Nimble AI Robotics Coding Challenge

Due: 7 days from date received

We've prepared a programming challenge to test your ability to jump into some of the things we're currently working on. All code should be done in Python. Feel free to use any libraries you want but make sure to cite which ones you've used. If the problem statement explicitly asks to implement your own version of an algorithm do not use a library's built-in function.

To meet the minimum requirement for this application select any subset of problems whose points sum to 10 or more with at least one problem from each of the two sections. If you exceed 10 points we will most certainly recognize that and it will strengthen your application. We've explicitly left some of the problems open-ended because real-world applications are rarely formulated like highly structured homework problems and there is typically more than one way to solve many problems.

To submit your challenge share a Github repository with all your code. All dependencies should be included with a requirements.txt file. Please email simon@nimble.ai if you have any questions. Good luck!

1 Kinematics

1.1 Forward Kinematics (3 Points)

To test your working knowledge of kinematics write the analytical forward kinematics solution for the (UR5 or Fanuc LR-mate 200iD) robot. Both robots are 6-DoF arms but have different kinematic configurations. The forward kinematics are a function of the robot's static physical parameters (link lengths, joint configurations (i.e. D-H parameters)) and current joint angles $\theta_1 - \theta_6$. Implement your solution as a Python class where each robot is initialized as an object requiring 1 input (4x4 tool transform). The physical parameters (link lengths, joint configurations, etc.) can be hard-coded since these will never change for any one robot. Your forward kinematics solver should be a function which can be called on the robot object. This function requires an input of the robot's current joint angles (1x6 numpy array or list). The output should be the 4x4 Transform (or 1x6 pose vector) of the end-effector relative to the robot's base frame.

1.2 Inverse Kinematics (6 or 8 points)

Implement the inverse kinematics for the (UR5 or Fanuc LR-mate 200iD) robot. You have the option to solve this analytically (8 points) or numerically (6 points). Analytical solutions are a bit trickier (a lot of trig and some tricks for robots with kinematic configurations that do not have spherical wrists which the the UR5 does not have) but are much faster since the final solution is implemented as a few lines of algebra. Numerical solutions run iterative optimization on constraint equations to find a solution which satisfies the robot's physical parameter constraints. They are mathematically simpler to implement but run much slower (up to 2 orders of magnitude slower depending on your optimization algorithm). For numerical solutions shoot for a computation time of less than 100 ms. The inverse kinematics should be a function in the robot class that requires as input the 4x4 end-effector transformation matrix (or 1x6 pose vector) and outputs 1x6 list or numpy array of joint angles to achieve the input pose. To choose a single solution from multiple valid solutions use a seed as input into the IK solver which chooses the solution whose joint angles are closest to the seed angles.

2 3D Vision

2.1 Generate a point cloud (2 points)

Given an RGB-D image with camera intrinsics (3x3) and extrinsics (6 x 1) write a function that generates a point cloud. Output should be a n x 6 array (x,y,z,r,g,b).

2.2 Point Cloud Surface Normal (3 points)

Implement in Python a surface normal calculation that's robust to noisy pointcloud data. The algorithm should take a nx6 point cloud and a xyz position where the surface normal should be approximated as input. The output should be a surface normal at that xyz position. Least-squares and RANSAC are two commonly used algorithms for plane fitting which can be used to estimate surface normals. Implement this from scratch using numpy only. Do not use PCL or Open3d libraries for this.

2.3 Point Cloud Registration (3 Points)

Combine two point clouds from different perspectives by transforming them into the world frame using their extrinsics. Fine tune alignment using ICP or SLAM techniques. Libraries can be used for ICP.

2.4 Point Cloud Smoothing (4 points)

Implement a smoothing function to filter out Gaussian noise in point clouds generated from low-cost depth sensors like Intel Realsense D415. Input to the function should be a nx3 point cloud and a smoothing term. The output should be an n x 3 point cloud with surfaces that are visually smoother where Gaussian noise is filtered. Plot images of the point cloud before and after. You may use any libraries for this.

2.5 Pointcloud Object Instance Segmentation (4 points)

There are many open-source deep learning models that perform object instance segmentation (Mask-RCNN, SD Mask-RCNN). Choose one and implement it such that it is stand alone and can take in an image and output a mask. This does not need to be done from scratch. Just implement an open source project and use it to segment objects from an RGB or Depth image. Take the mask from that algorithm and apply it to a point cloud to 'crop' out only the points in the cloud belonging to a single object. Plot the original cloud and the cropped cloud using Open3D.

2.6 Estimate Rigid Transforms (3 points)

The following problem arises in a large number of robotics and vision problems for calibration: Suppose $p1, ..., pn$ are the 3D coordinates of n points located on a rigid body in three-space collected by a depth sensor. $q1, ..., qn$ are the 3D coordinates of these same points after the body or point cloud has been translated and rotated by some unknown amount. Program an algorithm in python where SVD plays a central role for inferring the body's translation and rotation. See python starter code. The algorithm should be robust to noise given that the point clouds may be taken from real camera's. Implement from scratch using numpy only.

3 Resources

You may use any Python libraries to accomplish this challenge. Numpy, Math3d, and Open3D can be helpful tools.