Lab 3:
Stretch Perceives and Servos

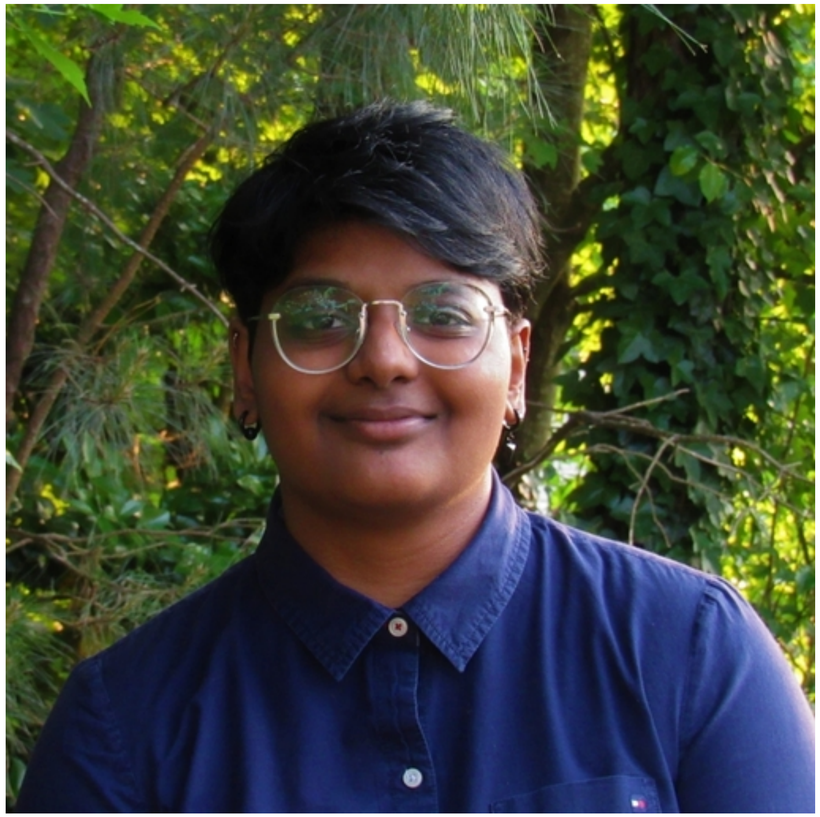
Zackory Erickson

Carnegie Mellon University
The Robotics Institute

16-762: Mobile Manipulation
Spring 2026

Acknowledgment

Thank you to Kavya Puthuveetil who helped create this lab.



Lab 2 Goals

In this lab, you'll be asked to build on the IK concepts you explored in Lab 2, this time using visual perception to identify, track, and grasp objects of interest, implemented using ROS2!

We'll be leveraging YOLO-E as part of our visual perception module, enabling zero-shot, real-time object detection and segmentation based on a text prompt.

Part #1

Detect a target object in the scene using the in-gripper camera and have the robot move both its base and gripper to follow it.

Part #2

Detect a target objects in the scene using the head camera and have the robot move to grasp them.

What you will submit

1. Your completed code files (.py) for parts #1 and #2 (see previous slide)
    - Part #1: `object_detector.py` and `target_following.py`
    - Part #2: `object_detector_pcd.py` and `grasp_objects.py`
2. Video (.mp4 or .mov) showing the robot following a target object as it moves between at least 3 distinct positions with respect to the robot
3. Videos (.mp4 of .mov) showing the robot reaching to and grasping at least 3 different objects of various shape/size and/or physical properties. You may submit one video for all three grasps or one video for each.
4. A single object_queries.yaml file that includes the full set of objects you used across both parts. It's okay if some items are commented out because they weren't used in part 2 but were in part 1 (or vice versa)

Lab 3

- Lab 3 files: https://github.com/Zackory/mm2026/tree/main/lab3
- **Wednesday, March 11th** (2 weeks + Spring break)
- Please submit your team member names so we can link you in Canvas: https://forms.gle/nNzqL7iCRX6jCxUj9

Pip installs

Put the battery into **SUPPLY** mode.

