Roblocks: An Educational System for AI Planning and Reasoning

by

Chirav Dave

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2019 by the
Graduate Supervisory Committee:

Siddharth Srivastava, Chair
Yu Zhang
Ihan Hsiao

ARIZONA STATE UNIVERSITY

December 2019

ABSTRACT

This research introduces Roblocks, a user-friendly system for learning Artificial Intelligence (AI) planning concepts using mobile manipulator robots. It uses a visual programming interface based on block-structured programming to make AI planning concepts easier to grasp for those who are new to robotics and AI planning. Users get to accomplish any desired tasks by dynamically populating puzzle shaped blocks encoding the robot's possible actions, allowing them to carry out tasks like navigation, planning, and manipulation by connecting blocks instead of writing code. Roblocks has two levels, where in the first level users are made to re-arrange a jumbled set of actions of a plan in the correct order so that a given goal could be achieved. In the second level, they select actions of their choice but at each step only those actions pertaining to the current state are made available to them, thereby pruning down the vast number of possible actions and suggesting only the truly feasible and relevant actions. Both of these levels have a simulation where the user plan is executed. Moreover, if the user plan is invalid or fails to achieve the given goal condition then an explanation for the failure is provided in simple English language. This makes it easier for everyone (especially for non-roboticists) to understand the cause of the failure.

ACKNOWLEDGMENTS

This thesis would have not been possible without the help of several people. Firstly, I would like to thank my advisor, Dr. Siddharth Srivastava for his constant advice and suggestions. Thanks to my committee members, Dr. Sharon Hsiao, and Dr. Yu Zhang for their time and advice. Thanks to all my lab-mates for supporting and helping me all the time. Special thanks to Sarath, Pulkit, Kislay, and Naman for their contribution towards my research. Finally, a big thanks to my family for their unconditional love and support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Artificial Intelligence (AI) is the science and engineering of making intelligent machines, especially intelligent computer programs. However, there is no fixed definition to explain what exactly do we mean by "intelligence". Some say that is the simulation of human intelligence processes like learning (the acquisition of information and rules for using the information), reasoning (using rules to reach approximate or definite conclusions) and self-correction while some aim at the creation of intelligence without concern for human characteristics. For my thesis, intelligence can be defined as the capability of an agent to perform an action based on the observations of the environment. Planning is a core part of AI research. It is a key ability for intelligent systems, helping them in increasing their autonomy and flexibility through the construction of a sequence of actions to achieve their desired goals. At the center of the problem of intelligent behavior is the problem of selecting the action to perform next. In AI, three different approaches have been used to address this problem. In the programming-based approach, a controller prescribing an action to do next is given by the programmer, usually in a suitable high-level language. In this approach, the problem is solved by the programmer in his or her head and the solution is expressed as a program or as a collection of rules or behaviors. In the learning-based approach, the controller is not given by a programmer but is learned from experience just like in reinforcement learning. Finally, in the model-based approach, the controller is not learned from experience but is derived automatically from a model of actions, sensors, and goals. Planning techniques have been applied in a variety of tasks including robotics, process planning, autonomous agents, and spacecraft mission control.

1.1 Motivation

This work is focused on model-based methods that require a model of the actions, sensors, and goals and face the computational problem of solving the model- a problem that is computationally intractable even for the simplest models where information is complete and actions are deterministic. For a simple blocks world problem with n blocks, we have more than n! states. My goal with this research is to help those who are new to AI planning and robotics understand the basic concepts of planning without allowing them to write code explicitly. I have tried leveraging the benefits of blocks-based programming style which has recently gained much popularity in the educational field. Anyone can use this system to solve a task like moving a robot in a warehouse or help a robot to place items on shelves by just connecting puzzled shaped blocks. And, if they are unsuccessful in solving the task then an explanation for the failure is provided to them in simple English language. This explanation will help them easily understand about the error. It will provide information that is easily understood, essential and without including its background details. For example, let's take one real-life example of a robot working inside a kitchen. You want him to bring you a cup of coffee and some donuts on a plate at once. But the robot is unable to perform this task as it has only one gripper. In this case, the explanation could be something like "(Bring donuts) could not be performed because the robot's gripper is not empty and the previous action, (Bring coffee) made (free gripper) false".

1.2 Planning Model

The state model M (Geffner and Bonet, 2013) for classical AI planning comprises of:

- a finite and discrete state space S,
- a known initial state $s_0 \in S$,
- a non-empty set $S_G \in S$ of goal states,
- actions $A(s) \subseteq A$ applicable in each state $s \in S$,
- a deterministic state transition function $s' = f(s, a)$ for $a \in A(s)$, and
- a cost function c: $A^* \to [0, \infty]$

Solutions to planning problems are paths from an initial state to a goal state in the transition graph. In this model, a solution or plan is a sequence of applicable actions mapping s_0 into S_G. More precisely, a plan $\pi = a_0, a_1, ..., a_{n-1}$ must generate a state sequence $s_0, s_1, ..., s_n$ such that $\forall a_i \in A(s_i), s_{i+1} \in f(s_i, a_i)$, and $S_G \in s_n$, for $i \in 0, 1, ..., n-1$. The cost of the plan is the sum of the action costs $c(s_i, a_i)$, and a plan is optimal if it has minimum cost over all the possible plans.

