vPlanSim: An Open Source Graphical Interface for the Visualisation and Simulation of AI Systems

Jamie O. Roberts [1,2], Georgios Mastorakis [3], Brad Lazaruk [3], Santiago Franco [3], Adam A. Stokes[2], Sara Bernardini [3]

[1]School of Engineering, Institute for Integrated Micro and Nano Systems, The University of Edinburgh
[2]EPSRC CDT in Robotics and Autonomous Systems, Edinburgh Centre for Robotics
[3] Department of Computer Science, Royal Holloway, University of London

Abstract

We introduce vPlanSim, an open source tool to aid in AI PDDL development. This tool is primarily aimed at researchers and developers who need a visual representation of their planning problem so that they can make useful insights into the performance of their system, and also to naturally convey their system to others. It is an open-source tool which allows a user to quickly and easily visualise a target environment to generate the problem files and also to visualise a plan. It is particularly well suited to spatial planning problems. This paper will demonstrate vPlanSim on 2D and 3D planning problems. vPlanSim is based on a small and carefully considered set of dependencies such as VTK and PyQt. It can be set up on different platforms and compiled from source with minimal effort. The code is and maintained via a clear code review mechanism. We welcome contributions from the open-source community.

Introduction

As the fields of Task Planning progress, the problems that they try to solve become increasingly more complex. This, in turn, introduces a difficulty in interpreting and comprehending the information of the initial state and plan output. This is particularly apparent in problems where the system attempts to tackle physical problems in 3-Dimensions. It is difficult for humans to accurately visualise a 3-D sequence, from a text or even integer output of a sequence of coordinates. Even more difficult is the communication and conveyance of these systems to stakeholders, new members of the development team or the general public without a common platform that is understandable.

To this end, we present vPlanSim: An Open Source Graphical Interface for the Visualisation of PDDL problems. This software is written in Python3, so as to be widely accessible to most programming literate users, with minimal dependencies. This software is open and is designed to allow a user to predefine how their system output should interact with vPlanSim across a range of modalities. The demonstration of this software will be framed in the context of Task Planning systems but it should be applicable across a wide variety of systems, providing the users observe the strict pipeline of the software.

Related Work

Explainable AI is a growing field in AI research with an increasing awareness around it as one of the major limiting factors in the practical deployment of AI systems (Adadi and Berrada 2018). Task Planning is a good candidate to satisfy the requirements to qualify for Explainable AI (Fox, Long, and Magazzeni 2017), but often lacks tools to aid in human intelligibility. Planimation (Chen et al. 2020) is an open source framework to visualise sequential planning solutions and uses a 2D representation to abstractly visualise plans. VisPlan (Glinskỳ and Barták 2011) is a graphical tool to allow for the user to analyse plans and find potential flaws. It requires a domain, problem and plan file as an input and so is a tool used in post-process. TransportEditor (Škopek and Barták 2017) allows a user to creating domains and visualising and editing problem instances, it is intended for problems in the transportation domain. Chakraborti et al. (2017) presents visualization capabilities of an Explainable AI Planning Agent such that a human operator can interact with a visual representation of the plan output but also the internal decision making processes. The Logic Planning Simulator (LPS) (Tapia, San Segundo, and Artieda 2015) is a graphical tool that allows a user to test and validate PDDL STRIPS domains and simulate plan outputs. PDDLGym (Silver and Chitnis 2020) is capable of simulating PDDL problems visually. There is also considerable work in the use of Minecraft to visualise specific planning problems. A good example of this is Kohn et al. (2020), where they present MC-Saar which is a platform for giving instructions between a human operator and the computer in complex tasks, particularly suited to construction tasks well approximated by a ‘blockworld’ model. In terms of the work towards aiding the user in editing planning files, PDDL Studio (Plch et al. 2012) offer considerable PDDL syntax tools such as error highlighting and autocompletion. (Vaquero et al. 2009) presents itSIMPLE$_{3.0}$, software that allows the user to construct a domain in UML which is automatically translated to PDDL syntax via XML. (Dolejsi et al. 2018) presents and end-to-end PDDL 2.2 developer environment with a variety of tools that integrates with Visual Studio, with (Long, Dolejsi, and Fox 2018) presenting another Visual Studio plug-in to support PDDL as a real-world problem modelling tool. myPDDL (Strobel and Kirsch 2014) presents a suite of tools for PDDL domain construction and analysis.

