Robot Vision and Learning: Final Project Report

Group members:

1. Lawrence Leroy Chieng Tze Yao 2401213369
2. Shui Jie 2401112104
3. Peterson Co 2401213365

1. Describe the Algorithm

A. <u>How Visual Information is Used</u>

   a. Picking Boxes:
      i. It uses segmentation data to identify boxes and compute their global positions.
      ii. Uses visual information to ensure boxes are reachable.
      iii. Prioritizes boxes with the most neighbors.
      iv. Returns the position of a reachable box with the most neighbors.

   b. Check If Spade is Empty:
      i. Use segmentation data to identify boxes and compute their global positions.
      ii. Only consider boxes that are reachable (rules out boxes outside pick up area and boxes inside bin)
      iii. Checks if among these boxes, there exists at least one elevated box (meaning it has been lifted by spade)
      iv. If true, spade is not empty, if false, spade is empty.

   c. Locate Bin:
      i. Uses bin's global position in the camera to retrieve its center position.

B. <u>Strategy to Pick Up Objects</u>

   a. Reuse baseline Phase 0 and Phase 1 code to align and do initial scoop
   b. On phase 2, reorient left spade downwards facing right spade, and reorient right spade slightly upwards to face left spade. This allows the boxes to slide down to the right spade only and empty left spade.
   c. On phase 3, the left spade does a slight sweep along the right spade to ensure remaining boxes are transferred and to push boxes already on the right spade inside (to prevent them from staying on spades edge). After this, left spade does a full movement and reorientation to face opposite side to prevent collisions when right spade moves towards bin

C. Strategy to Place Object (Phase 4)

a. Phase 4a: Achieve stable initial orientation before trajectory movement. Changes orientation while maintaining position.
b. Phase 4b: Locate center coordinates of bin using bin metadata; set a z-offset so that r2 reaches high enough to not collide with r1. Move r2 towards bin using the jacobian_drive() function
c. Phase 4c: continue moving towards bin, now with a negative y-offset (i.e., spade moves a little to the right) to provide more left-side room for dropping boxes. Motion stops when spade is above the centerpoint of the bin within a small tolerance.
d. Phase 4d: Rotate spade anticlockwise so boxes drop from the left of the spade into the bin.

D. How to plan the trajectory

a. Jacobian-Based Movement Control - Implements spatial twist control using the robot's Jacobian matrix to move to the bin.
   i. Computes dense Jacobian matrix (6 × DOF) for end-effector (index 9)
   ii. Separates linear (top 3 rows) and angular (bottom 3 rows) components
   iii. Uses pseudoinverse (J^+) of the Jacobian for inverse kinematics
   iv. Maps desired spatial twist (v) to joint velocities ($\dot{\theta}$) via $\dot{\theta} = J^+v$
b. Velocity scaling (0.3 by default) for controlled movement
c. Bin Approach
   i. Retrieves bin's global center position as target
   ii. Modifies target position:
      - Z-axis offset increased by 1.0 meters for clearance and avoid collision with r1 base link and bin
      - Y-axis adjusted by -0.1 meters for approach alignment
d. Uses position-based termination condition:
   - Monitors end-effector position relative to bin center
   - It uses 0.15m tolerance for position-matching
   - Checks both x and y coordinates for alignment

E. When to continue to the next scene

a. Phase 1: if pick_box does not return a target box, meaning there are no boxes within grasp range detected by the cameras, then the scene ends.
b. Phase 3: if spade is empty after scooping, then scene reiterates.

c. Phase 5: at the end of an iteration (i.e., pick - move - place), we have to decide whether to reiterate or to end. If boxes are placed into the bin in this iteration, then we simply return to phase 0, moving the spades back into ready position.
d. If no boxes are placed into the bin instead, we consider the following:
    i. fail_chances marks the iterations the robot is tolerated to make an empty grasp.
    ii. useless_time stores the unproductive time spent in this iteration
    iii. If the robot makes 3 consecutive empty grasps OR spends a continuous unproductive time above the given limit, the scene ends, deducing that the robot is unable to recover the remaining (observable) boxes

2. Describe your development process

We established a two-tiered approach to define our project objectives. Our baseline target focuses on achieving a performance score of 10 points through conventional hard-coded control mechanisms. This serves as a reliable benchmark for evaluating subsequent improvements. Upon establishing this baseline, we aim to leverage reinforcement learning techniques to enhance performance beyond the initial benchmark.

Given the inherent stochastic nature of reinforcement learning, this methodological approach ensures we meet fundamental project requirements while providing an opportunity to explore advanced reinforcement learning methodologies and broader robotics concepts.

2.1 Baseline

After we planned our approach, we then breakdown the baseline requirements and prioritize important tasks which we believe to be the key to achieving high performance. Namely, we identified several areas for improvement from the provided baseline.py

- Picking Up Boxes
    - We have noticed that baseline.py is very inconsistent when scooping up boxes, leading to it dropping a lot of boxes and not utilizing the end-effectors degree of freedom.

