Thin Plate Splines, as implemented in `DICOMautomaton`

Haley Clark

2020 June 18th

Introduction

Thin plate splines (TPS) refers to a physically-motivated interpolation technique that can be used to 'warp' one point cloud to another point cloud. The final warp is smooth and defined between the point, which makes the resulting warp attractive as mapping between coordinate systems. In `DICOMautomaton` TPS are implemented in $\mathbb{R}^3$, but the technique generalizes to other dimensions.

The TPS technique provides many convenient properties that make it a good choice for warping problems. In particular, the warp is smooth, straightforward to compute, and requires virtually no structure in the data. However, there are several limitations:

1. Correspondence between the point clouds must be known in advance.
2. Due to the correspondence issue, outliers are not handled.
3. Noise can result in spurious or dramatic warps.
4. Solving for warps is computationally costly, can suffer from numerical imprecision, and system degeneracy can cause stability issues.

`DICOMautomaton` implements TPS in multiple varieties to overcome some of these limitations. The correspondence problem and outliers can both be handled using an iterative solver ('TPS Robust Point Matching'). Regularization can, to some extent, control the impact of noise. Finally, use of robust matrix least-square solution techniques such as $LDL^{\intercal}$ decomposition or the Moore-Penrose pseudo-inverse, can improve computation speed and robustness.

Thin plate splines

There any many classical references for thin plate splines. Fred Bookstein provides a simplified, hands-on overview that we will mostly follow here [1]. Many of Grace Wahba's works provide comprehensive theoretical exhibition of splines if the reader is curious (e.g., [2]). Note that this work describes specifically how TPS are implemented in `DICOMautomaton`, so some familiarity is assumed. Also note that in the following we will assume $\mathbb{R}^3$ unless otherwise noted.

Consider a 'moving' point set (M) consisting of N points in $\mathbb{R}^3$, i.e., $M = \left\{ \vec{P}_n \middle| n \in [1:N] \right\}$. Our aim is to find a smooth function that warps the moving set to a 'stationary' point set (S) that also consists of N points in $\mathbb{R}^3$, i.e., $S = \left\{ \vec{V}_n \middle| n \in [1:N] \right\}$. Both sets are *ordered* so that the i^{th} point in each set correspond to one another: $\vec{P}_i \leftrightarrow \vec{V}_i$. We assume both point clouds have irregular shapes, and that points are spaced irregularly.

The (idealized) thin plate spline is described by the biharmonic equation $\Delta^2 U = 0$. The *fundamental solution* in $\mathbb{R}^3$ – that is, the solution to $\Delta^2 U(r) = \delta(r)$ with $\delta(r)$ the Dirac delta 'function' – is $U(r) = C\,|r|$ for some constant C. In $\mathbb{R}^2$ it is $U(r) = Cr^2 \log r^2$. In $\mathbb{R}^2$ U approximately describes the response of a thin strip of drafting spline that is bent; in $\mathbb{R}^3$ the interpretation is similar, except a thin *plate* is used. We can define the 'kernel' function $k(i,j) = U\left(\left|\vec{P}_i - \vec{P}_j\right|\right)$. Given any $\vec{P} \in M$, the purpose of k is to encode the proximity to every other moving point. In other words, the kernel function is the means in which the overall structure of M is accounted for.

We define a $(N+3+1) \times (N+3+1)$ matrix (K) consisting of the kernel function applied to every pair of points in the moving set M. K is symmetric and diagonal elements are all zero.

$$K \equiv \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & \dots \\ k_{1,0} & k_{1,1} & k_{1,2} & \dots \\ k_{2,0} & k_{2,1} & k_{2,2} & \dots \\ & \dots & & \end{bmatrix} \tag{1}$$

We can then define a $N \times (3+1)$ matrix (P) where each row consists of the ordered points from M expressed in *homogeneous coordinates*. The role of P is to encode the absolute positions of the points of interest that will be warped.

