Exercise 11 solutions

July 20, 2023

Exercise 1: ICP

Explain why the ICP algorithm is needed in robotics. In the context of SLAM, describe how you would approach each of the four stages of ICP (shown in the lecture on slide 12). You can choose from the variants presented in the slides or design new ones. Justify your choice. You can also choose a specific variant of SLAM and a specific sensor (e.g. 2D or 3D laser scanner) for your description.

The main application of ICP in robotics is the matching of (laser-) scans recorded from different robot poses. As the relative poses are not known (at least up to some uncertainty), the matching of scans is not trivial. ICP has proven to be efficient and quite reliable for this application.

The following choices for the four stages of ICP are made in the context of graph-based SLAM with a 3D laser scanner.

1. Point subsets (from one or both point sets) Feature-based sampling seems promising, in particular for the performance of ICP with 3D laser scanners, which produce large numbers of points. It might also be useful for reducing the amount of data stored for graph-based SLAM, where a history of scans is required for the full run-time.

You can play with the supplementary code *bunny_icp.py* to have a better feeling of how uniform sampling can help with convergence speed.

2. Weighting the correspondences Weighting of point pairs may help to judge the reliability of the scans, e.g. by giving lower weights to larger point-point distances. For example, a hedge will result in noisier scans than a wall. The points from one scan of the hedge will not well fit a consecutive scan and can be assigned lower weights for more reliable matching.

In the context of SLAM it can be helpful to define weights based on features. Of the points belong to a fixed object, like poles or walls (“landmarks”), they can be assigned larger weights than moving objects, like cars or humans.

The overall RMSE of the matched scans, based on the individual point-point distances and weights, can then be used as input to the pose constraints in the SLAM graph optimization.

You can play with the supplementary code *bunny_icp.py* to compare point-to-plane algorithm and point-to-point algorithm. You can also play with the *color_icp.py* to compare color ICP and point-to-plane in some difficult cases for point-to-plane ICP.

3. Data association In mostly static, but manifold geometric environments (like urban areas), a matching based on closest compatible points seems promising. The normals and curvatures help to match points of specific objects (e.g. poles vs. walls). If available, the color can also help to find matches more reliably.

However, in practice the compatibility of points is often not easy to define. Often the simple closest point approach is used instead.

If you are interested, you could check MULLS SLAM, RANSAC, global point cloud features.

4. Rejecting certain (outlier) point pairs Rejecting outliers is particular important for laser scan matching with a mobile robot. Scans from a specific view point will usually contain a large number of points that don't have a correspondence in a scan from a different view point. Since these unmatched points will usually come in whole areas / volumes, it might be worth to define an algorithm targeting this kind of outliers.

A simple distance threshold might prevent matching of scans from loop-closures, if the odometry noise causes the initial guess to be quite off.

In the context of dynamic environments, it is important to reject outliers that are caused by objects that move between frames or by objects that appeared / disappeared between (loop closure) runs.

Some state-of-the-art registration methods that can deal with outliers properly are TEASER++ (TEASER: Fast and Certifiable Point Cloud Registration, TRO 2020) and MAC (3D Registration with Maximal Cliques, CVPR 2023). If you are interested, feel free to have a look.

Exercise 2: Data association

In the icp_framework tarball, you will find a complete implementation of the basic ICP algorithm. By commenting in one of the lines `icp(...)` in `main()` of `icp_framework.py` you can test it on four different datasets.

The algorithm already works well for datasets with known correspondences (i.e. P1 and P2), but it does not work for datasets with unknown correspondences (i.e. P3 and P4). If the correspondences between the points are unknown, they have to be estimated at first.

Implement closest-point matching (see `TODO` in `icp_framework.py`) and test it using the two data sets P3 and P4. The closest point matching should be implemented as reordering of the elements in the vector P to match those in X. Note that not all points in P can be matched to their closest partner in X, since this would require multiple matches of single points, which is not possible with simple reordering.

