Object Selection from Table: A Multimodal Approach Using Text Input

Karthik Ragunath, Sasi Thomala, Weicheng Liu

Introduction

Our project aims to tackle the classic warehouse automation problem:

Efficient robotic arm pathing for reaching various objects on a table, directed by text input.

This project effectively combines object detection and reinforcement learning to guide a robot arm using the best gripper-to-object movement plan found during learning.

FASTER-RCNN: We make use of the FASTER-RCNN model, taking advantage of its pre-existing abilities and further refining it with a custom dataset for improved object recognition.

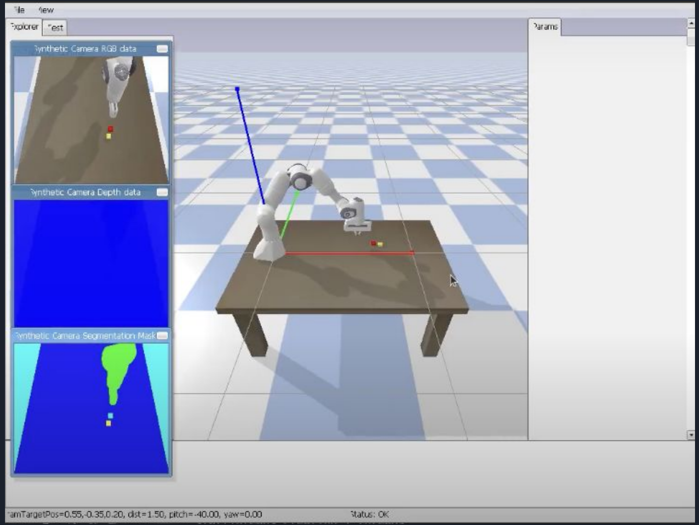
Optimal Path Planning: The Proximal Policy Optimization (PPO) algorithm is used to determine the best path for the robot arm based on object coordinates, label embeddings, and robot camera images.

Implementation

The robot arm was simulated using the **Pybullet** environment.

Real-time object detection was performed using the **Faster-RCNN** model which we fine-tuned using custom data.. Target objects were provided as text input.

The robotic arm path was learned (through our training) using the **Proximal Policy Optimization (PPO)** algorithm which accepted inputs from the object detection stage.



```
amTargetPos=0.55,-0.35,0.20, dist=1.50, pitch=-40.00, yaw=0.00          Status: OK
```

Custom Training Pipeline - Faster-RCNN

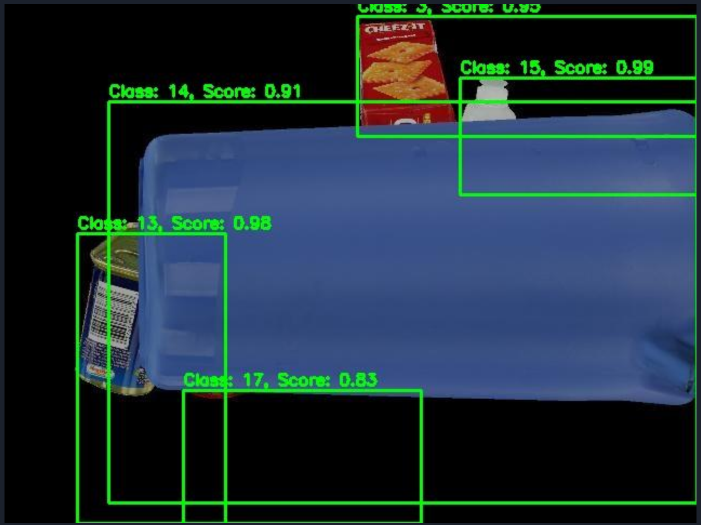
We were able to fine-tune the default Faster-RCNN model using a problem specific dataset, which in this case was Roboflow’s Household Objects Dataset accessible at here.

Roboflow’s dataset contains a comprehensive collection of household objects that were fed into the pretrained Faster-RCNN model.

Following fine tuning, our custom network was able to perform inference on Roboflow’s dataset, allowing us to extract bounding boxes, labels, and etc. from the predictions.

The extracted data was fed into our PPO algorithm to begin reinforcement learning for optimal robot path.

Inference Result On A Sample



Proximal Policy Optimization (PPO) - Path Planning through Reinforcement Learning

Our implementation of the PPO accepts the end effector joint coordinates, grasp finger conditions and also the target object's coordinates as input (observation state) to learn the robot's optimal path.

PPO is a type of policy gradient method for reinforcement learning, which alternates between **sampling data through interaction** with the environment and optimizing a **"surrogate" objective function**.

