEGO-esque labels without them being an existing LEGO piece (i.e., "banana piece", "single brick"). In the office space example, even with the hint, annotations such as "barrel" were provided. More work is needed to find a more suitable VLM for the labelling task or training a smaller more task-specific network for annotations.

From the increased piece experiment we find that segmentation improves as resolution increases. However, in most cases a resolution of 640x480 is sufficient for segmentation. We also observed that the success of the segmentation relies on balanced lighting rather than brightness. Balanced lighting mitigates shadows which decreases incorrect identifications.

In all the examples, the only locations to exist are the coordinates where an element is already located, which leaves other coordinates inaccessible. We attempted to allow for all coordinates within the bounds of the minimum and maximum element locations to be a valid location in the plan but this became too complex and introduced issues in the larger element examples. It was expected that the Cartesian mapping would increase the complexity of the domain because more Location objects exist relative to the grid approach in our preliminary work. So, as the domains grew larger, the time taken to determine a valid plan greatly increased.

While the grid approach from the preliminary experiments was not detailed enough. A combination of both approaches could map a larger environment to grid locations and map items to coordinates on a per-grid basis, thus decreasing the number of overall Location objects in a scene.

Additionally, although the cube bodies in the scenes were organised in a similar way to the real environment and the colours were a clear match, the digital reconstruction still fell short of realism. In our approach, we extract the width and depth measurements from the calculated area which is accurate for rectangular elements but not for circular shapes. Rotations are also ignored. Although labels are placed on each item, it was difficult to differentiate between similar looking items. Incorporating a mesh generation tool to create meshes from masks should aid in the explainability of the scene especially if methods are investigated to incorporate rotations. Since the pipeline is modular, it should be straightforward to integrate new methods into the system.

Conclusion and Future Work

This paper continues to lay the groundwork for dynamic scene reconstruction and highlights the main aspects needed to fully automate our pipeline in order to correctly recreate planning environments. Our current implementation is partially automated as properties and actions are generally prespecifed and our segmentation method requires user input. However, it is believed that with the landscape of available generative tools and alternative methods we can further automate other parts of the approach. In this implementation, each aspect of the pipeline is executed once to create the final scene. Future work will look at using cyberphysical systems to explore more complex environments (such as outdoor spaces) and introduce a dynamic feedback loop, where the scene is built as the system explores and if elements in the real world change, the domain and visualisation are updated. Furthermore, planning related tasks (such as navigation, assembly or assistance) will be executed in the simulation and relevant properties will be transferred back to the real system and corrected in the model if necessary.

References

Bavelos, A. C.; Anastasiou, E.; Dimitropoulos, N.; Michalos, G.; and Makris, S. 2025. Virtual reality-based dynamic scene

recreation and robot teleoperation for hazardous environments. *Computer-Aided Civ. and Infrastructure Eng.*, 40(3): 392–408.

Berrios, W.; Mittal, G.; Thrush, T.; Kiela, D.; and Singh, A. 2023. Towards Language Models That Can See: Computer Vision Through the LENS of Natural Language. arXiv:2306.16410.

Chen, G.; Ding, Y.; Edwards, H.; Chau, C. H.; Hou, S.; Johnson, G.; Syed, M. S.; Tang, H.; Wu, Y.; Yan, Y.; Tidhar, G.; and Lipovetzky, N. 2020. Planimation. arXiv:2008.04600.

Chen, K.; Wang, J.; Pang, J.; Cao, Y.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Xu, J.; Zhang, Z.; Cheng, D.; Zhu, C.; Cheng, T.; Zhao, Q.; Li, B.; Lu, X.; Zhu, R.; Wu, Y.; Dai, J.; Wang, J.; Shi, J.; Ouyang, W.; Loy, C. C.; and Lin, D. 2019. MMDetection: Open MMLab Detection Toolbox and Benchmark. arXiv:1906.07155.

Collins, J.; Chand, S.; Vanderkop, A.; and Howard, D. 2021. A Review of Physics Simulators for Robotic Applications. *IEEE Access*, 9: 51416–51431.

Dalal, N.; and Triggs, B. 2005. Histograms of oriented gradients for human detection. In *Proc. CVPR*, 886–893.

De Pellegrin, E.; and Petrick, R. P. 2024. Planning domain simulation: an interactive system for plan visualisation. In *Proc. ICAPS*, 133–141.

Downs, L.; Francis, A.; Koenig, N.; Kinman, B.; Hickman, R.; Reymann, K.; McHugh, T. B.; and Vanhoucke, V. 2022. Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items. In *Proc. ICRA*, 2553–2560.

Ferrara, E. 2023. Should ChatGPT be biased? Challenges and risks of bias in large language models. *First Monday*.

Gerevini, A. E.; and Saetti, A. 2020. An interactive tool for plan generation, inspection, and visualization. In *Knowledge Engineering Tools and Techniques for AI Planning*, 127–155.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Girshick, R. 2015. Fast R-CNN. In *Proc. ICCV*.

