NRSfM Tutorial

Paul Gafton

1 Introduction

The scope of this tutorial is to explore the NRSfM method presented in the lecture with a more hands-on approach, which finishes with an implementation. Following the same pattern as the lecture, the next section will present the statistics based NRSfM approach introduced by Bregler et al. [3] and the following section, the physically based NRSfM approach proposed by Parashar et al. [1].

2 Statistics Based NRSfM

At the core of all statistics based NRSfM lie the observation that the movement of the points from a deformable surface are highly correlated. Therefore, statistics based NRSfM make the assumption that all configurations of these points lie in low dimensional space and therefore can be expressed as a linear combination of a relatively small basis.

Therefore, knowing the cardinality of the basis K, we can assume that we can write any 3D configuration S with P points as:

$$S = \sum_{i=1}^{K} l_i S_i, \qquad l_i \in \mathbb{R}, S_i \in \mathbb{R}^{3\times P} \tag{1}$$

The camera model assumed by [3] is the Orthographic Projection:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2}$$

, where $(uv)^T$ is the observed 2D point on an image and $(XYZ)^T$ is the corresponding point in 3D.

Using this camera model, a 3D configuration is projected onto the 2D image plane by:

$$\begin{pmatrix} u_1 & u_2 & \dots & u_P \\ v_1 & v_2 & \dots & v_P \end{pmatrix} = R\sum_{i=1}^{K} l_i S_i + T = \begin{pmatrix} l_1 R & l_2 R & \dots & l_K R \end{pmatrix} \begin{pmatrix} S_1 \\ S_2 \\ \dots \\ S_K \end{pmatrix} \tag{3}$$

, where $R = \begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \end{pmatrix}$ is a rotation and projection matrix and T is a translation matrix. T can be ignored if we assume that all shapes are centered at origin.

The above theory was developed on a single frame. We can add all points in a measurement or tracking matrix W and factorize it using the result above:

$$W = \begin{pmatrix} u_1^{(1)} & u_2^{(1)} & \dots & u_P^{(1)} \\ v_1^{(1)} & v_2^{(1)} & \dots & v_P^{(1)} \\ u_1^{(2)} & u_2^{(2)} & \dots & u_P^{(2)} \\ v_1^{(2)} & v_2^{(2)} & \dots & v_P^{(2)} \\ & & \dots & \\ u_1^{(N)} & u_2^{(N)} & \dots & u_P^{(N)} \\ v_1^{(N)} & v_2^{(N)} & \dots & v_P^{(N)} \end{pmatrix} = \begin{pmatrix} l_1 R^{(1)} & l_2 R^{(1)} & \dots & l_K R^{(1)} \\ l_1 R^{(2)} & l_2 R^{(2)} & \dots & l_K R^{(2)} \\ & & \dots & \\ l_1 R^{(N)} & l_2 R^{(N)} & \dots & l_K R^{(N)} \end{pmatrix} \begin{pmatrix} S_1 \\ S_2 \\ \dots \\ S_n \end{pmatrix} = Q \cdot B \tag{4}$$

