ROB 422 HW 4

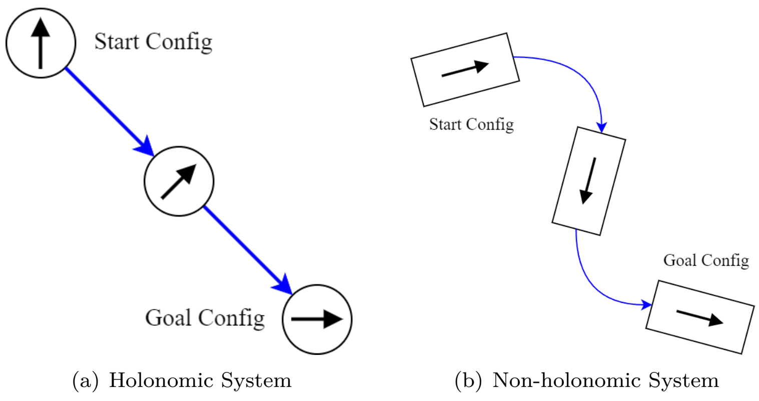
Yulun Zhuang
yulunz@umich.edu

November 8, 2023

Questions

1.

Defining a distance metric for a non-holonomic system like a car is more complex due to the constraints on its motion. For holonomic system, a widely used distance metric like Euclidean distance assumes that the vehicle can move between arbitrary nearby configurations using a straight line. Since the car cannot move directly to any point between configurations (Figure 1), we need to consider its dynamics and kinematics constraints. Moving between states (with no obstacles) in a non-holonomic system is defined as the two-point Boundary Value Problem (BVP), but this method can only be used locally i.e. small difference between states since it doesn't consider any obstacles and can take a long time to solve for large state difference.



(a) Holonomic System (b) Non-holonomic System

Figure 1: Moving between nearby configurations for Holonomic and Non-holonomic systems

Implementation

Iterative Inverse Kinematics (IK)

(c)

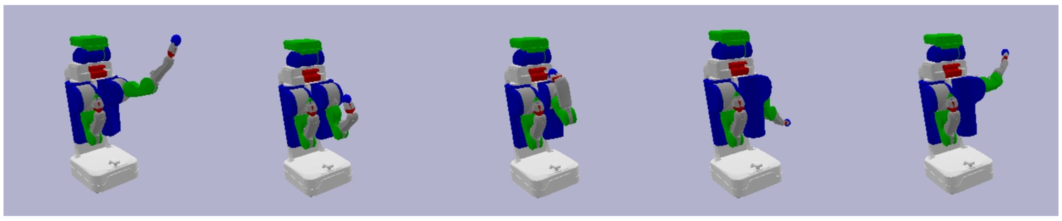


Figure 2: Visualization of the final configurations placed the end-effector at position targets 0 to 4. Configurations solved by iterative IK.

Configurations that placed the end-effector at target positions

1. Reached target 0 in 160 iterations with config:
   $[0.4392 - 0.24767 - 0.15007 - 0.6481 - 0.03273 - 0.374330.]$
2. Reached target 1 in 253 iterations with config:
   $[-0.1.04887 - 0. - 1.827360. - 1.337310.]$
3. Reached target 2 in 281 iterations with config:
   $[-0.224390.290650.13516 - 2.037070.05319 - 1.150140.]$
4. Reached target 3 in 292 iterations with config:
   $[1.935940.96999 - 0. - 0.34685 - 0. - 0.28347 - 0.]$
5. Reached target 4 in 435 iterations with config:
   $[1.72328 - 0.0391 - 0.61339 - 0.52134 - 0.24486 - 0.284760.]$

(d)

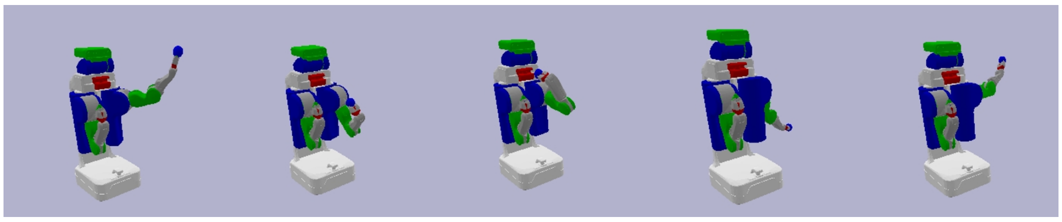


Figure 3: Visualization of the final configurations placed the end-effector at position targets 0 to 4 while repelling away from joint limits. Configurations solved by iterative Ik with a secondary task using left null-space projection.

