

Assignment 3

DS685

A.I. for Robotics

Yash Patel

yp359

31702347

1. Introduction

This assignment focused on converting a visual robot demonstration into symbolic planning models using PDDL (Planning Domain Definition Language). The idea was to simulate how a robot observes a task and then expresses the task in terms of objects, predicates, and actions that a planner can reason about.

Because my Gazebo simulator was not working correctly, I created simple synthetic frames that mimic the demonstration. These frames were passed through a Visual Language Model (VLM) to extract natural-language descriptions. From those descriptions, I constructed a PDDL domain and three problem files, and then validated them using the Unified Planning library.

The overall workflow ends up being very similar to the “Video2Plan” idea: starting with images, generating captions, interpreting symbolic relationships, and producing a planning formulation.

2. Generating the Demonstration Frames

To represent the demonstration, I created two simple images using Python’s Pillow library:

- **initial.png** – showing a red block and a blue block placed separately on a white surface
- **final.png** – showing the same two blocks, but with the red block now stacked on top of the blue block

Both images were stored inside:

/overlay_ws/video2plan/frames

```
(venv) root@Rag:/overlay_ws/video2plan# python3 - << 'EOF'
from PIL import Image, ImageDraw

# Frame 1: Two blocks on table
img1 = Image.new("RGB", (400, 300), "white")
d1 = ImageDraw.Draw(img1)
d1.rectangle([80, 150, 160, 230], fill="red") # block A
d1.rectangle([240, 150, 320, 230], fill="blue") # block B
img1.save("/overlay_ws/video2plan/frames/initial.png")

# Frame 2: Red block stacked on blue block
img2 = Image.new("RGB", (400, 300), "white")
d2 = ImageDraw.Draw(img2)
d2.rectangle([240, 150, 320, 230], fill="blue") # block B bottom
d2.rectangle([240, 80, 320, 160], fill="red") # block A top
img2.save("/overlay_ws/video2plan/frames/final.png")

print("Synthetic frames generated successfully!")
EOF
Synthetic frames generated successfully!
(venv) root@Rag:/overlay_ws/video2plan# cd frames
(venv) root@Rag:/overlay_ws/video2plan/frames# ls
final.png  initial.png
(venv) root@Rag:/overlay_ws/video2plan/frames# |
```


- The white region → **surface (table)**
- The robot must have a **gripper**
- The relationship between blocks → **stacked, on table, clear**, etc.

From this, I formulated the necessary predicates:

To express all this information symbolically, I settled on the following predicates:

- (on x y) – block *x* is on block *y*
- (on table x s) – block *x* is on surface *s*
- (clear x) – block *x* has nothing on top
- (handempty g) – the robot's gripper *g* is empty
- (holding g x) – the gripper *g* is holding block *x*
- (reachable g x) – gripper *g* can reach block *x*
- (reachablesurf g s) – gripper *g* can reach surface *s*

These predicates capture everything needed for the planner to reason about a simple block-stacking task.

These predicates describe exactly what conditions matter for a simple block-stacking task.

5. Designing the PDDL Domain

The domain file (domain.pddl) defines:

Types

block, surface, gripper

Predicates

These include all the relationships mentioned in the previous section.

Actions

To make the domain functional, I added four actions:

1. **pick** – picking up a block from a surface
2. **place** – placing a block back on a surface
3. **stack** – placing a block on top of another block

4. **unstack** – removing a block from another block

Each action has carefully written preconditions and effects so a planner can simulate each state transition. A screenshot of the domain.pddl file confirms the correct structure and formatting.

```
(venv) root@Rag:/overlay_ws/video2plan# nano domain.pddl
(venv) root@Rag:/overlay_ws/video2plan# ls
domain.pddl  frames  problems  scripts  venv
(venv) root@Rag:/overlay_ws/video2plan# |
```

```
(define (domain robot_blocks)
  (:requirements :typing :strips)
  (:types block surface gripper)

  (:predicates
    (on ?x - block ?y - block)
    (ontable ?x - block ?s - surface)
    (clear ?x - block)
    (holding ?g - gripper ?x - block)
    (handempty ?g - gripper)
    (reachable ?g - gripper ?x - block)
    (reachablesurf ?g - gripper ?s - surface)
  )

  (:action pick
    :parameters (?g - gripper ?x - block ?s - surface)
    :precondition (and (handempty ?g) (ontable ?x ?s) (clear ?x)
                       (reachable ?g ?x) (reachablesurf ?g ?s))
    :effect (and (holding ?g ?x)
                 (not (handempty ?g))
                 (not (ontable ?x ?s))))

  (:action place
    :parameters (?g - gripper ?x - block ?s - surface)
    :precondition (and (holding ?g ?x) (reachablesurf ?g ?s))
    :effect (and (handempty ?g)
```