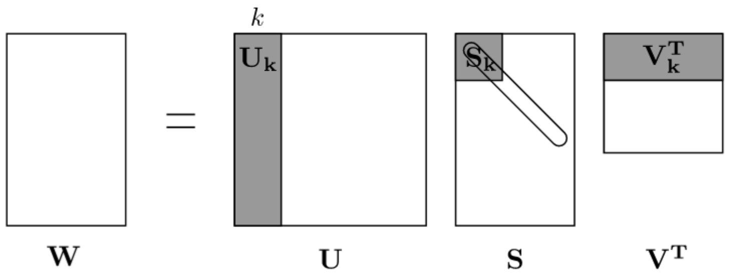

Figure 1: Low rank approximation

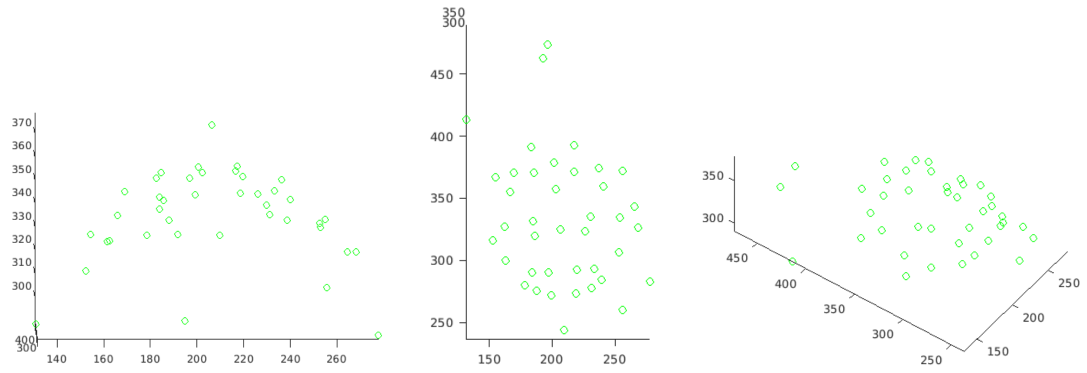

Figure 2: The first 3D shape in the Faces dataset

The B matrix obtained above is the shape basis, and the matrix Q contains all the information regarding the weighting of the basis, rotation and projection required for obtaining any given 2D image.

In order to find Q and B given the tracking matrix W, Bregler et la [3] uses a low rank approximation computed from SVD. More precisely, after doing SVD, keep only K principal vector and principal values and use the resulting matrix as an approximation. This is illustrated in Figure 1.

We demonstrate this by using the Faces dataset, which can be found here. The dataset contains 316 three dimensional configuration of a set of 40 points, which represent the points of a face tracked over time. Figure 2 shows one of these configurations from multiple views.

Note that all the shapes come in a large matrix `P3_gt`, which starts with all the X coordinates of all points, then the Y coordinates and then the Z coordinates. The code listing 1 shows how to load the dataset and do an ortographic projection to prepare the data for the NRSfM algorithm.

```
% Load the matrix P3_gt containing the ground truth data:
% P3_gt([t t+N t+2*N],:) contains the 3D coordinates of the P points at time t
% (N is the number of frames, P is the number of points)
load('face.mat');
[N, P] = size(P3_gt); N = N/3;

% 2D motion resulting from orthographic projection
% (input to the non-rigid sfm algorithm)
p2_obs = P3_gt(1:2*N, :);
```

Listing 1: Dataset Loading

Figure 2 was generated by using a figure generation similar to the one described in code listing 2.

```
figure
hold on
plot3(P3_gt(1,:), P3_gt(N+1,:), P3_gt(2*N+1,:), 'go')
axis equal
```

Listing 2: Plot 3D data

The matrix decomposition $W = Q \cdot B$ is described in code listing 3. We start by setting K, the size of the basis, which is a parameter to the algorithm. We form then W by rearanging the elements of the matrix `p2_obs` and we centralize them to remove translations. Finally, we do a singular value decomposition of W, select K principal values and principal vectors, and then obtain the matrices Q and B.

```matlab
% Choose K, the size of the shape basis
K = 2;
K = 3 * K;

% Form the measurement matrix
X = p2_obs;
A = X(1:N,:);
B = X(N+1:2*N,:);

W = reshape([A(:), B(:)]',2*N, []);

% Centralize
wm = mean(W,2);
W = W - wm*ones(1,size(W,2));

% Decompose the measurement matrix
[U, D, V] = svd(W);

U = U(:, 1:K);
D = D(1:K, 1:K);
V = V(:, 1:K);

Q = U * sqrt(D);
B = sqrt(D) * V';
```

Listing 3: Decomposition of W

As discussed before, the matrix Q combines the rotations and the basis weights. Looking closer at q, a row of Q, we notice that:

$$q = \begin{pmatrix} l_1R & l_2R & \dots & l_KR \end{pmatrix} = \begin{pmatrix} l_1r_1 & l_1r_2 & l_1r_3 & l_2r_1 & l_2r_2 & l_2r_3 \dots l_Kr_1 l_Kr_2 l_Kr_3 \\ l_1r_4 & l_1r_5 & l_1r_6 & l_2r_4 & l_2r_5 & l_2r_6 \dots l_Kr_4 l_Kr_5 l_Kr_6 \end{pmatrix} \tag{5}$$