1. Reached target 0 in 183 iterations with config:
   $[0.63309 - 0.267770.27886 - 0.6144 - 0.04292 - 0.428130.]$
2. Reached target 1 in 279 iterations with config:
   $[0.337720.955940.48873 - 1.78175 - 0.00363 - 1.38917 - 0.]$
3. Reached target 2 in 304 iterations with config:
   $[0.122120.152080.48221 - 2.00897 - 0.02983 - 1.15067 - 0.]$
4. Reached target 3 in 283 iterations with config:
   $[1.986070.947170.22095 - 0.28101 - 0.07283 - 0.37450.]$
5. Reached target 4 in 449 iterations with config:
   $[1.839490.02057 - 0.20015 - 0.49326 - 0.24007 - 0.35240.]$

Force Closure Evaluation

(a)

The volumes for the contact point 1 from Grasp 1 are:

1. True volume: 0.1227
2. 4-vector discretized: 0.0781
3. 8-vector discretized: 0.1105

The more the number of vectors, the smaller this approximation errors will be, since the approximated shape will get closer to a true friction cone.

(b)

The method for checking force closure:

- Input: Contact locations

- Output: Is the grasp in force closure? (YES or NO)

1. Approximate the friction cone at each contact points with a set of wrenches (4-edge or 8-edge polygonal pyramid cone)
2. Combine wrenches from all cones into a set of points S in wrench space
3. Compute the convex hull hull of S, and check for each boundary hyperplane, if the plane offset is less than zero i.e. if the origin is inside each half space.
4. If the origin is inside all half space i.e. the convex hull, return YES. If not, return NO.
5. If YES, pick the minimum absolute distance from the convex hull to the origin as the maximum hypersphere radius.

The force closure output and hypersphere radius if any:

1. Grasp 1: Not in force closure.
2. Grasp 2: In force closure. Maximum radius: 0.0141 on Mac and Linux; 0.0137 on Windows[1].

Knowing the hypersphere radius is useful because the maximum radius of the hypersphere represents the maximum external disturbance this grasp can resist, i.e. how stable this grasp is.

Principle Component Analysis (PCA)

(a)

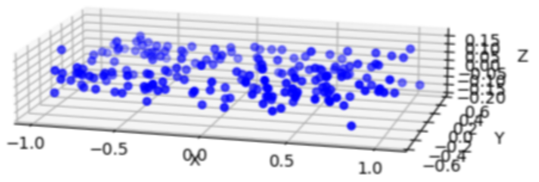


Figure 4: Rotated point cloud that aligns the surface in the point cloud with the XY plane.

$$V^T = \begin{bmatrix} -0.52935 & -0.00254 & 0.8484 \\ -0.05809 & -0.99754 & -0.03923 \\ 0.84641 & -0.07005 & 0.5279 \end{bmatrix}$$

(b)

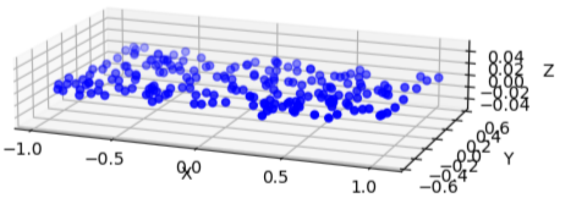


Figure 5: Rotated point cloud that aligns the surface in the point cloud with the XY plane while also eliminating the noise in the data (z = 0).