Fig. domain.pddl

6. Creating Multiple PDDL Problem Files

The assignment required at least three problem files. I created:

- **problem1.pddl**: two blocks on a table; goal: stack red on blue
- **problem2.pddl**: three blocks; goal: form a two-level stack
- **problem3.pddl**: four blocks across two tables; goal: create a taller stack

Each problem file includes:

- A list of objects
- The initial scene configuration
- Reachability assumptions for the robot gripper
- A goal description that represents the final stacked configuration

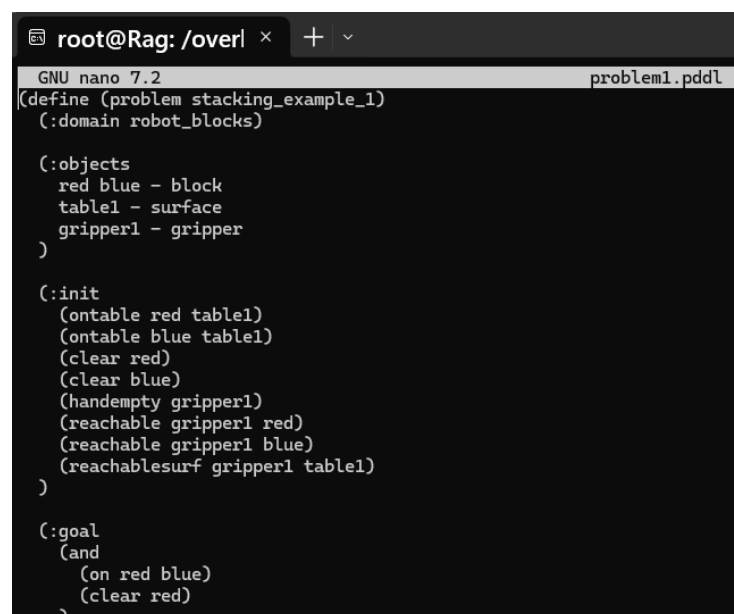
These files were saved in:

/overlay_ws/video2plan/problems

A screenshot shows all three files present.

```
(venv) root@Rag:/overlay_ws/video2plan/problems# ls
problem1.pddl  problem2.pddl  problem3.pddl
(venv) root@Rag:/overlay_ws/video2plan/problems# |
```

Fig. Problems list



```
GNU nano 7.2 problem1.pddl
(define (problem stacking_example_1)
  (:domain robot_blocks)

  (:objects
    red blue - block
    table1 - surface
    gripper1 - gripper
  )

  (:init
    (ontable red table1)
    (ontable blue table1)
    (clear red)
    (clear blue)
    (handempty gripper1)
    (reachable gripper1 red)
    (reachable gripper1 blue)
    (reachablesurf gripper1 table1)
  )

  (:goal
    (and
      (on red blue)
      (clear red)
    )
  )
)
```

Fig . Problem1.pddl

```
root@Rag: /overl x + v
GNU nano 7.2 problem2.pddl
(define (problem stacking_example_2)
  (:domain robot_blocks)

  (:objects
    red blue green - block
    table1 - surface
    gripper1 - gripper
  )

  (:init
    (ontable red table1)
    (ontable blue table1)
    (ontable green table1)
    (clear red)
    (clear blue)
    (clear green)
    (handempty gripper1)
    (reachable gripper1 red)
    (reachable gripper1 blue)
    (reachable gripper1 green)
    (reachablesurf gripper1 table1)
  )

  (:goal
    (and
      (on red blue)
    )
  )
)
```