  - We solve this by reorienting end effectors in a way to maximize r2's boxes and mainly using r1 as an assistant, described in detail in the previous section.
- Collision Avoidance
  - baseline.py r2 collides with r1 every single time in its trajectory to the bin, which almost always leads to r2 dropping even more boxes and putting them in a non-reachable area
  - We fixed this by making r1 do an almost 180 degree turn to the left to allow r2 a collision-free passage
- Trajectory Calculation
  - baseline.py uses a hardcoded target pose. This is an issue as it no longer works when the bin's position changes. Using cartesian IK also makes it very unstable, leading to shaky movement which drops boxes.
  - We use twist coordinates in the body frame and spatial_jacobian_twist to achieve a more smooth movement. We also calculate the bin's center position using visual information from the camera and use it as the target position.
- Final Positioning Consistency
  - baseline.py does not do minute fine tuning to properly align spades for placement, leading to inconsistent drops.
  - We make a condition that keeps fine tuning r2 as long as it does not align with bin center
- Object Placement Method
  - baseline.py slides the boxes down the bin by reorienting the end effector downward. This requires far more detailed alignment and consideration of spade's length relative to bin center.
  - We have found that spades can rotate 360 degrees clockwise and counterclockwise. We opted to use 360 degree rotation to drop boxes as it only requires an approximate alignment of the end-effector center with the bin center. Boxes will drop and land in roughly the same x and y axis position it occupied in the spade, unlike sliding it down.

<u>Failed Experiments:</u>
Our initial idea was to have both r1 and r2 pick up boxes and have both place their boxes in the bin. However, we found that this is not optimal for the following reasons:

- Having both do independent scooping is not optimal as it drops a lot of the boxes, same with baseline.py

    - Bin cannot accommodate two spades on top of it and so both spades take turns in dropping their respective boxes, taking a longer time and affecting efficiency
    - Cannot really consistently reset to phase 0 if spade is empty unless both spades are empty. If one spade has boxes and the other has not, the spade cannot just reset and attempt to rescoop again.

Because of this, we came up with the idea to maximize r2's boxes and use r1 mainly as an assistant to avoid box dropping.

<u>Success Rate and Efficiency Trade-Off</u>:
In our experiments, we have found that >50% success rate is far easier to achieve than >1.5 efficiency. We have encountered cases where we have already achieved >50% success rate and >1.5 efficiency at the beginning, but there will be a few difficult boxes that have not yet been successfully placed in the bin. Robots will repeatedly attempt and fail to handle these boxes, which although does not affect success rate, it negatively affects efficiency. Due to the nature of these two metrics, where one is fixed and the other is not. We have created kill conditions specifically to maximize our efficiency:

- If the whole process has not finished after a fixed time, we assume it is because the robot has encountered difficult boxes and failed to handle it. This scenario will kill the process.
- If a robot consecutively fails to place boxes in the bin in 3 attempts, the same assumption is made and we kill the process.
- If boxes are no longer contained within the robot's predefined reachable perimeter, we do not allow the robot to attempt to pick them up and we kill the process. This perimeter was found through trial and error by exploring how far in the x and y axis a box can be until robot malfunctions trying to scoop them..

2.2 SAC agent

In parallel with our baseline implementation, we explored the application of reinforcement learning for task automation. We selected the Soft Actor-Critic (SAC) algorithm as our primary learning framework due to its advantageous characteristics in handling complex control tasks.

Soft Actor-Critic combines actor-critic architecture with maximum entropy reinforcement learning principles, utilizing an actor network that generates probability distributions over actions and dual critic networks for reward estimation. A distinguishing feature of SAC is its incorporation of entropy maximization, which promotes systematic exploration of the action space. Given the absence of prior demonstration data for our specific task, SAC's inherent stability and its ability to autonomously balance exploration and exploitation through temperature parameter adjustment make it particularly well-suited for our application. The algorithm's capacity to learn effectively from scratch, coupled with its robust performance in continuous control tasks, aligns well with the challenges presented in our robotics project.

2.3 Experimentations
Due to lack of prior experience, we first decided to directly approach the training with very naive parameters.

2.3.1 naive attempt
We defined our states as:

1. end effector position vector and rotation matrix of each arm, together with the joint angle and joint velocity of the joint of each arm (52 parameters)
2. global location of bin and each box (33 parameters)

This makes a total of 85 parameters to describe the state of the simulation.

We defined our reward function as

1. each box in bin result in reward of 10
2. every timestep result in punishment of 1

We defined our action as

1. 2 sapien pose for each arm (14 parameters)
2. We utilize our own Jacobian drive to move the arms given these values.