The s**urrogate objective function** stops our algorithm from generating a new path plan (policy) that deviates too far from the new old plan (policy). This allows for smooths and regularizes our policy updates while the arm is searching for a path.

PPO Algorithm Implementation Details In Our RL - Environment

Observation State:
Tensor of length 8 which represents the state of robot arm and the environment at a given time.

States captured:
state_robot: Position of robot’s end effector (3 dim: x, y, z coordinates)
state_fingers: The state of two fingers of the robot’s gripper (2 dim)
state_object: The current position of the target object (3 dim: x, y, z coordinates)

—--------------------------
Action State:
Tensor of length 4
First 3 dimensions - dx, dy, dz, are the differentials to be added to get the new end effector position
The last element in action state denote the opening/closing condition of fingers

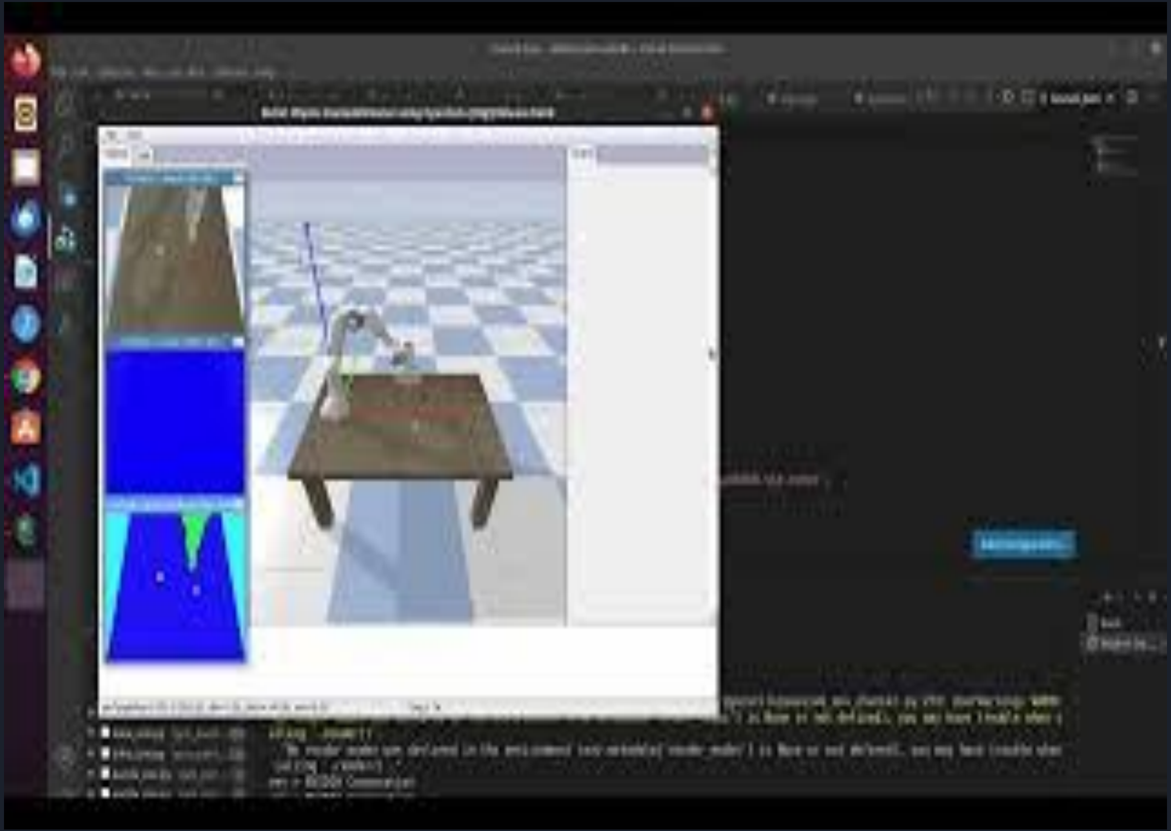
The new positions are fed as input to PyBullet’s inverse kinematic functions which calculates the required joint positions to achieve the required end-effector position

—-------------------------
Rewards - absolute value of difference between object’s position and end effector gripper position.

DEMO - Grasping different colors of Legos based on text input from the user.