$$P \equiv \begin{bmatrix} 1 & | & \vec{P}_1^{\intercal} \\ 1 & | & \vec{P}_2^{\intercal} \\ 1 & | & \vec{P}_3^{\intercal} \\ & \dots & \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ & & \dots & \end{bmatrix} \tag{2}$$

We can define a $(N+3+1) \times (N+3+1)$ aggregate matrix (L) that will be known as the 'system' matrix

$$L \equiv \begin{bmatrix} K & | & P \\ — & — & — \\ P^{\intercal} & | & 0 \end{bmatrix} = \begin{bmatrix} k_{1,1} & k_{1,2} & \dots & k_{1,N} & 1 & x_1 & y_1 & z_1 \\ k_{2,1} & k_{2,2} & \dots & k_{2,N} & 1 & x_2 & y_2 & z_2 \\ & & \dots & & & & \dots & \\ k_{N,1} & k_{N,1} & \dots & k_{N,N} & 1 & x_N & y_N & z_N \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ x_1 & x_2 & \dots & x_N & 0 & 0 & 0 & 0 \\ y_1 & y_2 & \dots & y_N & 0 & 0 & 0 & 0 \\ z_1 & z_2 & \dots & z_N & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

where 0 is a 4×4 zero matrix. Note that L is symmetric, and all diagonal coefficient are zero since $k_{i,i} = 0$. The system matrix can be shown to satisfy $LZ = Y$ where Y is a $(N+3+1) \times 3$ matrix that holds the points of the stationary set S padded at the bottom with four rows of zeros

$$Y \equiv \begin{bmatrix} \vec{V}_1^{\intercal} \\ \vec{V}_2^{\intercal} \\ \dots \\ \vec{V}_N^{\intercal} \\ 0^{\intercal} \\ 0^{\intercal} \\ 0^{\intercal} \\ 0^{\intercal} \end{bmatrix} = \begin{bmatrix} x'_1 & y'_1 & z'_1 \\ x'_2 & y'_2 & z'_2 \\ \dots & & \\ x'_N & y'_N & z'_N \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4}$$

and Z is a $(N+3+1) \times 3$ matrix that contains the TPS transformation parameters. Namely,

$$Z = \begin{bmatrix} \vec{W}_x & \vec{W}_y & \vec{W}_z \\ — & — & — \\ a_{x,0} & a_{y,0} & a_{z,0} \\ a_{x,x} & a_{y,x} & a_{z,x} \\ a_{x,y} & a_{y,y} & a_{z,y} \\ a_{x,z} & a_{y,z} & a_{z,z} \end{bmatrix} \tag{5}$$

where the vectors $\vec{W}$ represent warp parameters and the a values represent an affine transformation. In principle, we can compute Z via $L^{-1}Y = Z$ when L is well-behaved. However, this is not always the case, and even if it is, numerical instabilities can cause issues. We will discuss alternatives to inversion shortly.

Using Z, we can construct the TPS function as

$$f(\vec{r}) \equiv \begin{bmatrix} a_{x,0} + a_{x,x} r_x + a_{x,y} r_y + a_{x,z} r_z + \sum_{i=1}^{N} w_{x,i} U\left(\left|\vec{P}_i - \vec{r}\right|\right) \\ a_{y,0} + a_{y,x} r_x + a_{y,y} r_y + a_{y,z} r_z + \sum_{i=1}^{N} w_{y,i} U\left(\left|\vec{P}_i - \vec{r}\right|\right) \\ a_{z,0} + a_{z,x} r_x + a_{z,y} r_y + a_{z,z} r_z + \sum_{i=1}^{N} w_{z,i} U\left(\left|\vec{P}_i - \vec{r}\right|\right) \end{bmatrix} \tag{6}$$

where $w_{x,i}$ is the i^{th} coefficient of $\vec{W}_x$. It can be seen that the spline contributions to f are bounded and flatten asymptotically and thus act locally, whereas the affine contributions act globally. However, while the warp components flatten, they do not coalesce to a constant at infinity [1]. Therefore, while f is mathematically well-defined throughout all of $\mathbb{R}^3$, extrapolation it not generally recommended. Also note that when f does *not* fold, it represents a *diffeomorphism* [1]. This is an important feature when f needs to be inverted.