This work proposes the first step towards an all encompassing tool could incorporate many of the features exhibited in the literature presented, however with a crucial goal of allowing 3D manipulation of an environment to aid in the development of 3D planning problems, particularly in the field of robotics. vPlanSim operates independently of any domain files at this point, it functions as a framework to construct a domain specific set of predicates which can be related to 3D graphics. With that in mind, in should be noted that the most common visualisation and simulation tool used by the Robotics and AI community is of course ROS/Gazebo, and with respect to Task Planning, ROS/Gazebo with a particular emphasis on ROSPlan. Whilst this is a very powerful library and set of software tools, vPlanSim would be a suitable alternative for the following reasons; vPlanSim is lightweight with minimal dependencies and has an intuitive open framework so that users can adapt and add to its functionality to suit their needs. The tool is intended to be an aid to Task Planning for 3D problems, and is not intended to include physics and dynamic simulation, at which point a software tool such as ROSPlan/ Gazebo would be preferable.

Design & Implementation

The tool is built using several libraries offering user interaction (PyQt5 (Riverbank Computing Limited 2020), VTK (Schroeder, Martin, and Lorensen 2006)), real-time visualisation in 2D/3D (VTK), PDDL synthesis (Python libraries) and visual simulation (VTK). We chose to use PyQt because it runs without any pre-compilation and has easier and faster mechanisms to address dependency issues. VTK offers simple mechanisms to produce 2D and 3D visualisations. We implemented the tool using Python3 for several reasons such as minimising the time spend to compile and fix dependencies (i.e. if using C++), and also that QT and VTK have a very active Python community that has offered valuable support for the development of this tool.

In Figure 1 we show a high level pipeline of the vPlanSim design and the dependencies of the main functionalities. The user interacts with the User Interface (UI) using both a menu developed with PyQt and a VTK window that is also used for visualisation and simulation. The task of PDDL synthesis is implemented using native Python3 and NumPy (Oliphant 2006) libraries. Finally, annotated in a yellow box is the call to a planner to solve the PDDL problem, that currently runs externally. Nevertheless, it is planned to be embedded in the tool in a future release. Further discussion regarding the pipeline can be found at the GitHub repo (see Release Section).

Functionality

The new tool comprises several functionalities to assist users create visual scenes in 3D; set agent, goal and other visual

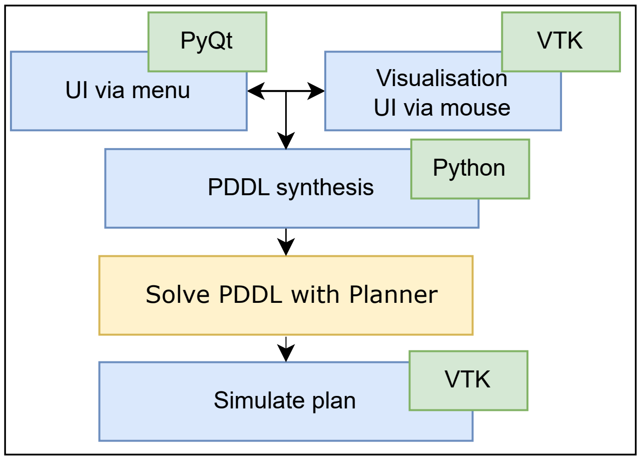


Figure 1: vPlanSim pipeline: blue boxes annotate automation offered by vPlanSim

features to reflect the predicates of the user domain on the 3D scene via mouse interaction; generate a PDDL problem in a semi-automatic mechanism; and finally simulate the output plan over the visualised 3D scene. Table 1 summarises the detailed functionalities of vPlanSim. The current functionality does not allow for automated PDDL domain file translation and synthesis.

Customised visualisation

As discussed, vPlanSim uses VTK to visualise the generated scene. The walls, obstacles and the floor panels make use of the vtkActor class. The colour properties of the vtkActor objects are manipulated to differentiate entry and goal points, and the positions of the objects can be calculated on demand. The undo/redo stack allows for objects to be removed or redone as necessary. Lists (i.e. Python List) of wall positions, entry points, and goal points can be generated by scanning the VTK colour and position properties of the various objects in the scene. The GUI can be used to add exterior and interior walls by setting the required dimensions. Obstacles, goal and entry points can be defined by entering a point (i.e. PyQt UI) and click mode or by specifying the exact position in the scene (i.e. VTK window). Wall and floor objects can be removed by selecting the delete mode or by clicking on a specific block to be deleted. The floor panels of the scene, annotations noting the positions of the wall blocks, the X, Y, and Z axes can be set to be visible or not.

When first executed, the tool initialises two vtkRenderer objects, one for the scene blocks and another for annotations such as the grid and axes. Scaling and rotation of the scene is enabled through VTK window interactions. All the walls and obstacles are composed of a vtkActor object of size 1x1x1 and a vtkCaptionActor2D. The vtkCaptionActor2D holds the text representation of the position of the cube. The floor panels are also vtkActor objects.