Run:

```
cd 16762/your_name
source env/bin/activate
```

This is new!

```
pip3 install ultralytics torch torchvision --index-url
https://download.pytorch.org/whl/cpu
```

Reminder: How to connect into the robot

- We have two robots — 3159 and 3160. These numbers are on the back of the robot near the on/off switch.
- Username: hello-robot
- Password: 16762cmu!
- stretch-se3-3159:
    - **SSH**: ssh -X hello-robot@stretch3159.wifi.local.cmu.edu
    - **Anydesk**: 1943010008
- stretch-se3-3160:
    - **SSH**: ssh -X hello-robot@stretch3160.wifi.local.cmu.edu
    - **Anydesk**: 1424568870

Lab 3 files

https://github.com/Zackory/mm2026/tree/main/lab3

Part #1: Position-Based Visual Servoing

Copy the following 5 files:

1. **`object_detector.py`**
    a. takes in color and depth frame's from the robot's in-gripper camera
    b. feeds color frames into an off-the-shelf object detector (YOLO-E) to get an object mask
    c. finds the 3D pose of the object and publishes it as the goal position for the robot's end effector
2. **`target_following.py`**
    a. finds waypoints between the robot's current gripper position and the goal (some small delta to move)
    b. uses IK (as implemented in lab 2) to reach to those waypoints, following the robot in "real time" (2Hz)
3. **`object_queries.yaml`**: used to specific objects you want to detect
4. **`ik_ros_utils.py`**: you need to copy your solution for Lab 2 into this file. Specific areas to copy over are listed as TODOs.
5. **`detection_utils.py`**: provided convenience functions for object_detector.py

Files 1-4 will require you to make changes. TODOs are delineated in the code files.

Part #1: Position-Based Visual Servoing

Detailed functionality/TODOs for `object_detector.py`: (THESE ARE IN THE README)

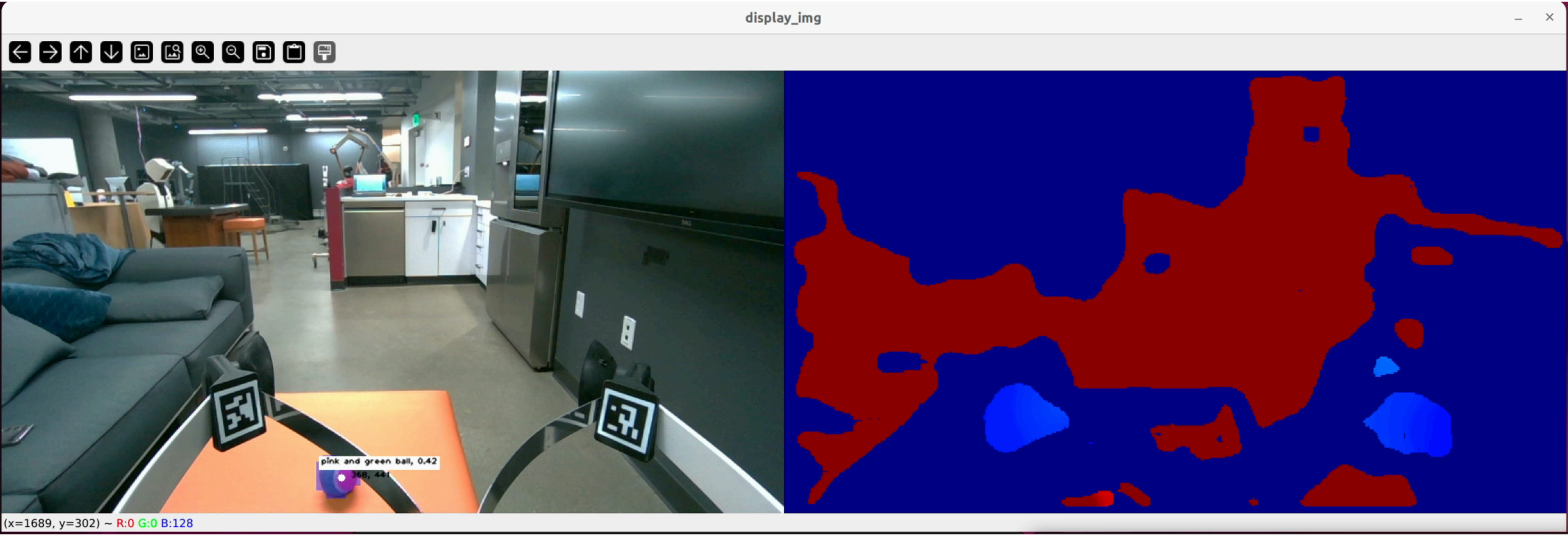
1. (Provided) Loads in `yolo-e-v26-small`.
2. **TODO**: populate `object_queries.yaml` with the object/s you want the robot to be able to detect
3. (Provided) Creates a synchronized subscriber to take in color and aligned-depth frames from the in-gripper camera, as well as the camera intrinsics.
4. **TODO**: implement a image callback function to unpack the incoming color and depth frames (cv2bridge). Once this is implemented correctly, you should see the live color and depth output plotted in a cv2 window by `detection_utils.visualize_detection_masks()` (without any masks yet though as you'll implement that next)
5. **TODO**: implement a timer callback function to feed color frames into YOLO-E at a fixed frequency.
6. **TODO**: Pass the results from YOLO-E to `detection_utils.parse_results()` to extract the detections (bounding, boxes, masks).
7. **TODO**: for each detected item in the color image, find the depth of the centroid of the object mask and project it to 3D using `detection_utils.pixel_to_3d()`.
8. **TODO**: convert the 3D position of the goal to a `PoseStamped msg`.
9. (Provided) Publish pose message as the goal point for the robot to track.

