ECEN-524 Robot Learning: Human Demonstrations

Physical AI-Driven Action Sequence Analysis and MDP Modeling Using NVIDIA Cosmos Reason

Anthony Angeles
Santa Clara University

February 19, 2026

Abstract

This project presents a physical AI system designed to teach a robot a sequence of object manipulation actions from video demonstrations. Using a dataset of recorded human demonstrations involving colored dice arranged in a target configuration, the system addresses two core challenges: recognizing objects and actions from video, and automatically generating structured action sequences. For object detection, NVIDIA's Cosmos Reason 8B vision-language model was integrated alongside Grounding DINO for zero-shot detection and SAM 2 for video segmentation and tracking, with HSV color space analysis used to classify dice by color. Action recognition was performed through natural language prompting of Cosmos Reason 8B, identifying discrete manipulation primitives such as reaching, grasping, lifting, moving, and releasing at 4 FPS temporal resolution. From these recognized actions, structured JSON-formatted action sequences were generated with formal preconditions and effects to capture state dependencies. The manipulation task was further formalized as a Markov Decision Process (MDP), defining nine states, eight actions, stochastic transition probabilities, and a reward function balancing task success with step efficiency. The resulting system provides an end-to-end pipeline from raw video to formal task representation for robotic planning and reinforcement learning, with limitations discussed in the context of occlusion handling, temporal resolution, and scalability to more complex multi-object scenarios.

1 Introduction

The target task involves rearranging a set of colored dice from a disorganized starting configuration into a specific ordered row. While simple in appearance, this task captures the essential challenges of real-world manipulation: detecting and distinguishing objects, recognizing the sequence of actions a human takes, understanding the logical dependencies between those actions, and modeling the task formally enough for a robot to reason about it. There are many methods that can be used to solve this problem, but in our implementation I opted for the use of reasoning models to help reach our solution.

To address this, the project is structured around two major goals. The first is object and action recognition from raw video, identifying what objects are present and what the human demonstrator is doing at each moment. The second is the automatic generation of structured action sequences and a formal Markov Decision Process representation, which together provide the kind of task model a robotic system would need to plan and execute the demonstrated behavior. By combining state-of-the-art vision-language models with classical task formalization techniques, I hope to reach an adequate result that is sufficiently apt to then be used in robots if one were to futher exapnd on this project.

2 Data Gathering

Data collection took place on February 4th in SCDI 4004, where the class performed single-handed object manipulation tasks on camera to build the demonstration dataset. Each participant recorded their own unique sequence of actions, deliberately avoiding replication of others' orderings to maximize variability across the dataset. During recording, participants were instructed to manipulate only one object at a time and to make hand gestures clearly visible, particularly during grasping and releasing actions, to ensure action boundaries were distinguishable in the footage. The demonstrations were captured in `.mov` format and centered on a tabletop workspace containing colored dice red, green, and blue which served as the manipulation objects. The goal of each demonstration was to transition the scene from a messy starting configuration to a structured target arrangement with the dice lined up in a row.

