APPENDIX: DETAILS ABOUT THE DISTANCE TRANSFORM

To speed up the closest-point distance computation, 3D Euclidean Distance Transform (DT) can be used in the proposed method. A DT is a uniform discretization of a bounded space, where the distances in the real-valued space are approximated by the grid distances in the discretized space. Each grid cell stores the closest distance to the "model grid", *i.e.* the discretized model points. In this way, the closest-point distances of the data points to the model point-set can be approximated by the grid distances and retrieved in $O(1)$ time. Figure I illustrates the use of a 3D DT.

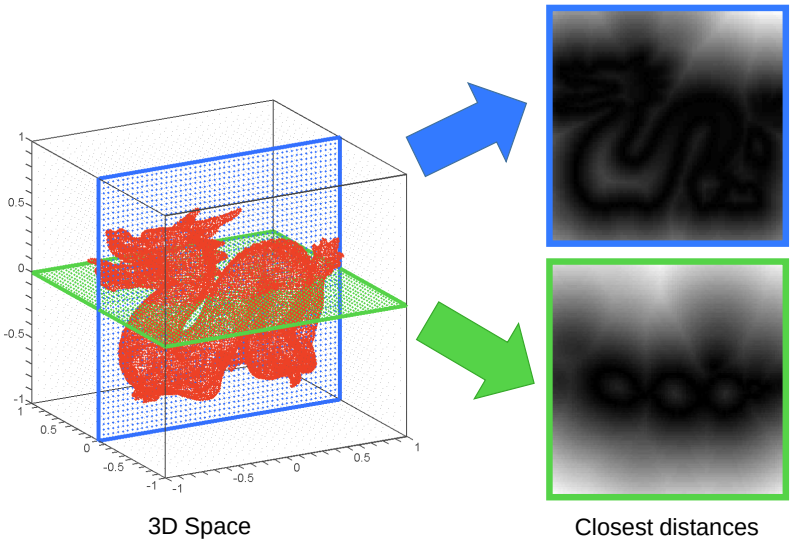

Figure I: Illustration of a 3D Euclidean Distance Transform. **Left:** the 3D space with a dragon model from the Standford 3D dataset. **Right:** distance values on the XY and YZ planes of the 3D DT grid (white for large distances and black for small distances).

In the following section, we present the implementation details of our instantiation of the DT, and show more experimental results and comparisons with a kd-tree.

Implementation Details

DT Construction. The Euclidean distance transform can be built sequentially or in parallel [1]. Following [2] we use a simple sequential method to construct a DT. Our DT construction procedure can be summarized as the following two steps:

- *Initialization.* Set the closest distance values of the grid cells containing any model point to zero; set the values of other grid cells to infinity.
- *Propagation.* Propagate the grid cell values to their neighbouring grid cells. Forward and backward propagations are run sequentially until stable.

Details of the construction algorithm are given as follows. The goal of DT construction is to build a 3D array D where $D(x, y, z)$ stores the distance from grid cell (x, y, z) to its closest model grid cell, say (x', y', z'). We also build auxiliary data structures D_1, D_2 and D_3 where $D_1(x, y, z)$, $D_2(x, y, z)$ and $D_3(x, y, z)$ store the distances from (x, y, z) to (x', y', z') in the x-, y-, and z-dimensions respectively. During initialization, we set $D(x, y, z) = D_1(x, y, z) = D_2(x, y, z) = D_3(x, y, z) = 0$ if (x, y, z) is a model grid cell, *i.e.*

if it contains any real-valued model point; otherwise they are set to infinity. The closest distances will then be propagated from the zero-distance set to the whole DT space in a sequential manner. Specifically, forward propagations are executed along arrays D_1, D_2, D_3 and D followed by backward propagations in the reverse order; during propagation, grid cell values of (x, y, z) are updated using the grid cell values of its neighbouring grid cells via the following equations:

$$
\begin{aligned}
(i^*, j^*, k^*) &= \underset{i,j,k\in\{-1,0,1\}}{\operatorname{argmin}} \big(D_1(x+i, y, z)+|i|\big)^2+\big(D_2(x, y+j, z)+|j|\big)^2+\big(D_3(x, y, z+k)+|k|\big)^2, \\
D_1(x, y, z) &\leftarrow D_1(x+i^*, y, z) + |i^*|, \\
D_2(x, y, z) &\leftarrow D_2(x, y+j^*, z) + |j^*|, \\
D_3(x, y, z) &\leftarrow D_3(x, y, z+k^*) + |k^*|, \\
D(x, y, z) &\leftarrow \sqrt{D_1(x, y, z)^2 + D_3(x, y, z)^2 + D_3(x, y, z)^2}.
\end{aligned}
$$

In our experiments, constructing a DT described above with a $300 \times 300 \times 300$ grid takes about 20 seconds. Naturally, a higher resolution can be used and a DT can be built extremely quickly, especially with modern processors [3, 4].