Fig. Problem2.pddl

```
root@Rag: /overl x + v
GNU nano 7.2 problem3.pddl

(:init
  (ontable red table1)
  (ontable blue table1)
  (ontable green table2)
  (ontable yellow table2)
  (clear red)
  (clear blue)
  (clear green)
  (clear yellow)
  (handempty gripper1)
  (reachable gripper1 red)
  (reachable gripper1 blue)
  (reachable gripper1 green)
  (reachable gripper1 yellow)
  (reachablesurf gripper1 table1)
  (reachablesurf gripper1 table2)
)

(:goal
  (and
    (on red blue)
    (on blue green)
    (on green yellow)
    (clear red)
  )
)
```

Fig.Problem3.pddl

7. Validating the PDDL Using Unified Planning

I used the Unified Planning library to confirm that:

- The domain is syntactically valid
- Each problem file is compatible with the domain

Using the PDDLReader class (following the official Unified Planning notebook examples), I parsed each file in sequence:

```
parse_problem(domain_file, problem_file)
```

Once the arguments were passed in the correct order, all files loaded successfully:

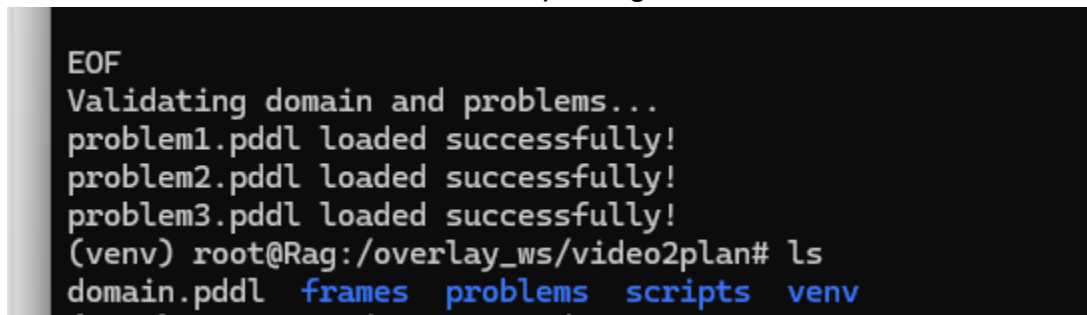
```
problem1.pddl loaded successfully!
```

```
problem2.pddl loaded successfully!
```

```
problem3.pddl loaded successfully!
```

This validation step is essential because it guarantees that the symbolic model is well-formed and ready for planners.

A screenshot confirms the successful parsing.

A terminal window with a dark background and light-colored text. The text shows the output of a script: 'EOF', 'Validating domain and problems...', 'problem1.pddl loaded successfully!', 'problem2.pddl loaded successfully!', 'problem3.pddl loaded successfully!', and a shell prompt '(venv) root@Rag:/overlay_ws/video2plan# ls' followed by the command 'ls' and its output 'domain.pddl frames problems scripts venv' where the directory names are highlighted in blue.

```
EOF
Validating domain and problems...
problem1.pddl loaded successfully!
problem2.pddl loaded successfully!
problem3.pddl loaded successfully!
(venv) root@Rag:/overlay_ws/video2plan# ls
domain.pddl frames problems scripts venv
```

8. Final Result

By combining synthetic images, VLM captioning, interpreted symbolic relations, and PDDL modeling, I completed a full demonstration-to-planning pipeline:

1. Created demonstration frames
2. Extracted natural-language descriptions using BLIP
3. Mapped the scene to symbolic predicates
4. Wrote a PDDL domain describing actions and relationships
5. Created multiple problem files representing different tasks
6. Validated everything using the Unified Planning library

All required screenshots have been collected to document each step.

Conclusion

This exercise helped me connect the visual side of robotics with symbolic planning. By turning simple images into predicates, defining a domain, and validating multiple problem files, I got a practical sense of how a planner reasons about actions. Despite a few formatting issues along the way, the full pipeline worked smoothly once everything was aligned. Overall, it was a useful introduction to building planning models directly from visual input.