Vision-to-Language Abstraction for MDPs

Jack Landers

I. Introduction

This project encodes sequences of actions extracted from visual demonstrations into a structured representation suitable for training a robotic policy via reinforcement learning with a Markov Decision Process (MDP) framework. Our data is a collection of videos in which participants move a randomly placed group of dice in a constant order using sequential pick and place actions. The Markov Decision Process probabilistically determines the next task to perform given the current and past states. This is learned over iterations of demonstrations, connecting future states as nodes that could potentially occur from the current state with likelihoods proportional to their representation in the videos.

II. Evaluating Approaches

We consider a variety of image processing techniques to develop an understanding of the actions occurring across the frames of each video. It is important to compare the ease of integration, the required computation, and the potential accuracy of each approach before committing to development.

A. Hand Recognition

One potential approach was to implement an open source hand recognition model from Google called Mediapipe. With this, our aim was to define pick and place actions as when and where the hand in the video would close open, respectively. Although Mediapipe was easy to integrate, computationally affordable, and highly accurate in some cases, we found that the dataset is dramatically biased towards lighter skin tones. This in addition to our high contrast environment, led to extremely disproportionate representations between different demonstrators. Such a difference was so polar that the hand could be perfectly predicted for all fingers across an entire demonstration performed by participants with light skin, while the hand could go completely undetected for demonstrators with dark skin.

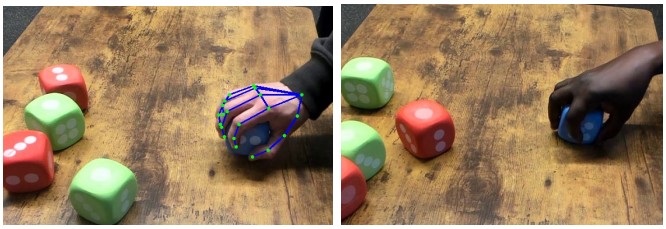


Fig. 1: Varying results for MediaPipe due to skin tone

Testing and identifying this dataset bias is extremely important. In the case of robotics, an unreliable detection of people and hazards can lead to dangerous environments. The event where a computer vision model is only trained and tested on subjects with light skin and is deemed reliable enough to determine when a hazard is posed, could lead to scenarios of injury occurring. For this reason, we consider other options.

B. Segmentation Models

Rather than reasoning about the state of the hand, we evaluate detecting and following the objects. In order to identify the exact color of the object as well as when it was grasped, we aimed to implement segmentation models in particular, where the exact pixels representing the object would be extracted. For its reportedly high accuracy and generalizability to video, we first tried Meta SAM, but found it to be overly computationally expensive and complex to implement. Contrastingly, open-sourced YOLO models were easily on a home computer, simple to integrate, but were still not reliable to the degree that the objects could be tracked. The dice were not very represented in the training data and were being related to distant objects. We considered YOLOe, where the final layers could be retrained with examples, but chose to pivot towards a different approach where success was more likely.

Fig. 2: YOLO segmentation could be very accurate, but was unreliable in some clear cases

C. Vision Transformers

Vision Transformers show potential as their success is highlighted across recent academic literature. These models use a transformer architecture to apply attention across regions of an image to interpret the objects depicted more relative to one another. Attention is very costly for hardware due to the requirement for quadratic operations to perform comparisons, and this scales further for image frames across video. In order to run such models, we would have to preprocess our data significantly by downsampling the video into a few frames. If the model could accurately detect the objects, this can work, but still cannot guarantee precise timing with the gaps in frames that would be required to compute, and outputs are not as structure giving room for error.

Professor Maria Kyrarini, Santa Clara University
Submission deadline February 19, 2026

III. Custom Vision Pipeline

Due to the computational expense of using state of the art models, we instead concluded on producing our own classical imaging model. Vision across video is especially costly due to the large number of images that must be processed having to perform them frame by frame. For this reason, a low level vision model enables real time computation ability at the cost of complexity in both feature recognition and reasoning. To overcome this boundary, we take advantage of the abstract information that remains constant across all frames. The first of these is that our objects are the most densely colored regions of each frame, being either extremely red, green, or blue. The second of which is that we are only acting in a constant region of interest (ROI) across all frames. These are our motivations for using both Max Pooling and Masking of the ROI.

A. Max Pooling

Max Pooling is a 2-dimensional filter performed across sections of the image, where the greatest pixel value among them is selected. It has the utility of downsampling the image while retaining most of the information in color and light. By isolating each color channel in a frame, we see that the dice are all unique highlights. With this, selecting the maximum values locates our objects reliably. However, there were still limitations to this vision method when standalone, because for many frames, the dice may be obscured and another part of the image would be selected. This happened most commonly, where the same dice would be selected twice, and colors on participants' clothing or the external environment would be misidentified.

