Exercise Medical Robotics CS4270 – Exercise Sheet 3

Summer Semester 2018
Institute for Robotics and Cognitive Systems
boettger@*rob.uni-luebeck.de*

Navigation and Registration

SOLUTIONS

Please submit your solutions before Monday 22.05.2017 at 14:15. The names of all group members must be included in the solution sheets/files. Handwritten solutions can be either submitted in the Institute for Robotics and Cognitive Systems (postbox in front of room 98) or scanned and uploaded in Moodle. MATLAB codes must be properly and briefly commented and uploaded in Moodle.

1 Iterative Closest Point (ICP) Algorithm (35 Points)

The ICP algorithm can be used to register two or more point clouds. As seen in Figure 1, let $Q \in \mathbb{R}^{3\times N}$ be the fixed point set and $P \in \mathbb{R}^{3\times N}$ the floating point set. The ICP algorithm attempts to find a transformation T with rotation R and translation t that matches P to Q, which is done in two steps: matching and moving. This first task deals with the latter only.

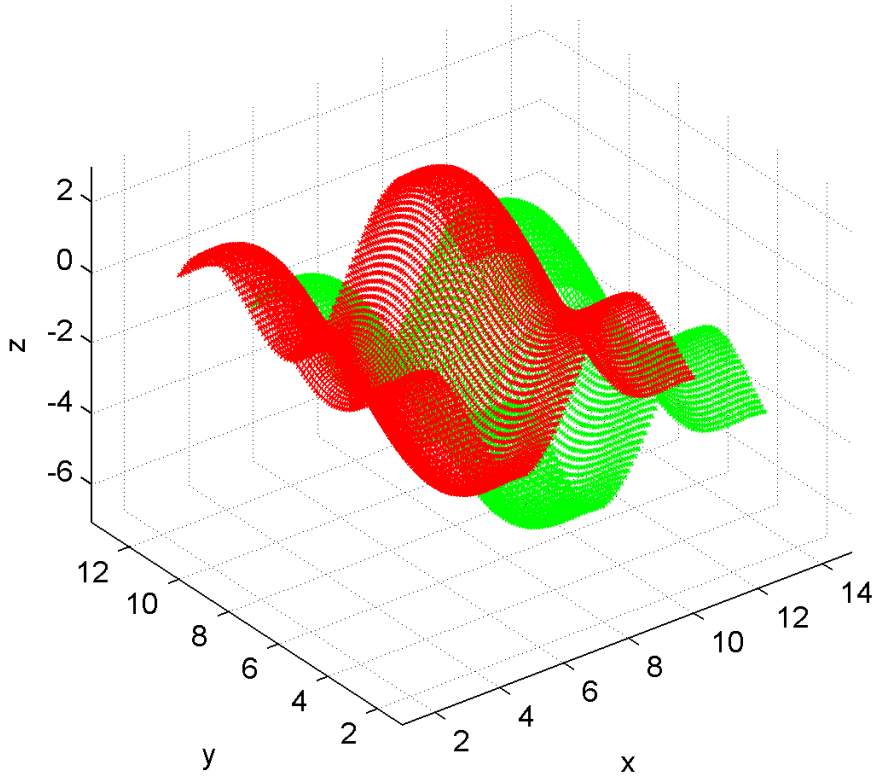


Figure 1: The two point clouds Q (red) and P (green) to be registered

(a) Let α, β and γ be the rotation angles about the x, y and z-axis and let $t = (t_x, t_y, t_z)$ be the translational vector. Given valid point-to-point correspondences between P and Q we can minimize the cost function

$$\sum_{i=1}^{N} \|Rp_i + t - q_i\|^2$$

to find a rotation R and a translation t that optimally match one point cloud to the other. To find the rotation angles, the lecture script outlines how to linearize the rotation matrix, find the derivatives with respect to the unknown motion parameters and solve a system of linear equations. Prove that $\sin(x) = x$ and $\cos(x) = 1$ for small x. (2P)