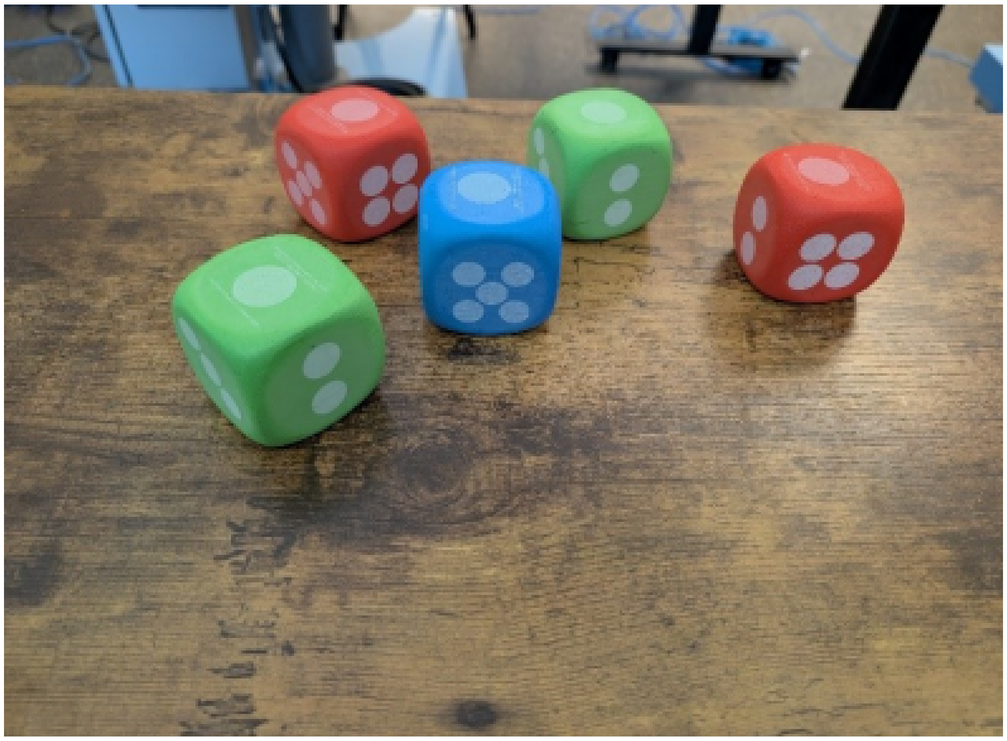

Figure 1: Example of starting environment.

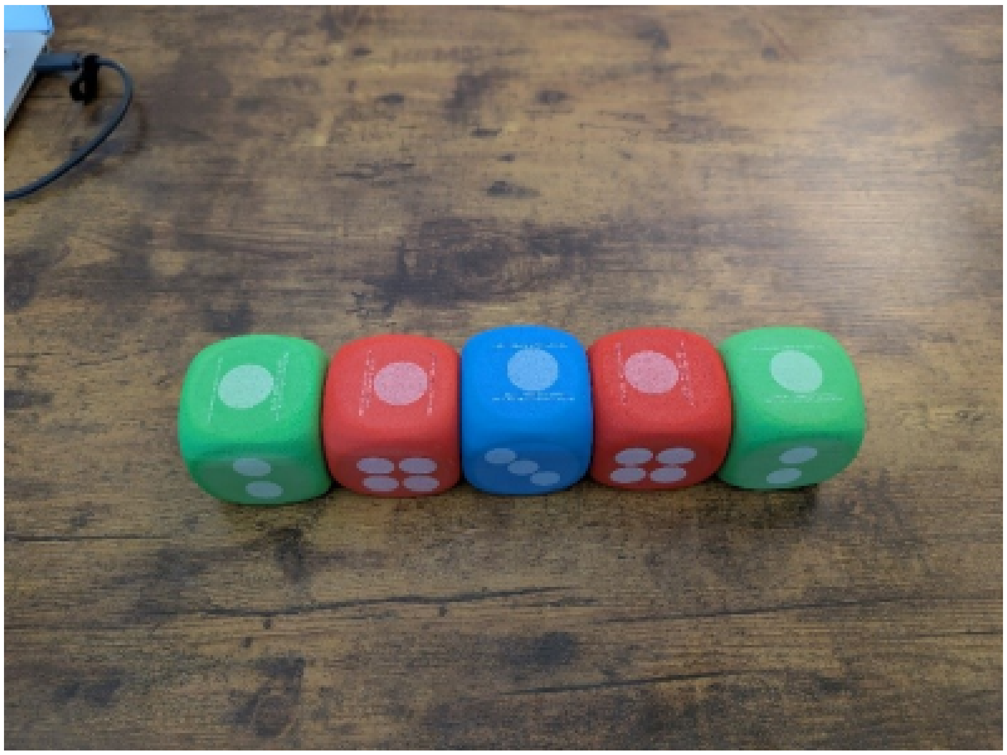

Figure 2: Example of end goal environment.