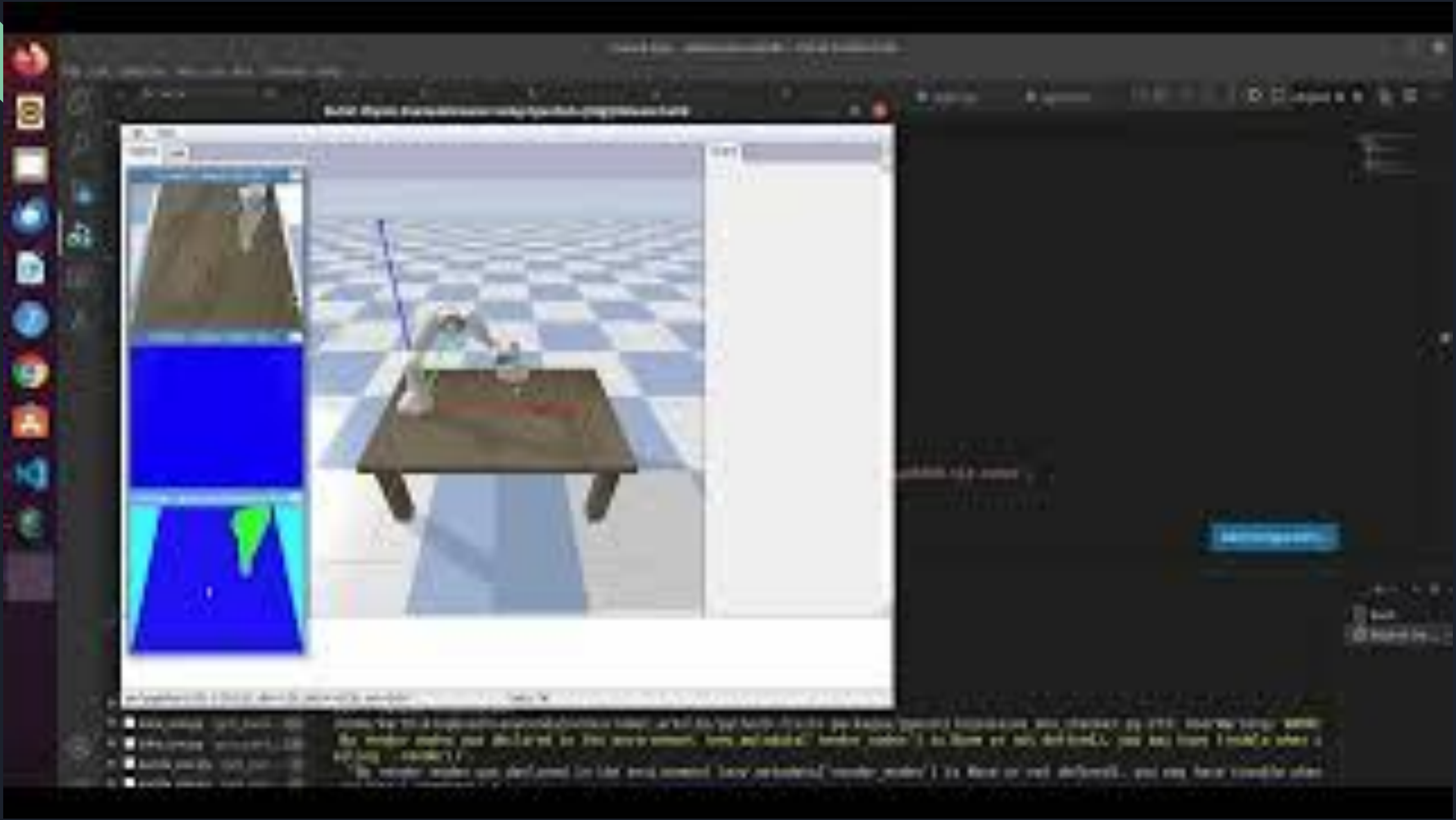
PPO RL Motion Planning - Reaching Red Cube