Expanded DT Space. In our experiments, the point-sets are normalized to be within $[-1, 1]^3$. Specifically, we first centralize the model and data point clouds independently to the origin, then scale them simultaneously to be within $[-1, 1]^3$. However, the DT is built upon a space larger than $[-1, 1]^3$ similar to [2]. An "expansion factor" s is used such that the DT is built on $[-s, s]^3$. Since all the points are within $[-1, 1]^3$, the out-of-bound issue can be easily avoided by setting $s \geqslant 1+2\sqrt{3}$ (see Figure II left). We have tried such a large DT space; however, in practice we found it very wasteful. Since the model and data point-sets are well centralized, we found an expansion factor $s = 2$ suffices for the registration task when they are of similar spatial extensions. When the model point-set has a much larger spatial extension (*e.g.* the experiments in Sec. 6.3), an even smaller expansion factor such as $s = 1.5$ can be used. If a data point $\mathbf{x}$ falls outside of the DT space, its closest-point distance is set to be the closest-point distance of $\mathbf{x}$'s nearest DT boundary point $\mathbf{y}$, plus $\|\mathbf{x} - \mathbf{y}\|$.

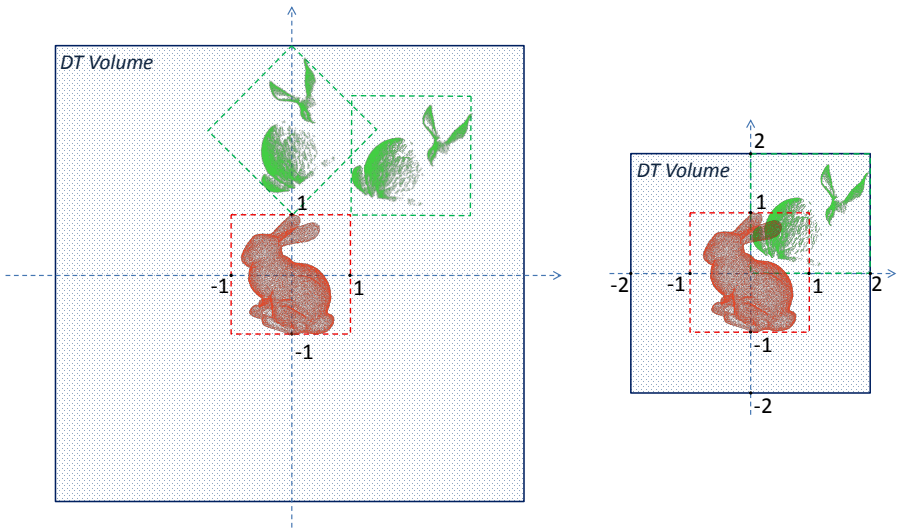

Figure II: Illustration of a Distance Transform with expansion factors (in 2D). **Left:** expansion factor $= 1+2\sqrt{2}$ (accordingly, $1+2\sqrt{3}$ in 3D). **Right:** expansion factor $= 2$.

More Results and Comparisons to Kd-tree[1]

In Sec. 6.1 of the main paper, we conducted experiments to show the remaining rotation and translation domains after BnB stops. Both synthetic points and real range data were used. Figure III shows typical remaining rotation domains on 2D slices of the rotation π-ball[2] for the synthetic points, and Figure IV shows typical remaining translation domains on three 2D slices of the $[-1,1]^3$ cube[3] for these. It can be seen that the remaining domains from the $300\times300\times300$ DT and the kd-tree are highly consistent. The differences between them are insignificant, especially considering they are well within the convergence basins of the optima, and the optima are well-contained by them. Figure V and Figure VI show the results for the bunny data and the dragon data we also tested. Again, the remaining domains from the DT and kd-tree were highly consistent; the differences being almost negligible.

