

Leveraging VLMs for Optimising Camera Inputs for Surgical Robotic Reinforcement Learning

Kushagra Raghuvanshi

MEDCVR Lab

University of Toronto

Toronto, Canada

kushagra2125@gmail.com

Abstract—Camera rendering and positioning play an important role in effective reinforcement learning (RL) for robotics tasks in surgical applications. This work explores optimizing camera inputs for a 3D surgical task - "Needle through a hoop" in Unity simulation. By leveraging Vision Language Models (VLMs) to evaluate and rank camera configurations, this work establishing a framework to predict the most effective combination of camera views and rendering types for a complex 3D surgical task. Experiments demonstrate that VLM predicted configurations align closely with actual performance. Moreover, this research also shows that camera inputs can potentially lead to better VLM reward generation than text-only prompts for reward generation and can yield results comparable to human-defined reward functions. These findings highlight the potential of VLMs in enhancing RL-driven robotic learning pipelines in surgical environments.

Index Terms—Vision Language Model (VLMs), Reinforcement Learning (RL), Simulation, Cameras, Rendering.

I. INTRODUCTION

The quality of the camera input plays an important role in the training of RL models. This is of particular concern in high-degree-of-freedom tasks involving 3D movement. A single RGB camera input is often not sufficient to provide the necessary information for learning due to the ambiguity it can produce in complex 3D scenes. For surgical robotics, where precision is of great importance, this limitation is especially pronounced. Multi-camera setups with differing views and rendering types can help overcome these challenges by providing richer visual data for model learning.

This paper explores the optimization of camera inputs for RL training in a 3D surgical robotics task in simulation. A VLM is utilized to evaluate and predict optimal combinations of camera view and rendering types (Figure 1) for a simulation "Needle through a Hoop" task, a representative example of surgical task requiring precision in 3d space. By comparing VLM predictions to actual performance and integrating camera inputs for VLM reward generation, I demonstrate how VLMs can enhance camera input optimization which can aid in reward generation.

II. RELATED WORKS

Works by Su et al. [1] [2] highlight the need of multiple camera views for robotic learning in robot assisted minimally

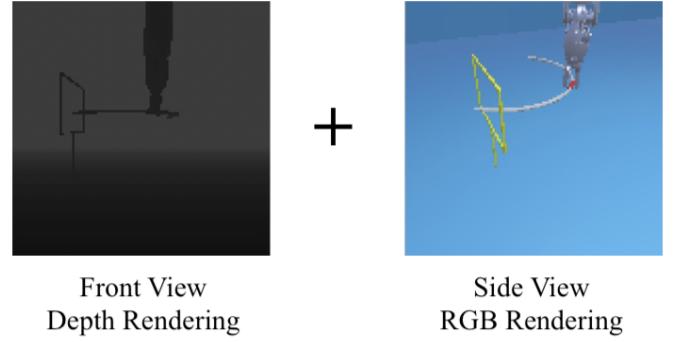


Fig. 1: Example of a Camera Combination

invasive surgeries. They propose the use of multiple cameras for good 3D reconstruction and use deep RL for optimizing camera positions. My works uses the concept of multi camera use as a foundation to explore VLMs ability in choosing optimal camera combinations. Additionally, I also study the effect of different rendering types on model training performance.

There has been significant progress on usage of VLMs for RL training. Earlier works made use of text description of agent's goal as input to VLMs for reward generation and made use of human feedback on roll-out performance via re-prompting to optimize the reward, thereby reducing the need for human input in terms of human defined rewards [3]. However, lack of visual input to VLM makes this an infeasible approach for complex tasks . Wang et al. propose a framework for automatically generating and improving reward policy for agents using text descriptions of task goals as well as images of task progress, leveraging feedback from VLM models via creation of preference labels [4]. Even though it is able to work on a variety of tasks, it requires re-prompting the VLM numerous times periodically over the course of training which makes it resource intensive. Furthermore, works by Venuto et al. [5] motivates my use of initial and goal state image of the task for VLM prompting. My work takes inspiration from these study for utilizing text + visual information for VLM prompting and attempts to build a less resource intensive zero shot framework for reward generation.

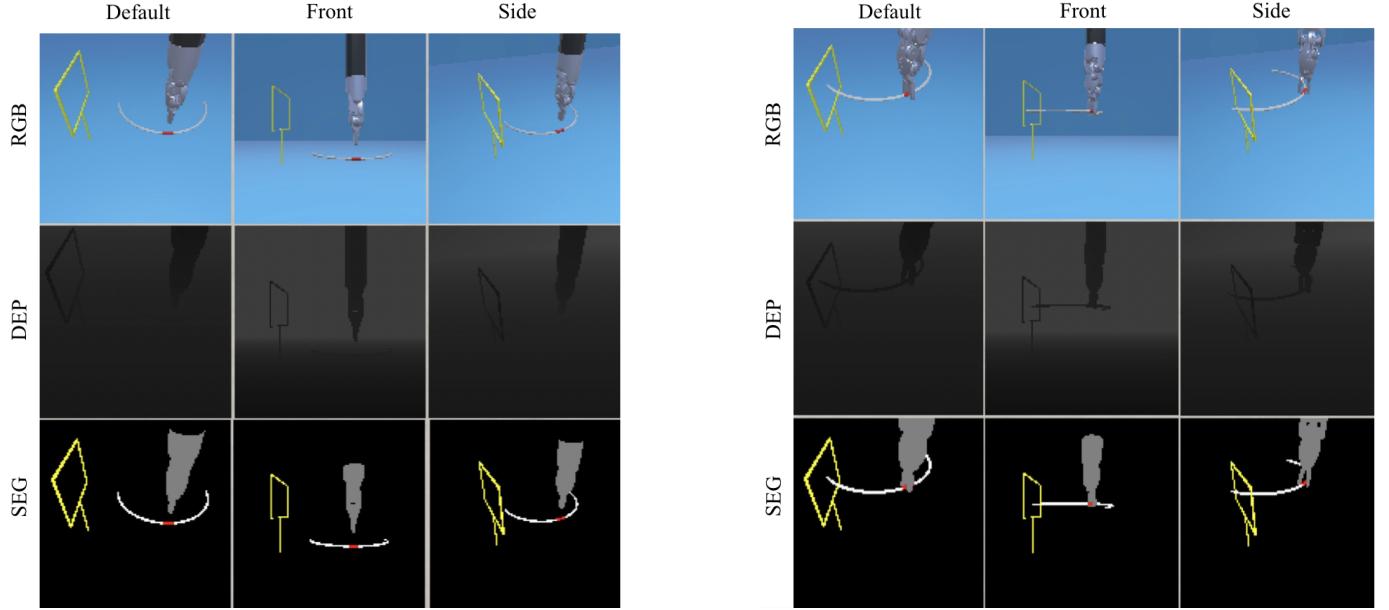


Fig. 2: Initial and Goal states of the scene over all camera views and rendering types.

III. METHODOLOGY

My methodology is comprised of A "Needle through a Hoop" simulation built on a modern physics engine with dVRK PSM model, a human-defined reward function, and a pipeline that makes use of a VLM for questioning and prediction, compares predictions with actual performance and utilizes more VLM instances for reward generation from top performing models.

A. Simulation

The simulation is built upon Unity 3D game engine. Unity provides accurate physics for robotic contacts, collisions and articulations. I use the URDF (Unified Robot Description Format) files (that has had its base capabilities provided by the importer extended to allow for mimic joints and a closed kinematic chain to be simulated, and its system fine-tuned to have realistic visual models and efficient convex collision models [6]) to create a simulation of the PSM with accurate kinematics [7] and dynamics [8]. In addition to the default RGB rendering (**RGB**), I make use of two additional renderings: RGBD or Depth rendering (**DEP**) which supplements RGB data with depth information and Segmentation rendering (**SEG**) which allows focus on main objects in the scene with selective coloring.