We can rearrange these elements into:

$$\overline{q} = \begin{pmatrix} l_1r_1 & l_1r_2 & l_1r_3 & l_1r_4 & l_1r_5 & l_1r_6 \\ l_2r_1 & l_2r_2 & l_2r_3 & l_2r_4 & l_2r_5 & l_2r_6 \\ & & \dots & & & \\ l_Kr_1 & l_Kr_2 & l_Kr_3 & l_Kr_4 & l_Kr_5 & l_Kr_6 \end{pmatrix} = \begin{pmatrix} l_1 \\ l_2 \\ \dots \\ l_K \end{pmatrix} \begin{pmatrix} r_1 & r_2 & r_3 & r_4 & r_5 & r_6 \end{pmatrix} \tag{6}$$

As you can notice, $\overline{q}$ is of rank 1 and can be factorized using the same SVD technique into a rotation and a weights vector. This decomposition is illustrated in the code listing 4 for the first frame.

```matlab
% Try frame 1
q = Q(1:2,:);

% Rearrange q into q bar
qb = [q(1,1), q(1,2), q(1,3), q(2,1), q(2,2), q(2,3);
      q(1,4), q(1,5), q(1,6), q(2,4), q(2,5), q(2,6)];


% Decompose q bar
[u, d, v] = svd(qb);

% Get the first principal vector
u = u(:, 1);
d = d(1,1);
v = v(1,:);

% Get the weights and the rotation vectors
w = u * sqrt(d);
r = sqrt(d) * v';
```

Listing 4: Decomposition of Q

After the decomposition of Q, we can extract the bases and interpolate them, as shown in the code listing 5.

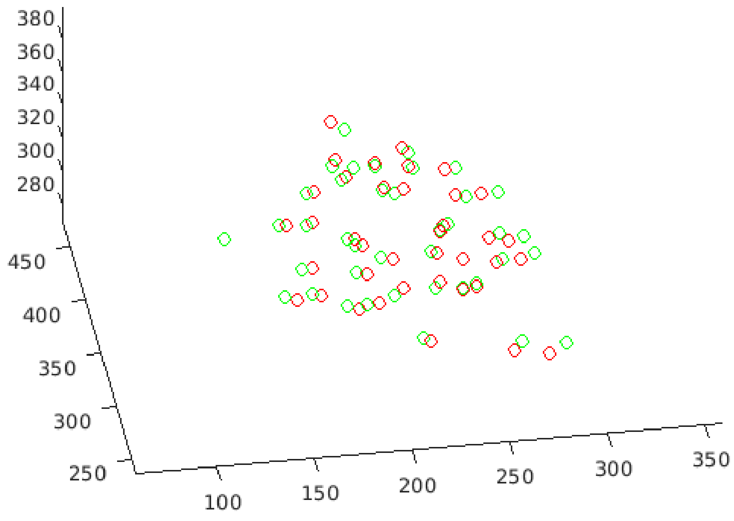

Figure 3: Comparison of the reconstruction, in red, with the ground truth, in green

```
% Extract the two bases
B1 = B(1:3,:);
B2 = B(4:6,:);

% Interpolate the estimated shape
est = w(1) * B1 + w(2) * B(2);
```

Listing 5: Decomposition of Q

We can now visualize the resulting reconstruction and compare it with the dataset. The code for the visualization is shown in listing 6 and the results in figure 3.

```
% Align reconstruction with ground truth
[~, est, ~] = absor(est, [P3_gt(1,:); P3_gt(T+1,:); P3_gt(2*T+1,:)], 'doScale', true);

% Draw both reconstruction and groudtruth on the same figure
figure
hold on
plot3(est(1,:), est(2,:), est(3,:), 'go');
plot3(P3_gt(1,:), P3_gt(T+1,:), P3_gt(2*T+1,:), 'ro');
axis equal
```

Listing 6: Visualisation of results

3 Physically based NRSfM

We shall now explore a physically based method for NRSfM, proposed by [1], which models isometric deformation using differential constraints.

