COM SCI 188: Introduction to Robotics (winter 2026)

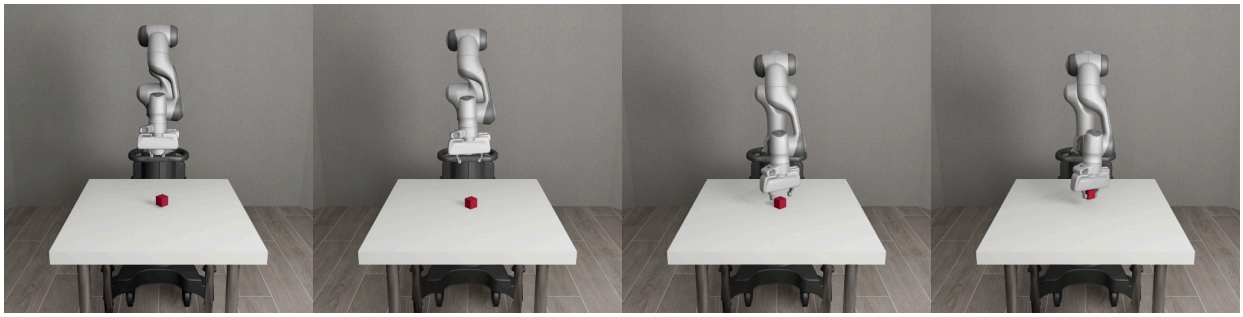
Coding Assignment 2:
Camera Calibration & Vision-Based Manipulation

Deadline: February 6, 2026, 11:59 PM

Overview

In Coding Assignment 1, you had direct access to ground-truth simulation state (e.g., object positions in world coordinates). In real-world robotics, however, you can only observe the environment through **cameras**. To enable vision-based manipulation, you must perform **camera calibration** to map pixel observations to the robot's action space.

In this assignment, you will implement a **hand-eye calibration** system that computes transformation matrices from camera coordinates to robot base coordinates. Using these transformations, the robot will perform the **Lift task** using only RGB-D images and proprioceptive data (joint angles, end-effector position).



The robot must detect and grasp the red cube using only camera observations.

By completing this assignment, you will:

- Understand the pinhole camera model and how to convert 2D pixels to 3D points
- Implement color-based object detection using computer vision techniques
- Apply the Kabsch algorithm to solve for rigid transformations between coordinate frames
- Generate diverse calibration waypoints with position and orientation variations
- Gain hands-on experience with hand-eye calibration for robot vision systems
- Debug and tune a complete perception-to-action pipeline

COM SCI 188: Introduction to Robotics (winter 2026)

Assignment Tasks

You will implement **5 functions** in camera_utils.py:

- **Task 1: Detect Red Blob (detect_red_blob)**
  - Implement a function that detects the largest red-colored object in an RGB image and returns its center pixel coordinates.
- **Task 2: Pixel to 3D Camera Coordinates (pixel_to_camera3d)**
  - Implement the pinhole camera deprojection formula to convert a 2D pixel and its depth to a 3D point in camera coordinates.
- **Task 3: Generate Calibration Waypoints (generate_calibration_waypoints)**
  - Generate a diverse set of waypoints by sampling both positions and orientations in the workspace.
- **Task 4: Solve for Rigid Transformation (solve_for_rigid_transformation)**
  - Implement the **Kabsch algorithm** to find the optimal rigid transformation (rotation + translation) between two sets of corresponding 3D points.
- **Task 5: Calibration Offset Tuning (get_calibration_offset)**
  - Determine an offset vector to compensate for systematic calibration errors.

Goal: Achieve >= 90% success rate over 10 trials of Lift task

COM SCI 188: Introduction to Robotics (winter 2026)

Provided Code Structure

Files You'll Modify:

- camera_utils.py - Implement the four functions above

Files Provided (Do Not Modify):

- policies.py - PID controllers for robot motion
- test.py - Evaluation script that runs 10 trials
- Helper functions in camera_utils.py:
    - get_intrinsics() - Extracts camera parameters
    - merge_point_clouds() - Combines multi-camera depth maps
    - run_hand_eye_calibration() - Main calibration loop

COM SCI 188: Introduction to Robotics (winter 2026)

Calibration Procedure

The run_hand_eye_calibration() function performs the following steps:

1. **Initialization**: Robot picks up the red cube
2. **Data Collection**:
    - Robot moves cube to 75 waypoints in its workspace
    - At each waypoint, cameras observe the cube
    - For each camera:
        - Detect red blob → Get pixel (u, v)
        - Convert to 3D point in camera frame using depth
        - Record pair: (point_camera, gripper_position_robot)
3. **Transform Solving**:
    - For each camera, solve T using Kabsch algorithm
    - Save transforms as T_frontview.npy, T_agentview.npy, etc.
4. **Verification**: Print reprojection errors

Expected Calibration Output:

```
=== CALIBRATION VERIFICATION ===
frontview:
  Mean error: 0.0049m
  Max error:  0.0085m
  Num points: 31
agentview:
  Mean error: 0.0055m
  Max error:  0.0112m
  Num points: 48
sideview:
  Mean error: 0.0037m
  Max error:  0.0107m
  Num points: 46
```

COM SCI 188: Introduction to Robotics (winter 2026)

Testing Your Implementation

Running Calibration:

python camera_utils.py

This generates T_*.npy files.

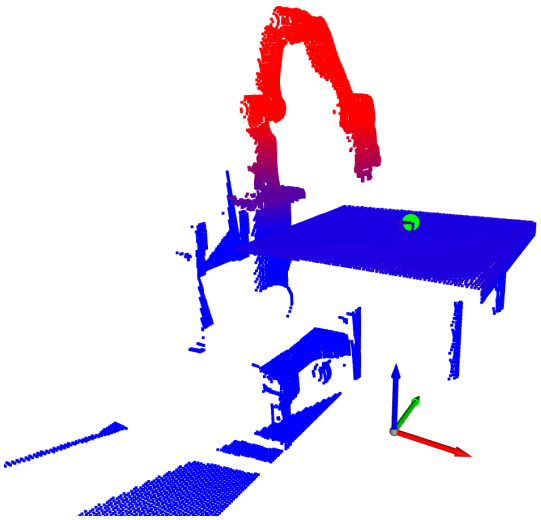
Running Test Trials:

python test.py

This loads calibration files and runs 10 pick-and-place trials.

Expected Behavior:

- Green circle appears on red cube in camera views
- Robot reaches toward detected cube position
- Success rate >= 90% after offset tuning



(merged pointcloud view, use ‘q’ to quit viewer and continue to testcases)

COM SCI 188: Introduction to Robotics (winter 2026)

Submission Requirements

Submit to Gradescope:

1. **camera_utils.py** - Your implementation file
2. **T_frontview.npy** - Calibration matrix for front camera
3. **T_agentview.npy** - Calibration matrix for agent camera
4. **T_sideview.npy** - Calibration matrix for side camera

Important Notes:

- Do **not** rename any functions (autograder depends on exact names)
- Do **not** modify function signatures (except generate_calibration_waypoints)
- Your code must run without errors on the autograder environment

Resources

- **Robosuite Documentation**: https://robosuite.ai/docs/overview.html
- **OpenCV Python Tutorials**: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html