The function f minimizes the 'bending energy'

$$E_{TPS}(f) \quad = \quad \iiint_{\mathbb{R}^3} \left(\frac{\partial^2 f}{\partial x^2}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 + \left(\frac{\partial^2 f}{\partial z^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial z}\right)^2 + 2\left(\frac{\partial^2 f}{\partial y \partial z}\right)^2 dx\, dy\, dz \tag{7}$$

and further satisfies the least-squares criteria

$$\sum_{i=1}^{N} \left(\vec{V}_i - f\left(\vec{P}_i\right)\right)^2 = 0 \tag{8}$$

since, barring numerical matters, every point is interpolated exactly by f.

Kernel function performance

While the $\mathbb{R}^3$ kernel function *should* be used when points are in $\mathbb{R}^3$, in practice the $\mathbb{R}^2$ kernel function was found to be more stable and provide overall more sensible warps than the $\mathbb{R}^3$ kernel. Though the implementation exclusively handles point sets in $\mathbb{R}^3$, the default kernel function in `DICOMautomaton` is currently the $\mathbb{R}^2$ kernel function.

Tikhonov regularization

In practice, the bending energy might be inappropriately large, even when minimized, which can result in an insufficiently smooth warping function. This is commonly seen when the point clouds are ill-conditioned, noisy, or contain outliers since the warp function becomes contorted in order to ensure every point is interpolated.

Additional smoothness can be forced through the use of *regularization*, which gives the warp function a small amount of 'wiggle room' in the sense that not all points will necessarily be interpolated. This is accomplished by introducing a regularization parameter λ and redefining $K \to K'$ where $K' = K + I\lambda$ so that all diagonal coefficients are non-zero. Note that K is altered, but the bottom-right corner of L is not since the 4×4 O matrix serves to ensure the warp coefficients sum to zero [1,3]. This effectively causes the system matrix L to provide an altered representation of the structure of M. It also lowers the *condition number* of L, improving numerical stability.

Since regularization causes f to no longer exactly interpolate all points, the E_{TPS} is altered:

$$E'_{TPS}(f) = \sum_{i=1}^{N} \left(\vec{V}_i - f\left(\vec{P}_i\right) \right)^2 + \lambda \iiint_{\mathbb{R}^3} \left(\frac{\partial^2 f}{\partial x^2}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 + \left(\frac{\partial^2 f}{\partial z^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial z}\right)^2 + 2\left(\frac{\partial^2 f}{\partial y \partial z}\right)^2 dx\, dy\, dz. \quad (9)$$

Note that as $\lambda \to 0$ the sum term approaches 0 and the bending energy is suppressed by λ. However, the bending energy is still minimized when $\lambda = 0$, which recovers the original E_{TPS}.

Determining which value of λ is optimal, or even merely acceptable, is a trade-off that must be evaluated by the end-user. It depends on the problem, especially noise and how the point sets are sampled. Typically a small value, on the order of 1-2 orders of magnitude smaller than typical near-diagonal K coefficients, is suitable. Larger values are needed when data is noisier.

Avoiding matrix inversion

Matrix inversion is computationally costly, may not be possible, and is often not explicitly necessary to find a solution. Standard methods for solving systems of linear systems include LU decomposition, Cholesky decomposition (or the closely related $LDLT^{intercal}$ decomposition), QR decomposition, and the Moore-Penrose pseudo-inverse. Because each method has strengths and weaknesses, `DICOMautomaton` supports multiple methods.

Moore-Penrose pseudo-inverse

The simplest solution is to replace L^{-1} with L^{*}, the Moore-Penrose pseudo-inverse. The major benefit is that L^{*} reduces to L^{-1} when L^{-1} exists. It also provides a reasonable *polyfill* when L^{-1} does not exist. The major downside is that computation of L^{*} is slow. The meaning of L^{*} as a solution is also blurry when L^{-1} does not exist. In practice this is usually an acceptable trade-off for warping functions (which are by nature inexact and only somewhat physically motivated anyways), but it can also produce completely invalid results.