https://youtu.be/51LI3-b4MFQ



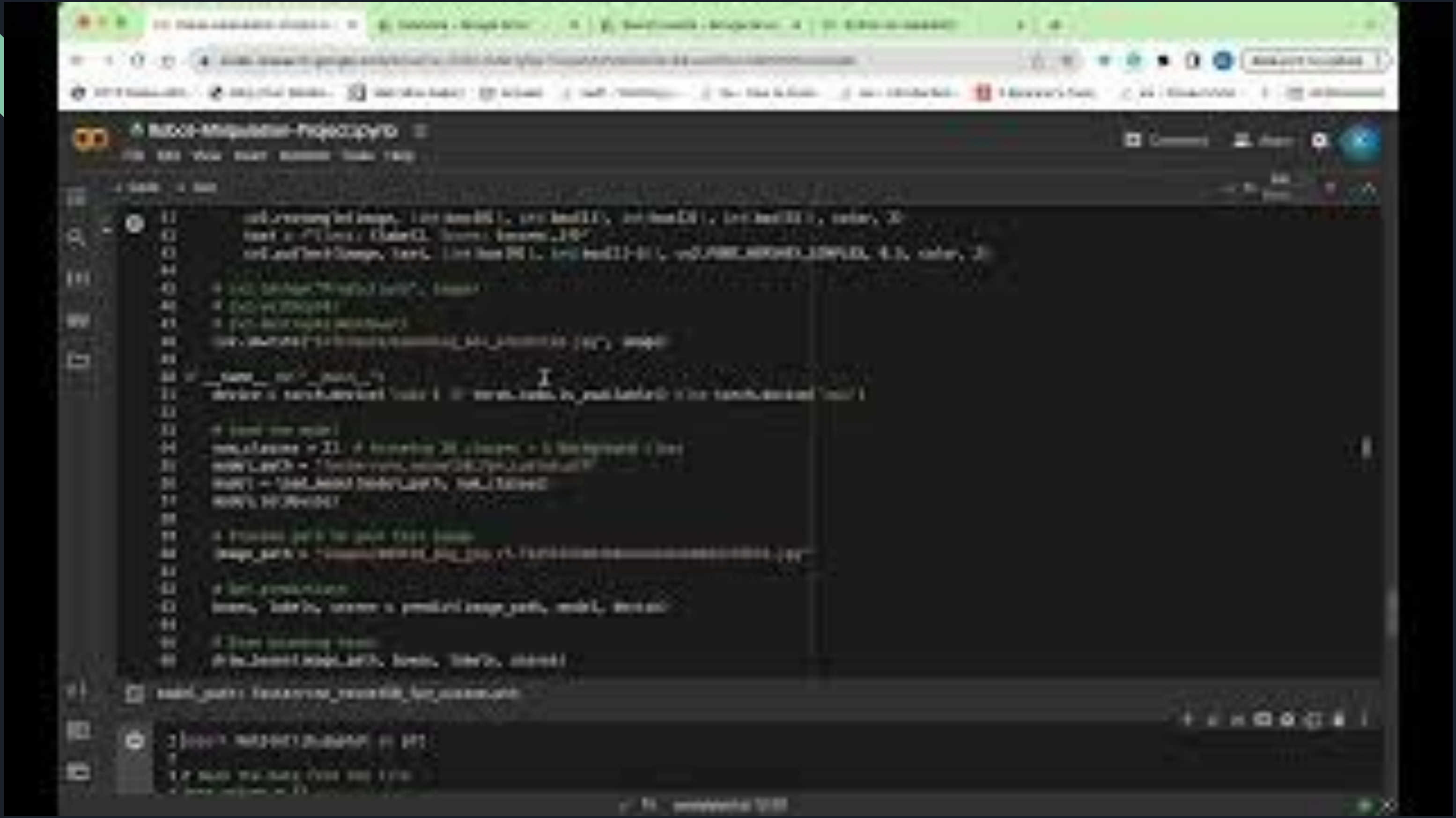
PPO RL Motion Planning - Reaching Yellow Cube

https://youtu.be/CxDWeZtJwTI



```
[illegible]
```

Demo - Faster RCNN Component (https://youtu.be/BO6FHZmbYDo)



```
        cv2.rectangle(image, (int(box[0]), int(box[1])), (int(box[2]), int(box[3])), color, 2)
        text = f"Class: {label}, Score: {score:.2f}"
        cv2.putText(image, text, (int(box[0]), int(box[1])-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    # cv2.imshow("Predictions", image)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    cv2.imwrite("[illegible].jpg", image)

if __name__ == "__main__":
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

    # Load the model
    num_classes = 21 # [illegible]
    model_path = "[illegible]"
    model = load_model(model_path, num_classes)
    model.to(device)

    # [illegible]
    image_path = "[illegible].jpg"

    # Get predictions
    boxes, labels, scores = predict(image_path, model, device)

    # Draw bounding boxes
    draw_boxes(image_path, boxes, labels, scores)
```

Future Scope

Although we were able to use RL to find a optimal path from gripper to object, the current implementation of the project does not take into consideration obstacles. We hope improve by running our algorithms in the sim with obstacles to observe how the robotic arm will behave.

Furthermore, currently our planning algorithm allows the the robot arm to optimally reach the object. However, because we are using RL (rather than using purely inverse kinematics based approach) to plan the path we have yet to look into getting the gripper to grab and lift the target object.

Instead of specifying the objects current positions in code directly, using sensor info to identify the required object based on visual characteristics with the Faster-RCNN model available in our pipeline.

Thank you!