1.3 Planning Languages

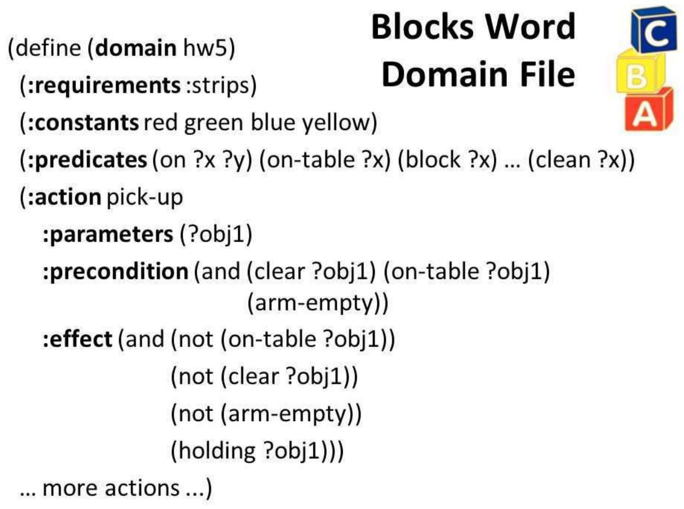
The most commonly used language for representing planning domains and planning problems is the Planning Domain Definition Language (PDDL) (Ghallab *et al.*, 1998). It is based on Standford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971) formalism. In STRIPS, states and actions are specified in terms of propositional state variables also called as grounded atoms.

A state is a conjunction of grounded atoms that define how does the world look now. i.e $At(kitchen) \wedge Empty(Gripper)$. Atoms that are not present are assumed to be false. An action defines how an agent can change the world. It has preconditions and effects. Precondition defines a set of grounded atoms that needs to be true before the action could be applied. Effect defines a set of grounded atoms that will become true and a set of grounded atoms that will no longer remain true once the action has been executed.

PDDL is intended to express the "physics" of a domain, that is, what predicates there are, what are the objects in the environment, what actions are possible and their effects. In PDDL, we define a planning problem using two files, a *domain file* and a *problem file.*

1.3.1 Domain File

A domain file contains the name of the domain, a list of predicates and a list of actions with parameters, preconditions, and effects. Below is an example domain file of the *blocks world* planning problem.



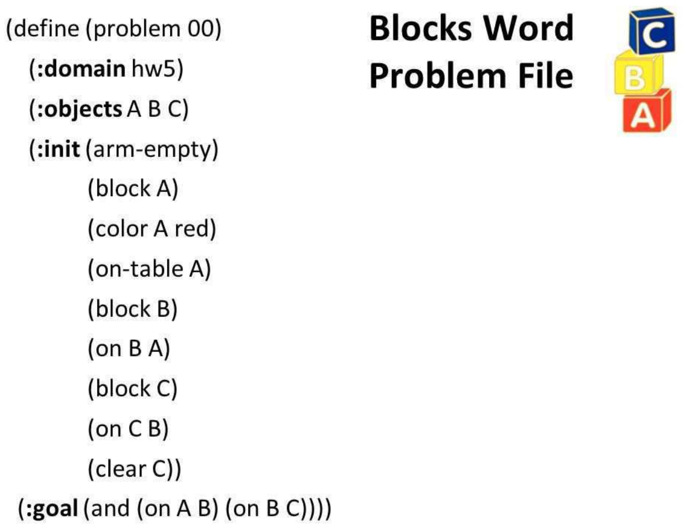
Blocks Word Domain File

```
(define (domain hw5)
 (:requirements :strips)
 (:constants red green blue yellow)
 (:predicates (on ?x ?y) (on-table ?x) (block ?x) ... (clean ?x))
 (:action pick-up
   :parameters (?obj1)
   :precondition (and (clear ?obj1) (on-table ?obj1)
                      (arm-empty))
   :effect (and (not (on-table ?obj1))
                (not (clear ?obj1))
                (not (arm-empty))
                (holding ?obj1)))
 ... more actions ...)
```

Figure 1.1: Blocks World Domain File (Drew McDermott, 2016)

1.3.2 Problem File

A problem file contains the name of the domain, objects in the environment, a description of the initial state (using the predicates listed in the domain file) and description of the goal state (again, using predicates listed in the domain file). Below is an example problem file of the *blocks world* planning problem.



```
(define (problem 00)
   (:domain hw5)
   (:objects A B C)
   (:init (arm-empty)
          (block A)
          (color A red)
          (on-table A)
          (block B)
          (on B A)
          (block C)
          (on C B)
          (clear C))
  (:goal (and (on A B) (on B C))))
```

Blocks Word
Problem File
C
B
A

Figure 1.2: Blocks World Problem File (Drew McDermott, 2016)

1.4 Block-Based Programming

Block-based programming, sometimes known as block-based coding, is coding within a programming language where instructions are mainly represented as blocks. If a language does not involve blocks as the main part of its programming language but instead is mostly oriented around text, then it is known as a text-based language. Both text-based and block-based languages have advantages and disadvantages. Block-based languages usually have a set of commands to choose from, preventing the need to memorize them, and are therefore easy for the beginners to start coding.