Using Taylor series to represent the sin and cos functions

$$f(x)|_a = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots$$

we get

$$\sin(x)|_0 = 0 + \frac{1}{1!}(x-0) + 0 - \frac{1}{3!}(x-a)^3 + \cdots$$

$$\cos(x)|_0 = 1 + 0 - \frac{1}{2!}(x-0)^2 + 0 + \cdots$$

At $x \cong 0$ (small x) the factorial of $(x-a)$ grows with each differentiation, reducing its influence on the overall function. Therefore, for the sine function we cancel out all the terms starting the second factorial (third term). This leaves us with

$$\sin(x)|_0 = \mathrm{x}$$

$$\cos(x)|_0 = 1$$

(b) Derive the linearized version of the three-dimensional rotation matrix. (6P)
Hint: Consider the three elementary rotation matrices (for each axis x, y and z) separately. In each matrix, replace the sin and cos functions by their linear approximations. Then multiply the three matrices, while setting products of two small angles to zero.

The elementary rotation matrices are

$$R(x,\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_\alpha & -s_\alpha \\ 0 & s_\alpha & c_\alpha \end{pmatrix}, R(y,\beta) = \begin{pmatrix} c_\beta & 0 & s_\beta \\ 0 & 1 & 0 \\ -s_\beta & 0 & c_\beta \end{pmatrix}, R(z,\gamma) = \begin{pmatrix} c_\gamma & -s_\gamma & 0 \\ s_\gamma & c_\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

We replace s_x with x and c_x with 1 for linearization and multiply all three matrices to yield the linearized 3d rotational matrix

$$R = R(x,\alpha) \times R(y,\beta) \times R(z,\gamma) = \begin{pmatrix} 1 & -\gamma & \beta \\ \alpha\beta + \gamma & -\alpha\beta\gamma + 1 & -\alpha \\ -\beta + \alpha\gamma & \beta\gamma + \alpha & 1 \end{pmatrix}$$

Since the angles α, β and γ are small, multiplying them results in a very small value that can be neglected, leaving us with

$$R = \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix}$$

(c) Exemplarily calculate the derivative of the cost function with respect to the rotation about the x-axis and the translation t_x in x-direction (4P).

The cost function is broken down to

$$f = \sum_{i=1}^{n} \|Rp_i + t - q_i\|^2 = \sum_{i=1}^{n} \left\| \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix} \begin{pmatrix} p_{ix} \\ p_{iy} \\ p_{iz} \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} - \begin{pmatrix} q_{ix} \\ q_{iy} \\ q_{iz} \end{pmatrix} \right\|^2$$

$$= \sum_{i=1}^{n} \left\| \begin{pmatrix} p_{ix} - \gamma p_{iy} + \beta p_{iz} + t_x - q_{ix} \\ \gamma p_{ix} + p_{iy} - \alpha p_{iz} + t_y - q_{iy} \\ -\beta p_{ix} + \alpha p_{iy} + p_{iz} + t_z - q_{iz} \end{pmatrix} \right\|^2$$

The derivative with respect to translation on the x-axis is

$$\frac{df}{dt_x} = \sum_{i=1}^{n} 2(p_{ix} - \gamma p_{iy} + \beta p_{iz} + t_x - q_{ix})$$

And the derivative with respect to rotation around the x-axis is

$$\frac{df}{d\alpha} = \sum_{i=1}^{n} -2(\gamma p_{ix} + p_{iy} - \alpha p_{iz} + t_y - q_{iy}) p_{iz} + 2p_{iy}(-\beta p_{ix} + \alpha p_{iy} + p_{iz} + t_z - q_{iz})$$

(d) Present a Matlab implementation which uses the linear approximation above to match two point clouds. Instead of finding all derivatives and to solve the system of linear equations you can use the Matlab function $fminunc()$ to minimize the cost function. Start with zeros as an initial guess. Finally use your implementation to match the cloud pairs $cloud1$, $cloud2$ and $cloud3$ in $data.mat$. You can assume that the order of points in the arrays is already matched by array index. Give the root mean square (RMS) matching error and list the motion parameter output from the optimization. (14P)

See attached code *ICP_Medrob.m* and related files.