B. Reinforcement Learning

This work uses a vision based "Needle through a Hoop" task, which consists of 4 degrees-of-freedom (3D movement + jaw rotation around robot arm's axis), as a representative of a complex 3D surgical task. I use the Unity ML-Agents package for RL training using pytorch. The agent's goal is to insert the needle in the hoop. I train the agent using a Proximal Policy

Optimization (PPO) algorithm from ML-Agents. The agent is trained for 1.5 million steps on a Ryzen 7800X3D, Nvidia 4090, Cuda v12.1 and 32 GB of Ram. batch size of 256, buffer size of 10240, learning rate of 0.0003, beta of 0.001, epsilon of 0.2, lambda of 0.95, 3 epochs, constant beta scheduling and linear epsilon scheduling, a ResNet visual encoder, hidden layer size of 256 with 3 layers used, reward gamma of 0.99, rewards strength of 1.0, and a time horizon of 128.

C. VLM Prompting

I employ Gemini 1.5 Flash at temperature 2 for VLM Prompting due to its quick and creative reasoning for most tasks. A single instance is used for the VLM questiong and VLM Ranking parts while one new instance is used per VLM Reward Generation.

D. Pipeline

The workflow can be divided into two phases.

Camera Optimization Workflow

Camera Inputs: First, nine images are captured for both the initial and goal states across all camera views and renderings. These images, as illustrated in Figures 2 and 3, are generated from a simulation of the scene.

VLM Questioning: Second, using RGB-rendered images from three camera views of the initial state, the VLM is queried to break the "needle-through-hoop" task into two subtasks. This ensures the VLM can effectively reason about the task structure. Subsequently, all eighteen images (initial and goal states) are provided to the VLM, accompanied by a series of questions to identify which camera views and rendering types are best suited for the subtasks. Detailed prompts for this step can be found in Appendix B.

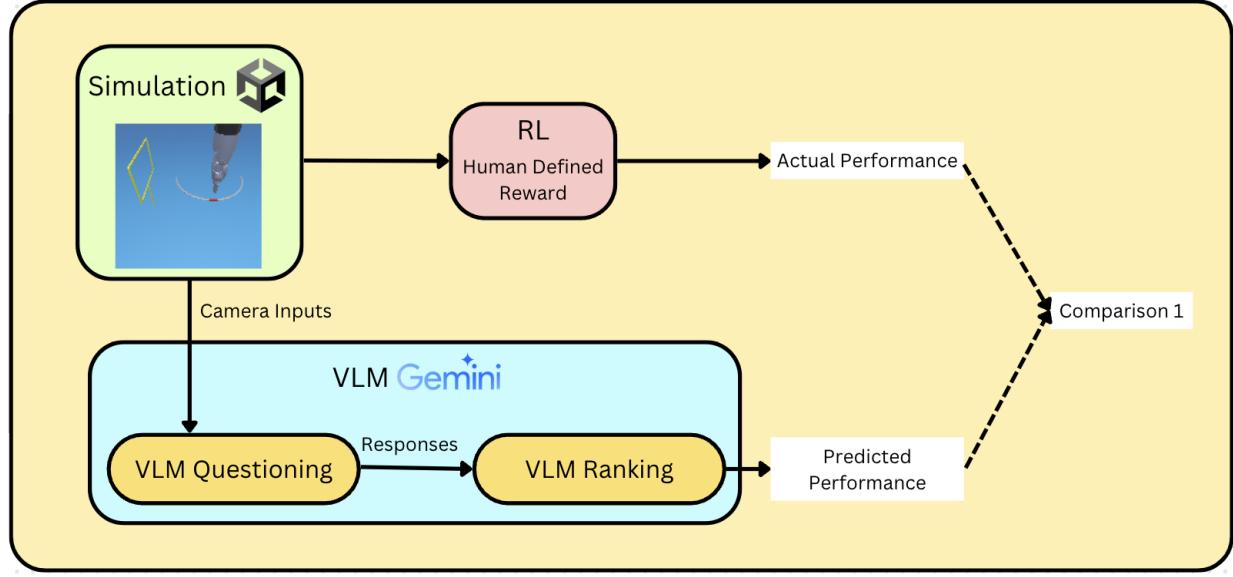


Fig. 3: Camera Optimization Workflow: VLM is used to predict performance of camera combinations which is then compared with actual performance

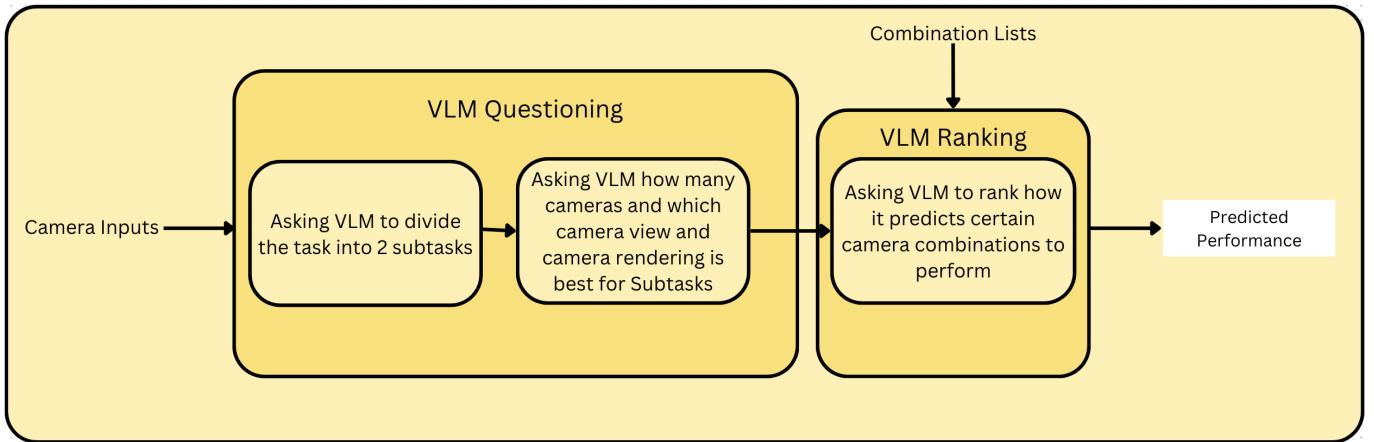


Fig. 4: VLM Questioning and VLM Ranking: VLM is prompted to (i) simplify the scene into subtasks (ii) reason about best camera view and rendering for each subtask and then (iii) asked to predicting performance of camera combinations

VLM Ranking: Next, the VLM is given three lists of camera combinations, one for each pair of renderings (RGB,DEP,SEG), and asked to rank how it expects combinations within the list to perform in terms of RL model training. The specific prompts for this step are included in Appendix C. A human defined reward ranging from a range of about 0 to 9 from start of task to goal state is used for the experiments. It can be found in Appendix Section A.

Comparison 1: Finally, the VLM’s predicted performance rankings are compared against the actual performance outcomes to assess whether VLM can be utilized to optimize the camera inputs that are used for RL training.

Figure 3 provides a visual representation of the camera optimization workflow and Figure 4 shows what happens inside

parts of the workflow.

VLM Reward Generation Workflow

VLM Reward Generation: Based on the performance of the camera combinations, the top 3 performing models i.e. top 3 camera combinations are selected and then 4 rewards functions are generated using four instances of the VLM. One reward is based only on human provided text information of the environment: scene setup, objects and state variables as input while the other 3 are based on the text information as well as the initial state image of the camera combination as input. This, results in 6 models, 2 per camera combination, one using the text information only input reward (that’s common for all the three camera combination) and the other using the text+camera input reward. prompts for this step can be found