Use of the Moore-Penrose pseudo-inverse is optional in `DICOMautomaton` but is not recommended for large N.

LU factorization

LU factorization is often applied to solve linear systems of equations. While it works well and is well-behaved in general, it is relatively slow for large N and has therefore not been implemented in `DICOMautomaton`.

$LDL^{\intercal}$ decomposition

A faster approach is to make use of $LDL^{\intercal}$ decomposition. The basic premise is to altogether avoid computing L^{-1} by breaking the problem into three stages: first, the decomposition; second, solving a simplified system with a using forward substitution; and finally solving another simplified system with back substitution. $LDL^{\intercal}$ decomposition exploits the symmetry of the TPS system matrix L and is therefore more efficient than LU decomposition [4]. It can also decompose any symmetric matrix. Computationally, the stages can be broken up so the decomposition can be re-used. This is especially helpful when iterative solutions are required.

Use of $LDL^{\intercal}$ decomposition is the default in `DICOMautomaton`.

QR decomposition

QR can be used to more explicitly disentangle the warp components from the affine components. See [5] or [6] for more details. Lira et al. recommend Cholesky decomposition over QR decomposition [7]. While $LDL^{\intercal}$ decomposition is not equivalent to Cholesky decomposition, $LDL^{\intercal}$ decomposition must be used for matrices that cannot be shown to be positive definite in general (i.e., the system matrix L). Furthermore, QR decomposition was implemented in `DICOMautomaton`, but was found to be less stable than the Moore-Penrose pseudo-inverse and $LDL^{\intercal}$ decomposition techniques, so was removed.

Thin plate spline – robust point matching

Thin plate splines require the correspondence between points to be known *a priori*. An iterative approach that does **not** require *a priori* correspondence, the so-called 'Thin plate spline – robust point matching' (TPS-RPM) technique, was developed by Hali Chui and Anand Rangarajan in 2001 [5,8]. The core idea is to pair the TPS machinery with a 'softassign' procedure. Correspondence is updated using the existing TPS solution, and the TPS solution is updated using the current correspondence – this back-and-forth continues iteratively. TPS-RPM makes use of simulated annealing to control the range of influence each point is subject to by introducing a 'temperature' parameter T. When T is high, each point in M can be influenced by each point in S. As T reduces, points in M are only influenced by nearby points in S.

In practice, TPS-RPM behaves similarly to a the well-known iterative closest point algorithm, but is less dependent on the initial starting position of the moving mesh, outliers, and noise.

The TPS-RPM algorithm is as follows:

1. Pre-compute static coefficients of L.
2. Populate Z with an identity affine transformation and null warp coefficients.
3. Populate the correspondence matrix C so that all points correspond equally with one another and all points are weakly and equally seeded as being outliers.
4. Initialize deterministic annealing parameters.
   - Initial and final T.
   - Initial λ.
5. Anneal until final T reached:
   a. Use softassign to refine the correspondence (using current f).
   b. Solve for updated F via the aforementioned least-squares technique (using current correspondence).
   c. Reduce T and λ.
6. Extract final f and correspondence.

The TPS-specific parts are mechanically no different than the standard TPS fitting procedure, except that:

1. Instead of having N points in both M and S we now have N_M and N_S, respectively.
2. A correspondence matrix is used to fill Y rather than using the points of S directly.
3. E_{TPS} is modified to suppress the effects of outliers (see [5]).

Softassign

The essential action of softassign is to convert the binary correspondence, which is needed for TPS, into a continuous function that we can *gradually* binarize. The binarization process is controlled by the deterministic annealing schedule, or more specifically the current T.

We introduce a $(N_M + 1) \times (N_S + 1)$ matrix C such that coefficient $c_{i,j}$ represents the correspondence between $\vec{P}_i$ and $\vec{V}_j$. There is one additional row and one additional column which are used to indicate whether $\vec{P}_i$ or $\vec{V}_j$ are outliers. When $T = 0$ then $c_{i,j} \in [0, 1]$ (i.e., $c_{i,j}$ are binary) but otherwise $c_{i,j} \in [0 : 1]$ (i.e., $c_{i,j}$ are continuous). Allowing *partial* correspondence provides a smooth energy landscape that is easier to optimize.