For the purpose of exploring our implementation, we propose the Kinect dataset, available here. Some sample images are shown in Figure 4. The dataset includes the 2d keypoint correspondences of about 1500 across all 191 frames, as well as 3D groundtruth. Our algorithm can be ran from the `example_infp.m` script.

Remember the reconstruction scenario discussed in the lecture, shown in Figure 5, where we observe the projections of two manifolds and we are interested in recovering the image embedding functions ϕ_1 and ϕ_2.

We start by assuming that we can model the image registration using a continuous function $\eta_{ij} : \mathbb{R} \to \mathbb{R}$. This function, which we sometimes call image warp, is in practice computed using a Bicubic B-Spline interpolation. Our implementation uses the BBS toolbox available here. It requires the compilation of some Matlab MEX modules (a copy of the toolbox is included with the project). Code listing 7 demonstrates the use of this toolbox.

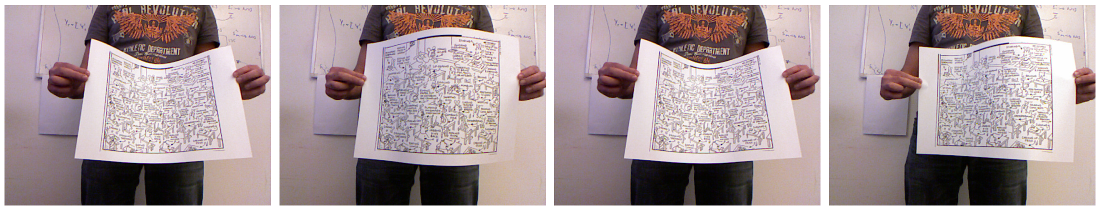

Figure 4: The Kinect dataset

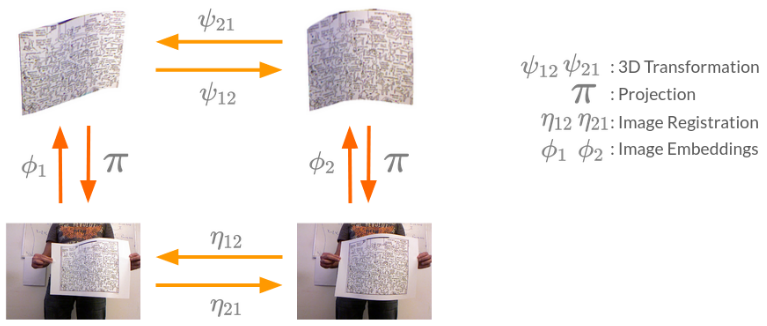
ψ_{21}
ψ_{12}
ϕ_1
π
ϕ_2
π
η_{12}
η_{21}
ψ_{12} ψ_{21} : 3D Transformation
π : Projection
η_{12} η_{21}: Image Registration
ϕ_1 ϕ_2 : Image Embeddings

Figure 5: Reconstruction Setting

```
1      % Parameters
2      er = 1e-5;
3      t= 1e-3;    % Interpolation domain boundary
4      nC = 40;    % Number of Control Points
5
6      % Determine domain bounds
7      umin = min(q_n{i}(1,:))-t; umax = max(q_n{i}(1,:))+t;
8      vmin = min(q_n{i}(2,:))-t; vmax = max(q_n{i}(2,:))+t;
9
10     % Create BBS structure (just stores the parameters)
11     bbs = bbs_create(umin, umax, nC, vmin, vmax, nC, 3);
12
13     % Compute colocation and bending matrices
14     coloc = bbs_coloc(bbs, q_n{i}(1,:), q_n{i}(2,:));
15     lambdas = er*ones(nC-3, nC-3);
16     bending = bbs_bending(bbs, lambdas);
17
18     % Find the control points
19     cpts = (coloc'*coloc + bending) \ (coloc'*P2_n{i}');
20     ctrlpts = cpts';
21
22     % Compute the first order derivatives of the warp at each point
23     dqu = bbs_eval(bbs, ctrlpts, q_n{i}(1,:)',q_n{i}(2,:)',1,0);
24     dqv = bbs_eval(bbs, ctrlpts, q_n{i}(1,:)',q_n{i}(2,:)',0,1);
```