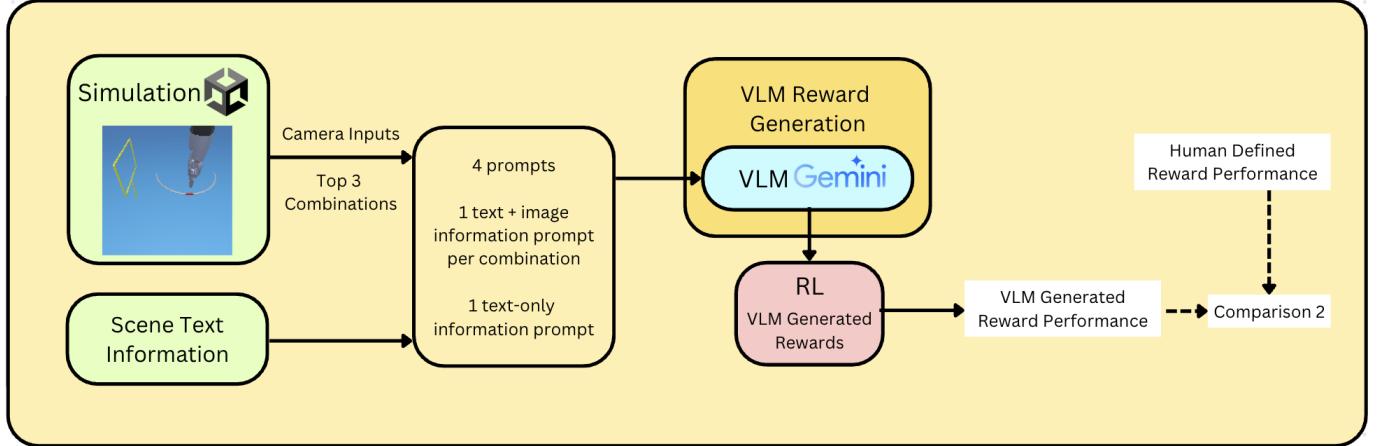


Fig. 5: VLM Reward Generation Workflow: VLMs with (i) Text only and (ii) Text + Image scene information are used to generate rewards and then trained in order to compare with Human defined reward's performance for the best camera combinations

TABLE I: VLM Questions and Responses Summary

S. No.	Question	VLM Response
1	Divide task into 2 subtasks	(i) Needle Pickup + Orientation (ii) Needle Insertion
2	Which camera view is best for subtask 1	Front view
3	Which camera rendering is best for subtask 1	Depth Rendering
4	Which camera view is best for subtask 2	Side view
5	Which camera rendering is best for subtask 2	Segmentation Rendering
6	Which camera view combination is best in general	Front and Side combo

in Appendix D.

Comparison 2: The performance of these 6 models is compared with the 3 top performing models that used the human defined reward. This comparison evaluates whether VLM-generated reward functions can perform at a level comparable to human-defined rewards.

Figure 5 provides a visual representation of the VLM reward generation workflow.

E. Experiments

The experiments primarily consist of training 20 models over 14 combinations of camera views and rendering types. These configurations are aimed at covering a wide array of scenarios. The combinations are based on pairs of camera with same or different rendering types or a single camera. For each of these rendering pair combinations, 3 types of camera views are used: default view (which shows the scene at an angle of 45° i.e. mid way between top and front view), front view (which shows the scene from front) and side view (which shows the scene midway between front view and left side view). Figure 2 shows the initial and goal state images of the scene over all camera views and rendering types respectively.

The 6 extra models trained (20-14) come from training the top 3 performing camera combinations each using a (i) text information only input VLM generated reward and a (ii) text information + initial state camera view image(s) input VLM generated reward.

IV. RESULTS

TABLE II: Comparison 1 - Actual vs Predicted performance.

List 1: RGB and Depth Combinations		
S. No.	Actual Performance Ranking	VLM Predicted Ranking
1	DEP (F) + RGB (S) [8.92]	DEP (F) + RGB (S)
2	RGB (F) + DEP (S) [8.05]	RGB (F) + RGB (S)
3	RGB (F) + RGB (S) [3.40]	RGB (D) + DEP (D)
4	RGB (D) + DEP (D) [2.41]	RGB (F) + DEP (S)
5	RGB (D) [1.56]	RGB (D)
List 2: RGB and Segmentation Combinations		
S. No.	Actual Performance Ranking	VLM Predicted Ranking
1	RGB (F) + SEG (S) [8.92]	RGB (F) + SEG (S)
2	RGB (D) + SEG (D) [7.97]	SEG (F) + RGB (S)
3	SEG (F) + RGB (S) [6.76]	RGB (F) + RGB (S)
4	RGB (F) + RGB (S) [3.40]	RGB (D) + SEG (D)
5	RGB (D) [1.56]	RGB (D)
List 2: Depth and Segmentation Combinations		
S. No.	Actual Performance Ranking	VLM Predicted Ranking
1	DEP (F) + SEG (S) [8.96]	DEP (F) + SEG (S)
2	SEG (F) + SEG (S) [2.56]	SEG (F) + DEP (S)
3	SEG (F) + DEP (S) [2.37]	DEP (F) + DEP (S)
4	SEG (D) [2.29]	SEG (F) + SEG (S)
5	DEP (F) + DEP (S) [1.98]	SEG (D)
6	DEP (D) [0.79]	DEP (D)

*The ones marked red performed better than predicted by the VLM
*Reward after training is shown in blue

Brief description of question prompts and answer responses are provided in Table 1. Overall 25 VLM prompts are made utilizing 14,137 tokens. Detailed prompts and responses can be found in Appendix Sections B, C and D.

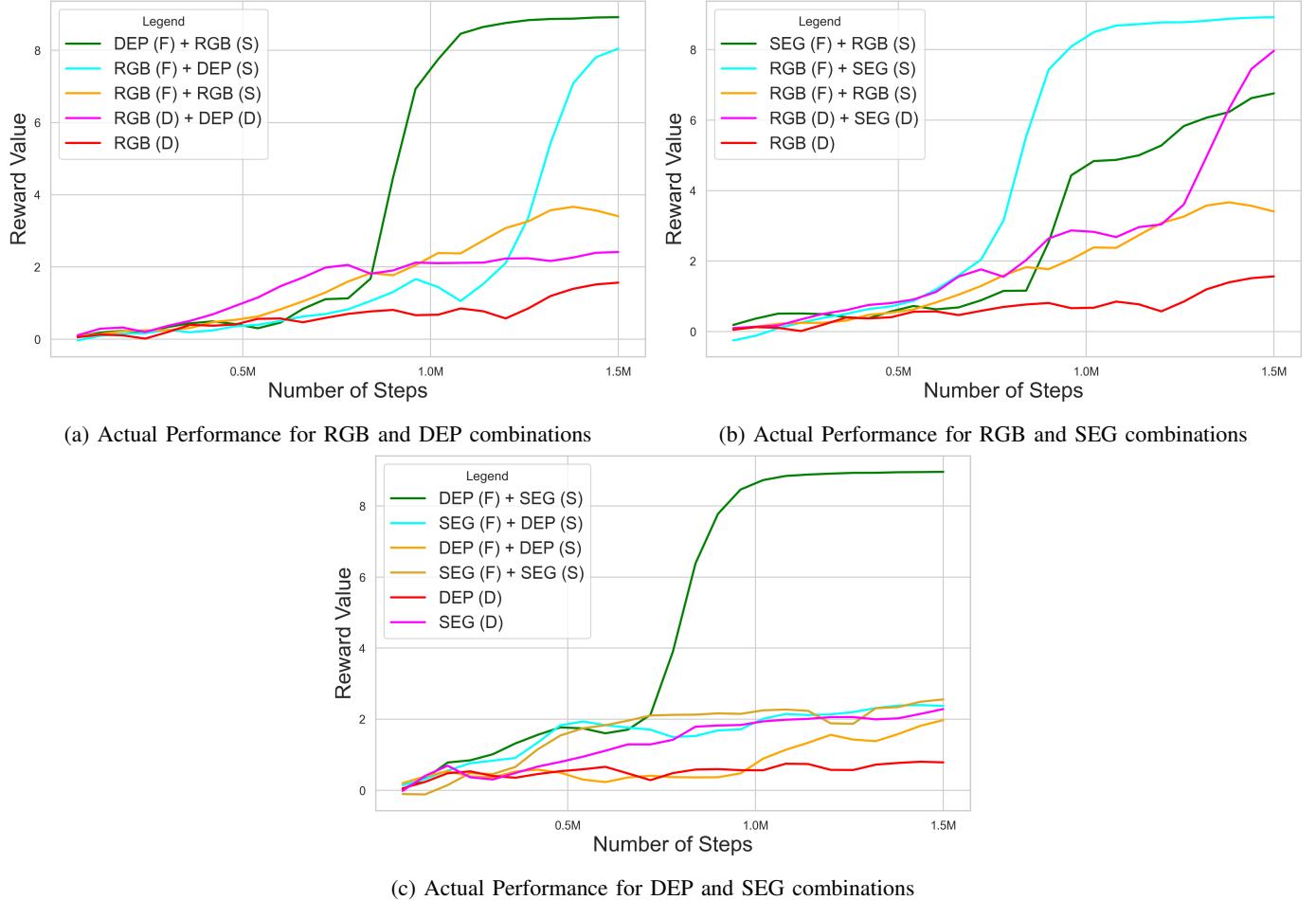


Fig. 6: Actual Performance over all the camera combinations considered: Depth rendering with front view and Segmentation rendering with side view tend to be the best performers

Figure 6 shows the cumulative mean reward of actual training of all the camera combinations over eight environments used to train 1.5 million steps. Table 2 shows the comparison between actual performance of the camera combinations using human defined reward and the predictions made by VLM on how these combinations are expected to perform.