In Sec. 6.2 of the main paper, running times of the proposed method were analyzed with "partial" to "full" registration experiments. We used 10 partial scans of the bunny and dragon datasets as data point-sets and their reconstructed models as the model point-sets. In the first experiment, we performed 100 tests with random initial rotations and translations, for each data point-set. As expected, Go-ICP achieved 100% correct registration on all the registration tasks, with both the DT and kd-tree. The comparisons of results using the DT and the kd-tree are presented in Figure VII and Figure VIII. It can be seen that, the rotation differences and the translation differences are all below 1 degree and 0.01, respectively.

References

[1] C. R. Maurer Jr, R. Qi, and V. Raghavan, "A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 25, no. 2, pp. 265–270, 2003.

[2] A. Fitzgibbon, "Robust registration of 2D and 3D point sets," *Image and Vision Computing*, vol. 21, no. 13, pp. 1145–1153, 2003.

[3] Y.-R. Wang and S.-J. Horng, "An $O(1)$ time algorithm for the 3D euclidean distance transform on the CRCW PRAM model," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 10, pp. 973–982, 2003.

[4] T.-T. Cao, K. Tang, A. Mohamed, and T.-S. Tan, "Parallel banding algorithm to compute exact distance transform with the GPU," *Pro. ACM SIGGRAPH Symp. Interactive 3D Graphics and Games*, pp. 83–90, 2010.

[5] M. Muja and D. Lowe, "Scalable nearest neighbour algorithms for high dimensional data," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 36, pp. 2227–2240, 2014.

[1] A kd-tree implementation from the FLANN library [5] was used.

[2] The 2D π-slices were chosen as follows. For the irregular tetrahedron data where only one optimal solution exists, we use 3 slices passing through the optimal solution and X-, Y-, Z-axes respectively. For other data where multiple optimal solutions exist, we chose 3 slices where each slice passes two randomly-selected optimal solutions plus the origin (due to shape symmetry there exists multiple optimal solutions).

[3] The three 2D slices of the $[-1,1]^3$ cube were chosen such that they pass the optimal solution and are parallel to the XY-, YZ- and XZ-planes respectively.