However, it may be slower to code in a block-based language than one where a user can quickly input commands using their keyboard. On the other end, syntax errors are common in text-based languages. This could be frustrating for those transitioning from block to text.

Recently, block-based programming has been the de facto way to teach kids introductory programming in the US. Instead of traditional, text-based programming, block-based coding involves dragging and dropping "blocks" of instructions. The most popular, by far, the block-based app is Scratch (Resnick *et al.*, 2009). As you can see in the picture below, you drag blocks of instructions on the left into the editor on the right!

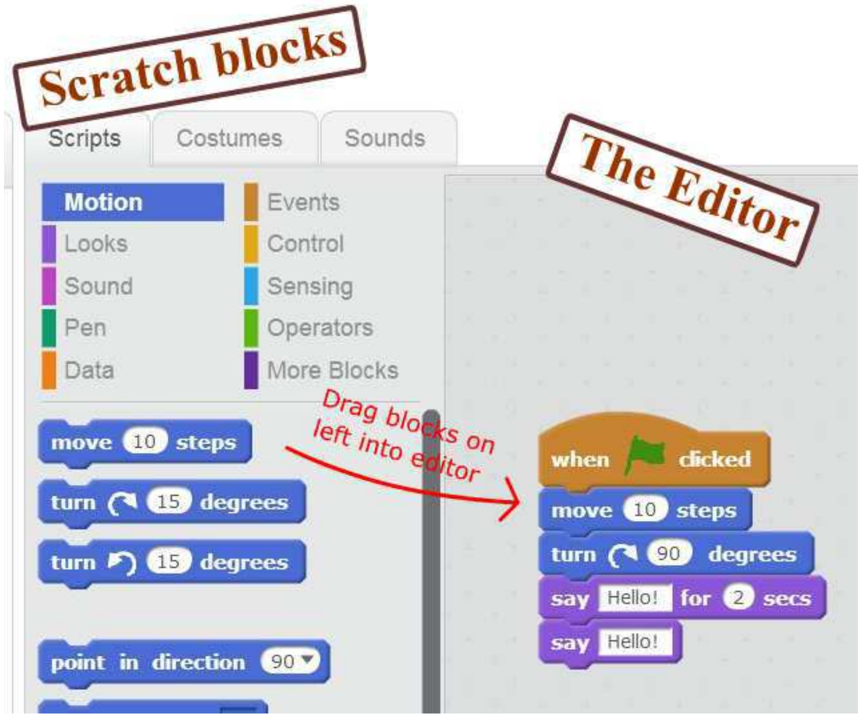


Figure 1.3: Scratch App (Resnick *et al.*, 2009)

Chapter 2

RELATED WORK

Roblocks is motivated from the work done in Code3 (Huang and Cakmak, 2017), an end-to-end system for programming mobile manipulator robots for novices and experts. Code3 integrates components for perception, manipulation, and high-level programming. The perception component helps users define a library of object and scene parts that the robot can later detect. The manipulation component lets users define actions for manipulating objects or scene parts through programming by demonstration. Finally, the high-level programming component provides a drag-and-drop interface with which users can program the logic and control flow to accomplish a task using their previously specified perception and manipulation capabilities. However, their system lacks in addressing explanations or possible reasons for plan failures.

Some similar systems like ROS Commander (Nguyen *et al.*, 2013) and RoboFlow (Alexandrova *et al.*, 2015) have integrated robot actions, including those programmed by demonstration, with a visual programming language using a data flow model (Hils, 1992). The authors from the above works have pointed out that in some cases these flow-based interfaces do not scale well, especially when the state space is large. In contrast, Roblocks uses simple HTML and Javascript for developing the control flow which makes it robust, simple and easy to play with.

Others have designed robot programming interfaces for creating social interactions. These include the TiViPE (Lourens, 2004), Choregraphe (Pot *et al.*, 2009), and Interaction Blocks (Sauppé and Mutlu, 2014) interfaces for programming the Nao robot. All of them have combined flow-based programming interfaces with support for timing, social dialogue, and gestures.

RoboStudio (Datta *et al.*, 2012) is a system for authoring UI and control flow for healthcare robots. However, Roblocks focuses on programming manipulation tasks, such as fetching and carrying, delivering objects, and manipulating the environment. It provides explanations for plan failure in simple English language which helps anyone to understand the cause of the failure very easily.