Result:	(cloud1)	Rotation ≈ (0, 0, 0)	Translation ≈ (-5, -3, 4)
	(cloud2)	Rotation ≈ (-0.02, -0.03, -0.05)	Translation ≈ (-5.3, -2.7, 3.9)
	(cloud3)	Rotation ≈ (-0.2, -0.27, -0.53)	Translation ≈ (-7.9, -0.65, 3.5)

(e) Let c_q and c_p be the centroids of the point clouds and $q_{ic} = q_i - c_q$ and $p_{ic} = p_i - c_p$ the centralized point sets. In 1987, Arun proved that the optimal matching can be obtained using singular value decomposition (SVD):

$$SVD(Q_c P_c^T) = U\Lambda V$$

$$R = UV^T$$

$$t = c_q - Rc_p$$

Implement this approach (using Matlab's $svd()$ function) and compute the remaining RMS error for all three point clouds. Compare the RMS error for both approaches. What can you say about the matching quality for each point cloud? Discuss your result with respect to the motion parameters and the values you obtained for the derivatives. Why do the latter show a different behavior than the RMS error? (6P)

See attached code *ICP_Medrob.m* and related files.

The SVD approach yields more accurate results with lower RMS matching error, and it runs faster.

(f) List at least three shortcomings of this basic ICP approach (assuming a brute force method for the matching step). (3P)

i. Runs slow for finding closest point pairs, therefore not very suitable for large data and/or real-time processing.

ii. Requires preprocessing steps, ex. for calculating closest points.

iii. Works better for small features relative to object size, therefore can be used for local rather than global optimization.

2 k-d-trees (22 Points)

To accelerate the algorithm, brute force search in the matching step can be replaced by k-d-trees. The k-d tree algorithm computes the nearest neighbor of p_i in the point cloud Q. A simplified version works as follows:

1. Sort the points in Q according to ascending x-axis.
2. Find the mean value of all x-axis values in Q (see Figure 2).
3. Partition Q into two subsets by the mean line, i.e. a vertical line along the mean value.
4. Repeat this process for both subsets in the y-direction. The result is a tree structure, shown in Figure 2.
5. To select the nearest neighbor of p_i, simply insert p_i into the tree, following the tree downwards from the top node. In the example, we obtain q_3 as the nearest neighbor of p_i.

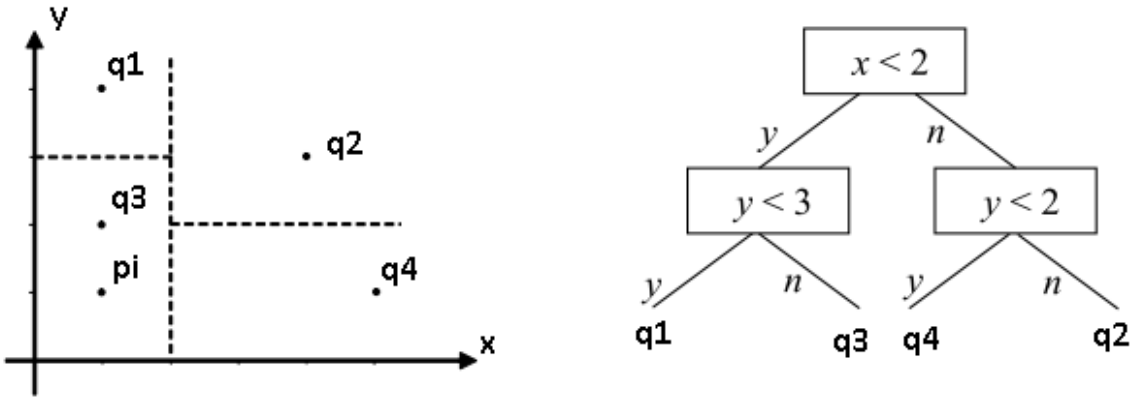


Figure 2: k-d-tree

(a) Given the two point clouds

$$Q = \{(2,1), (3,0.5), (4,3), (3.5,2), (1,1.2), (2,1)\}$$

and

$$P = \{(4,1), (1.5,2.5), (3,4)\},$$

build a k-d-tree using the data from Q (pen and paper work). Start with the first data dimension to find the first separating plane. (8P)
Hint: use the arithmetic mean to find the separating plane.