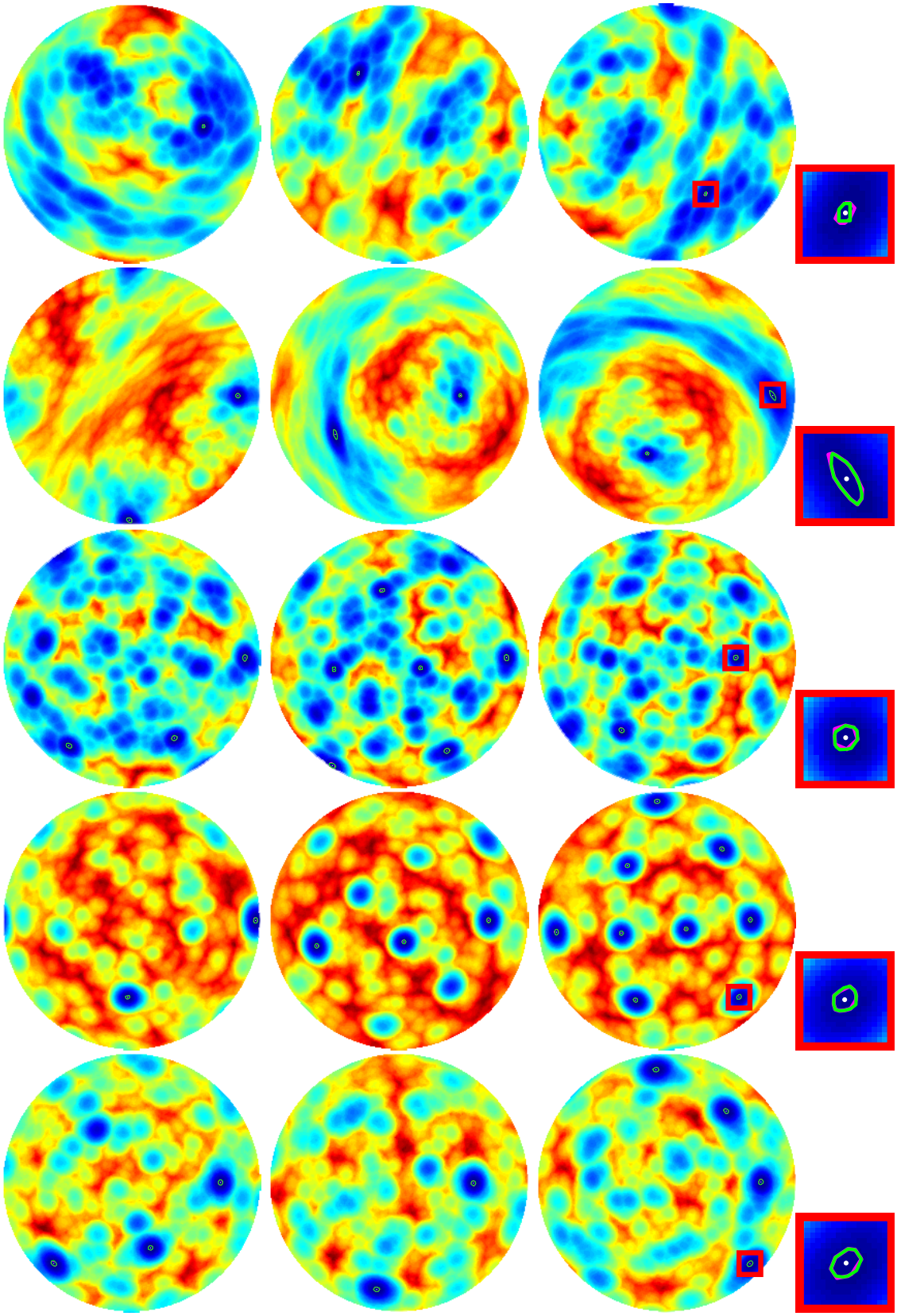

Figure III: The remaining rotation domains (on 2D slices of the π-ball) of the outer rotation BnB using a **300×300×300 DT (within magenta polygons)** and a **kd-tree (within green polygons)** for the synthetic points. From top to bottom: an irregular tetrahedron, a cuboid with three different side-lengths, a regular tetrahedron, a regular cube, and a regular octahedron. The white dots denote optimal solutions. The colors indicate registration errors evaluated via inner translation BnB with rotations on these slices (red for high error and blue for low error). ***Best viewed on screen with zoom.***