This raw video data served as the foundation for every subsequent stage of the project. The recorded demonstrations were first passed through the object detection and action recognition pipeline to identify the dice and label the manipulation actions frame by frame. These labeled sequences were then used to construct structured action representations with formal preconditions and effects. Finally, patterns observed across all demonstrations informed the design of the MDP, where recurring state transitions and action frequencies guided the definition of states, actions, transition probabilities, and reward values. In this way, the collected data drove both the recognition and formalization components of the system from start to finish.

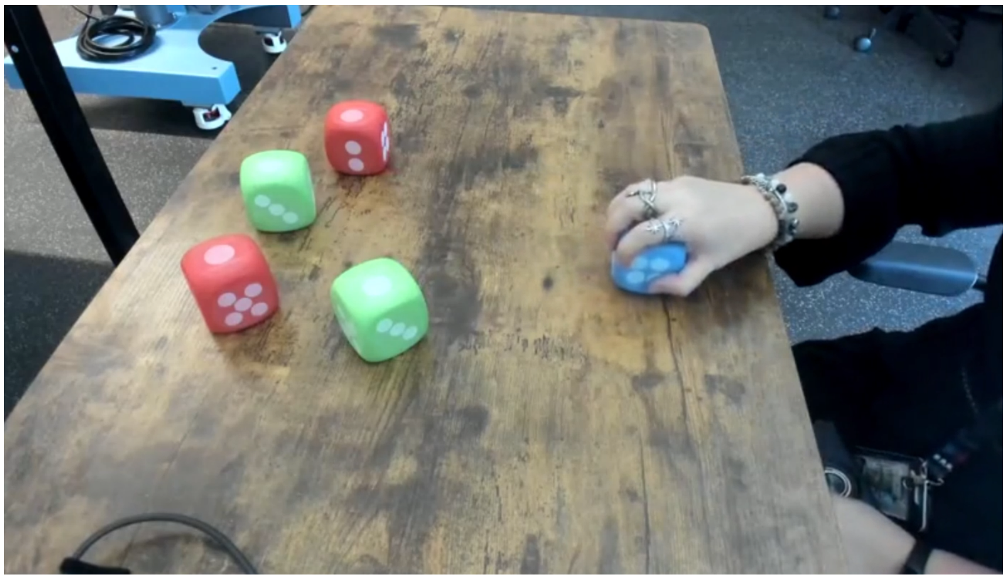
Figure 3: Frame of Raw Demonstration Video.

3 Object and Human Action Recognition

3.1 Object Detection

To detect objects across all demonstration videos, we employed a multi-model pipeline combining Grounding DINO, SAM 2, and NVIDIA Cosmos Reason 8B. The process began by loading the pretrained Grounding DINO model from Hugging Face (`IDEA-Research/grounding-dino-base`), a zero-shot object detection model capable of identifying objects described through natural language prompts. For each video, the first frame was extracted and passed to Grounding DINO using a text prompt specifying the target objects: `"green dice . red dice . blue dice . green cube . red cube . blue cube."` The model returned bounding boxes with associated confidence scores and labels for any detected dice above a threshold of 0.20.

Initially, MediaPipe's hand detection model was explored as a candidate for tracking both hands and objects within the scene. However, after implementation and comparative testing, SAM 2 consistently outperformed MediaPipe across both tasks. Where MediaPipe required separate models for hand and object detection, SAM 2 unified these into a single pipeline.

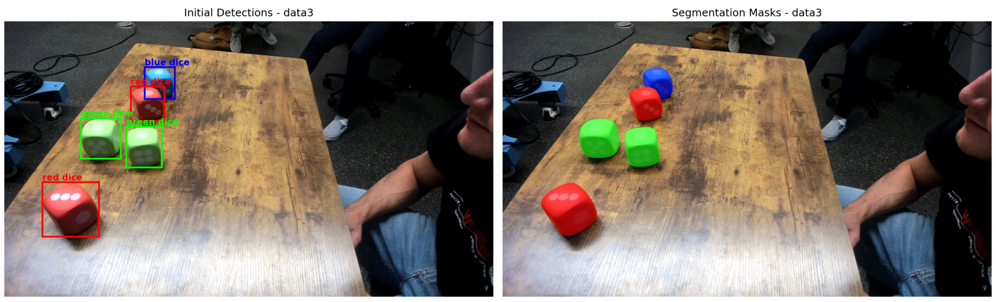

Figure 4: Initial Frame by Frame Extraction.