[1]The computation results from Qhull are in little different between Linux and Windows systems, while the same result is achieved on Mac and Linux. Results are tested with the same script but different laptops under the Honor Code.

$$V_s^T = \begin{bmatrix} -0.52935 & -0.00254 & 0.8484 \\ -0.05809 & -0.99754 & -0.03923 \end{bmatrix}$$

(c)

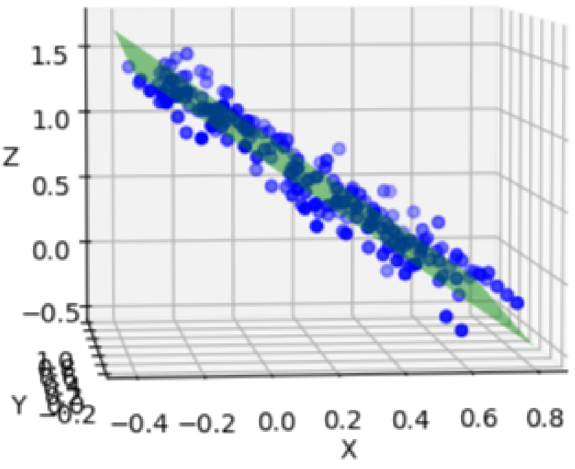


Figure 6: Use PCA to fit a plane (shown in green) to the point cloud.

The plane normal is $[0.84641,\ -0.07005,\ 0.5279]^T$ with the offset -0.4549.

RANdom SAmple Consensus (RANSAC)

(a)

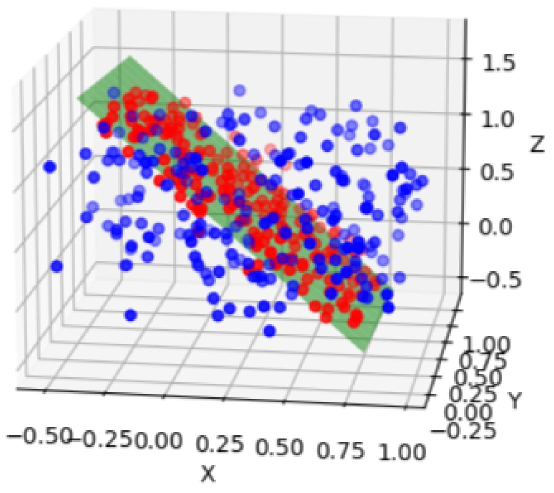
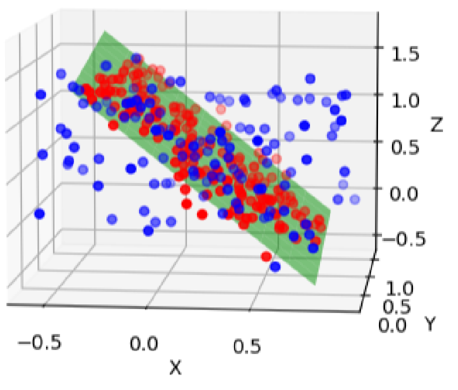


Figure 7: Use RANSAC to fit a plane (shown in green) to the point cloud. The inliers for the plane are in red, and all other points are in blue.

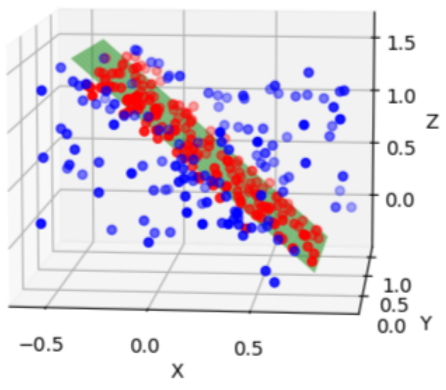
The plane normal is $[0.83917,\ 0.01051,\ 0.54376]^T$ with the offset -0.54022.

PCA v.s. RANSAC

(a)



(a) PCA



(b) RANSAC

Figure 8: Comparison of the plane fitting results between PCA and RANSAC. The fitted plane is shown in green, inliers are shown in red and other points are shown in blue.