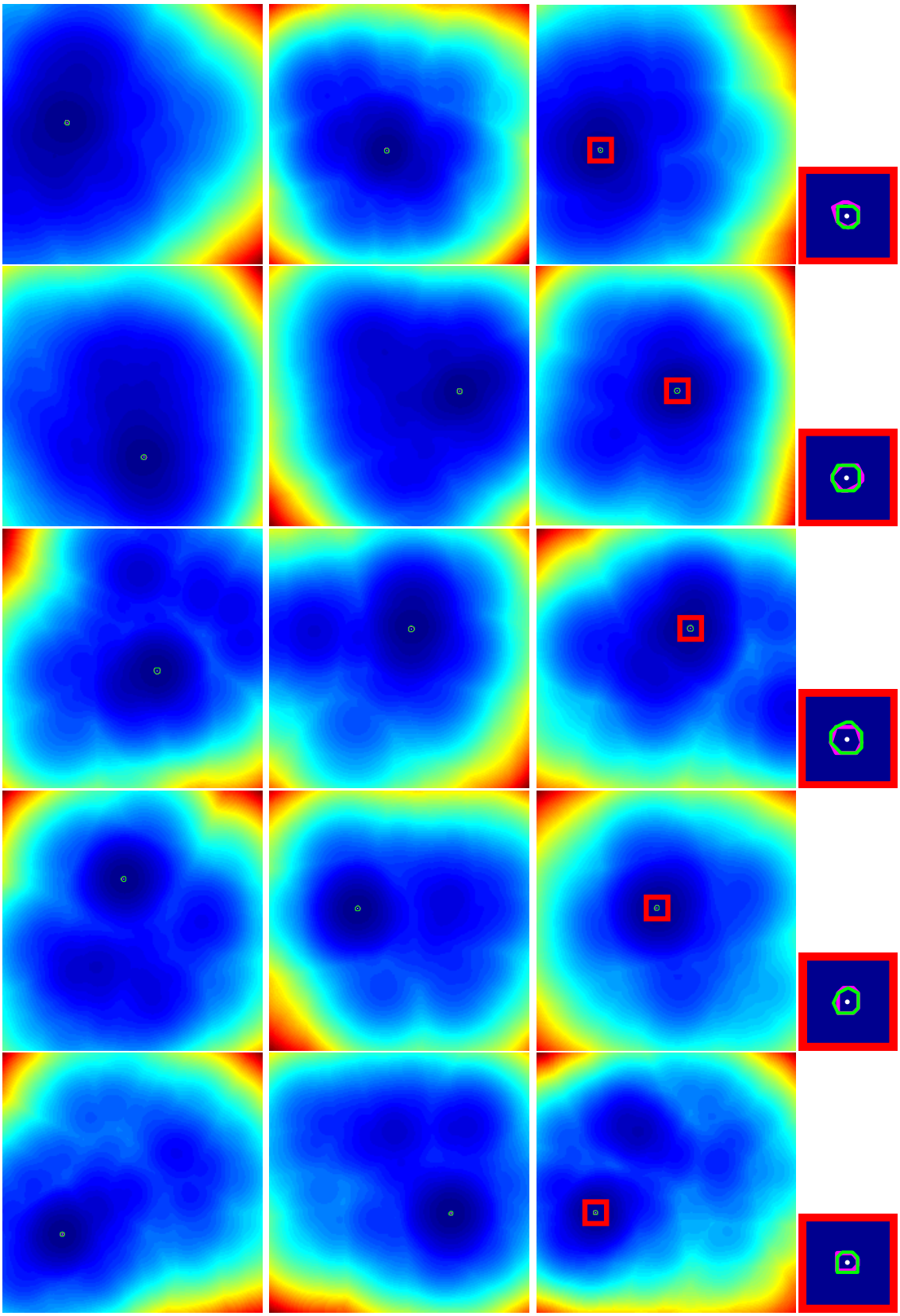

Figure IV: The remaining translation domains (on 2D slices of the $[-1, 1]^3$ cube) of the inner translation BnB, using a **$300 \times 300 \times 300$ DT (within magenta polygons)** and a **kd-tree (within green polygons)** for the synthetic points. From top to bottom: an irregular tetrahedron, a cuboid with three different side-lengths, a regular tetrahedron, a regular cube, and a regular octahedron. The white dots denote optimal solutions. The error maps contain only a handful of local minima partially because optimal rotations were used. ***Best viewed on screen with zoom.***