Listing 7: BBS Example

Lines 6-20 of the code listing 7 represent the standard way a BBS interpolation is created. Note that on lines 7 and 8 we compute the size of the interpolation domain based on the value of `q_n{i}` and the colocation matrix is computed for the same variable. Then, we compute the control points using `P2_n{i}`. This models the warp between `q_n{i}` and `P2_n{i}`. The function `bbs_eval` evaluates this warp at the points of `q_n{i}` (is just a coincidence that this is also the starting point of the interpolation - the warp can be evaluated at any points as long as they are in the domain). The last two parameters to `bbs_eval` represent the order of the derivative in the two directions. For example, passing $(0, 0)$ evaluates the interpolation, $(1, 0)$ is the first derivative with respect to the first coordinate, $(1, 1)$ represents the first derivative with respect to both coordinates, $(0, 2)$ represents the second order derivative in the second coordinate.

The function `create_dataset` takes the keypoint correspondences and computes the registrations between the first frame (which shall be named reference frame) and each of the other ones. It interpolates the keypoints in a regular grid and computes the first and second order derivatives (as Jacobian and Hessian matrices) at each point of the grid which will be needed further in the reconstruction algorithm. For example, the variable `J21a` represents the first element of the warp jacobian when going from the second image to the first.

The lines from code listing 8 run a refinement on the warps, which imposes the Schwarzian constraint [2]. This step will be relevant for the project.

```
    % COMMENT THESE 2 TO DISABLE SCHWARPS
    ctrlpts = optimPanalSchwarz(bbs,ctrlpts,q2',q1',[xv(:),yv(:)],par(i));
    ctrlpts = ctrlpts';
```

Listing 8: Running Scnwarzian Optimization

Now that we established the modelling of the image registration functions η_{12} and η_{21}, we can turn our attention towards the deformation model and the resulting reconstruction equations.

We assume that the projection is a perspective projection function given by:

$$\pi : \mathbb{R}^3 \to \mathbb{R}^2, \qquad \pi \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{1}{z} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} \tag{7}$$

Our goal is to estimate all the functions ϕ_i which are the inverses of π at different configurations of the manifold (note that, while π is unique, ϕ_i is different for each frame). ϕ_i is therefore a function of the form:

$$\phi_i : \mathbb{R}^2 \to \mathbb{R}^3, \qquad \phi_i \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{1}{\beta(u,v)} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{8}$$

, where β is the inverse depth function - a way of parametrizing the function ϕ_i. We also introduce two variables, $k_1 = \frac{\beta_u}{\beta}$ and $k_2 = \frac{\beta_v}{\beta}$, with $\beta_u = \frac{\partial \beta}{\partial u}$ and $\beta_v = \frac{\partial \beta}{\partial v}$.

The Jacobian of ϕ_i, parametrized by k_1 and k_2, is given by:

$$J_{\phi_i} = \frac{1}{\beta} \begin{pmatrix} 1 - uk_1 & -uk_2 \\ -vk_1 & 1 - vk_2 \\ -k1 & -k_2 \end{pmatrix} \tag{9}$$

In order to constrain deformation to isometric transformations only, we will use the Metric Tensor, a mathematical object which captures the notions of distance, angle and area, and can be used to model the way they change during a transformation. The metric tensor can be computed only from the tangent plane at a point, which is given by the Jacobian. Therefore, the metric tensor of an embedding ϕ is given by:

$$g = J_\phi^T J_\phi \tag{10}$$

The key observation of this reconstruction method is the following relation:

$$\phi_2 = \psi_{12} \circ \phi_1 \circ \eta_{21} \tag{11}$$

Since the functions above are equal, we can use them to express quantities such as the metric tensor of ϕ_2 in terms of the metric tensor of ϕ_1. Therefore, we have that:

$$J_{\phi_2}^T J_{\phi_2} = J_{\eta_{21}}^T J_{\phi_1}^T J_{\psi_{12}}^T J_{\psi_{12}} J_{\phi_1} J_{\eta_{21}} \tag{12}$$

