PyCPD: Pure NumPy Implementation of the Coherent Point Drift Algorithm

Anthony A. Gatti[1,2] **and Siavash Khallaghi**[3¶]

1 Stanford University, USA **2** NeuralSeg Ltd., Canada **3** Independent Researcher, Canada ¶ Corresponding author

DOI: 10.21105/joss.04681

Software

- Review
- Repository
- Archive

Editor: Hugo Ledoux

Reviewers:

- @zexinyang
- @rickie95

Submitted: 14 July 2022
Published: 15 December 2022

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Background

Point cloud registration is a common problem in many areas of computer science, particularly computer vision. Point clouds come from many types of data such as LIDAR commonly used for self-driving vehicles, and other sorts of 3D scanners (e.g., structured light) are commonly used to map the surface of physical objects. Point clouds are also used to represent the surface of an anatomical structure extracted from a medical image. Point cloud registration finds a transformation from one point cloud to another. Point cloud registration has use cases in many fields from self-driving vehicles to medical imaging and virtual reality. Typically, point cloud registration is classified into rigid (only rotations or translations), affine (rigid + shearing and scaling) and non-rigid also called deformable registration (non-linear deformation).

Point cloud registration typically requires 2 point clouds. The first point cloud is the "fixed" or "target" point cloud and the second is the "moving" or "source" point cloud. We try to find the transformation that will best align the moving (or source) point cloud with the fixed point cloud. One of the most well known rigid point cloud registration algorithms is the Iterative Closest Point (ICP) algorithm (Besl & McKay, 1992; Chen & Medioni, 1992). ICP is an iterative algorithm where the following steps are iterated:

(1) for every point on the moving point cloud find the closest point on the fixed point cloud
(2) use a least squares approach to find the optimal transformation matrix (rotation, translation, scaling, shear) to align the point correspondences found in (1)
(3) apply the transformation from (2) to the moving point cloud

These steps are repeated until the root mean squared point-to-point distances from (1) converge.

The coherent point drift (CPD) algorithm was created by Myronenko and Song (Myronenko & Song, 2010) to overcome many of the limitations of ICP and other previous registration methods (Besl & McKay, 1992; Chen & Medioni, 1992; Fitzgibbon, 2003; Rusinkiewicz & Levoy, 2001). Namely, these other methods did not necessarily generalize to dimensions greater than 3 and they were prone to errors such as noise, outliers, or missing points. The CPD algorithm is a probabilistic multidimensional algorithm that is robust and works for both rigid and non-rigid registration. In CPD the moving point cloud is modelled as a Gaussian Mixture Model (GMM) and the fixed point cloud is treated as observations from the GMM. The optimal transformation parameters maximize the Maximum Likelihood / Maximum A Posteriori (MAP) estimation that the observed point cloud is drawn from the GMM. A key point of the CPD algorithm is that it forces the points to move coherently by preserving topological structure. The CPD algorithm is also an iterative algorithm that iterates between an expectation (E) step and a maximization (M) step until convergence is achieved. The E-step estimates the posterior probability distributions of the GMM centroids (moving points) given the data (fixed points) then the M-step updates the transformation to maximize the posterior probability that the data belong to the GMM distributions. The E- and M-steps are iterated until convergence.

Statement of need

Due to the robustness and the broad array of uses for the CPD algorithm the original CPD paper has currently (March 2022) been referenced >2000 times. The CPD algorithm is available in Matlab (*The MathWorks Inc.*, n.d.) and an open-source C++ version has been implemented (Gadomski, n.d.). However, to the best of our knowledge, no open-source python version previously existed. In this paper we present a pure NumPy (Harris et al., 2020) version of the CPD algorithm to enable general use of CPD for the Python community. Furthermore, the full implementation in NumPy makes the algorithm accessible for others to learn from. To help in learning, a blog post that coincides with this library has previously been published (Khallaghi, 2017).

Summary

The PyCPD package implements the CPD algorithm in NumPy. The library itself includes a module to implement the Expectation Maximization (EM) algorithm. Sub-modules inherit the EM functionality and implement rigid, affine, and deformable registration using EM. CPD registration using affine, rigid, and deformable methods all allow for the transformation learned from CPD to be applied to any point cloud. Thus, it is possible to learn the transformation on a subset of the points and then apply it to the whole point cloud to reduce computation time. Finally, the low-rank approximation for deformable registration that was described by Myronenko and Song (Myronenko & Song, 2010) was implemented. A low rank approximation of the Gaussian kernel is used to reduce computation time and has the added benefit of regularizing the non-rigid deformation.