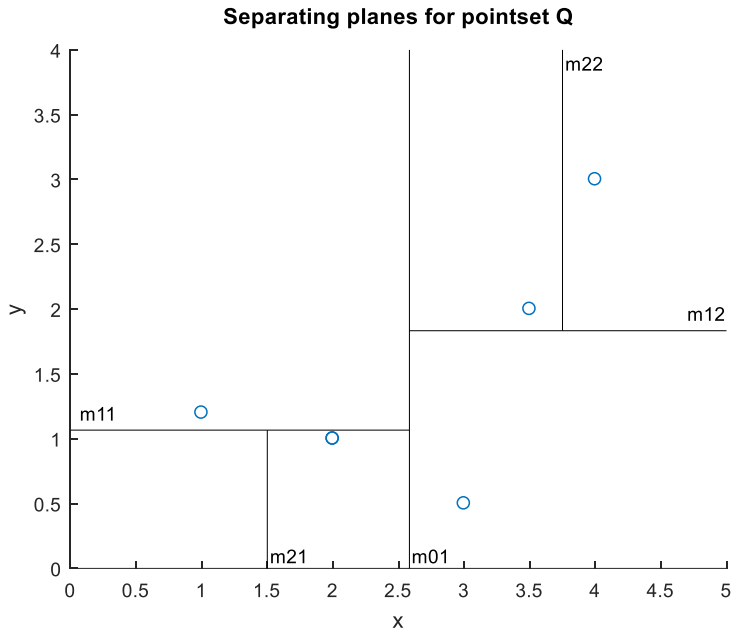


Figure 3: Exercise 2(a)

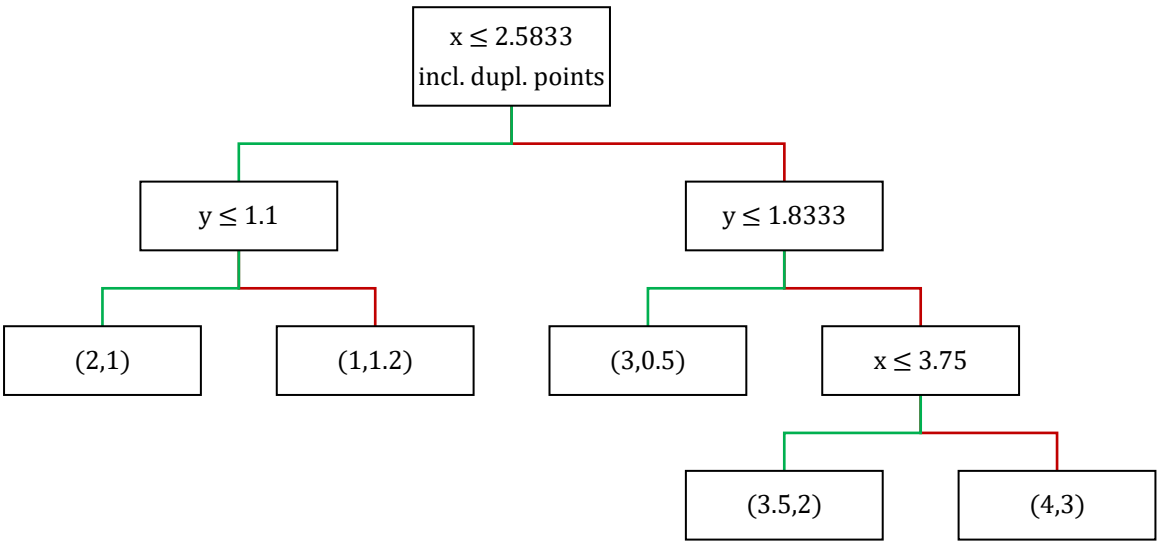
$m01 = \frac{2+3+4+3.5+1+2}{6} = 2.5833$ contains both duplicate points (2, 1), (2, 1)