Chapter 3

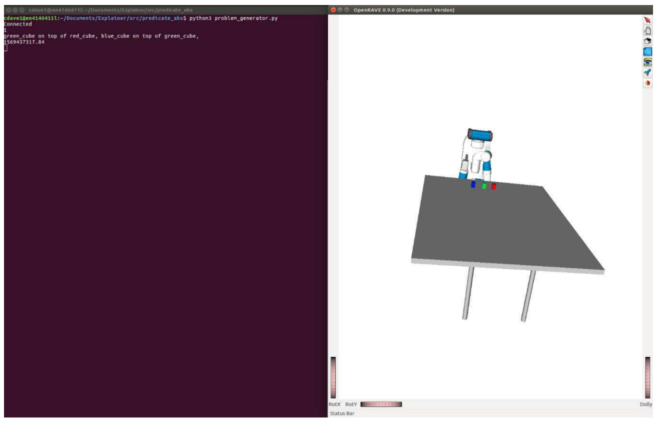
SYSTEM OVERVIEW

3.1 Components Overview

There are four main components of the system - Problem Generator, GUI, Feedback Generator, and Plan Executer.

3.1.1 Problem Generator

Problem Generator generates a problem instance randomly for a given domain. The randomness comes in the form of different initial state or goal state or both. It creates three copies of the problem file, one each for the GUI, Feedback Generator and Plan Executor component and an environment file for executing the plan in a simulation. Once all these files are generated, it calls the GUI component which will then start preparing the action blocks. Below is a picture of the Problem Generator component.



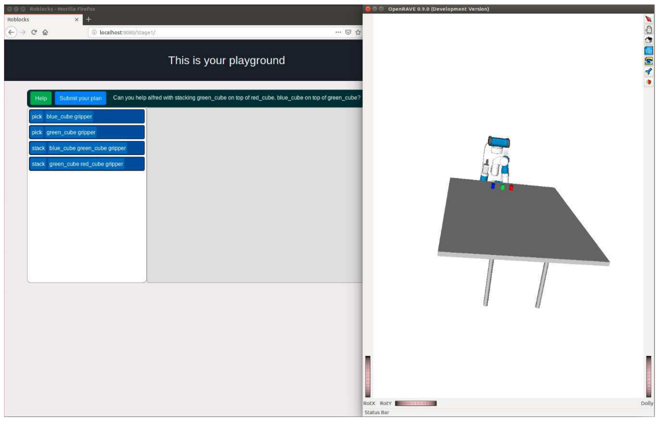
```
cdave1@en4146411l:~/Documents/Explainer/src/predicate_abs$ python3 problem_generator.py
Connected
1
green_cube on top of red_cube, blue_cube on top of green_cube,
1569437317.84
```

cdave1@en4146411l: ~/Documents/Explainer/src/predicate_abs
OpenRAVE 0.9.0 (Development Version)
RotX
RotY
Dolly
Status Bar

Figure 3.1: Problem Generator

3.1.2 GUI

GUI takes the domain file and the problem file generated by the problem generator component as inputs and creates action blocks and the problem question for the front-end. An end-user will then create a plan using these action blocks to achieve the goal condition. Once the plan is submitted, it will be processed and verified by the Feedback Generator component. Below is a picture of the GUI component.



Can you help alfred with stacking green_cube on top of red_cube, blue_cube on top of green_cube?

pick blue_cube gripper
pick green_cube gripper
stack blue_cube green_cube gripper
stack green_cube red_cube gripper

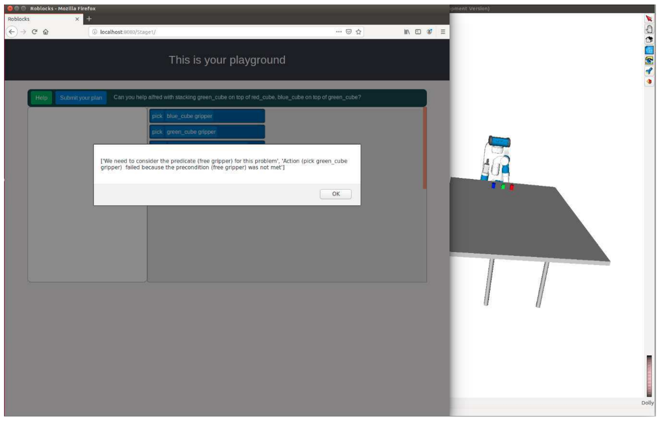
Figure 3.2: GUI

The process of generating blocks differs for the two stages of the system. For stage 1, the Fast Forward (FF) (Hoffmann, 2001) planner is run on the domain and the problem file to get a valid plan. Once the plan is generated, it is sent to the front-end where all the actions in the plan are jumbled up and displayed to the end-user. The task for the end-user is to arrange these jumbled actions into its correct sequence. Again for stage 2, at every step the FF planner is run to get all the feasible actions which can be applied in the current state. The user has to select an action from this list of feasible actions. Once the user is done selecting an action, a list of parameter configurations is displayed corresponding to the selected action. The user then has to select one of the configuration from this list of parameter configurations.