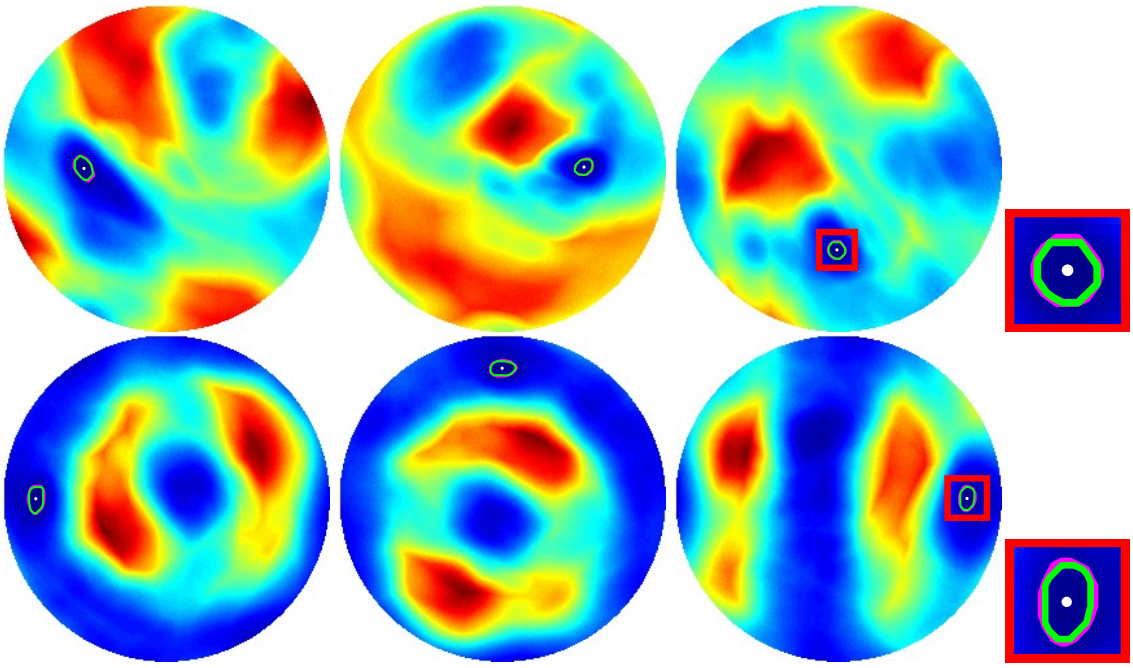

Figure V: The remaining rotation domains (on 2D slices of the π-ball) of the outer rotation BnB using a **$300 \times 300 \times 300$ DT (within magenta polygons)** and a **kd-tree (within green polygons)** for bunny (first row) and dragon (second row) data. The white dots denote optimal solutions. The error maps on the slices were computed via inner translation BnB. ***Best viewed on screen with zoom.***

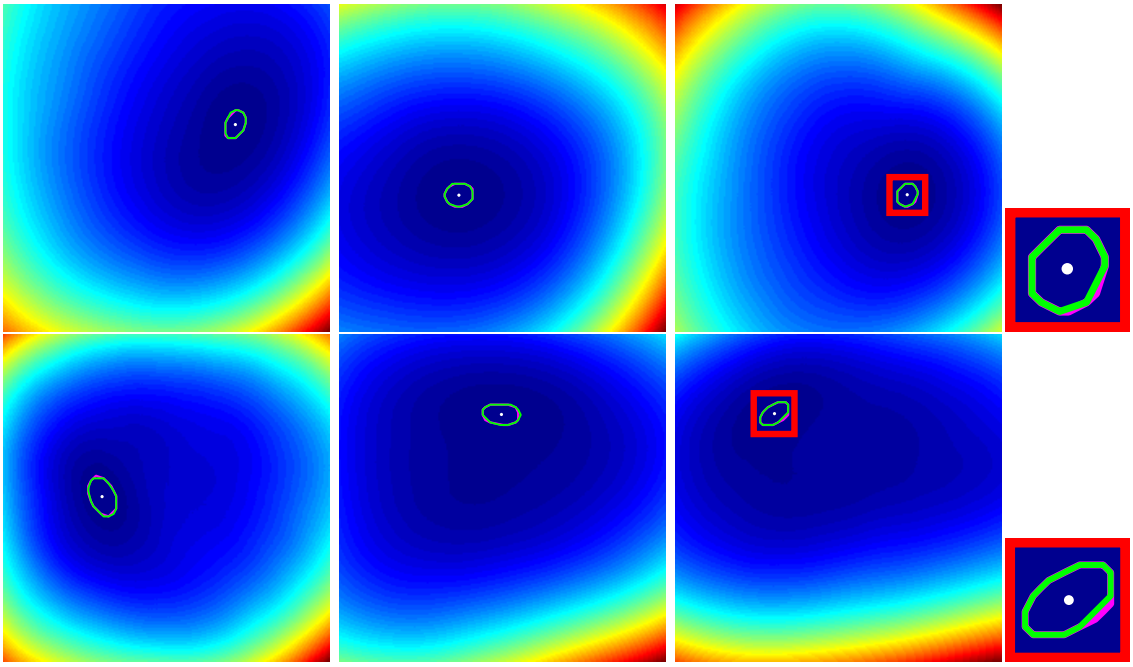

Figure VI: The remaining translation domains (on 2D slices of the $[-0.5, 0.5]^3$ cube) of the inner translation BnB, using a **$300 \times 300 \times 300$ DT (within magenta polygons)** and a **kd-tree (within green polygons)** for bunny (first row) and dragon (second row) data. The white dots denote optimal solutions. ***Best viewed on screen with zoom.***