We defined our goal as

1. All the boxes in the bin.

After 2500 episodes, the agent failed to achieve any success on the tasks. Observation of rendered episodes shows the agent unable to make meaningful processes such as picking up the object.

2.3.2 semi-rl approach
While our baseline approach worked well, it was limited by having to move only one arm at a time to avoid collisions. This led us to refine our focus - training the SAC agent specifically for moving both arms to the bin while minimizing collisions.

We simplified the state space after realizing that the spade rotation needed to stay relatively constant for successful box transport. By removing rotation from the agent's considerations, we reduced complexity and allowed it to focus on the core task of coordinated arm movement.

We defined our states as:

1. end effector position vector and rotation matrix of each arm (24 parameters)
2. global location of bin and each box (33 parameters)

This makes a total of 57 parameters to describe the state of the simulation.

We modified our reward function. First we reduced the punishment for timestep as we realised its way too large for any form of success score to matter. Secondly, we focus more on getting close to the bin and minimal collision.

1. for every join of both arms that gets too close within a threshold, a punishment of 0.5
2. every timestep result in punishment of 0.1
3. each reduction in the distance from the end effector to the bin is rewarded: distance reduction * 1.
4. each increase in the distance from the end effector to the bin is punished: distance increment * 1.

We defined our action as

1. 2 global position for each arm (6 parameters)
2. We utilize our own Jacobian drive to move the arms to these positions with fixed rotations.

We defined our goal as

1. Both spades reach a point directly above the center of the bin.

After observing the model's failure to converge, we identified that the vast exploration space, combined with frequent negative rewards, was hindering effective learning. The absence of a prior dataset for guided learning further complicated this challenge.

To address this, we leveraged our baseline implementation as a trajectory generator. We focused specifically on phase 4d of the task, where boxes are already secured and the left arm follows a predetermined path to the bin. By recording these successful trajectories from our baseline, we were able to generate a dataset of effective movement patterns that successfully completed the bin-reaching objective. This approach provided the agent with valuable demonstration data, effectively reducing the

After implementing the trajectory-based training approach, we observed notable improvements in the agent's stability and performance, with the robot successfully navigating approximately halfway to the bin. This validates our strategy of using baseline-generated trajectories as prior knowledge to guide the learning process.

However, this approach introduced new challenges. The limited variety in our demonstration trajectories led to biased behavior, with the agent predominantly favoring right arm movement. Additionally, the agent began exhibiting unstable behavior patterns, characterized by abrupt movements that frequently dislodged boxes from the spades. These issues indicate that while our trajectory-based training provided initial guidance, the limited diversity in our demonstration data potentially restricted the agent's ability to learn robust, balanced dual-arm coordination.

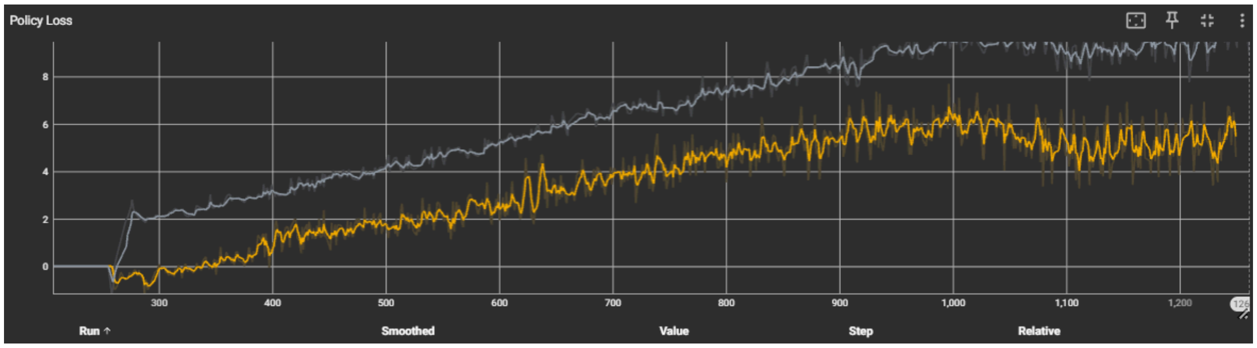


figure 1. Policy loss over 1250 episodes when compared between agent with prior trajectories (yellow) and without prior trajectories (grey). Though both agents failed to converge to a stable, decreasing policy loss, the agent with prior successful trajectories shows significant improvement with the lower policy loss.