Finally, the selected action along with the selected parameter configuration is applied to the current state to get the next state. The task for the end-user is to reach the goal state.

3.1.3 Feedback Generator

Feedback Generator validates the plan submitted by an end-user from the GUI. It creates a lattice of abstract models where the top-most node is the most abstract model and the bottom-most node is the actual or most concrete model. It then tries to search the most abstract model that can explain the failure in the user plan. This makes the feedback generation cost-effective because the deeper you go down the lattice the costlier it is. Moreover, feedback is generated based on the user's level of understanding and capability. There will be a different scale of explanation based on user knowledge. The feedback typically includes the information of an unmet precondition in the failed action along with an entire trace of all the previous actions within the plan that affected the value of that precondition. All of the above information is presented in simple English language which makes it easier for the end-user to understand the root cause of the failure and simplifies the debugging. However, if the plan is valid then it sends the plan to the plan executor component which executes it on a simulation.
The explanation system is based on the work done on user-specific contrastive explanations by Sreedharan *et al.* (2018). Below is a picture of the Feedback Generator component.



Can you help alfred with stacking green_cube on top of red_cube, blue_cube on top of green_cube?

['We need to consider the predicate (free gripper) for this problem', 'Action (pick green_cube gripper) failed because the precondition (free gripper) was not met']

Figure 3.3: Feedback Generator

3.1.4 Plan Executor

Plan Executor executes the plan submitted by the Feedback Generator component in OpenRAVE (Diankov and Kuffner, 2008) which is a very famous robotics tool that provides an environment for testing, developing, and deploying motion planning algorithms in real-world robotics applications. It uses the Task & Motion Planning (TMP) framework which is being developed and maintained at Autonomous Agents Intelligent Robots (AAIR) lab for executing the user plan in the simulation.

It takes the plan, environment file, and action-specific pose generator scripts as inputs and runs the submitted plan. The pose generator scripts contain motion planning information for every action specified in the domain file. Below are some pictures showing plan execution in OpenRAVE using the TMP framework.

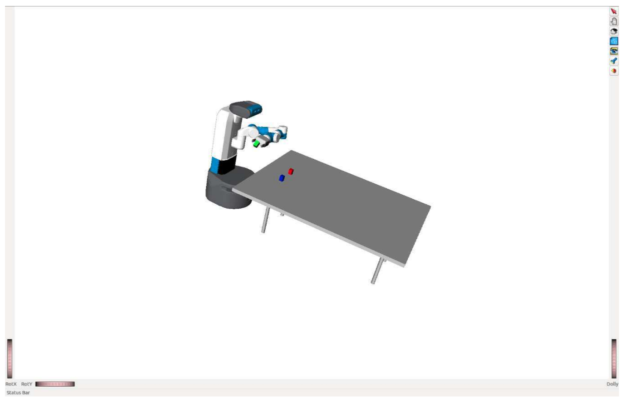


Figure 3.4: Robot Performing First Action (picking green cube) in the Plan

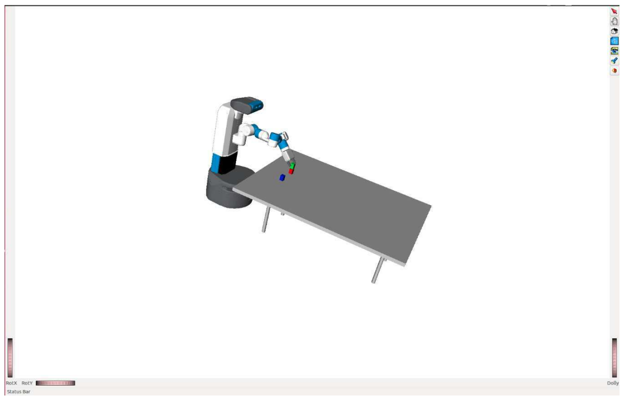


Figure 3.5: Robot Performing Second Action (stacking green cube on red cube) in the Plan