(b)

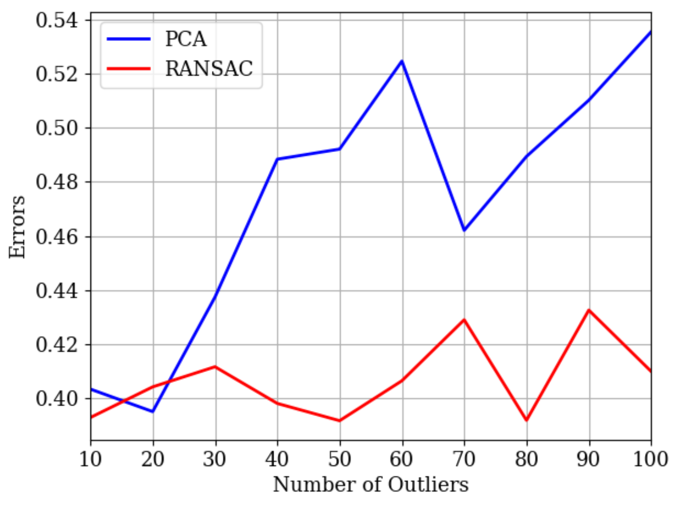


Figure 9: Comparison of inlier fitting errors between PCA and RANSAC as the number of outliers grows.

The Average computation times of PCA is 0.0017 seconds and that of RANSAC is 0.0686 seconds.

(c)

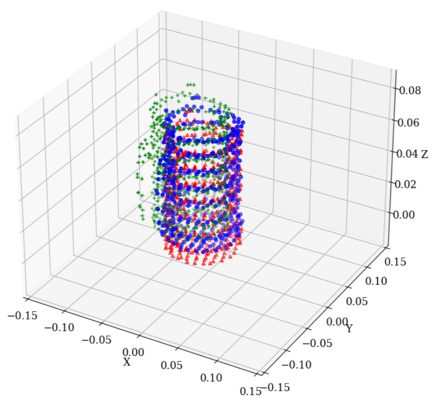
Overall, RANSAC usually performs better than PCA in terms of fitting errors, while PCA runs much one order of magnitude faster than RANSAC on average.

RANSAC utilize the random sampling method and use a clear error bound to select inliers and outliers based on sampled model parameters, this technique ensure it to be robust to noise and outliers in the data, but the solution may be sub-optimal and it can run slower due to the number of iterations.

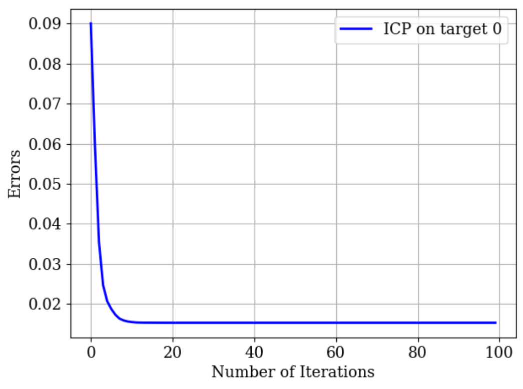
On the other hand, PCA works well when the data has less noise and can run much faster than RANSAC, but as the number of outliers growing, it will perform worse.

Iterative Closest Point (ICP)

(b)