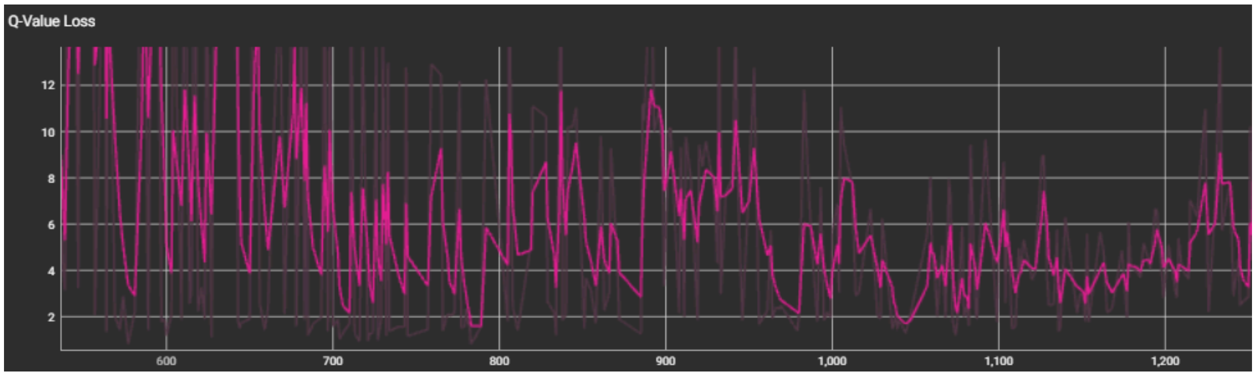


figure 2. Shows the Q-value loss for the agent with prior success trajectories, showing a decreasing trend

3. Future work and limitation

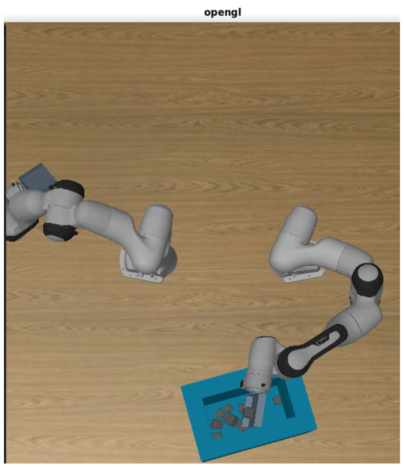
In conclusion, we managed to achieve decent performance with our baseline, mechanical approach. However, we did not manage to successfully train a RL agent that can help overcome the bottleneck in our baseline.

For future development, we believe the key lies in providing better defined states, reward functions and we will need to use our baseline to generate more successful trajectories as training dataset for our agents.

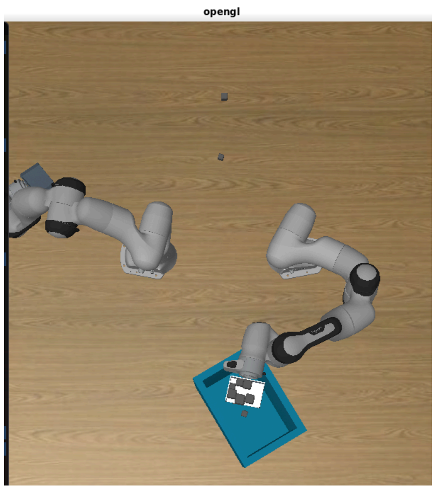
Experiment Results

Random seed	Execution time (s)	Success rate (%)	Efficiency (boxes per minute)	Remarks
1	142.61	100	4.21	
2	170.69	70	2.46	box fell out of robot grasp range
3	245.85	100	2.44	
4	241.38	100	2.49	
5	201.77	80	2.38	box fell out of robot grasp range
6	146.85	80	3.27	box fell out of robot grasp range & bin
7	216.46	90	2.49	box fell out of robot grasp range
8	179.65	100	3.34	
123	145.28	100	4.13	
321	220.76	100	2.72	
Average	**191.13**	**92** (mode=100)	**2.993**	

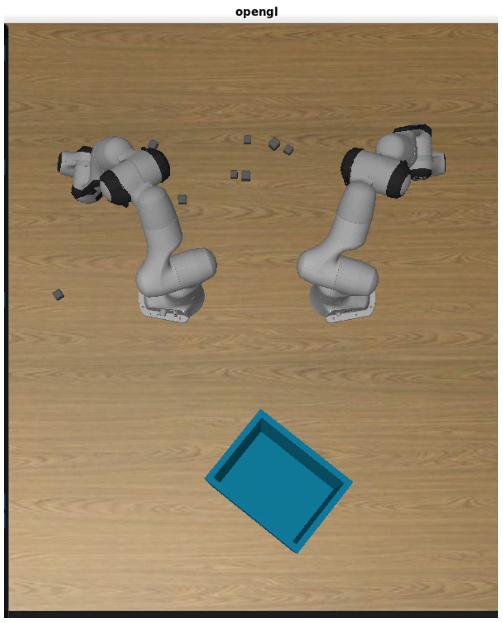
Screenshots



[successfully placing all boxes in bin]



[transporting as much as 6 boxes in one iteration]



[box already out of grasp at the start of scene (seed=2)]