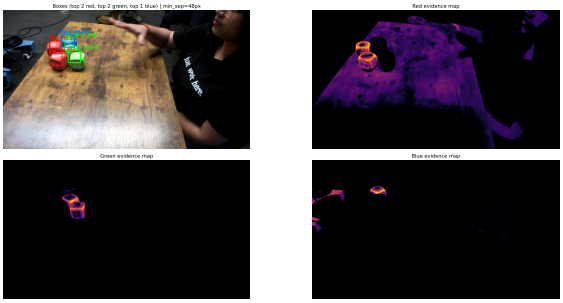


Fig. 3: Using Max Pooling to isolate the targets

B. Masking Region of Interest

For most cases, masking a region of interest resolved these issues. In order to do this, we manually selected four corners in the image to isolate the workspace, as well as adding a mask within a radius of the first red and green dice selected, so that they would not be highlighted twice. With this addition to the model, we had a functioning system that was selecting the objects more confidently than any of the more expensive approaches we had previously considered.

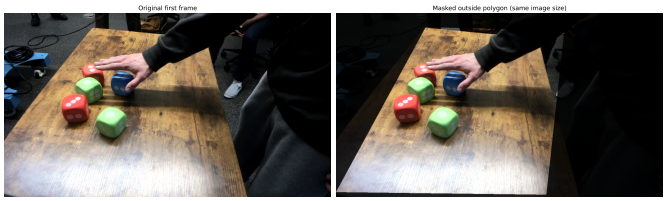


Fig. 4: The masked region of interest

IV. Action Representation

To construct a Markov Decision Process, reliably detecting the objects is not enough, our next goal was to reason across the frames of the images to determine in what sequence each of the dice is moved.

A. Trajectory Following

As a first approach, we attempted to follow the trajectory of each dice by relating the position to the nearest position in the previous frame. With this we could then threshold the center position and evaluate when each dice is moved to the final sequence. However, the detection was still imperfect for every frame, especially for cases where the objects would become obstructed, and consequently the position might jump erratically. Visually, the trajectories of each object were clearly mapped, but even with velocity smoothing and outlier dropouts, the trajectory information was insufficient for predicting the sequence of actions.

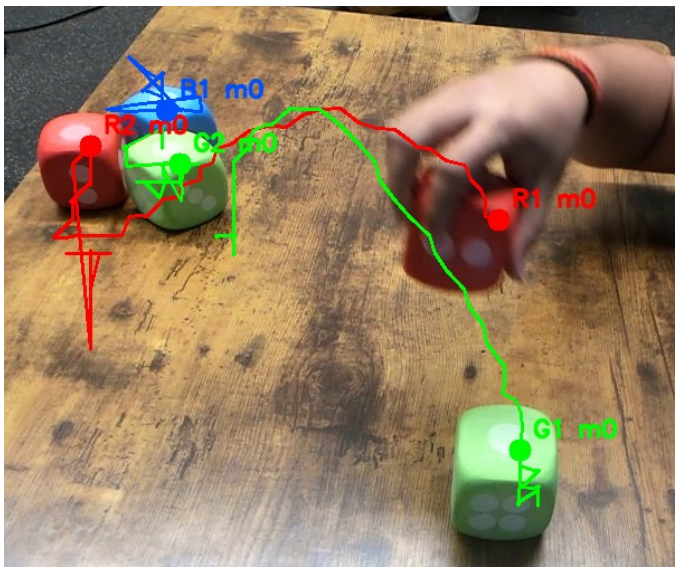
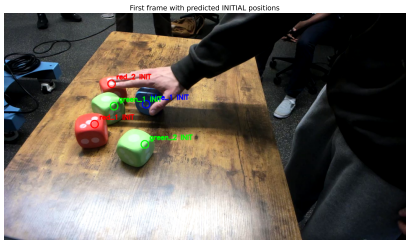


Fig. 5: Trying to follow the trajectories from detection was unreliable

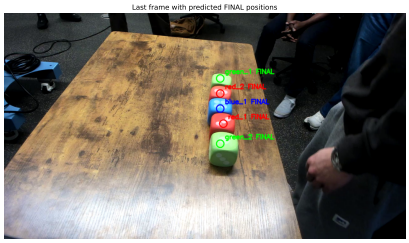
B. Sequence Prediction

From this, we pivot our approach to consider predicting the sequence of actions based exclusively on initial and final positions of the objects. To find these positions we design our algorithm to detect the positions of each dice within a

range of frames at the start and end of each video, within an error distance in the event that the dice are bumped slightly or go undetected for some frames. After optimizing these two parameters we find that evaluating across a 30 frame range to determine the points of interest, 4 frames to detect whether the object has been moved or settled, and a 4% error distance, was most reliable.



(a) The predicted initial position



(b) The predicted final position

Fig. 6: Resting Locations

With our sequence of actions predicted, we overlay the connected initial and final positions and highlight them individually at each time of moving to validate our results with the video. Here we found our reasoning model could consistently determine when and where to each object was being placed.