But we assumed that the manifold transform isometrically. Then, we have that $J_{\psi_{12}}^T J_{\psi_{12}} = I_{3\times 3}$. Therefore, the relation above becomes:

$$J_{\phi_2}^T J_{\phi_2} = J_{\eta_{21}}^T J_{\phi_1}^T J_{\phi_1} J_{\eta_{21}} \tag{13}$$

Christoffel Symbols Γ_{mn}^p are a second order differential quantity which can be used to capture properties such as surface curvature. The Christoffel Symbols of the image embedding ϕ at the point $x \in \mathbb{R}^2$ are given by:

$$\Gamma_{mn}^1[\phi(x)] = \begin{pmatrix} -2k_1 & -k_2 \\ -k_2 & 0 \end{pmatrix} \qquad \Gamma_{mn}^2[\phi(x)] = \begin{pmatrix} 0 & -k_1 \\ -k_1 & -2k_2 \end{pmatrix} \tag{14}$$

, where $\beta_u = \frac{\partial \beta}{\partial u}$ and $\beta_v = \frac{\partial \beta}{\partial v}$.

The Christoffel Symbols are related to the metric tensor and its derivatives by the following relation:

$$\Gamma^p_{mn}[\phi_i] = \frac{1}{2} g^{pl}[\phi_i](g_{lm,n}[\phi_i] + g_{ln,m}[\phi_i] - g_{mn,l}[\phi_i]) \tag{15}$$

, where g^{pl} is the inverse of the metric tensor and $g_{ij,k}$ is the derivative of the metric tensor with respect to k. The change of coordinates of the Christoffel symbol can be written in terms of $k_1 = \frac{\partial \beta_u}{\beta}$ and $k_2 = \frac{\partial \beta_v}{\beta}$ as:

$$\begin{pmatrix} \overline{k_1} \\ \overline{k_2} \end{pmatrix} = J^T_{\eta_{21}} \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} J^{-1}_{\eta_{21}} \frac{\partial^2 \eta_{21}}{\partial \overline{u} \partial \overline{v}} \tag{16}$$

, where the variables overlined indicate a different frame than the variables without overline.

As shown in [1], the deformation constraints imposed by the metric tensor and the Christoffel symbol change of coordinates are enough to construct a closed form system of two bicubic bivariate polynomials for each pair of frames. The polynomials are parametrized by $k_1 = \frac{\partial \beta_u}{\beta}$ and $k_2 = \frac{\partial \beta_v}{\beta}$, where β is the inverse depth function and $\partial \beta_u$ and $\partial \beta_v$ are its first order derivatives.

The coefficients of these polynomials are computed in the function `create_polynomial_coefficients_non_square`. The function `create_polynomial_coefficients` computes in addition the coefficients for the squared polynomials (which become polynomials of degree 6 in two variables). The coefficients of the squared polynomials might be useful for certain approaches to solving.

The function `solve_polynomial` solves the squared polynomials for k_1, k_2 using a Matlab toolbox named Gloptipoly. The first task of the project is about replacing this solver with one based on an explicit non-linear least squares cost.

The roots of the polynomials, k_1 and k_2 can then be used to compute the normals at each point. This is shown in the code listing 9.

```matlab
% recover first order derivatives on rest of the surfaces
% res(:,1) is k1 on the reference frame
% k1 on the rest of the frames are computed with a change of coodinates given by registration
k1_all = [res(:,1)';a.*repmat(res(:,1)',length(idx)-1,1) + b.*repmat(res(:,2)',length(idx)-1,1) +
    t1];
k2_all = [res(:,2)';c.*repmat(res(:,1)',length(idx)-1,1) + d.*repmat(res(:,2)',length(idx)-1,1) +
    t2];

% I1u are the u coordinates of the points in the reference (first) frame (one row)
% I2u are the u coordinates of the points in the other frames (n-1 rows)
u_all = [I1u;I2u];
v_all = [I1v;I2v];

% find normals on all surfaces N = [N1;N2;N3] based on k1,k2
N1 = k1_all; N2 = k2_all; N3 = 1-u_all.*k1_all-v_all.*k2_all;
n = sqrt(N1.^2+N2.^2+N3.^2);
N1 = N1./n ; N2 = N2./n; N3 = N3./n;

N = [N1(:),N2(:),N3(:)]';
N_res = reshape(N(:),3*length(idx),length(u_all));
```

Listing 9: Extracting Normals from the Solutions of the Reconstruction Equations

The normals are then integrated to obtain the final 3D points in the function `calculate_depth` as shown in listing 10.

```matlab
P_grid=calculate_depth(N_res,u_all,v_all,1e0);
```

Listing 10: Normal Integration

Figure 6 show the reconstruction on a few frames from the Kinect dataset, compared with the ground truth (shown in green).