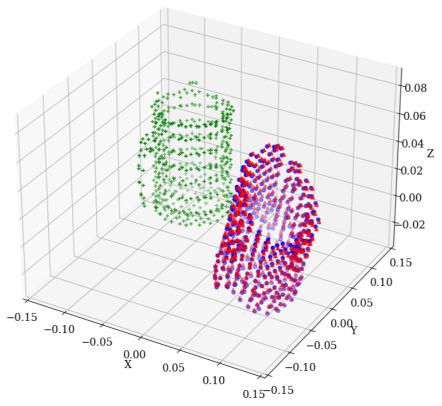


(a) Registration Result

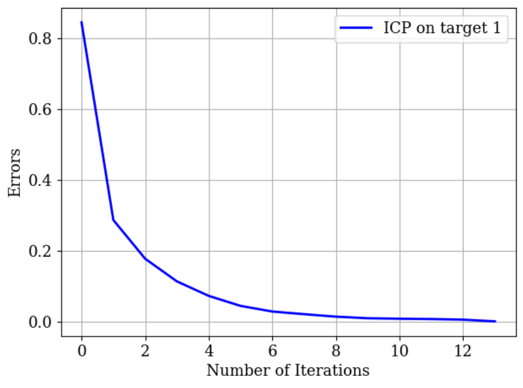


(b) Error vs. Iteration

Figure 10: ICP result for target 0. Source in green, aligned source in blue and target in red.

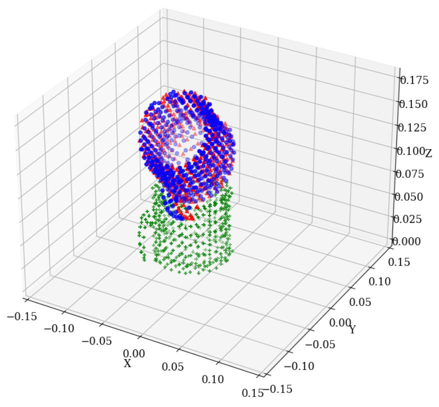


(a) Registration Result

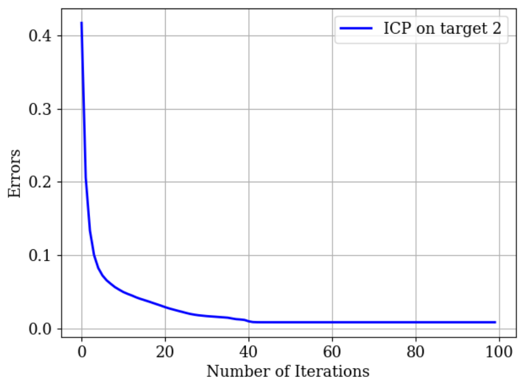


(b) Error vs. Iteration

Figure 11: ICP result for target 1. Source in green, aligned source in blue and target in red.

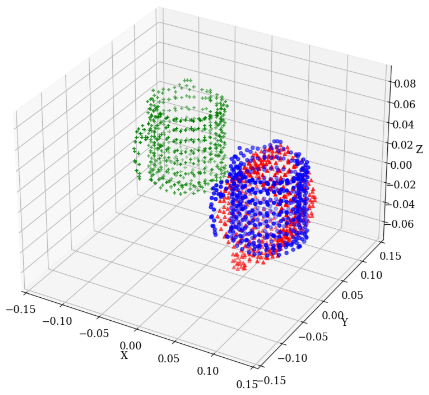


(a) Registration Result

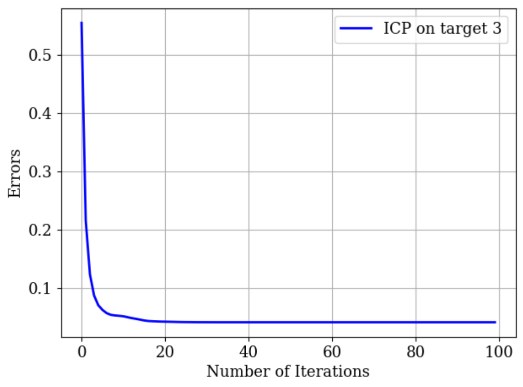


(b) Error vs. Iteration

Figure 12: ICP result for target 2. Source in green, aligned source in blue and target in red.



(a) Registration Result



(b) Error vs. Iteration

Figure 13: ICP result for target 3. Source in green, aligned source in blue and target in red.

(c)

As shown in above figures, target 3 is the most difficult one for ICP. This is because it has a very large orientation difference for the source and target point clouds, and the distance metric we used is just the Euclidean distance. Under this distance metric, it's easy to align the positions of two point clouds, but it's hard to align orientations since it will converge to a sub-optimal closest solution e.g. "embedded" source into target with a totally different orientation (Figure 13).