Finally, Figure 7 provides the observations of how VLM text only input generated rewards and VLM text + image input generated rewards that make use of best camera combinations perform compared to the human defined reward's performance for the these camera combinations.

V. DISCUSSION

Performance of Camera combinations

From the observations over 1.5 million steps of training, some camera combinations are able to reach rewards near 9 which indicates task completion.

None of the single camera combinations train well, which can be explained being due to the complexity of the task and the ambiguity single camera perspective brings to 3D scenes.

RGB and Depth Rendering combination setups perform significantly better than RGB cameras only setups for similar camera views. This can be attributed to additional depth information provided by the Depth rendering. The low reward for combination of RGB and Depth camera from the same default view also tells us that using different rendering types from just one view is not helpful due to single camera scene ambiguity explained above. Moreover, Depth rendering performs better when used for the front view than side view. This aligns with VLMs responses that depth rendering and front view are best for the needle pickup subtask due to the depth information and the greater visual clarity of the height of needle relative to the hoop opening.

Similarly, RGB and Segmentation Rendering combinations also perform significantly better than RGB only combinations. The difference this time is that Segmentation rendering performs better when used for the side view which also matches VLMs responses that segmentation rendering and side view are

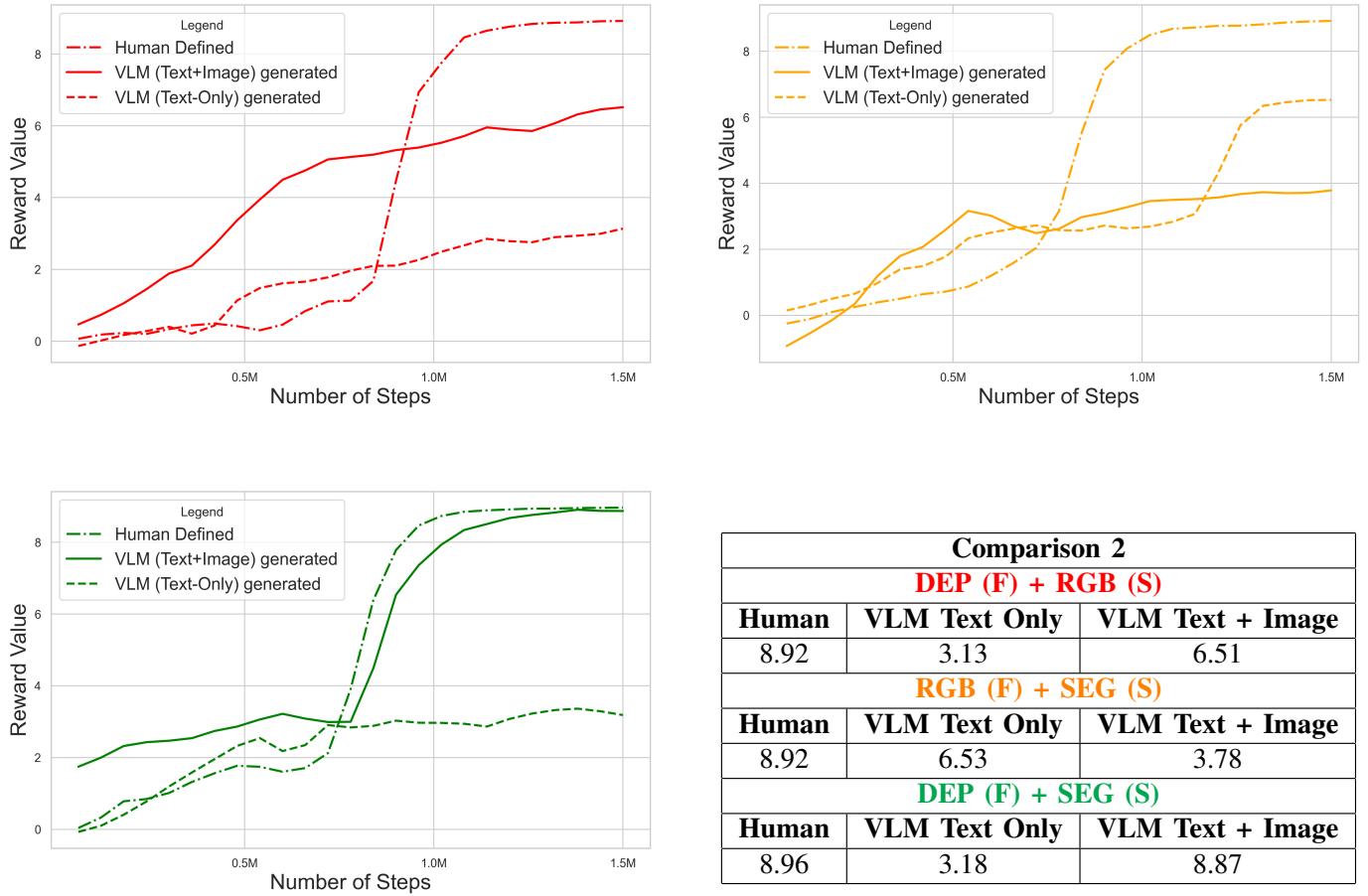


Fig. 7: Comparing the performance of VLM (text+image) generated reward with Human defined reward and text-only VLM generated reward. VLM generated rewards are able to reach human comparable performance for the right camera combination

best for the needle insertion subtask due to side view giving us the profile perspective directly looking at the hoop’s plane and segmentation rendering being the best at clearly showing when needle is inside the hoop. Another difference is the unexpected performance of same default view combination of the two renderings. This could be due to segmentation rendering being able to quickly learn the needle insertion subtask once the needle pickup task is learned, which is supported by poor performance of RGB (F) + SEG (S) combo for the first two-thirds of the training and the jump in progress in the last one-third.

For Depth and Segmentation Rendering combinations, similar observations are made. Individually, neither Depth or Segmentation cameras perform well. However using them in combination with optimal camera views leads to the best performing model. Depth rendering for front view in combination with Segmentation rendering for side view performing well while the opposite rendering order performing bad bolsters the results from previous rendering pair combinations and underscores the importance of the interplay between camera rendering type and camera view.

By utilizing optimal camera views and renderings we are

able to go from failed trainings with rewards of 1.56 for single camera RGB and 3.4 for two camera RGB to successful task completion training with scores reaching 8.96.

VLM Predictions

Many of the VLMs predictions about the different combinations held in actual performance. The VLM was correctly able to predict the best and worst performing camera combinations for all three lists. Some combinations ended up performing better than what the VLM predicted due to two reasons (i) The VLM expected Depth rendering with side view to not be useful and lead to complications, something that actual performance of RGB (F)+DEP(S) combination at 1 million steps also suggests. However, it can be reasoned that the additional depth information is still beneficial in learning (ii) The VLM underestimated Segmentation cameras performance due to reasoning that less info provided by Segmentation camera would lead to worse performance. However it doesn’t consider that by only focusing on the important objects, segmentation camera is able to better learn than a RGB camera, which is reinforced by a single segmentation camera performing slightly better than a single RGB camera (2.29 vs 1.56).

Performance of VLM generated rewards

As seen in Figure 7, using rewards generated with only text information of the environment like scene setup, objects and state variables do not lead to good performance. The only notable progress is observed in the RGB (F) + SEG (S) combo which is able to reach a final reward of 6.53. However, that is a result of the agents learning to 'cheat' by reaching the goal without having the needle being between the hoops.

On the other hand, VLM rewards generated using initial state images along with the earlier described text description perform better. Agent doesn't learn to cheat its way to goal completion and the best performing camera combination from training on the human defined reward is able to actually reach human reward performance using this VLM generated reward in 1.5 million steps of training.

