Simultaneous Localization and Mapping using Iterative Closest Point

Autonomous Robotics Labs

Tomáš Petříček

March 22, 2020

1 Iterative Closest Point

Optimal least-squares alignment $\mathtt{R}, \mathtt{t}$ of corresponding points $\mathtt{y}_i = \mathtt{R}\mathtt{x}_i + \mathtt{t}$, $i = 1, \ldots, N$, $\mathtt{x}_i \in \mathcal{X} \subseteq \mathbb{R}^D$, $\mathtt{y}_i \in \mathcal{Y} \subseteq \mathbb{R}^D$, can be found using the absolute orientation algorithm [1]. The *Iterative Closest Point* (ICP) [2] algorithm provides an approximate solution when the correspondences $(\mathtt{x}_i, \mathtt{y}_i)$ are not known in advance. Instead, it solves the correspondence and alignment problem iteratively, with ad hoc correspondences $(\mathtt{x}_i, \operatorname{argmin}_{\mathtt{y} \in \mathcal{Y}} \|\mathtt{R}\mathtt{x}_i + \mathtt{t} - \mathtt{y}\|)$ using alignment $\mathtt{R}, \mathtt{t}$ from the previous iteration. In the first iteration, an initial estimate or identity $\mathtt{R} = \mathtt{I}$, $\mathtt{t} = \mathtt{0}$ can be used.

Recall the ICP and absolute orientation algorithms from the lecture. Note that we will provide the ICP algorithm with an initial estimate different from the identity used in the lecture. This initial alignment is to be applied to the input points $\mathtt{x}_i$ prior the nearest neighbor search. Thus a small change to the algorithm from the lecture is needed, see Alg. 1.

Algorithm 1 Iterative Closest Point

Input: input points $\mathcal{X} = \{\mathtt{x}_i\}_i$ to align,
reference points $\mathcal{Y} = \{\mathtt{y}_j\}_j$ to align to,
initial alignment $\mathtt{R}$, $\mathtt{t}$.

1: Establish correspondences $\mathcal{C} \leftarrow \{(\mathtt{x}, \operatorname{argmin}_{\mathtt{y} \in \mathcal{Y}} \|\mathtt{R}\mathtt{x}_i + \mathtt{t} - \mathtt{y}\|) \mid \mathtt{x} \in \mathcal{X}\}$.
2: Compute distance threshold for inliers $d_{\text{inl}} \leftarrow r Q_d(p)$ where $Q_d(p)$ is the p-quantile of distances $d_i = \|\mathtt{x}_i - \mathtt{y}_i\|$, $(\mathtt{x}_i, \mathtt{y}_i) \in \mathcal{C}$, and $r > 0$ is a multiplier.
3: Construct set of inliers $\mathcal{C}' \leftarrow \{(\mathtt{x}_i, \mathtt{y}_i) \mid (\mathtt{x}_i, \mathtt{y}_i) \in \mathcal{C} \wedge d_i \leq d_{\text{inl}}\}$.
4: Solve absolute orientation $\mathtt{R}, \mathtt{t} \leftarrow [1](\mathcal{C}')$, for the inliers.
5: Go to line 1 if not done.

Output: alignment $\mathtt{R}$, $\mathtt{t}$,
average distance for the inliers $(\sum_{d_i \leq d_{\text{inl}}} d_i)/(\sum_{d_i \leq d_{\text{inl}}} 1)$,
inlier indicators $a_i = [d_i \leq d_{\text{inl}}]$.

Rigid transform $\mathtt{R}, \mathtt{t}$ can be represented by single matrix $\mathtt{T} = \begin{bmatrix} \mathtt{R} & \mathtt{t} \\ \mathtt{0}^\mathsf{T} & 1 \end{bmatrix}$.

2 Simultaneous Localization and Mapping

When a robot explores an unknown environment, it has to solve localization and mapping simultaneously, hence the name *Simultaneous Localization and Mapping* (SLAM). We will focus on the *online SLAM problem* in which we are interested only in the current robot pose and map given all previous measurements. In other words, past robot poses are not updated when new measurements arrive.