Since Grounding DINO's text-based labels were not always precise in distinguishing dice colors, a secondary color classification step was applied using HSV color space analysis. For each detected bounding box, the region of interest was extracted and converted to HSV, then compared against predefined color ranges for red, green, and blue. The color with the highest pixel coverage above a 15% threshold was assigned as the die's label. Non-maximum suppression (NMS) with an IoU threshold of 0.4 was also applied to remove duplicate detections of the same object.

To ensure only relevant objects on the tabletop were tracked, a workspace detection algorithm was applied using a `lower_half` heuristic, masking out the upper 20% of the frame where background elements were more likely to appear. Detections falling outside this region were discarded.

With the dice detected and identified on the first frame, SAM 2 (`sam2_hiera_large`) was used to propagate segmentation masks across the entire video. Each detected die was registered as a tracked object using its bounding box as the initial prompt, and SAM 2's video predictor propagated the segmentation forward through all subsequent frames. The resulting per-frame masks were used to extract centroids, bounding boxes, and pixel areas for each tracked die, which were saved to JSON tracking files for downstream use.

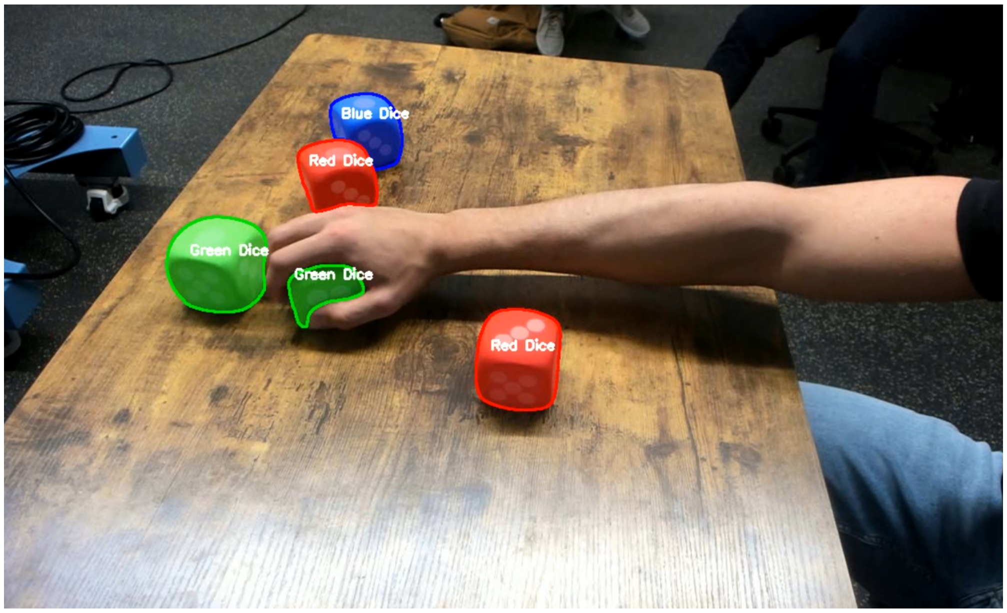

Figure 5: Frame taken from masked object detected video.