Thus even though VLM generated rewards struggle, it is possible to reach human comparable performance with the right camera setup

Limitations and Future Work

A limitation of this work is that the VLMs were provided detailed description of the scene for reward generation, implying that human input is still required, though much less as the main reward logic is decided by the VLM. Moving forward, I plan to explore this methodology on more scenes with similar complexity to further validate my findings. Additionally, I plan to look into depth based segmentation rendering integrates both segmentation and depth renderings by layering colors of objects based on depth. This approach has the potential to improve model training and may even make it possible for optimal training using only a single camera setup.

This work motivates further study on different types of views and renderings to consider for RL training and brings a new perspective to exploration of VLM reward generation.

VI. CONCLUSION

This work demonstrates potential of progress in robotic learning by emphasizing the significance of choosing the correct camera view and rendering combinations for RL training, particularly in complex 3D tasks such as surgical tasks that require high precision like "Needle through a Hoop". By optimizing camera inputs through a combination of camera views and rendering types, this study establishes a framework for improving RL performance.

Additionally, Rewards generated by VLMs, when combined with well-chosen camera perspectives, can achieve performance levels comparable to those of rewards defined by humans. This approach provides a compelling alternative to text only based reward generation and brings a new perspective to VLM reward generation.

ACKNOWLEDGMENT

I am deeply grateful to Amey Pore for his support and guidance throughout the project which was invaluable towards the projects completion. Special thanks to Mustafa Haiderbhai

for his help in setup of accurate physics for Unity scene and depth and segmentation renderings. I would also like to express my gratitude to Professor Lueder Khars for his insightful feedback and suggestions that helped in the projects success.

REFERENCES

- [1] Y.-H. Su, K. Huang, and B. Hannaford, "Multicamera 3D reconstruction of dynamic surgical cavities: Autonomous optimal camera viewpoint adjustment," *Proc. Int. Symp. Med. Robotics*, pp. 103–110, 2020, doi: 10.1109/ISMR48331.2020.9312951.
- [2] Y.-H. Su, K. Huang, and B. Hannaford, "Multicamera 3D viewpoint adjustment for robotic surgery via deep reinforcement learning," *J. Med. Robotics Res.*, vol. 6, 2140003, 2021, doi: 10.1142/S2424905X21400031.
- [3] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, "Text2Reward: Reward shaping with language models for reinforcement learning," *arXiv*, vol. abs/2309.11489, 2024.
- [4] Y. Wang, Z. Sun, J. Zhang, Z. Xian, E. Biyik, D. Held, and Z. Erickson, "RL-VLM-F: Reinforcement learning from vision language foundation model feedback," *arXiv*, vol. abs/2402.03681, 2024.
- [5] D. Venuto, S. N. Islam, M. Klissarov, D. Precup, S. Yang, and A. Anand, "Code as reward: Empowering reinforcement learning with VLMs," *arXiv*, vol. abs/2402.04764, 2024.
- [6] M. Haiderbhai, R. Gondokaryono, T. Looi, J. M. Drake, and L. A. Kahrs, "Robust Sim2Real transfer with the da Vinci Research Kit: A study on camera, lighting, and physics domain randomization," in *Proc. 2022 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2022, pp. 3429–3435, doi: 10.1109/IROS47612.2022.9981573.
- [7] R. Gondokaryono, A. Agrawal, A. Munawar, C. J. Nycz, and G. S. Fischer, "An approach to modeling closed-loop kinematic chain mechanisms, applied to simulations of the da Vinci surgical system," *Acta Polytechnica Hungarica*, vol. 16, no. 5, pp. 29–48, 2019.
- [8] Y. Wang, R. Gondokaryono, A. Munawar, and G. S. Fischer, "A convex optimization-based dynamic model identification package for the da vinci research kit," *IEEE Robotics and Automation Letters*, pp. 3657–3664, 2019.

Appendix

A: Human Defined Reward

```
public override void UpdateReward()
{
    switch (rewardMode)
    {
        case RewardMode.Sparse:
            base.UpdateReward();
            break;
        case RewardMode.Dense:
            float negStepReward = -(float)StepCount / (float)MaxStep;
            // negStepReward = 0f;
            float liftProgress = 0f;
            float hoopProgress = 0f;
            float graspProgress = 0f;

            if (!grasped)
            {
                graspProgress = AgentHelper.ExpDistanceReward(
                    ToolMidpoint.transform.position,
                    NeedlePickupPoint.transform.position,
                    0.05f,
                    100f,
                    4.0f);
            }
            else
            {
                graspProgress = 1.0f;
            }

            if (!lifted)
            {
                Vector3 liftPosition = new Vector3(
                    needleStartPose.position.x,
                    Goal.transform.position.y,
                    needleStartPose.position.z
                );
                liftProgress = AgentHelper.ExpDistanceReward(
                    Needle.transform.position,
                    liftPosition,
                    0.05f,
                    100f,
                    4.0f);
            }
            else
            {
                liftProgress = 1.0f;
                hoopProgress = AgentHelper.ExpDistanceReward(
                    NeedleEndPoint.transform.position,
                    Goal.transform.position,
                    0.05f,
                    100f,
                    4.0f);
            }

            graspProgress *= GraphProgressRewardScaling;
            liftProgress *= LiftDistanceRewardScaling;
            hoopProgress *= HoopDistanceRewardScaling;

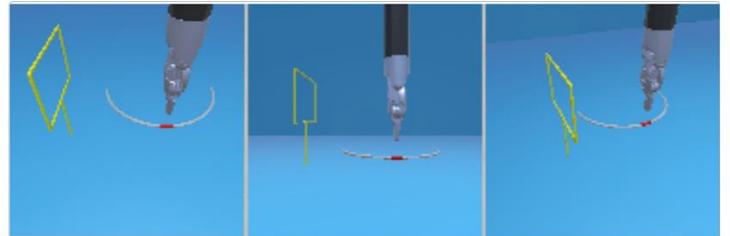
            float sparseGraspReward = 0.0f;
            // Discourage oscillations by encouraging the jaw to be
            // desired shut when the block is grasped
            // if (grasped)
            if (grasped &&
                Arms[0].controller.GetJointCommand("jaw") < 0.1f)
            {
                sparseGraspReward = 2.0f;
            }

            float current_reward = negStepReward + liftProgress + graspProgress +
                hoopProgress + sparseGraspReward;

            AddReward(current_reward - reward);
            reward = current_reward;

            // Debug.Log("Reward: " + reward + " Grasp " + graspProgress + " Lift " + liftProgress +
            //           " Hoop " + hoopProgress + " Sparse " + sparseGraspReward);
            break;
        default:
            break;
    }
}
```

B: VLM Questioning Prompts



I plan to RL train a "Needle though a Hoop" where a needle placed on ground needs to be inserted through an elevated hoop. Divide this task into 2 simple subtasks by understanding the scene using different camera views.

Here's a breakdown of the "Needle Through a Hoop" task into two simpler subtasks, leveraging different camera views for effective RL training:

Subtask 1: Needle Pickup and Orientation

- **Camera View:** A low, close-up perspective focused on the needle and the robot's manipulator. This view provides a detailed understanding of the needle's position and orientation on the ground. Ideally, a view directly above the needle would be best for grasping.
- **Objective:** The robot must successfully grasp the needle and bring it into an upright, near-vertical orientation. This requires precise movement and manipulation skills.
- **Reward Function:** The reward could be based on:
 - How close the robot's gripper is to the needle's center.
 - How successfully the robot grasps the needle (maybe using pressure sensors in a simulation).
 - How vertically oriented the needle becomes relative to the hoop (a vector or angle comparison).
 - Penalty for excessive movements (to avoid erratic behaviour).

Subtask 2: Hoop Insertion

- **Camera View:** A broader perspective encompassing both the hoop and the robot holding the oriented needle. This view provides spatial context, allowing for proper trajectory planning towards the hoop. This camera angle could be a little below and towards the hoop to clearly identify its vertical extent and centre.
- **Objective:** With the needle upright, the robot must move the needle and precisely insert it through the hoop. This requires good spatial reasoning and precise trajectory control.
- **Reward Function:** The reward could be based on:
 - The distance of the needle's tip from the hoop's center.
 - The alignment of the needle's long axis to the hoop's normal axis.
 - The needle fully penetrating the hoop.
 - Penalty for collisions with the hoop, or needle slipping during movement.

Training Strategy:

The two subtasks can be trained sequentially or in an integrated fashion.