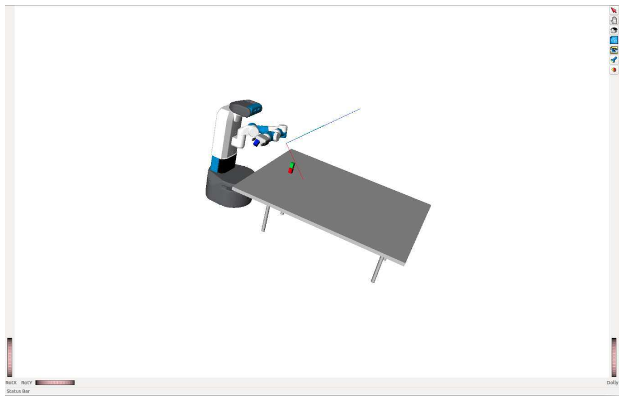


Figure 3.6: Robot Performing Third Action (picking blue cube) in the Plan

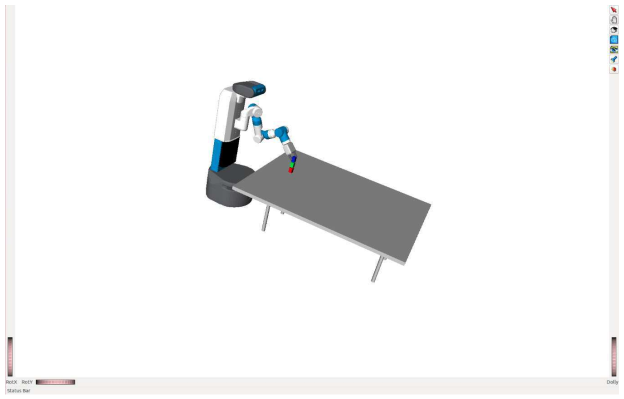


Figure 3.7: Robot Performing Last Action (stacking blue cube on green cube) in the Plan

3.2 System Pipeline

Below is a block diagram showing the architecture of the entire system:

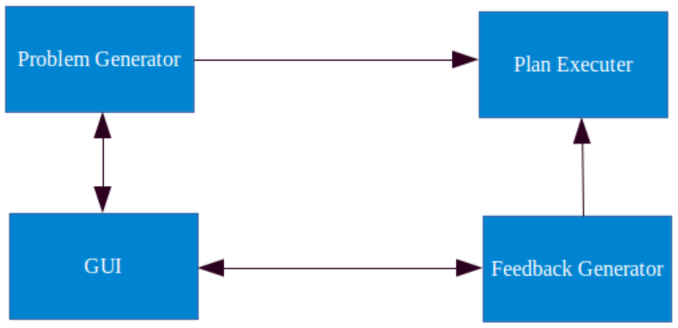


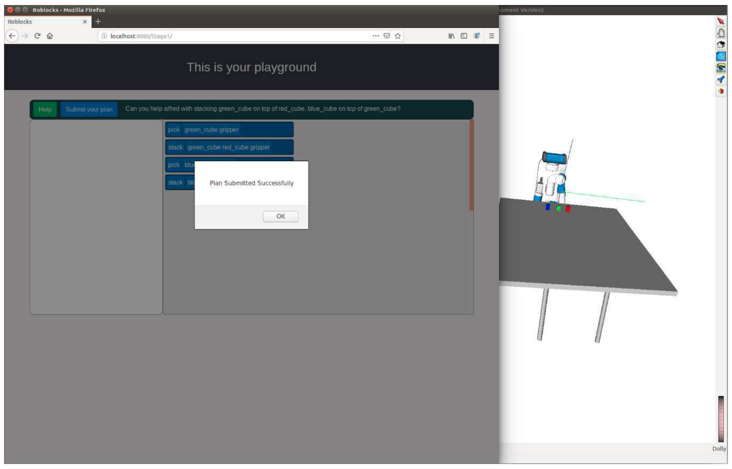
Figure 3.8: System Architecture

3.3 System Levels Overview

There are two levels in Roblocks, one where a user has to solve a puzzle of actions to get the actual plan and second where they need to come up with a valid plan. The two levels are described below:

3.3.1 Level 1

In level 1, an end-user is shown a set of actions of a valid plan, all jumbled up in some random order. The goal of the end-user is to arrange this set of jumbled actions in the correct order so that a valid plan could be formed. Once the plan is submitted, it is verified and if it is valid, it is executed on a simulation. The objective here is to explain end-users about predicates, actions, preconditions, and plans. This stage makes the use of the Feedback Generator component to explain these concepts. If the user plan is invalid or fails to achieve a given goal condition, an explanation is provided for the failure. It shows all the missing facts from the goal condition on the GUI when the user plan fails in achieving the goal condition. And, if the plan fails because of an action that could not be performed then it shows one of the preconditions from the failed action which was not met along with the entire trace of all the previous actions within the plan that affected the value of this precondition. All of the above information is presented in simple English language which makes it easier for the end-user to understand the root cause of the failure and simplifies the debugging.



This is your playground

Can you help alfred with stacking green_cube on top of red_cube, blue_cube on top of green_cube?