Part #1: Position-Based Visual Servoing

Detailed functionality/TODOs for `target_following.py`: (THESE ARE IN THE README)

1. **TODO**: Run the code once before making any changes. The robot should stow itself, then move to its ready configuration, which is designed such that: the lift is up at ~table height, wrist yaw is in line with the base, pitch is slightly downward, gripper is open. Adjust initial lift height and pitch as needed for your workspace (`READY_POSE_P1` found in `ik_ros_utils.py`).
2. (Provided) Creates a joint state callback that unpacks joint state messages for what works with/is expected by ikpy
3. **TODO**: subscribe to the `object_detector/goal_pose` topic in `self.main()`, published by your object detection node, to find the target pose the robot should move to.
4. **TODO**: Also create a TF buffer and listener in `self.main()`
5. **TODO**: Transform the goal pose to the same coordinate system as the robot's base
6. **TODO**: Transform the gripper pose to the same coordinate system as the robot's base
7. **TODO**: Move the robot a maximum of 2cm (or some other small displacement) towards the goal pose at each timestep. If the distance between the gripper and the target object is greater than this threshold, find an appropriate waypoint within the threshold that approaches the goal.
8. **TODO**: Use your IK implementation in `ik_ros_utils.py` to move the robot to the waypoint.

Solution



Screen capture of Color-Depth Frame Viewer (provided in `detection_utils.py`)

Part #2: 3D Perception for Grasping

Make copies of your implementations for Part 1 and rename them as follows:

1. `object_detector.py` → `object_detector_pcd.py`
2. `target_following.py` → `grasp_objects.py`

You should modify `object_detector_pcd.py` to use the head camera instead of the in-gripper camera and to use the centroid of the target object's point cloud (instead of the 2D mask) as the goal position.

You will also modify `grasp_objects.py` so that you move to and grasp a object (instead of continuously following it), before lifting the object and retracting the arm.