- **Sequential:** First, train the agent extensively to achieve mastery of Subtask 1 (needle pickup and orientation). After a satisfying performance on this subtask, transfer these trained parameters as the base for Subtask 2, thereby making it an easier task for the reinforcement learning agent.
- **Integrated:** Train the agent to complete the whole task at once (although with a multi-part reward system). An epsilon-greedy exploration strategy should help with the more complex decision-making requirements that might occur. This has the potential for better generalization but might require longer training.

By breaking down the complex task and using carefully designed reward systems with distinct visual perspectives for the respective phases of this training task, reinforcement learning becomes significantly more efficient and scalable. Each of these camera perspectives adds necessary information to succeed, making RL training manageable and improving learning outcomes substantially.

I am trying to understand which camera position(s) and camera types will be best for this task. I have a camera view called default view that is between front view and top view when I am testing with just 1 camera and I have 2 more view, one front view and other side view when I am using two cameras. As for the camera types there are three types: rgb camera which is a normal camera, depth camera that shows the depth of objects via different shades and finally segmentation camera that gives strong colors to only important object needle, hoop and robot. I will share the images of these views and camera types and then give you a list of the combos I am interested in. Also you then have to rank them in order of how good they will be for the training. for each view+type combo I will share a init image that shows the initial state and a goal image that shows how the scene will look at completion

Please share the images. I need to see the different camera views and types to effectively rank the combinations for your reinforcement learning task. Once you provide the images, I can assess how well each view and camera type captures the relevant information for each subtask (needle picking, lifting, insertion). My ranking will consider factors like:

- **Observability of the Needle:** How clearly is the needle's position and orientation visible? This is crucial for all three subtasks.
- **Observability of the Hoop:** Similarly, clear visibility of the hoop's position and hole is important, especially for subtasks 2 and 3.
- **Observability of the Robot:** The robot's position and gripper are essential for understanding its actions and providing feedback during training.
- **Depth Information Relevance:** Depth information might be extremely helpful for judging distances and avoiding collisions, particularly during the lifting and insertion phases.
- **Segmentation Benefits:** Segmentation simplifies the scene and allows the agent to focus only on essential elements. This should reduce complexity and speed up training, but it might also limit the agent's robustness to unforeseen situations.
- **Viewpoint Completeness:** A single viewpoint might miss crucial information. Multiple views provide redundancy and a richer understanding of the scene.

I look forward to seeing your images so I can give you a detailed and informative ranking.

I will share the images over various prompts and then when I am done giving all images, only then will I give you the list of combos, don't make predictions about the images before that

Excellent! I now have all the image pairs. Please provide the list of combinations you want me to rank. I'll then provide a ranked order based on their suitability for your reinforcement learning task.

before i give my list, i will ask u some questions

Great! Ask away. I'll do my best to answer your questions.

q1) which camera type is best to know that goal state has been reached, i.e. that the needle is in the hoop

For definitively knowing the goal state (needle in the hoop), the **segmentation camera** is the best option. While the RGB camera *might* show the needle in the hoop, there's a higher chance of ambiguity or misinterpretation, especially if the lighting is poor or the needle is partially obscured. The depth camera would provide some distance information which could help determine if the needle is close enough to the hoop, but doesn't explicitly show the *successful* insertion. The segmentation camera, by highlighting the needle and hoop with distinct colors, offers the most reliable way to verify the goal state through straightforward pixel-level checks.

q2) which camera view is the best to know that the needle is inside the hoop

The best camera view to determine if the needle is inside the hoop would depend on the hoop's orientation and the needle's insertion angle. However, a **side view** offering a profile perspective directly looking at the hoop's plane is likely the best. A front view might obscure whether the needle is truly *through* the hoop or just resting against it. A top-down view (your 'default' view which seems somewhat between top and front) would be ambiguous. The side view minimizes occlusion and gives a clearer view of the needle's relationship to the hoop's opening.

q3) which camera type is best to know that the needle has been lifted

For determining if the needle has been lifted, the **depth camera** is the best choice. It provides depth information, allowing the RL agent to easily perceive the change in distance between the needle and the surface from which it was lifted. While an RGB camera might show a visual change, it wouldn't give a quantitative measure of the lift height. The segmentation camera would just highlight the needle, but wouldn't explicitly convey height information. The depth information directly addresses the requirement of knowing whether a lift has happened.

q4) which camera view is the best to know that the needle has been lifted

q5) which camera view combination is better: (1) one camera with default view (2) one camera with depth view and another with side view (3) two different camera with default view (different as differnt camera types). rank them

Ranking the camera view combinations from best to worst for the needle-threading task:

1. (2) **One camera with front view and another with side view:** This combination offers complementary perspectives. The front view is excellent for monitoring the vertical lifting and initial insertion, while the side view provides a clear profile view crucial for determining if the needle fully penetrates the hoop. The depth camera, while it could also be useful, will be far less crucial for this particular decision.
2. (3) **Two different cameras with front view (different camera types):** This offers some redundancy, allowing you to evaluate lift height in two ways using a depth and RGB cameras or, other pairings to reduce reliance on any one type, for example RGB+segmentation). Although potentially powerful if you leverage differences properly, it needs the specific capabilities combined (like a front RGB to clearly view position combined with the Front depth view for lift). It might be only marginally less useful than option (2).
3. (1) **One camera with default view:** The default view, being somewhere between a front and top-down view, compromises the clarity of both. It might not provide the reliable signals needed for accurate control.

The main idea here is redundancy of views and to make the job less challenging by removing unnecessary reliance on the relative angle, position etc. The challenge is not the view, rather in how clearly can the specific action's success or failure be easily observed by your RL agent, a segmentation view aids in extracting those specifics by clearly marking and removing all irrelevant items from view

C: VLM RANKING PROMPTS

now for the lists that I want u to rank, I will give u multiple lists over various next prompts and u have to rank the combos for each list i.e. for list 1 rank all combos in list 1, for list 2 rank all combos in list 2 and so on

Understood. I will wait for your lists and rank the combinations within each list separately. Please provide them one at a time.

list 1: combinations of depth and rgb camera types. remember the responses u gave me for the 5 questions i asked before that depth camera best for front view and front + side view cameras best.

1. 1 rgb camera default view
2. 1 rgb camera default view + 1 depth camera default view
3. 1 rgb camera front view + 1 rgb camera side view
4. 1 rgb camera front view + 1 depth camera side view
5. 1 depth camera front view + 1 rgb camera side view

Here's a ranking of the combinations in List 1, taking into account your previous questions and the strengths of depth and RGB cameras:

1. (5) 1 depth camera front view + 1 RGB camera side view: This is the strongest combination. The depth camera front view provides excellent information for lifting, and the RGB camera side view offers a good view for assessing needle placement into the hoop. This leverages each camera type's strengths effectively.
2. (3) 1 RGB camera front view + 1 RGB camera side view: While lacking depth information for precise lifting, this offers decent visual information. The combination of views provides good overall context, though slightly less reliable information concerning lift heights and accurate needle position within the hoop than (5).
3. (2) 1 RGB camera default view + 1 depth camera default view: The default view, being neither pure front nor pure side, is less useful for either of the subtasks. This option also suffers because both cameras observe the same slightly poor view point resulting in only redundant information
4. (4) 1 RGB camera front view + 1 depth camera side view: This combination isn't ideal as depth information from the side view isn't as useful for the vertical lift needed to get to the right position. This will cause unnecessary complications.
5. (1) 1 RGB camera default view: This is the weakest combination as it relies solely on one camera type (RGB) and suboptimal viewpoint. Depth information for critical positioning is entirely missing making this setup most likely far harder to get good results.