Standard VTK colour properties are used to differentiate the entry and goal points. Creation of the outside walls takes three inputs from the GUI, representing the width, height,

Module	GUI spec	Functionality
Visualisation	Outside walls	Set the initial geometry of the 3D scene
	Internal walls	Set cells
	Delete blocks	Delete parts of geometry
	Set obstacles	Introduce obstacles
	Select entry	Initialisation point
	Select goal	Termination point
	Redo/Undo	Undo/Redo visualisation steps
PDDL Synthesis	Generate PDDL	Formulate PDDL description Producing 2D and 3D problems Preview PDDL description
Simulation	Visualise path	Simulate plan via Avatars class

Table 1: vPlanSim modules and functionalities

and depth of the outside walls. These are rendered along the X, Y, and Z axes respectively. Looping over the values, a vtkPropAssembly object for each section of the outside walls is added and then the scene is rendered. It is initially assumed that the scene will have a floor. Adding interior walls is divided into two functions allowing the addition of walls along the X axis or along the Z axis. The starting position along the X or Z axis along with the height and length of the walls are taken from inputs in the GUI. The delete blocks module uses a vtkPicker class to interpret mouse clicks on the scene. Undo and redo functionality in provided through the inclusion of two basic action stacks, implemented as Python Lists. On demand, the scene can be scanned to calculate the positions and type of each object. The entry points and goals are differentiated by their colour properties. Separate lists of the wall objects, entry points and goals can be generated. Obstacles can be added to the scene by specifying the geometry directly in the GUI, or by entering a selection mode, clicking on the scene at the point the obstacle should be created and clicking save. The vtkPicker class is used to determine where the user has clicked in the scene. The entry point and goals can similarly be created by the user through point and click or by directly setting a position in the GUI.

PDDL Problem Generation

The second functionality of the tool generates a PDDL problem file based on the visualisation environment discussed in the previous subsection. Given a particular domain (e.g. Sokoban), the user codes the necessary commands in Python to *align* the domain with a skeleton PDDL problem. Effectively, the user must map the object names and predicates to a process that would generate the PDDL problem. We give two examples of PDDL problems where videos can be found in the GitHub repo with corresponding problem files. The PDDL generation then maps the 2D/3D geometry designed by the user to parametrize the PDDL predicates. The name of predicates and objects are defined earlier by the user and the tool will automatically generate the numerical values representing the scene, obstacles, agents and goals, etc. More specifically, the tool takes into account the outside wall geometry to define the search space. In some problems, outside walls play a role in the plan (e.g. a wall has an entrance for an agent to enter) while in some other, the outside wall's role is to delimit the permitted coordinates for agents, goals etc. Coordinates of internal walls and obstacles alongside with the outside wall blocks are passed to the generatePDDLproblem function that assigns the correct object/predicate to these values. Furthermore, coordinates of the agents, elements and goals are also passed to the same function. The tool generates a full PDDL problem that can then be solved by a planner.

Visualisation Foundation

VTK The visual representations are built entirely on VTK, which is a powerful open source graphics library and allows for flexible implementation of a users own needs.

Integration with GUI The point of integration for a user's own graphics visualisation with vPlanSim is the class vtkActor. A user can bring their own graphics in many forms, from predefined .STL files to 3D models using VTK's own primitives. The GUI will accept the vtkActor container for the models and so can be easily visualised in the program.

On top of this, the user will just need to define a response of the 3D models, given an action in the plan produced.

Plan Simulation

As previously mentioned, vPlanSim utilises the flexible nature of the VTK pipeline and particularly the vtkActor class as the point of integration for all modules in vPlanSim. As vPlanSim is intended for user modification/ addition to enable their application specific domain representations.

For plan visualisation, there are processes to account for: plan interpretation (action parsing), agent-to-graphic assignation and finally plan execution.

Plan Interpretation (Action Parsing) Each domain will have a unique set of actions and naming conventions. A text parser is required to generate a usable plan for the visualisation. The relative complexity or simplicity of this parser is entirely dependent on the resolution of the action details that the user would like to visualise in vPlanSim. In our examples, the plan is parsed such that the names of the actions can be compared against the user defined functions in vPlanSim that perform the intended action.

The examples of Sokoban (2D deployment) and a bespoke drone operation (3D deployment) domains have an important difference which must also be accounted for in the Plan Interpretation process, and is particularly important for domains which are focused on many agents. Sokoban has a specific agent construction in its domain, whilst the drone domain relies on an unspecified agent construction.