pick green_cube gripper

stack green_cube red_cube gripper

pick blu

stack bl

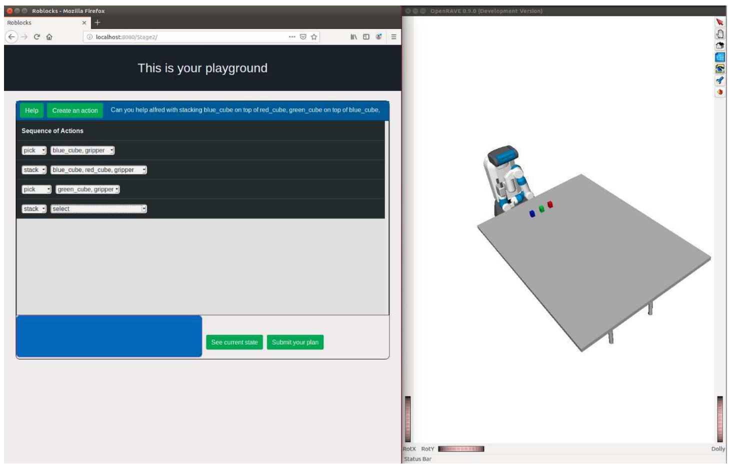
Plan Submitted Successfully

Figure 3.9: Level 1 of Roblocks

3.3.2 Level 2

In level 2, an end-user has to come up with a valid plan to solve a particular problem. At every step, the FF planner is run to get all the feasible actions that can be applied in the current state. The user has to select an action from this list of feasible actions. This way, the system prunes down the vast number of possible actions and suggests only the truly feasible and relevant actions. Once the user is done selecting an action, a list of parameter configurations is displayed corresponding to the selected action. The user then has to select one of the configurations from this list of parameter configurations. Finally, the selected action along with the selected parameter configuration is applied to the current state to get the next state. The task for the end-user is to reach the given goal state. The objective here is to help end-users understand how to come up with the best plan and how the pruning of the vast search space is beneficial to reach the goal state as fast as possible. Again, if the goal state is reached then the plan is executed on a simulation. However, if the goal state is not achieved then the system shows all the missing facts in the goal condition on the GUI.

All of the above information is presented in simple English language which makes it easier for the end-user to understand the root cause of the failure and simplifies the debugging.



This is your playground

Can you help alfred with stacking blue_cube on top of red_cube, green_cube on top of blue_cube,

Sequence of Actions

pick | blue_cube, gripper
stack | blue_cube, red_cube, gripper
pick | green_cube, gripper
stack | select

Figure 3.10: Level 2 of Roblocks

3.4 System Evaluation

For evaluating the system, ten users having no idea about AI planning and robotics tried using the system. The only information given to them was the meaning of every action in the problem domain. Both the levels included tasks where the users had to arrange some blocks in a given order. All the users had to clear level 1 before moving to the level 2. Also, there were 2 rounds (level 1 + level 2) of experiment, first where no feedback was provided to them and second when feedback was provided to them. After completing each round, they provided their feedback on a 5-scale Likert chart. This chart measured how did they feel about using the system, how easy or difficult it was to use the system. Along with this, the time taken for completing each stage was also recorded.

Below are the three figures, one showing the average value on the 5-scale Likert chart and the other two showing the average time taken to complete each stage with and without the feedback component.

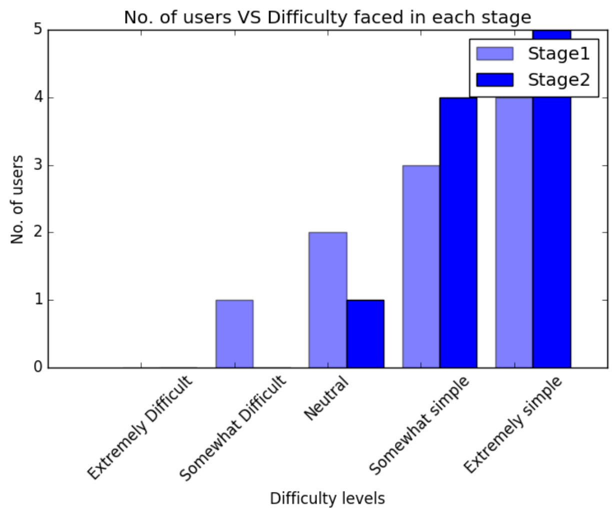


Figure 3.11: No. of Users VS Difficulty Faced in Each Stage.

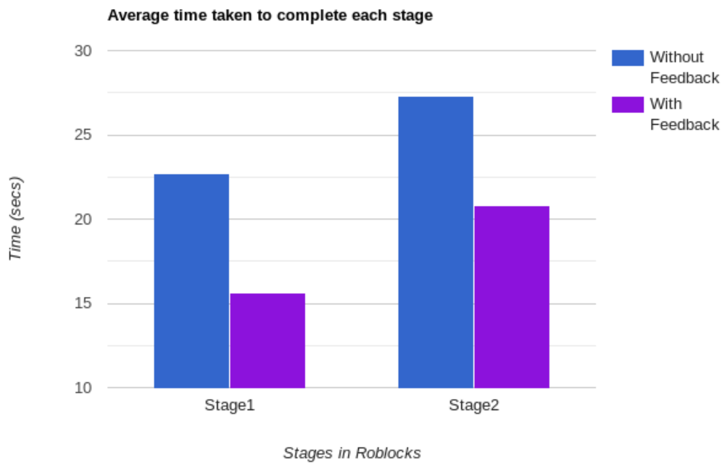


Figure 3.12: Average time taken to complete both the stages with and without the feedback component.