One can find many approaches both in literature and deployed in field. Main differences come from the sensors used (camera, lidar, sonar, etc.) and the assumptions taken (2D vs 3D, static vs dynamic world). A few of them can be seen in action below:

- 2D lidar + IMU, indoor (Hector)
- Stereo, outdoor (RTAB-Map)
- 3D lidar, indoor (BLAM)
- 3D lidar, outdoor with dynamic obstacles (ICP Mapper)

3 Assignment: ICP-based 2D SLAM

We will implement a ROS node similar to the *ICP Mapper* above but we will restrict ourselves to two dimensions, localizing and mapping only in horizontal x-y plane. We assume that the robot operates on a flat ground since the TurtleBots used in the labs cannot traverse larger obstacles anyway.

Your task is to get familiar with the provided *aro_slam* ROS package and to implement the *icp* function in the *aro_slam.icp* Python module[1] it contains. Then compress the whole *aro_slam* package excluding the data folder and any bag files into a *zip* archive and upload it into Brute.

Many parts of the package, including the *icp_slam_2d* node[2] with subscribers and publishers, map updating etc., are already provided, nevertheless, feel free to experiment with these as well. The node builds a global point map from lidar scans which it first aligns with the map via ICP. Odometry can provide an initial estimate for the alignment, if it is available. See Fig. 1 and the corresponding video for an example run with the robot.

The transforms related to navigation, organized in a directed tree structure, are *map* → *odom* → *base_link*. The transform *odom* → *base_link* is provided by odometry of the TurtleBot platform, by fusing the information from encoders in wheels and an *inertial measurement unit* (IMU). The transform *map* → *odom* is therefore only a correction of odometry-only localization, which would slowly drift by accumulating errors over time.

While building the map, the node avoids excessive map updates and reduces map drift by keeping only the first measurement in every cell. It also keeps only the points from stable occupied cells, the rest is discarded each time the map is updated.

[1] `aro_slam/src/aro_slam/icp.py`
[2] `aro_slam/scripts/icp_slam_2d`

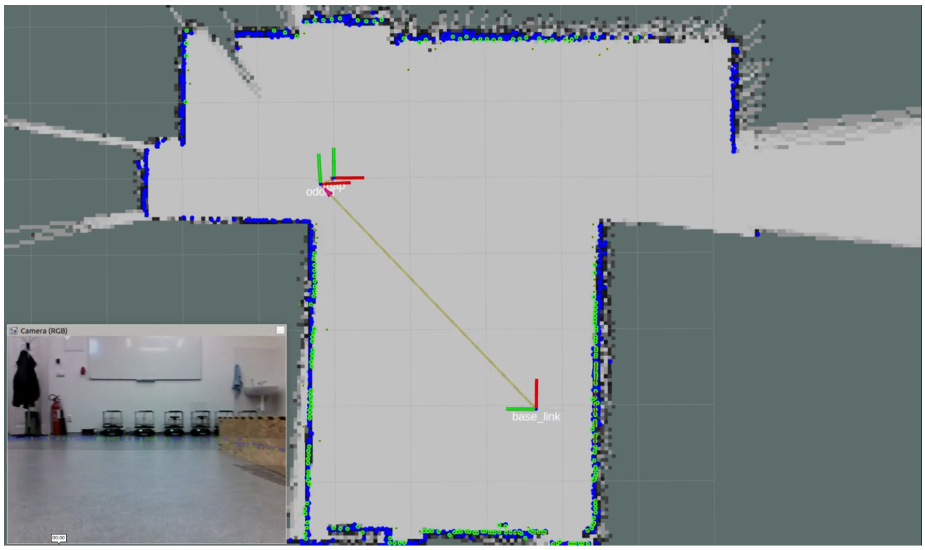

Figure 1: An example of the ICP-based 2D SLAM results. The red points correspond to the input point cloud, the blue points to the point map, the smaller green points to the registered points, and the larger green points correspond to the inliers from the last ICP iteration. Dark gray to black in occupancy map correspond to likely occupied cells, light gray correspond to empty cells. See the video.

3.1 Occupancy Map

To identify the stable occupied cells, an extra occupancy grid is maintained and updated with aligned scans. The occupancy model of each grid cell o_i is updated with every scan: the cells containing the measured points have their probability of being occupied increased and other cells incident with the rays from the sensor towards these points have their probability of being occupied decreased. A point $\mathtt{x}_i$ is kept in the point map only if the corresponding occupancy o_i is higher than threshold o_{occupied}.