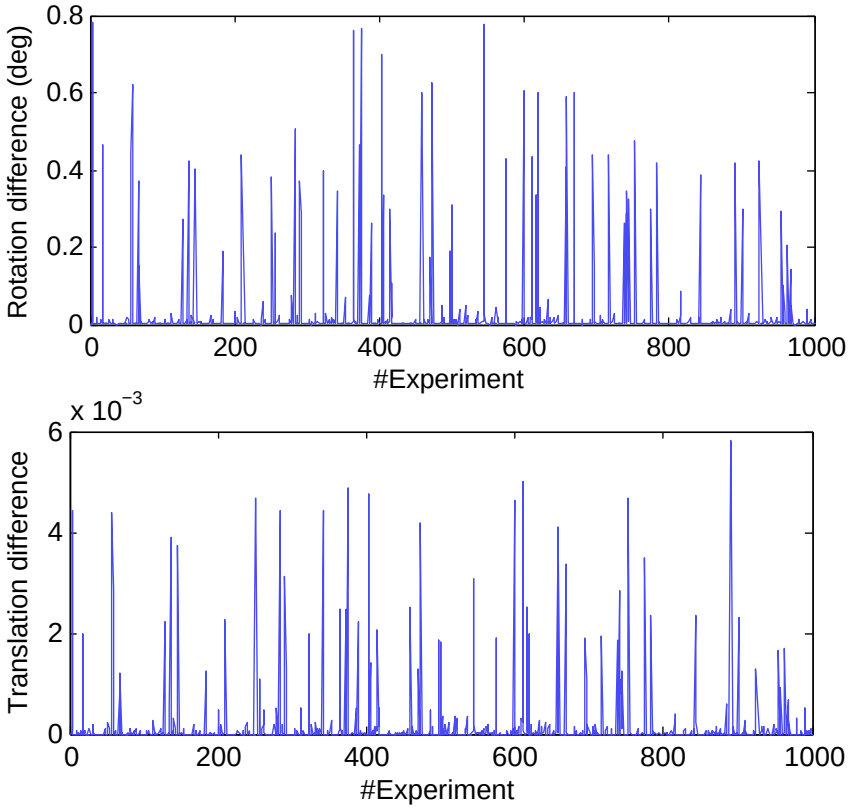

Figure VII: Comparison of the obtained solutions using a DT and a kd-tree on the bunny data (10 data point-sets with 100 random poses).

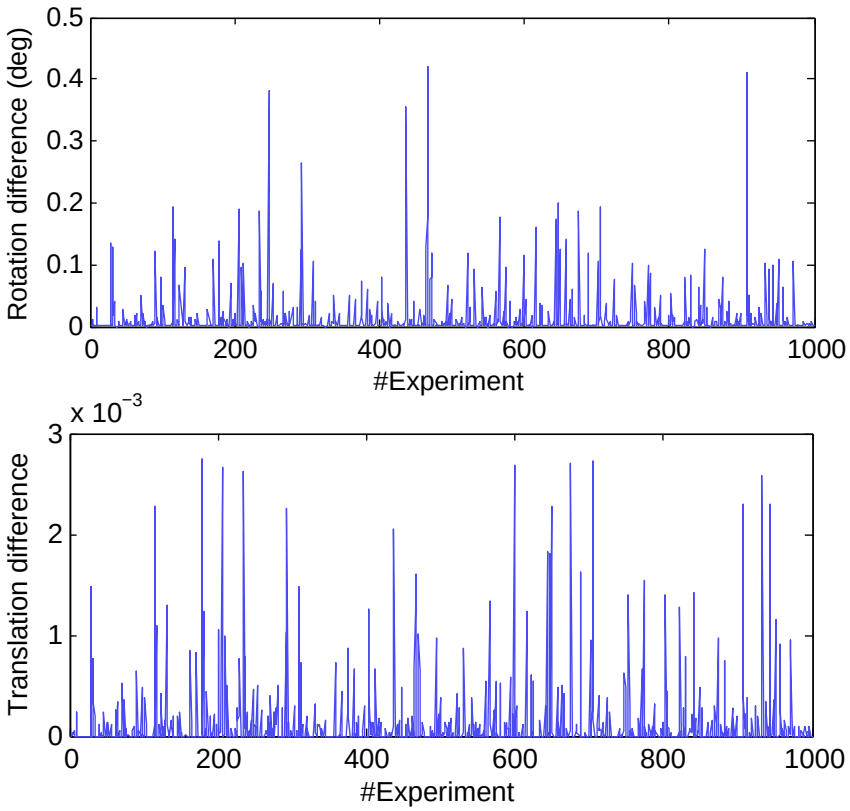

Figure VIII: Comparison of the obtained solutions using a DT and a kd-tree on the dragon data (10 data point-sets with 100 random poses).