The ranking prioritizes combinations that use depth information for lifting and leverage side and front views to avoid occlusion issues in checking needle placement within the hoop.

list 2: combinations of segmentation and rgb camera types. remember the responses u gave me for the 5 questions i asked before that segmentation camera best for side view and front + side view cameras best

1. 1 rgb camera default view
2. 1 rgb camera default view + 1 segmentation camera default view
3. 1 rgb camera front view + 1 rgb camera side view
4. 1 segmentation camera front view + 1 rgb camera side view
5. 1 rgb camera front view + 1 segmentation camera side view

Here's a ranking of the combinations in List 2, considering the strengths of segmentation and RGB cameras and the insights from your previous questions:

1. (5) 1 RGB camera front view + 1 segmentation camera side view: This is likely the best combination. The RGB camera front view provides good context for the robot's actions and general needle positioning for initial positioning. The segmentation camera side view offers the most unambiguous verification for determining if the needle successfully passes through the hoop, by isolating the needle from the background.
2. (4) 1 segmentation camera front view + 1 RGB camera side view: This is a strong alternative. It will offer extremely good results to know if the needle has passed through the hoop. However, a side camera view is only needed for this determination meaning you might end up relying far too heavily on that information which could complicate the process. Although similar to option 1 this does still benefit from the segmented camera front view ensuring good, solid verification of correct actions early on in the operation
3. (3) 1 RGB camera front view + 1 RGB camera side view: This is a decent baseline, but lacks the clear goal-state verification of the segmentation camera, meaning many more samples will be required for sufficient learning. It might miss some information that it will be hard to fix during the training stage making a perfect placement much harder than what would happen with the use of the segmented camera
4. (2) 1 RGB camera default view + 1 segmentation camera default view: The default view, being a compromise between front and top-down views, isn't ideal for either needle manipulation or verifying successful needle passage. It might also result in difficulties as one of the cameras (RGB) uses up data without clearly assisting in determining the critical information and instead relying on the information found using the other segmented camera, similar to problems found with options 4 & 5. This combination doesn't utilise the strength of either cameras as the viewpoints don't strongly favor either camera.
5. (1) 1 RGB camera default view: This is the weakest combination as it completely lacks the clear goal state recognition a segmentation camera would offer.

The ranking emphasizes the segmentation camera's role in goal state confirmation and combines this with an RGB view offering better context around the positioning and movement of the robot and needle during the operation itself. Better results are likely with the stronger pairings and clear views in 4 & 5 versus 1, 2 & 3

list 3: combinations of depth and segmentation camera types. remember the responses u gave me for the 5 questions i asked before that depth camera best for front view, segmentation camera best for side view and front + side view cameras best

1. 1 depth camera default view
2. 1 segmentation default view
3. 1 depth camera front view + 1 depth camera side view
4. 1 segmentation camera front view + 1 segmentation camera side view
5. 1 segmentation camera front view + 1 depth camera side view
6. 1 depth camera front view + 1 segmentation camera side view

Here's a ranking of the combinations in List 3, prioritizing depth information for lifting and segmentation for precise placement verification:

1. (6) 1 depth camera front view + 1 segmentation camera side view: This is the optimal combination. The depth camera front view excels at monitoring the vertical lift, while the segmentation camera side view provides the clearest and most unambiguous signal for successful needle placement through the hoop.
2. (4) 1 segmentation camera front view + 1 depth camera side view: This is a strong contender. The front segmentation view still offers robust goal verification, although, depth information is less important in the front view since we primarily need a side view in order to get a precise sense of whether it passed through or not. However, relying too heavily on one type of information could also hinder performance and create unnecessarily complex conditions. Having redundant means of performing the same task tends to make RL operations considerably simpler as it removes some of the reliance on having perfect accuracy, removing this type of complexity greatly assists with efficient data processing.
3. (2) 1 depth camera front view + 1 depth camera side view: While both depth cameras might help somewhat, this completely lacks a high degree of certainty when determining if the goal was achieved, and is more heavily weighted towards a single means of completing this which will almost certainly reduce learning efficiency and results in lower quality placement and completion accuracy.
4. (5) 1 segmentation camera front view + 1 segmentation camera side view: Redundant segmentation cameras might provide more robust verification but at high cost of processing complexity, redundant cameras offer improved chances of avoiding incorrect signals but comes at high computational cost. Similar issues found in option 3 however only amplified by redundancy.
5. (3) 1 segmentation camera default view: Relying on a single view that isn't optimal for lifting, placement detection can yield inconsistent and problematic learning which means lower quality overall resulting placements. Without any depth information or clear view for evaluating actions or their effectiveness, successful training is considerably lower and less reliable.
6. (1) 1 depth camera default view: Similar to option (3), the suboptimal view makes it less efficient, which will affect learning considerably. Using only depth cameras will have more issues related to accurately identifying goal conditions which will end up negatively affecting completion and learning in the long run

The top rankings prioritize the combination of depth for lifting and segmentation for the clear verification needed to reliably detect the achievement of the needle insertion goal state. redundancy should not be seen as a purely negative thing as it improves learning and placement, although some methods to create redundancy create un-necessary processing overheads.

D: VLM REWARD GENERATION PROMPTS

RL reward generation for a needle through hoop scene.

You have these things at your disposal:

gameObject NeedlePickupPoint: represents the point the needle needs to be picked up at
Vector3 liftPosition: represents point just above the needle at the same height as the hoop hole
gameObject NeedleEndPoint: represents the end point of needle that needs to pass through hoop
gameObject ToolMidpoint: represents the mid point of the jaws
Goal: represents the goal object

You also already have a variable 'grasped' that is set to true/false in another function when the jaws grasp/ungrasp the needle and a variable 'lifted' that starts as false and becomes true when needle is lifted to liftPosition and stays true after that.

Sum all the different rewards in a variable curr_reward before calling the following:

```
float sparseGraspReward = 0.0f;  
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {  
    sparseGraspReward = 2.0f;  
}
```

```
float curr_reward += sparseGraspReward;  
AddReward(curr_reward - reward); // reward already initialized  
reward = curr_reward;  
Debug.Log("Reward: " + reward)
```

Only the sparseGraspReward is a sparse reward, all else rewards you need to make them dense distance based rewards.

Another function will do the success on goal touch stuff, you just need to make the reward that makes the needle go to goal

Make the sum of reward range from 0 to 9.

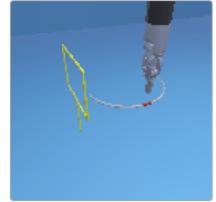
Only give me the body not the function header.

```
float distNeedlePickup = Vector3.Distance(NeedlePickupPoint.transform.position, ToolMidpoint.transform.position);  
float pickupReward = 1.0f - Mathf.Clamp01(distNeedlePickup / 0.5f); // Adjust 0.5f to control pickup reward falloff distance  
  
float distLift = Vector3.Distance(liftPosition, ToolMidpoint.transform.position);  
float liftReward = grasped ? (1.0f - Mathf.Clamp01(distLift / 0.3f)) * 2f : 0f; // Stronger reward for lifting, adjust falloff  
  
float distNeedleGoal = Vector3.Distance(NeedleEndPoint.transform.position, Goal.transform.position);  
float goalReward = 1.0f - Mathf.Clamp01(distNeedleGoal / 0.7f); // Adjust 0.7f to control goal reward falloff distance  
goalReward *= lifted ? 1f : 0.5f; // Reduce reward if not lifted  
  
float dist_needle_hoop = Vector3.Distance(NeedleEndPoint.transform.position, Goal.transform.position);  
float dist_reward = 0f;  
if (dist_needle_hoop < 0.1f) { // Reward for needle end proximity to hoop  
    dist_reward = 3f;  
}  
else {  
    dist_reward = 1.0f - Mathf.Clamp01(dist_needle_hoop / 1.0f) * 2f;  
}  
  
float curr_reward = pickupReward * 2f + liftReward + goalReward * 3f + dist_reward;  
  
// Clamp the reward to the range [0, 9]  
curr_reward = Mathf.Clamp(curr_reward, 0f, 9f);  
  
float sparseGraspReward = 0.0f;  
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {  
    sparseGraspReward = 2.0f;  
}  
  
curr_reward += sparseGraspReward;  
AddReward(curr_reward - reward);  
reward = curr_reward;  
Debug.Log("Reward: " + reward);
```

 Use code with caution.

C#

« »



The two images show two views of looking at a needle through a hoop scene.

The first image depth_front shows the scene in front view as it looks in a depth camera and the second image rgb_side shows a side view of the scene as it looks in a default camera

You have these things at your disposal:

gameObject NeedlePickupPoint: represents the point the needle needs to be picked up at
Vector3 liftPosition: represents point just above the needle at the same height as the hoop hole
gameObject NeedleEndPoint: represents the end point of needle that needs to pass through hoop
gameObject ToolMidpoint: represents the mid point of the jaws

Goal: represents the goal object

You also already have a variable 'grasped' that is set to true/false in another function when the jaws grasp/ungrasp the needle and a variable 'lifted' that starts as false and becomes true when needle is lifted to liftPosition and stays true after that.