Gu, Q.; Kuwajerwala, A.; Morin, S.; Jatavallabhula, K. M.; Sen, B.; Agarwal, A.; Rivera, C.; Paul, W.; Ellis, K.; Chellappa, R.; Gan, C.; de Melo, C. M.; Tenenbaum, J. B.; Torralba, A.; Shkurti, F.; and Paull, L. 2023. ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning. arXiv:2309.16650.

Karami, E.; Prasad, S.; and Shehata, M. 2017. Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images. arXiv:1710.02726.

Kargar, S. M.; Yordanov, B.; Harvey, C.; and Asadipour, A. 2024. Emerging Trends in Realistic Robotic Simulations: A Comprehensive Systematic Literature Review. *IEEE Access*, 12: 191264–191287.

Kerbl, B.; Kopanas, G.; Leimkühler, T.; and Drettakis, G. 2023. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4): 1–14.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The planning domain definition language. Tech. Report, Yale Center for Computational Vision and Control.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Naik, D.; Naik, I.; and Naik, N. 2024. Large Data Begets Large Data: Studying Large Language Models (LLMs) and Its History, Types, Working, Benefits and Limitations. In *Contributions Presented at C3AI*, 293–314.

Ni, Z.; Hupkes, J.; Eriksson, P.; Leijonhufvud, G.; Karlsson, M.; and Gong, S. 2025. Parametric Digital Twins for Preserving Historic Buildings: A Case Study at Löfstad Castle in Östergötland, Sweden. *IEEE Access*, 13: 3371–3389.

NVIDIA Corporation. 2025. *PhysX SDK Documentation*.

Othman, A. A.; De Pellegrin, E.; and Petrick, R. P. 2026. Dynamic Scene Reconstruction for Planning Environments. In *Proc. UKPlanSIG 2026*.

Pellegrin, E. D.; and Petrick, R. 2024. Simulating Robotics Planning Domains with PDSim and ROS. In *ICAPS 2024 System's Demonstration track*.

Raschka, S. 2024. *Build a Large Language Model (From Scratch)*. Shelter Island, NY: Manning Publications.

Ravi, N.; Gabeur, V.; Hu, Y.-T.; Hu, R.; Ryali, C.; Ma, T.; Khedr, H.; Rädle, R.; Rolland, C.; Gustafson, L.; Mintun, E.; Pan, J.; Alwala, K. V.; Carion, N.; Wu, C.-Y.; Girshick, R.; Dollár, P.; and Feichtenhofer, C. 2024. SAM 2: Segment Anything in Images and Videos. *arXiv preprint arXiv:2408.00714*.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *Proc. CVPR*.

Roberts, J. O.; Mastorakis, G.; Lazaruk, B.; Franco, S.; Stokes, A. A.; and Bernardini, S. 2021. vPlanSim: an open source graphical interface for the visualisation and simulation of AI systems. In *Proc. ICAPS*, 486–490.

Rosinol, A.; Gupta, A.; Abate, M.; Shi, J.; and Carlone, L. 2020. 3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans. arXiv:2002.06289.

Schneider, A.; Sturm, J.; Stachniss, C.; Reisert, M.; Burkhardt, H.; and Burgard, W. 2009. Object identification with tactile sensors using bag-of-features. In *Proc. IROS*, 243–248.

Tapia, C.; San Segundo, P.; and Artieda, J. 2015. A PDDL-based simulation system. In *Proceedings of the IADIS International Conference Intelligent Systems and Agents*, 81–88.

Unity Technologies. 2025. Unity. Game development platform.

Verdouw, C.; Tekinerdogan, B.; Beulens, A.; and Wolfert, S. 2021. Digital twins in smart farming. *Agricultural Systems*, 189: 103046.

Verykokou, S.; and Ioannidis, C. 2023. An Overview on Image-Based and Scanner-Based 3D Modeling Technologies. *Sensors*, 23(2).

Vidal, J.; Vallicrosa, G.; Martí, R.; and Barnada, M. 2023. Brickognize: Applying Photo-Realistic Image Synthesis for Lego Bricks Recognition with Limited Data. *Sensors*, 23(4): 1898.

Wang, Z.; Han, K.; and Tiwari, P. 2021. Digital Twin Simulation of Connected and Automated Vehicles with the Unity Game Engine. In *Proc. DTPI*, 1–4.

Zhiheng, X.; Rui, Z.; and Tao, G. 2023. Safety and Ethical Concerns of Large Language Models. In Sun, M.; Qin, B.; Qiu, X.; Jiang, J.; and Han, X., eds., *Proc. Chinese Nat. Conf. on Computational Linguistics (Tutorial Abstracts)*, 9–16.

Zhou, Q.-Y.; Park, J.; and Koltun, V. 2018. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847*.