The agent-to-graphic stage of the pipeline must be in agreement with the parsing format that reflects this difference by storing the graphical representations of the agents as a list, where the addresses would be the index of the list, or as a dictionary.

Agent-To-Graphics This part of the process generates a unit graphic to represent the agent. The basic unit in vPlanSim is a cube, which is taken from vtkCubeSource and as such the user can customise each cube to the degree that the existing architecture of vtkCubeSource and vtkActor allows. VPlanSim has a standard set of graphical units in Avatars.py which the user can call. From Avatars.py, there are standard classes for a cube, sphere, arrow, cylinder and a cone. These are all based from the vtkSource classes and have accessible functions to modify colour, orientation, position and physical dimensions. There is also a standard function to import .stl files into the software so that users can expand on the complexity of the graphics to files generated on standard CAD packages. This function has the standard colour and position functions as well as scale and rotation.

To accompany the graphical objects, it is required to have a list of functions that are callable in place of the raw plan output. These are user defined to match the desired action effects but in the example domains given, each action results in a physical displacement of the agent. Note that calibration of an object in each dimension may be required for placement, achieved through predefined offsets.

Plan Execution This is the final stage of visualising a plan and occurs by stepping through the interpreted plan and executing the associated function. This implementation of this user defined, but in these domains it uses a counter of the plan step which updates every time the 'step through' button is pressed by the user.

Release

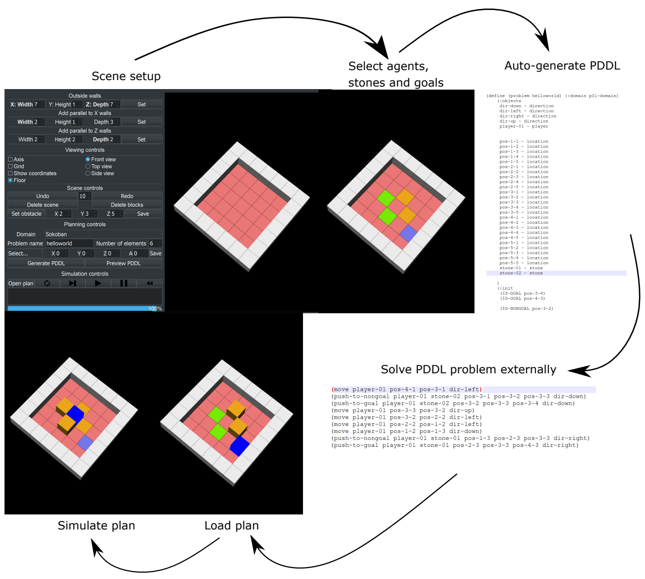
vPlanSim is released open-source under the GNU GPLv3 licence and is available at a GitHub repo (https://github.com/mastrogiorgis/vPlanSim). Ongoing development is coordinated via GitHub. Code changes are integrated via the following steps.

1. An issue is opened for a feature request or a bug fix
2. A developer starts a new branch on a personal fork of vPlanSim, writes code, and submits a pull request when ready
3. The code change is reviewed and discussed in the pull request Modifications are made to address issues raised in the discussion
4. One of the admins merges the pull request to the master branch and closes the issue

The code review process maintains a consistent coding style and adherence to modern Python3 programming guidelines. Pull requests are automatically checked by a continuous integration service. We hope that vPlanSim will be a useful tool to a broad community of researchers and developers who aim to export and simulate PDDL problems with the ease of a visualisation tool.

Examples

The tool comes with two example problems (see drop down menu), Sokoban (Muise 2015) and a bespoke drone domain. More plans and video demos such as a 3D planning problem for drones can be found in the GitHub repo.