NVIDIA Cosmos Reason 8B was then used in addition to provide high-level natural language descriptions of the objects in each video. Using the `analyze_video` function, the model was prompted with a detailed instruction asking it to count each die by color, describe their initial arrangement, and note any other scene elements. The model was loaded using the Hugging Face `transformers` library as `Qwen3VLForConditionalGeneration` in `bfloat16` precision with automatic device mapping. Video frames were sampled dynamically using OpenCV to count total frames, and the model produced chain-of-thought reasoning wrapped in `<think>` tags followed by a structured answer in `<answer>` tags. Results for all videos were saved to a JSON file for reference.

```
Video: data3_tracked.mp4 | Frames: 783 | FPS: 29.95 | Duration: 26.15 secs

Analyzing video: demonstrations/objects_tracked/data3_tracked.mp4
Question: You are analyzing a video with colored dice on a table. Pay EXTREMELY
    CLOSE ATTENTION to counting.

```

```
...

 Objects in data3_tracked.mp4:
----------------------------------------
Total number of GREEN dice: 2
Total number of RED dice: 2
Total number of BLUE dice: 1
Total number of ALL dice: 5}
```

3.2 Action Recognition

For human action recognition, NVIDIA Cosmos Reason 8B was used exclusively, leveraging its vision-language reasoning capabilities to identify and describe manipulation actions from video. Each video was processed using the `analyze_video` function with a fixed sampling of 60 frames and reasoning enabled. The model was prompted with a detailed instruction asking it to produce a chronological list of all observed fine-grained manipulation actions including reaching, grasping, lifting, moving, placing, releasing, stacking, and adjusting. For each action, the model was asked to name the action, identify the object involved, and describe the state of the dice arrangement after the action was performed.

```python
def analyze_video(video_path, question, model, processor, nframes,
    enable_reasoning=True):

    # Prepare the system prompt (with reasoning format if enabled)
    if enable_reasoning:
        system_prompt = """Answer the question in the following format:
            <think>
                your reasoning
            </think>

            <answer>
            your answer
            </answer>
        """
    else:
        system_prompt = "You are a helpful assistant that analyzes videos."

    # Prepare the conversation messages
    messages = [
        {
            "role": "system",
            "content": system_prompt
        },
        {
            "role": "user",
            "content": [
                {
                    "type": "video",
                    "video": str(video_path),
                    "nframes": nframes  # Explicit frame count to avoid metadata
    warning
                },
                {
                    "type": "text",
                    "text": question
                }
            ]
        }
    ]

    ...

    # Parse reasoning and answer if reasoning was enabled
    if enable_reasoning:
        reasoning = ""
        answer = ""

        if "<think>" in output_text and "</think>" in output_text:
            reasoning = output_text.split("<think>")[1].split("</think>")[0].strip
    ()

        if "<answer>" in output_text and "</answer>" in output_text:
            answer = output_text.split("<answer>")[1].split("</answer>")[0].strip
    ()
        elif "</think>" in output_text:
            # Sometimes the answer comes after </think> without tags
            answer = output_text.split("</think>")[1].strip()
        else:
            answer = output_text

        return {
            "reasoning": reasoning,
            "answer": answer,
            "full_output": output_text
        }
    else:
        return {
            "reasoning": "",
            "answer": output_text,
            "full_output": output_text
        }
```

The model's chain-of-thought reasoning block allowed it to work through the video step by step before generating a final structured answer, which helped reduce errors in identifying subtle transitions between actions. Results were collected across all demonstration videos and saved to a JSON file. This approach required no custom training and relied entirely on the pretrained model's visual reasoning ability, though it was constrained by the model's fixed vocabulary and susceptibility to occlusion or rapid motion between sampled frames.

```
================================================================================
Processing video 4/11: data3_tracked.mp4
================================================================================

Analyzing video: demonstrations/objects_tracked/data3_tracked.mp4
Question: Analyze this video and identify ALL human actions performed required to
    complete the pattern of dice in this video. The dice will be lined up in a
    specific pattern.

...

 Actions in data3_tracked.mp4:
----------------------------------------
1. Reaches toward the Green Dice at the bottom left corner of the table.
2. Grasps the Green Dice and lifts it slightly off the table.
```

```
3. Moves the Green Dice horizontally toward the center of the table.
4. Places the Green Dice in a line with the other dice, forming the sequence:
    Green, Red, Blue.
5. Adjusts the position of the Green Dice to align it neatly with the others.
6. Releases the Green Dice, completing its placement in the line.
7. Reaches toward the Red Dice located near the center o...
```