Part #2: 3D Perception for Grasping

Detailed TODOs for `object_detector_pcd.py`: (THESE ARE IN THE README)

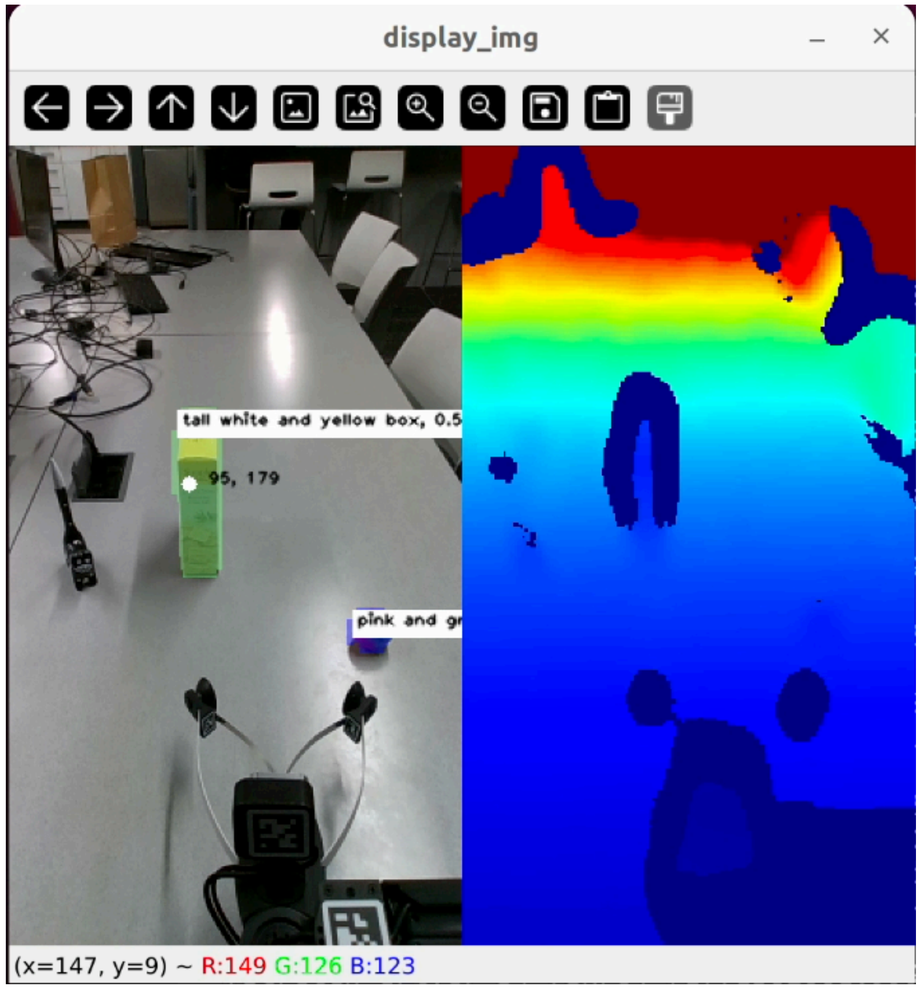
1. **TODO**: Modify color, depth, and cam info subscribers/callback to use the head camera instead of the in-gripper one.
2. **TODO**: Add 3-4 more candidates to grasp to `object_queries.yaml`. You should select objects with interesting variations in geometry or physical properties.
3. **TODO** (optional): if you are having a lot of trouble with missing depth values near where the centroid of the object is (common for reflective objects), Add another step of processing that fills in missing depth values across the mask of the detected object.
4. **TODO**: Edit `get_goal_pose()` to project all the points in the mask to 3D, then compute the centroid of that resulting point cloud to get the goal/grasp point.