Sum all the different rewards in a variable curr_reward before calling the following:

```
float sparseGraspReward = 0.0f;  
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {  
    sparseGraspReward = 2.0f;  
}
```

```
float curr_reward += sparseGraspReward;  
AddReward(curr_reward - reward); (reward already initialized)  
reward = curr_reward;  
Debug.Log("Reward: " + reward)
```

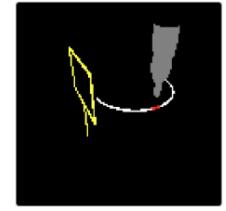
Only the sparseGraspReward is a sparse reward, all else rewards you need to make them dense distance based rewards.

Another function will do the success on goal touch stuff, you just need to make the reward that makes the needle go to goal

Make the reward range from 0 to 9.

Only give me the body not the function header.

```
float distanceToPickup = Vector3.Distance(ToolMidpoint.transform.position, NeedlePickupPoint.transform.position);  
float pickupReward = 9f * (1f - Mathf.Clamp01(distanceToPickup / 0.1f)); // Adjust 0.1f as needed for the desired pickup range.  
  
float distanceToLift = 0f;  
if (grasped) {  
    distanceToLift = Vector3.Distance(NeedlePickupPoint.transform.position, liftPosition);  
    float liftReward = 9f * (1f - Mathf.Clamp01(distanceToLift / 0.1f)); // Adjust 0.1f as needed for lift distance  
    curr_reward += liftReward;  
}  
  
float distanceToGoal = Vector3.Distance(NeedleEndPoint.transform.position, Goal.transform.position);  
float goalReward = 9f * (1f - Mathf.Clamp01(distanceToGoal / 0.3f)); // Adjust 0.3f as needed for the desired goal distance.  
  
curr_reward += pickupReward;  
curr_reward += goalReward;  
  
float sparseGraspReward = 0.0f;  
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {  
    sparseGraspReward = 2.0f;  
}  
  
curr_reward += sparseGraspReward;  
AddReward(curr_reward - reward);  
reward = curr_reward;  
Debug.Log("Reward: " + reward);
```



The two images show two views of looking at a needle through a hoop scene.

The first image `depth_front` shows the scene in front view as it looks in a depth camera and the second image `seg_side` shows a side view of the scene as it looks in a segmentation camera that just shows the needle, hoop and robot

You have these things at your disposal:

```
gameObject NeedlePickupPoint: represents the point the needle needs to be picked up at
Vector3 liftPosition: represents point just above the needle at the same height as the hoop hole
gameObject NeedleEndPoint: represents the end point of needle that needs to pass through hoop
gameObject ToolMidpoint: represents the mid point of the jaws
Goal: represents the goal object
You also already have a variable 'grasped' that is set to true/false in another function when the jaws grasp/ungrasp the needle and a variable 'lifted' that starts as false and becomes true when needle is lifted to liftPosition and stays true after that.
Sum all the different rewards in a variable curr_reward before calling the following:
float sparseGraspReward = 0.0f;
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {
sparseGraspReward = 2.0f;
}

float curr_reward += sparseGraspReward;
AddReward(curr_reward - reward); (reward already initialized)
reward = curr_reward;
Debug.Log("Reward: " + reward)
```

Only the `sparseGraspReward` is a sparse reward, all else rewards you need to make them dense distance based rewards.
Another function will do the success on goal touch stuff, you just need to make the reward that makes the needle go to goal
Make the sum of reward range from 0 to 9.
Only give me the body not the function header.

```
float dist_needle_pickup = Vector3.Distance(NeedlePickupPoint.transform.position, ToolMidpoint.transform.position);
float dist_needle_endpoint_goal = Vector3.Distance(NeedleEndPoint.transform.position, Goal.transform.position);
float dist_needle_lift = Vector3.Distance(NeedleEndPoint.transform.position, liftPosition);

float graspReward = 0f;
if (grasped) {
    graspReward = 1f / (1f + dist_needle_pickup); //Inverse distance reward for grasping
}

float liftReward = 0f;
if (grasped && !lifted) {
    liftReward = 1f / (1f + dist_needle_lift); //Inverse distance reward for lifting
}
else if (lifted) {
    liftReward = 1f; //Full reward when lifted
}

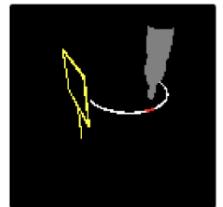
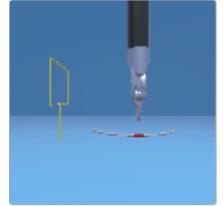
float goalReward = 0f;
if(lifted) {
    goalReward = 1f / (1f + dist_needle_endpoint_goal); //Inverse distance reward for reaching the goal
}

// Normalize rewards to range [0, 1] and scale.
float normalizedGraspReward = Mathf.Clamp01(graspReward);
float normalizedLiftReward = Mathf.Clamp01(liftReward);
float normalizedGoalReward = Mathf.Clamp01(goalReward);

// Combine and scale rewards to [0,9] range. Weights can be adjusted.
float totalReward = 3f * normalizedGraspReward + 3f * normalizedLiftReward + 3f * normalizedGoalReward;

float sparseGraspReward = 0.0f;
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {
    sparseGraspReward = 2.0f;
}

float curr_reward = totalReward + sparseGraspReward;
AddReward(curr_reward - reward);
reward = curr_reward;
Debug.Log("Reward: " + reward);
```



The two images show two views of looking at a needle through hoop scene.

The first image `rgb_front_init` shows the scene in front view as it looks in a default camera and the second image `seg_side_init` shows a side view of the scene as it looks in a segmentation camera that just shows the needle, hoop and robot

You have these things at your disposal:

`gameObject NeedlePickupPoint`: represents the point the needle needs to be picked up at
`Vector3 liftPosition`: represents point just above the needle at the same height as the hoop hole
`gameObject NeedleEndPoint`: represents the end point of needle that needs to pass through hoop
`gameObject ToolMidpoint`: represents the mid point of the jaws

`Goal`: represents the goal object
You also already have a variable 'grasped' that is set to true/false in another function when the jaws grasp/ungrasps the needle and a variable 'lifted' that starts as false and becomes true when needle is lifted to liftPosition and stays true after that.

Sum all the different rewards in a variable `curr_reward` before calling the following:

```
float sparseGraspReward = 0.0f;
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {
    sparseGraspReward = 2.0f;
}

float curr_reward += sparseGraspReward;
AddReward(curr_reward - reward); //reward already initialized
reward = curr_reward;
Debug.Log("Reward: " + reward)
```

Only the `sparseGraspReward` is a sparse reward, all else rewards you need to make them dense distance based rewards.

Another function will do the success on goal touch stuff, you just need to make the reward that makes the needle go to goal

Make the sum of reward range from 0 to 9.

Only give me the body not the function header.

```
float dist_needle_pickup = Vector3.Distance(NeedlePickupPoint.transform.position, ToolMidpoint.transform.position);
float dist_needle_endpoint_goal = Vector3.Distance(NeedleEndPoint.transform.position, Goal.transform.position);
float dist_needle_lift = Vector3.Distance(NeedleEndPoint.transform.position, liftPosition);

float graspReward = 1.0f - Mathf.Clamp01(dist_needle_pickup / 0.2f); //Scale reward to 0-1 based on distance. Adjust 0.2f as needed
float liftReward = lifted ? 1.0f - Mathf.Clamp01(dist_needle_lift/0.2f):0f; //Give reward only if lifted. Adjust 0.2f as needed
float goalReward = 1.0f - Mathf.Clamp01(dist_needle_endpoint_goal / 0.4f); //Scale reward to 0-1 based on distance. Adjust 0.4f as needed

float curr_reward = graspReward * 2.0f + liftReward * 3.0f + goalReward *4.0f; //Weight rewards. Adjust weights as needed.

float sparseGraspReward = 0.0f;
if (grasped && Arms[0].controller.GetJointCommand("jaw") < 0.1f) {
    sparseGraspReward = 2.0f;
}

curr_reward += sparseGraspReward;
AddReward(curr_reward - reward);
reward = curr_reward;
Debug.Log("Reward: " + reward);
```