4 Automatic Generation of Sequence of Actions

4.1 Structured Action Sequence Generation

To generate structured action sequences from the demonstration videos, NVIDIA Cosmos Reason 8B was again used, this time prompted to produce output in a JSON-formatted schema. Each video was passed through the `analyze_video` function with 60 sampled frames and a prompt that instructed the model to output a complete step-by-step action sequence using a standardized set of eight action verbs: REACH, GRASP, LIFT, MOVE, PLACE, RELEASE, ADJUST, and STACK. For each step, the model was asked to populate a JSON object with the step number, action verb, object being manipulated, start and end states, a list of preconditions that must be true before the action can be taken, and a list of effects that result from taking the action.

```
"data3_tracked.mp4": {
    "reasoning": "...
    There are five dice shown: Green Dice 1, Green Dice 2, Red Dice 1, Blue Dice,
   and Red Dice 2. Each has specific positions and labels. The agent's actions
   involve moving these dice into a vertical line.\n\nLooking at the impact of the
    agent's actions: The hand moves from right to left, interacting with each die.
    The sequence starts with Green Dice 1 being moved to the center, then Red Dice
    1 stacked on it, followed by Blue Dice, Red Dice 2, and finally Green Dice 2
   completing the stack.\n\nI need to list each action step by step. For each die,
    determine the action taken (move, stack, place) and note the start and end
   states. Also, check if any conditions were met before each action, like
   ensuring the previous die was placed correctly.\n\nLet me verify the order. The
    initial setup has Green Dice 1 on the left, Green Dice 2 near center-left, Red
    Dice 1 near center-right, Blue Dice above Red Dice 1, and Red Dice 2 below Red
    Dice 1. The hand moves them into a vertical line starting from the center.\n\
   nSo the sequence should be: Green Dice 1 moved to center, then Red Dice 1
   stacked on it, Blue Dice added, Red Dice 2 placed, and Green Dice 2 on top.
   Each step involves moving or stacking the respective die onto the previous one
   .\n\nI need to make sure each action is clearly defined with the correct verbs
   and objects. Also, check that the preconditions (like previous placement) are
   noted where applicable. Finally, structure all this into the required JSON
   format without missing any steps."
    ...
  },
```

This structured format was chosen because it maps naturally to task planning representations used in robotics, allowing logical reasoning about action dependencies and state transitions. The preconditions and effects in particular are critical for enabling a robot to verify whether a given action is executable before attempting it, and to update its world model after completing each step. Results for all videos were parsed from the model's output and saved to a JSON file. A key limitation of this approach is that the model must infer all of this structure purely from visual input without any access to ground truth state information, which means errors in action recognition at stage B.2 can propagate forward into incorrect precondition or effect assignments. The approach also does not generalize well to tasks with more complex object relationships, significantly longer action sequences, or environments with significant visual clutter.

4.2 Markov Decision Process Design

Based on the action sequences observed across all demonstration videos, a formal Markov Decision Process was designed to represent the dice manipulation task. The MDP was defined with nine states capturing the full lifecycle of a single object manipulation: `S0_IDLE` (hand empty, objects on table), `S1_REACHING`, `S2_GRASPING`, `S3_HOLDING`, `S4_MOVING`, `S5_POSITIONING`, `S6_RELEASING`, `S7_STACKED` (a successful placement, reward state), and `S8_COMPLETE` (all objects placed, terminal state). Eight actions were defined corresponding to the verbs used in the action sequence generation step: `A0_WAIT`, `A1_REACH`, `A2_GRASP`, `A3_LIFT`, `A4_MOVE`, `A5_LOWER`, `A6_RELEASE`, and `A7_ADJUST`.