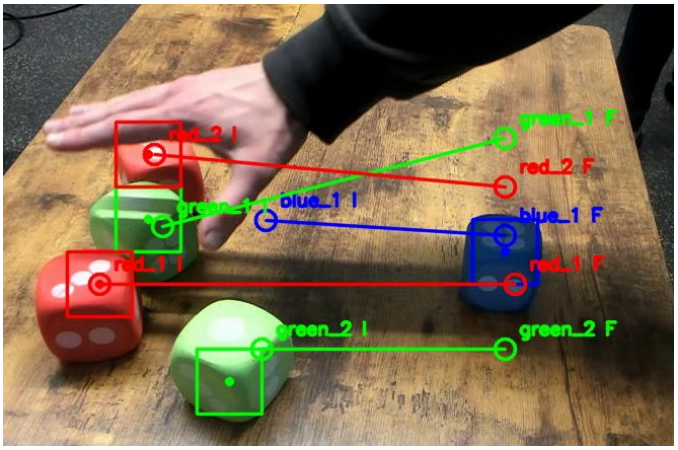


Fig. 7: The paths of the objects could be accurately found from the resting locations aggregated across multiple frames

V. Natural Language Abstraction

To represent the unique sequence of movements occurring for each video, we develop 5 pairs of tuples. One tuple would list the colors unmoved, with all starting at ('green', 'red', 'blue', 'red', 'green'), and the other tuple, would indicate which dice had been moved to final position, and in which order, all finishing at this same order by the fifth step. As we construct these tuples we simultaneously print the action encoded, showing the conversion from video to natural language that we have abstracted from the tasks presented.

```
Demo: data3

Step 0:
Red moved.
Red was placed.
This move completed at frame 250 (t=4.17s).
The placed sequence is now ('red',).

Step 1:
Green moved.
Green was placed below Red.
This move completed at frame 768 (t=12.82s).
The placed sequence is now ('green', 'red').

Step 2:
Red moved.
Red was placed above Red.
This move completed at frame 775 (t=12.94s).
The placed sequence is now ('green', 'red', 'red').

Step 3:
Blue moved.
Blue was placed between Red and Red.
This move completed at frame 1030 (t=17.20s).
The placed sequence is now ('green', 'red', 'blue', 'red').

Step 4:
Green moved.
Green was placed above Red.
This move completed at frame 1226 (t=20.47s).
The placed sequence is now ('green', 'red', 'blue', 'red', 'green').
```

Fig. 8: The output description of states and actions for the third demonstration

VI. Constructing the Markov Decision Process

From the states reached in each of our four demonstrated paths we can begin to see the formation of the Markov Decision Process. As more states are represented with more demonstrations, the probabilities become more complex, and there is a greater range of the possible combinations to reach. To build a reinforcement learning model, we can define rewards and punishments based on our objective. The greatest reward is to reach the final alignment of the cubes. As the model determines which color to pick it should be rewarded for selecting colors that still remain, and punished for trying to pick a color that is already at the objective. For placing, the model should compound reward for reaching states that do not break the order of the dice, and punished in the cases where does. Iterations of such a reinforcement model enable the robotic policy to learn to maximize reward by performing the pick and place Markov Decision Process. This produces intelligent robotic policy with an understanding for planning and sequence of action control.

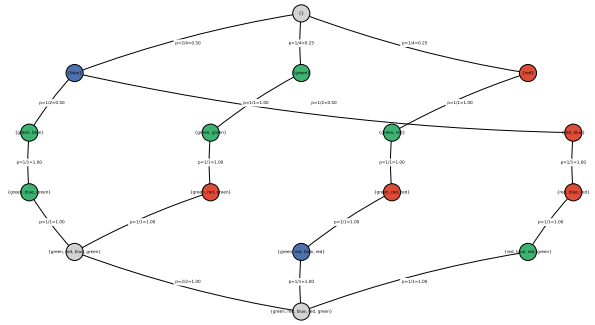


Fig. 9: The Markov Decision Process

ACKNOWLEDGMENT OF AI

Code for this project was generated by LLMs:

- ChatGPT 5.2

REFERENCES

[1] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep reinforcement learning for robotics: A survey of real-world successes," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 8, pp. 153–188, May 2025. doi:10.1146/annurev-control-030323-022510. :contentReference[oaicite:0]index=0

[2] M. Lauri, D. Hsu, and J. Pajarinen, "Partially observable Markov decision processes in robotics: A survey," *IEEE Trans. Robot.*, vol. 39, no. 1, pp. 21–40, Feb. 2023. doi:10.1109/TRO.2022.3200138. :contentReference[oaicite:1]index=1

[3] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Lecture Notes in Computer Science, vol. 8689, Springer, 2014, pp. 818–833. doi:10.1007/978-3-319-10590-1_53. :contentReference[oaicite:2]index=2

[4] B. Chiang and J. Bohg, "Representations and representation learning," Stanford Univ. course notes for CS231A, Feb. 2022. [Online].

[5] J. Landers, 'https://github.com/JacktheLander/Lab-Projects/blob/main/Robot-Learning/Markov-Decision-Processes/MDP.ipynb'