Scene setup
Select agents, stones and goals
Auto-generate PDDL
Solve PDDL problem externally
Simulate plan
Load plan

```
(define (problem helloworld) (:domain p01-domain)
    (:objects
     dir-down - direction
     dir-left - direction
     dir-right - direction
     dir-up - direction
     player-01 - player


     pos-1-1 - location
     pos-1-2 - location
     pos-1-3 - location
     pos-1-4 - location
     pos-1-5 - location
     pos-2-1 - location
     pos-2-2 - location
     pos-2-3 - location
     pos-2-4 - location
     pos-2-5 - location
     pos-3-1 - location
     pos-3-2 - location
     pos-3-3 - location
     pos-3-4 - location
     pos-3-5 - location
     pos-4-1 - location
     pos-4-2 - location
     pos-4-3 - location
     pos-4-4 - location
     pos-4-5 - location
     pos-5-1 - location
     pos-5-2 - location
     pos-5-3 - location
     pos-5-4 - location
     pos-5-5 - location
     stone-01 - stone
     stone-02 - stone

    )
    (:init
     (IS-GOAL pos-3-4)
     (IS-GOAL pos-4-3)

     (IS-NONGOAL pos-3-2)
```

```
(move player-01 pos-4-1 pos-3-1 dir-left)
(push-to-nongoal player-01 stone-02 pos-3-1 pos-3-2 pos-3-3 dir-down)
(push-to-goal player-01 stone-02 pos-3-2 pos-3-3 pos-3-4 dir-down)
(move player-01 pos-3-3 pos-3-2 dir-up)
(move player-01 pos-3-2 pos-2-2 dir-left)
(move player-01 pos-2-2 pos-1-2 dir-left)
(move player-01 pos-1-2 pos-1-3 dir-down)
(push-to-nongoal player-01 stone-01 pos-1-3 pos-2-3 pos-3-3 dir-right)
(push-to-goal player-01 stone-01 pos-2-3 pos-3-3 pos-4-3 dir-right)
```

Figure 2: GUI interface showing a typical Sokoban example

Conclusion

We have introduced and demonstrated vPlanSim, a lightweight and open source software package for Task Planning researchers to quickly and easily visualise their systems for the purposes of communication and development. vPlanSim does not contain any physics as standard but is instead meant as a quick and effective visualiser. vPlanSim allows the user to construct a 3D representation to aid them in developing their systems by visually generating problem files and simulating their plans in the same environment.

Acknowledgments

EPSRC and the CDT in Robotics and Autonomous Systems at Heriot-Watt and The University of Edinburgh (EP/L016834/1). Partially supported by the InnovateUK Connect-R (TS/S017623/1).

References

Adadi, A.; and Berrada, M. 2018. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6: 52138–52160.

Chakraborti, T.; Fadnis, K. P.; Talamadupula, K.; Dholakia, M.; Srivastava, B.; Kephart, J. O.; and Bellamy, R. K. 2017. Visualizations for an explainable planning agent. *arXiv preprint arXiv:1709.04517* .

Chen, G.; Ding, Y.; Edwards, H.; Chau, C. H.; Hou, S.; Johnson, G.; Syed, M. S.; Tang, H.; Wu, Y.; Yan, Y.; et al. 2020. Planimation. *arXiv preprint arXiv:2008.04600* .

Dolejsi, J.; Long, D.; Fox, M.; and Besançon, G. 2018. PDDL Authoring and Validation Environment for Building end-to-end Planning Solutions. In *Proc. of the 28th Int. Conference on Automated Planning and Scheduling (ICAPS)*.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *arXiv preprint arXiv:1709.10256* .

Glinskỳ, R.; and Barták, R. 2011. Visplan–interactive visualisation and verification of plans. *KEPS 2011* 134.

Köhn, A.; Wichlacz, J.; Schäfer, C.; Torralba, A.; Hoffmann, J.; and Koller, A. 2020. MC-Saar-Instruct: a Platform for Minecraft Instruction Giving Agents. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 53–56.

Long, D.; Dolejsi, J.; and Fox, M. 2018. Building support for PDDL as a modelling tool. *KEPS 2018* 78.

Muise, C. 2015. AI Planning classical domains. URL https://github.com/AI-Planning/classical-domains.

Oliphant, T. E. 2006. *A guide to NumPy*, volume 1. Trelgol Publishing USA.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL studio. *System Demonstrations and Exhibits at ICAPS* 15–18.

Riverbank Computing Limited, T. Q. C. 2020. PyQt5. URL https://www.riverbankcomputing.com/static/Docs/PyQt5/index.html. [Online; accessed 8-September-2020].

Schroeder, W.; Martin, K.; and Lorensen, B. 2006. *The Visualization Toolkit–An Object-Oriented Approach To 3D Graphics*. Kitware, Inc., fourth edition.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. In *International Conference on Automated Planning and Scheduling (ICAPS) PRL Workshop*. URL https://github.com/tomsilver/pddlgym.

Škopek, O.; and Barták, R. 2017. Transporteditor–creating and visualising transportation problems and plans. In *International Conference on Automated Planning and Scheduling*.

Strobel, V.; and Kirsch, A. 2014. Planning in the wild: modeling tools for PDDL. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, 273–284. Springer.

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*.

Vaquero, T. S.; Silva, J. R.; Ferreira, M.; Tonidandel, F.; and Beck, J. C. 2009. From Requirements and Analysis to PDDL in itSIMPLE3. 0. *Proceedings of the Third International Competition on Knowledge Engineering for Planning and Scheduling, ICAPS 2009* 54–61.