The correspondence is iteratively updated, but rows and columns need to be normalized such that both

$$\sum_{n=1}^{N_M+1} c_{n,j} = 1 \tag{10}$$

$$\sum_{n=1}^{N_S+1} c_{i,n} = 1 \tag{11}$$

with $i \in [1, N_M]$ and $j \in [1, N_S + 1]$. This condition results in M being a *doubly-stochastic* matrix, with the exception of the row and column that contain the bottom-right coefficient.

Coefficients $c_{i,j}$ can be updated with $i \in [1, N_M]$ and $j \in [1, N_S]$ as

$$c_{i,j} = \frac{1}{T} \exp\left(\frac{\zeta}{T}\right) \exp\left(-\frac{\left(\vec{V}_j - f\left(\vec{P}_i\right)\right)^2}{T}\right) \tag{12}$$

$$c_{i,N_S+1} = \frac{1}{T_0} \exp\left(-\frac{\left(\vec{\phi}_S - f\left(\vec{P}_i\right)\right)^2}{T_0}\right) \tag{13}$$

$$c_{N_M+1,j} = \frac{1}{T_0} \exp\left(-\frac{\left(\vec{V}_j - \vec{\phi}_M\right)^2}{T_0}\right) \tag{14}$$

$$\tag{15}$$

where: ζ is another regularization parameter that can be introduced to help suppress outliers; T_0 is the starting temperature; and $\vec{\phi}_S$ and $\vec{\phi}_M$ are the centre of masses of S and M, respectively.

Coefficients are normalized using the Sinkhorn method, which can be used to iteratively normalize rows and columns of C (see [8] or [9]). First, with $i \in [1, N_M]$ and $j \in [1, N_S + 1]$:

$$c_{i,j} = \frac{c_{i,j}}{\sum_{b=1}^{N_S+1} c_{i,b}} \tag{16}$$

and then second with $i \in [1, N_M + 1]$ and $j \in [1, N_S]$:

$$c_{i,j} = \frac{c_{i,j}}{\sum_{a=1}^{N_M+1} c_{a,j}}. \tag{17}$$

Correspondence is implemented during the TPS solution phase by summing the weighted contribution from every stationary point when constructing Y. The first N_M rows are changed from simply $\vec{V}_i$ to

$$y'_i = \sum_{j=1}^{N_S} c_{i,j} \vec{V}_j. \tag{18}$$

Note that the normalization of M is required so that the weights used in the TPS solution accurately scale the warp components. The TPS-RPM algorithm described by [5] includes homogeneous coordinates that protect against un-normalized M, but lack of convergence, with or without normalization, will result in inaccurate warping.

The TPS-RPM algorithm is implemented in `DICOMautomaton`. It supports both λ and ζ regularization parameters. `DICOMautomaton` will also halt the Sinkhorn procedure automatically when the desired tolerance level is reached.

Double-sided outlier handling

In 2011 Jinzhong Yang demonstrated that the TPS-RPM algorithm handles outliers in the moving and stationary point sets in different ways [6]. The basic TPS-RPM algorithm requires a small augmentation to achieve more balanced outlier handling in this way, which effectively amounts to a per-point regularization process. It adjusts the E_{TPS} and requires further modification of Y. For more details see [6].

This addendum is implemented in `DICOMautomaton`. It has been translated from Yang's version (which makes use of the QR decomposition described by Wahba and references therein) to the style of Bookstein so that pseudo-inverses and the $LDL^{\intercal}$ decomposition can be used.

Forced correspondence

A subset of points in either point cloud that are known to correspond can be handled by adjusting M immediately prior to the Sinkhorn method.

`DICOMautomaton` implements this functionality.

Known outliers

Points in either cloud can be explicitly marked as outliers and will be disregarded during the Sinkhorn technique. As with forced correspondence, M is adjusted immediately prior to the Sinkhorn method.