References

[1] Parashar, S., Pizarro, D and Bartoli, A.: "Isometric Non-Rigid Shape-from-Motion with Riemannian Geometry Solved in Linear Time," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 10, pp. 2442-2454, 1 Oct. 2018, doi: 10.1109/TPAMI.2017.2760301.

[2] Pizarro, D., Khan, R. and Bartoli, A. Schwarps: Locally Projective Image Warps Based on 2D Schwarzian Derivatives. Int J Comput Vis 119, 93–109 (2016). https://doi.org/10.1007/s11263-016-0882-9

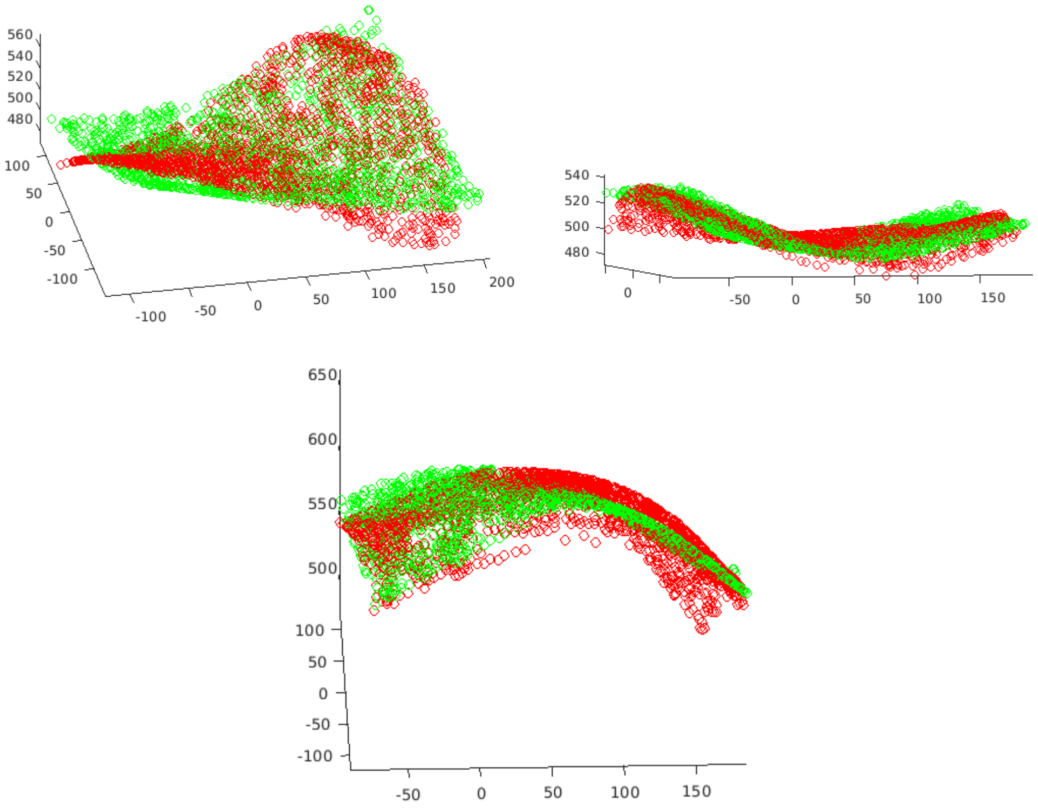

Figure 6: Results of the presented physically based NRSfM method on the Kinect dataset

[3] Bregler, C and Hertzmann, A and Biermann, H: Recovering non-rigid 3D shape from image streams. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2000), doi: 0.1109/CVPR.2000.854941