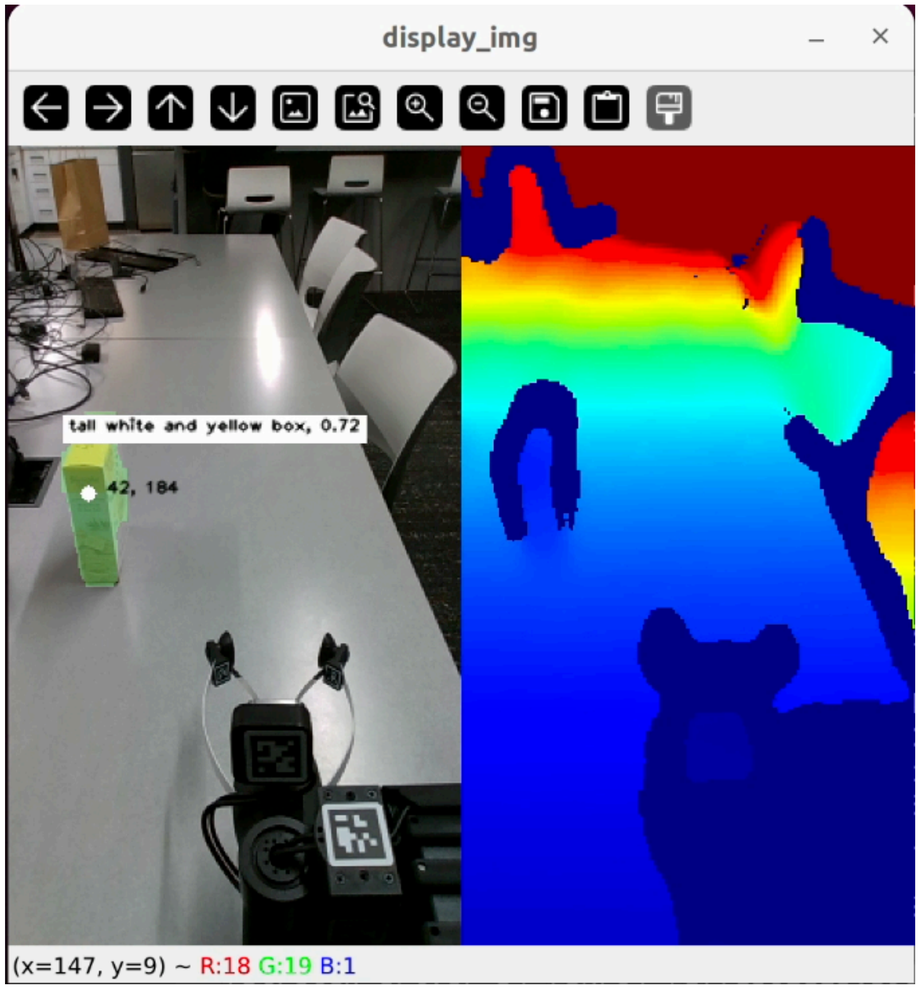
Part #2: 3D Perception for Grasping

Detailed TODOs for `grasp_objects.py`: (THESE ARE IN THE README)

1. **TODO**: Change the ready pose by uncommenting the line for part 2. Run the code once before making any further changes. The robot should stow itself, then move to its ready configuration, which is designed such that: the lift is up at ~table height, wrist yaw is normal to the base, pitch is slightly downward, gripper is open, and head is looking down the arm. Adjust initial lift height and pitch as needed for your workspace (`READY_POSE_P1` found in `ik_ros_utils.py`).
2. **TODO**: Edit the code to move to and grasp an object (instead of continuously following it). You should be able to pick up a given object (lift the object off of the table) and retract the arm to bring it closer to the base.
3. **TODO**: Rerun for all the objects.

Solution





Screen capture of Color-Depth Frame Viewer (provided in `detection_utils.py`)

Notes/Troubleshooting

- Avoid reflective objects as the target as these can give you poor depth values. See the `object_queries.yaml` file for more notes on selecting objects to detect.
- CV2 expects color frames in BGR, YOLO-E expects them in RGB format. Further, YOLO outputs in pixel coordinates, while array indexing is the opposite - this can trip you up!
- Cameras can go down, seemingly at random. If you find your code randomly stops working, make sure to check the terminals running the cameras.
- If you have to runstop the robot, you might want to restart the strech driver and camera nodes - handling the robot can mess up where the robot thinks it is in the world, which can introduce error in your transforms.
- While we provide a tool for visualizing the color and depth frames with the detected object masks, Rviz is your friend when it comes to visualizing things in 3D (goal poses, transforms, etc.)! For example, you might might use Rviz to verify where your published goal pose is with respect to where you expect it to be in the point cloud to make sure your detection is working well! You should be using it often, especially when implementing the nodes that move the robot.