`DICOMautomaton` implements this functionality.

Forbidding outliers

Points in either cloud can be *forbidden* from being classified as outliers by adjusting the 'gutter' row and column of M (i.e., the bottom row and right-most column). Both the whole moving and whole stationary point clouds can be handled this way, however the point set with the largest number of points *cannot* be handled this way since the Sinkhorn method will (generally) fail to converge.

`DICOMautomaton` implements this functionality. It will also detect when the Sinkhorn procedure stalls and abort the calculation.

Limitations of TPS-RPM and troubleshooting

The TPS-RPM algorithm combines multiple algorithms (i.e., TPS, softassign, deterministic annealing) and the interaction between these algorithms is complicated.

While TPS-RPM is designed to gracefully handle outliers, it can still suffer from outlier biases and noise.

Failure to achieve an appropriate registration are often caused by inappropriate settings or data. However, it can be hard to guess what inappropriate means. Here are some troubleshooting tips:

- Configure the start and end T so that the system has enough freedom to move.
  - If the point cloud centroids are not (roughly) aligned, and they should be, increase the starting T.
  - If multiple points collapse in the registration, then overfitting is likely occuring, so increase the end T.
  - Consider multi-stage registration; use a simpler rigid registration for rough alignment, and then reduce the starting T to penalize affine-like deformations.
- Start with small regularization parameters.
  - The scale of λ and ζ are difficult to assess. Starting very small (or even at zero) will often result in reduced algorithmic complexity and thus fewer instances where numeric imprecision can creep in.
- Observe how tweaking parameters impacts the registration.
  - It can be hard to determine if increasing or decreasing parameters will improve or worsen a registration. Trying many and visualizing how the deformation changes will help.

- Force correspondence when known.
    - If no correspondences are known, it may be worthwhile to add points to, for example, fix a boundary.

It is also worthwhile to remember that TPS-RPM is a *point cloud* registration technique. If the points are connected in some way (e.g., as a mesh) the TPS-RPM algorithm is not aware of the connections. Point clouds can be pinched, folded, or twisted in ways that are nonsensical when connections are considered. The TPS-RPM algorithm *can* be augmented to account for domain-specific connections, but not in a completely generic way.

Applications of TPS and the TPS-RPM algorithm

Thin plate splines have practical value in a number of domains:

- Identifying coordinate system warps using a small number of features (i.e., typically derived from images).
    - Fiducial marker mapping.
- Identifying coordinate system warps using more complex data sets.
    - Grids.
    - Surface meshes.
    - Embedded shapes (e.g., planar contours).
- Interpolation of complex shapes.
    - Interpolating between two geometric shapes (e.g., planar contours).
- Applying physically-sensible deformations to structures.
    - Point clouds.
    - Surface meshes.
    - Volumetric meshes.
- Quantifying distortion.
    - Primarily as a means of identifying the correspondence; quantifying distortion likely requires more complex models or even simulations.
- Pose estimation.

References

[1] Bookstein FL. Principal warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on Pattern Analysis and Machine Intelligence 1989;11:567–85.

[2] Wahba G. Spline models for observational data. vol. 59. Siam; 1990.

[3] Eberly D. Thin plate splines (2019 update). Geometric Tools Inc 2019;1996:9.

[4] Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical recipes 3rd edition: The art of scientific computing. Cambridge university press; 2007.

[5] Chui H, Rangarajan A. A new point matching algorithm for non-rigid registration. Computer Vision and Image Understanding 2003;89:114–41.

[6] Yang J. The thin plate spline robust point matching (tps-rpm) algorithm: A revisit. Pattern Recognition Letters 2011;32:910–8.

[7] Lira M, Iyer R, Trindade AA, Howle V. QR versus cholesky: A probabalistic analysis. Matrix 2016;1:3–4.

[8] Chui H. Non-rigid point matching: Algorithms, extensions and applications 2001.

[9] Gold S, Rangarajan A. Softassign versus softmax: Benchmarks in combinatorial optimization. In:. Advances in neural information processing systems, 1996, pp. 626–32.