$m11 = \frac{1+1.2}{2} = 1.1$

$m12 = \frac{0.5+2+3}{3} = 1.8333$

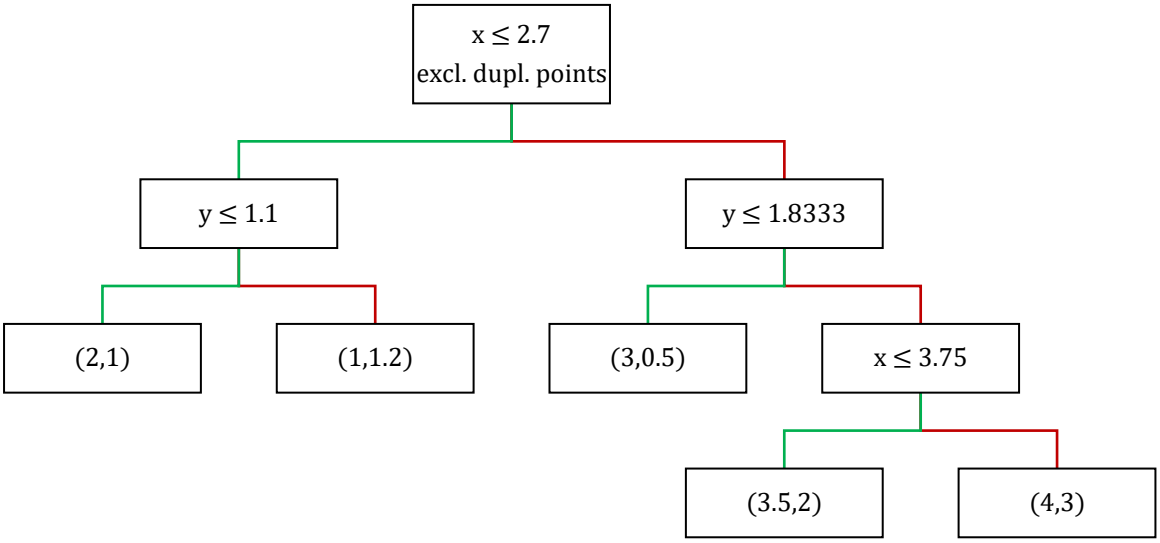
$m21 = \frac{1+2}{1} = 1.5$ (this does not separate any further points)

$m22 = \frac{3.5+4}{2} = 3.75$



(b) Implement a function $kdtree()$ which takes a point cloud and outputs a data struct that contains all codes (thresholds, node number and child nodes) of your k-d-tree. (5P)
Hint: The function can call itself recursively to build the tree. Always start with the x-direction when partitioning the intervals.

See attached code *kdtrees.m* and related files. (Don't include duplicate points into kdtree)



(c) Implement a function $kdtreeclassify()$ which takes the node struct and a point and outputs the nearest point neighbor according to your tree. (4P)

See attached code *kdtreeclassify.m* and related files.

$Q = \{(2,1), (3,0.5), (4,3), (3.5,2), (1,1.2), (2,1)\}$
$P = \{(4,1), (1.5,2.5), (3,4)\}$

P		Q
(4, 1)	→	(3, 0.5)
(1.5, 2.5)	→	(1, 1.2)
(3, 4)	→	(3.5, 2)

(d) Use the function to find point-to-point correspondences with P according to your tree. (3P)

See attached code *kdtrees_example.m* and related files.

(e) Which problem might frequently occur when using the median instead of the mean to find a new plane? (2P)

Points would lie on the separating plane yielding wrong results when looking for the nearest point. These points would be either on a negative or a positive subset (depending on the usage of equal sign) in a way they could be overlooked.

3 *Optional:* ICP Implementation (10 Points)

Use your implementations from task 1 (Arun87 moving step) and 0 (k-d-tree matching step) to implement the entire ICP algorithm. Apply the algorithm to the point clouds Q and P given in $data\_optional.mat$, where the latter is the floating point cloud. Plot the RMS error over 100 iterations and give the result for the total transformation matrix.

Hint: It might be useful to start with a moving step and random point-to-point matching. In some cases with a large translational offset this might prevent the k-d-trees from assigning all points of P to only one or two in Q.

See attached code *ICP_optional_example.m* and related files.