The algorithm is implemented by first calculating the distances between all pairs of elements in P and X. The reordering then starts with the closest pair and stores the corresponding element of P to P_matched, then moves to the next closest pair. This implementation might not be ideal, as the pairs that are assigned in the end might get very suboptimal matches. Please find the code listing below.

```python
def closest_point_matching(X, P):
  """
  Performs closest point matching of two point sets.

  Arguments:
  X -- reference point set
  P -- point set to be matched with the reference

  Output:
  P_matched -- reordered P, so that the elements in P match the elements in X
  """

  P_matched = P

  # calculate distances
  D = np.Inf * np.ones((P.shape[1], X.shape[1]))
  for i in range(P.shape[1]):
      for j in range(X.shape[1]):
          D[i,j] = np.linalg.norm(P[:,i] - X[:,j])

  # reorder P
  P_matched = np.zeros(P.shape)
  for k in range(P.shape[1]):
    ij_min = np.argmin(D)
    i_min, j_min = np.unravel_index(ij_min,(P.shape[1],X.shape[1]))
    P_matched[:,j_min] = P[:,i_min]
    D[i_min,:] = np.Inf
    D[:,j_min] = np.Inf

  return P_matched
```

It is worth noting that the original implementation of ICP doesn't consider the reflection case $det(U \cdot V^T) = -1$. You need to modify the rotation matrix computation to the following to handle the case. If the mentioned case happens, we need to change the last column of V to the negative $V[:, -1] = -V[:, -1]$ before computing the rotation matrix. For details explanation please check the paper "Least-Squares Fitting of Two 3-D Point Sets" (PAMI, 1987). You can also find the detailed proof of the registration solution if you didn't manage to follow the proof during the tutorial session.

```python
    # calculate rotation and translation
    R = np.dot(U, V.T)
    val = np.linalg.det(R)
    if val < 0:
        V[:, -1] = -V[:, -1]
        R = np.dot(U, V.T)
    t = mx - np.dot(R, mp)
```

Exercise 3: ICP and SVD

Recall the formulas on the slides 5-7 of the ICP lecture and prove the following:

$$X' = P' \quad \Rightarrow \quad R = I \tag{1}$$

Hint: Find out, how singular value decomposition (SVD) and eigenvalue decomposition (EVD) are related to each other.
Since $X' = P'$:

$$W = \sum_{i=1}^{N_p} x'_i {p'_i}^\intercal = \sum_{i=1}^{N_p} p'_i {p'_i}^\intercal \tag{2}$$

$$\tag{3}$$

W is symmetric:

$$W^\intercal = \sum_{i=1}^{N_p} \left(p'_i {p'_i}^\intercal\right)^\intercal = \sum_{i=1}^{N_p} p'_i {p'_i}^\intercal = W \tag{4}$$

$$\tag{5}$$

Hence, W is normal:

$$W^\intercal W = W W^\intercal \tag{6}$$

W is also positive-semidefinite: for any column vector z

$$z^\intercal p'_i {p'_i}^\intercal z = \left({p'_i}^\intercal z\right)^\intercal \left({p'_i}^\intercal z\right) = \left|\left|{p'_i}^\intercal z\right|\right|^2 \geq 0 \tag{7}$$

$$\Rightarrow \quad z^\intercal W z = \sum_{i=1}^{N_p} z^\intercal p'_i {p'_i}^\intercal z \geq 0 \tag{8}$$

In general, the singular value decomposition (SVD) of a real matrix W $(m \times n)$ is:

$$W = U \Sigma V^\intercal, \tag{9}$$

where Σ is a rectangular diagonal matrix ($m \times n$). This simplifies for positive-semidefinite normal matrices W to (details on Wikipedia):

$$W = UDU^{-1}, \tag{10}$$

where D is a diagonal matrix ($n \times n$) and one can set $\Sigma = D$. This is the eigenvalue decomposition (EVD) of W. Hence, we have $V^{\mathsf{T}} = U^{-1}$ and

$$R = UV^{\mathsf{T}} = UU^{-1} = I \tag{11}$$