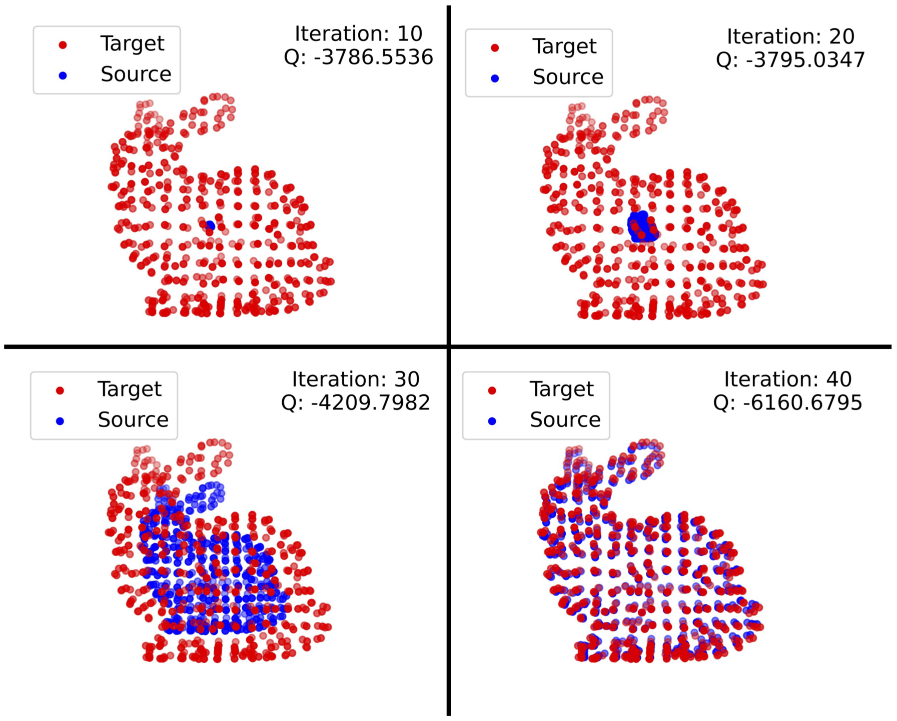

Figure 1: Visualization of the 3D rigid registration from the examples included in the library. Each panel represents a different iteration in the registration process. The Q parameter is the objective function that is optimized using the EM-algorithm during registration.

Examples of how to use the PyCPD algorithm are included in the package, Figure 1 displays the visualization corresponding with a 3D rigid registration example. Examples are available for 2D and 3D versions of all registration methods (rigid, affine, deformable). Examples of how to use the low-rank approximation as well as how to use a sub-set of the points for registration are also included in the examples.

Acknowledgements

We acknowledge contributions from:

- Alvin Wan for testing on Python 3.x.
- Talley Lambert for pointing out that the moving point cloud should be immutable during registration.
- Matthew DiFranco for fixing the check for target point clouds.
- normanius for pointing out that the contribution of uniform distribution was not being added in the E-step.
- Kai Zhang for finding a bug when transforming a point cloud using rigid registration parameters.
- sandyhsia for finding a bug when updating the variance during deformable registration.
- Arthur Porto for contributions to the community, both in the issues and adding a priors option to cpd

References

Besl, P. J., & McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *14*(2), 239–256. https://doi.org/10.1109/34.121791

Chen, Y., & Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, *10*(3), 145–155. https://doi.org/10.1016/0262-8856(92)90066-C

Fitzgibbon, A. W. (2003). Robust registration of 2D and 3D point sets. *Image and Vision Computing*, *21*(13), 1145–1153. https://doi.org/10.1016/j.imavis.2003.09.004

Gadomski, P. (n.d.). *Gadomski/CPD: C++ implementation of the coherent point drift point set registration algorithm.* https://github.com/gadomski/cpd

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Khallaghi, S. (2017). *PyCPD: Tutorial on the coherent point drift algorithm.* http://siavashk.github.io/2017/05/14/coherent-point-drift/

Myronenko, A., & Song, X. (2010). Point set registration: Coherent point drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(12), 2262–2275. https://doi.org/10.1109/TPAMI.2010.46

Rusinkiewicz, S., & Levoy, M. (2001). Efficient variants of the ICP algorithm. *Proceedings Third International Conference on 3-d Digital Imaging and Modeling*, 145–152. https://doi.org/10.1109/IM.2001.924423

The MathWorks inc. (version 9.5 (R2018b)). (n.d.). [Computer software]. https://www.mathworks.com/help/vision/ref/pcregistercpd.html