Occupancy values are clipped to interval $[o_{\text{low}}, o_{\text{high}}]$ to compensate for the simplifying assumption of independence of individual measurements z_j, $P(\text{cell } i \text{ occupied} \mid z_1, \ldots, z_N) = \prod_{j=1}^{N} P(\text{cell } i \text{ occupied} \mid z_j)$, which allows us to use additive occupancy updates of form $\Delta o_i = \log(P(\text{cell } i \text{ occupied} \mid z_j)/(1-P(\text{cell } i \text{ occupied} \mid z_j)))$. We use unit updates $\Delta o_i \in \{-1, 1\}$ and let parameters o_{low}, o_{high}, o_{occupied} be tuned to get the required trade-off between noise removal and response time to dynamic changes in the environment.

3.2 Outlier Correspondence Rejection

In some cases, such as going through a narrow opening such as a door, large portions of the new measurements can correspond to the previously unseen areas, i.e., for many points from the input point cloud a true correspondence in the map cannot be found. In such cases, the relative overlap can easily decrease to as low as 1/2. To handle smaller overlaps between the point clouds, we will employ the following outlier rejection strategy in the ICP algorithm.

Let $\mathcal{C} = \{(\mathtt{x}_i, \mathtt{y}_i)\}_i$ be the set of correspondences from the nearest neighbor search, $i \in \{1, \ldots, N\}$, and d_i the distance between corresponding points, $d_i = \|\mathtt{x}_i - \mathtt{y}_i\|$. Construct the set of inlier correspondences $\mathcal{C}'$ to be used in optimization of $\mathtt{R}, \mathtt{t}$, $\mathcal{C}' = \{(\mathtt{x}_i, \mathtt{y}_i) \mid (\mathtt{x}_i, \mathtt{y}_i) \in \mathcal{C} \wedge d_i \leq rQ_d(p)\}$, using p-quantile of the distances d_i, $Q_d(p)$, and a multiplier r. To handle relative overlaps at least 1/2, the *icp_slam_2d* node uses the following parameters and default values: $p = \texttt{inlier\_ratio} = 1/2$ and $r = \texttt{inlier\_dist\_mult} = 2$. See Alg. 1 for the full algorithm.

4 Workspace Configuration

In general, the package should run both in ROS Kinetic and Melodic if the dependencies are met.

4.1 Robolab Singularity Image

This Singularity image is used at the labs—it already provides all the dependencies. To build your customized image, use the scripts from the Robolab deploy repository.

Enter the shell within the Singularity image:

```
singularity shell /path/to/robolab_melodic.simg
```

In the labs, an expanded image is available:

```
singularity shell /opt/singularity/robolab/melodic
```

4.2 Catkin Workspace

Then, the workspace can be configured as follow:

```
ws=~/workspace/aro
mkdir -p "${ws}/src"
cd "${ws}/src"
curl -sSL https://cw.fel.cvut.cz/b192/_media/courses/aro/tutorials/aro_slam.zip -O
unzip *.zip
cd "${ws}"
catkin init
catkin config --extend /opt/ros/aro
catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release
catkin build -c
```

4.3 Extras for a Generic Ubuntu

On a generic Ubuntu with ROS Melodic (not with the Singularity image above), dependencies must be install prior `caktin build -c` from the accompanied rosinstall file and via rosdep.

```
cd "${ws}/src"
wstool init
wstool merge aro_slam/dependencies.rosinstall
wstool up -j 4
cd "${ws}"
rosdep install --rosdistro melodic --from-paths src --ignore-src -r -y
```

5 Package Usage

Don't forget to source your workspace:

```
source "${ws}/devel/setup.bash"
```

Test the node on a provided bag file, with or without odometry:

```
roslaunch aro_slam bag.launch
roslaunch aro_slam bag.launch odom:=0
```

Eventually, run it on the robot:

```
roslaunch aro_slam live.launch
```

I encourage you to test the `icp` function separately before the ROS node test.

References

[1] K. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-9, no. 5, pp. 698–700, 1987.

[2] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/026288569290066C