Chapter 4

CONCLUSION

The primary contribution of this thesis is the introduction of a new tool to make AI planning concepts easier to grasp. Roblocks as a system has many applications and can be used in various domains like education, industrial warehouses, and research. Primarily it is built for the educational purpose where it can be used as a tool to teach technical concepts of AI planning like predicates, actions, preconditions, states and how to write domain and problem files in PDDL. However, it can be of real help in industrial and domestic environments. i.e. Amazon warehouse where the workers might need to control a robot but they don't have any knowledge about robotics. Roblocks can help them to control a robot by just connecting puzzled shaped blocks and only showing those actions which could be performed on the current state of the robot and if there are any issues with the robot or if the robot is unable to carry out an action then the explanation component can explain them about the cause in simple English language so that they can understand it very easily. Explanations play a vital role in these kinds of situations where we need to explain things based on user's capability and understanding.

It can also be used for research purposes where one can use it to create a data set of plans and places where one would like to replace complex robotics control system with a simple user interface.

4.1 Future Work

A larger user study needs to be done to evaluate the true potential or the educational efficacy of the system. Two types of studies need to be done, one where feedback is not given to the end-user and second where feedback is given to the end-user. In both the studies, end-users should be randomly given different levels to start from and there should be separate training and test data set.

The reason for having randomness in the levels to start with is to prevent experience learned from one level affecting the performance on the other level and having separate studies with and without the feedback module can help us to understand the impact of the feedback generation.

Additionally, It would be very interesting to see how to add more advanced explanation capabilities into the system while keeping the system very intuitive to use. These explanations should have support for explaining more about predicates and operator schemata. Moreover, it will be a nice opportunity to see how it could be used to create PDDL files (domain and problem) by using blocks representing facts in the current state to fill out the preconditions and effects for all the actions. A more significant thing would be to see how the system can be used to represent policies that can then help to solve many real-world problems where action can only be taken after observing the surroundings.

4.2 Acknowledgements

I would like to thank Sarath Sreedharan for providing me with the support for the explanation module and Naman P Shah for providing me with the support for the plan execution module.

REFERENCES

Alexandrova, S., Z. Tatlock and M. Cakmak, “Roboflow: A flow-based visual programming language for mobile manipulation tasks”, 2015 IEEE International Conference on Robotics and Automation (ICRA) pp. 5537–5544 (2015).

Datta, C., C. Jayawardena, I.-H. Kuo and B. Macdonald, “Robostudio: A visual programming environment for rapid authoring and customization of complex services on a personal service robot”, pp. 2352–2357 (2012).

Diankov, R. and J. Kuffner, “Openrave: A planning architecture for autonomous robotics”, Tech. Rep. CMU-RI-TR-08-34, Carnegie Mellon University, Pittsburgh, PA (2008).

Drew McDermott, “Hw5: Pddl planning domain description language based on strips”, (2016).

Fikes, R. E. and N. J. Nilsson, “Strips: A new approach to the application of theorem proving to problem solving”, in “Proceedings of the 2Nd International Joint Conference on Artificial Intelligence”, IJCAI’71, pp. 608–620 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1971), URL `http://dl.acm.org/citation.cfm?id=1622876.1622939`.

Geffner, H. and B. Bonet, *A Concise Introduction to Models and Methods for Automated Planning: Synthesis Lectures on Artificial Intelligence and Machine Learning* (Morgan & Claypool Publishers, 2013), 1st edn.

Ghallab, M., C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun and D. Weld, “Pddl - the planning domain definition language”, (1998).

Hils, D. D., “Visual languages and computing survey: Data flow visual programming languages”, J. Vis. Lang. Comput. **3**, 69–101 (1992).

Hoffmann, J., “Ff: The fast-forward planning system”, AI Magazine **22**, 57–62 (2001).

Huang, J. and M. Cakmak, “Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts”, 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI pp. 453–462 (2017).

Lourens, T., “Tivipe ” tino’s visual programming environment”, in “Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01”, COMPSAC ’04, pp. 10–15 (IEEE Computer Society, Washington, DC, USA, 2004), URL `http://dl.acm.org/citation.cfm?id=1025117.1025469`.

Nguyen, H., M. T. Ciocarlie, K. Hsiao and C. C. Kemp, “Ros commander (rosco): Behavior creation for home robots”, 2013 IEEE International Conference on Robotics and Automation pp. 467–474 (2013).

Pot, E., J. Monceaux, R. Gelin and B. Maisonnier, “Choregraphe: a graphical tool for humanoid robot programming”, pp. 46 – 51 (2009).

Resnick, M., J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman and Y. Kafai, “Scratch: Programming for all”, Commun. ACM **52**, 11, 60–67, URL `http://doi.acm.org/10.1145/1592761.1592779` (2009).

Sauppé, A. and B. Mutlu, “Design patterns for exploring and prototyping human-robot interactions”, in “CHI”, (2014).

Sreedharan, S., S. Srivastava and S. Kambhampati, “Hierarchical expertise level modeling for user specific contrastive explanations”, in “Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18”, pp. 4829–4836 (International Joint Conferences on Artificial Intelligence Organization, 2018), URL `https://doi.org/10.24963/ijcai.2018/671`.