Transition probabilities were estimated from patterns observed across the demonstration videos rather than computed analytically, reflecting the stochastic nature of real manipulation. For example, executing `A1_REACH` from `S0_IDLE` transitions to `S1_REACHING` with probability 0.95, with a 0.05 probability of remaining idle due to a failed approach. More uncertain transitions such as lowering an object to a target position (`S4_MOVING` $\rightarrow$ `S5_POSITIONING` via `A5_LOWER`) were assigned a lower success probability of 0.85 to account for variability in placement precision. The reward function was designed to balance task success with efficiency: a reward of $+10$ is given for successfully placing an object (entering `S7_STACKED`), $+100$ for completing the full task (entering `S8_COMPLETE`), -5 to -10 for failure events such as dropped objects or failed grasps, and a step cost of -0.5 per action to encourage shorter solutions. The complete MDP was visualized as a state transition diagram using `matplotlib`, with states color-coded by type (initial, intermediate, reward, terminal), and also expressed in Mermaid format for external rendering. All MDP components were saved to a JSON definition file alongside the other project outputs.

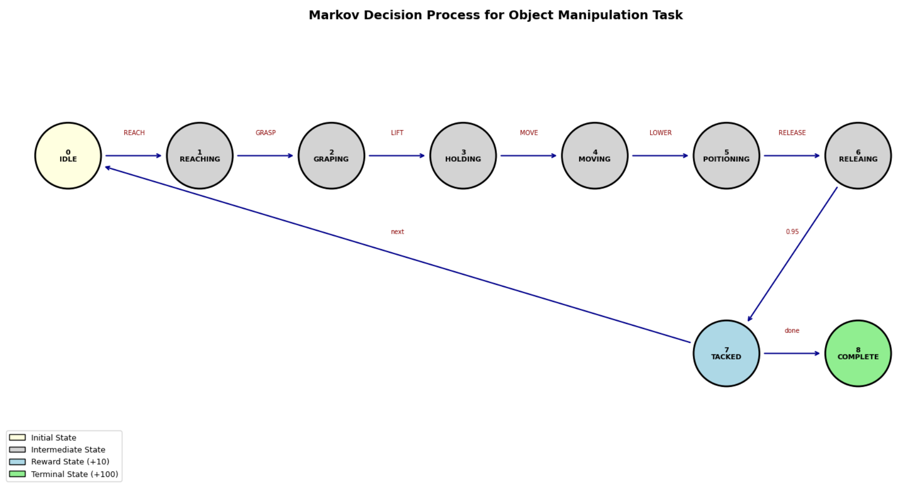

Figure 6: Markov Decision Process Diagram Generated from Results.

5 Conclusion

I explored how we can generate an MDP from video recorded human demonstrations to define states and actions for a specific pick and place task goal. Starting with video recorded demonstrations, after implementing methods to accomplish our goals, we measured their effectiveness. The code related to this project and the plots associated can be found here: `https://github.com/aneangel/cosmos-2b`

SAM2 was a very effective model at handling object segmentation and detection, most of the minor performance issues that I encountered where mainly user error, in which I improved my implementation on how I used SAM2 exactly to obtain the result I wanted and needed.

The reasoning models were very effective in detailing what we needed for our MDPs, however this wasn't until we supplied it with the sufficiently contextualized data. The results of feeding it the videos without the masking done from SAM2 and the labeling versus when we gave it the raw videos was drastic. I would recommend that going forward, to truly improve the output of the reasoning model that we provide it as much contextualized data about the scene and environment as possible for the best results.

In my short experience trying to obtain the best results, if I were to be given the choice of using the reasoning model as I have it now or re-implement differently I think I would ultimately choose to implement differently. While reasoning models are effective, I think it requires a lot of context and prompt engineering to truly make it an effective piece within the pipeline that I have designed for MDP creation from raw video. Not to say that NVIDIA's Cosmos-8B is not good, I think it did quite well given the expectations I had for it, however I think it might have over complicated the end goal and potentially with a simpler approach I might have achieved better results or the same with a less complicated design to reach it.

I must also highlight that Meta's SAM2 is a wonderful model for object detection and segmentation, I have used it before in other research and stand by it in its performance.