Dense Depth Estimation of a Complex Dynamic Scene without Explicit 3D Motion Estimation: Supplementary Material

Suryansh Kumar[1] Ram Srivatsav Ghorakavi[3] Yuchao Dai[4] Hongdong Li[3], Luc Van Gool[1,2]
[1]ETH Zurich, [2]KU Leuven, [3]Australian National University, [4]Northwestern Polytechnical University
suryansh.kumar@vision.ee.ethz.ch

Abstract

In this supplementary material, we first discuss the potential limitation of our algorithm. Secondly, we provide the MATLAB simulation code on two synthetic examples. These examples explains and show the utility of as rigid as possible constraint to recover the 3D points in a dynamic scene setting without estimating motion. Additionally, we provide few statistical experiment results about the behavior of our algorithm under noisy initialization and different $d_{i\sigma}$ values (if the second constraint is used with Φ^{arap}). Although some of the evaluations are also provided in the main paper, we provide it again with numerical examples for completeness and easy understanding.

1. Limitation and Discussion

Even though our method works well for diverse dynamic scenes, there are still a few challenges associated with the formulation. Firstly, very noisy depth initialization for the reference frame can provide unsettling results. Secondly, our method is challenged by the instant arrival or removal of the dynamic subjects in the scene, and in such cases, it may need re-initialization of the depth. Lastly, well-known limitations such as occlusion and temporal consistency, especially around the regions close to the boundary of the images can also affect the accuracy of our algorithm.

Discussion: In defense, we would like to state that motion based methods to structure from motion is also prone to noisy data [2, 1]. Algorithms like motion averaging [3], M-estimators and random sampling [13] are quite often used to rectify the solution.

(a) *What do we gain or lose by our approach?*

Estimating all kinds of conceivable motion in a complex dynamic scene from images is a challenging task, in that respect, our method provides an alternative way to achieve per pixel depth without estimating any 3D motion. However, in achieving this we are allowing the gauge freedom between the frames (temporal relations in 3D over frames).

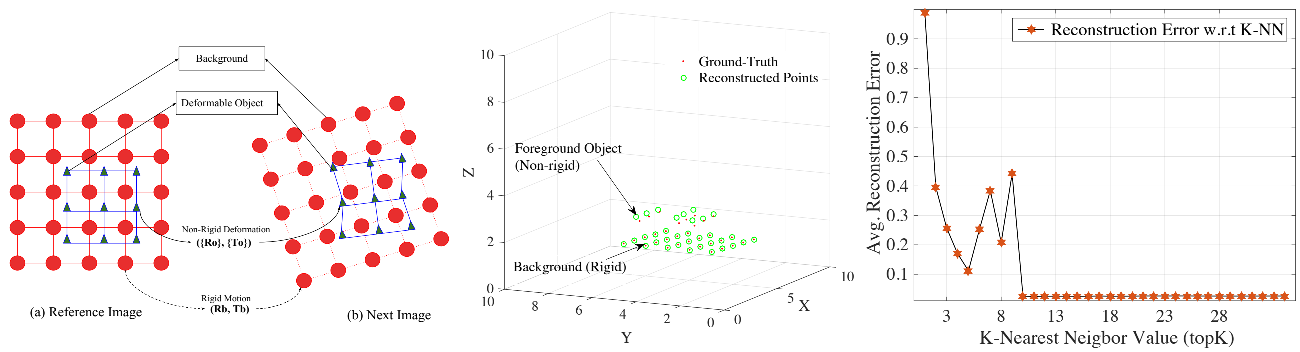
(b) *Depth results has some blocky effects?* Few blocky artifacts can be observed in the depth results due to discrete piece-wise planar decomposition of the scene. Although we smooth the solution using TRW-S [4], the number of particles for each move is taken as 10 to reduce the convergence time, hence, some artifacts can be observed.

(c) *Limitations of as rigid as possible assumption?*

In a general dynamic scene, its quite intuitive to assume that the changes in the scene between successive frames is gradual. Therefore, to have an assumption that the scene undergoes as rigid as possible transformation in consecutive frames holds in general. However, there are situations were such an assumption may not hold and the solution to depth estimation problem under ARAP regularisation can provide unreasonable results. Couple of the such examples are: (a) When the displacement of objects between frames are large. (b) When the subject is shrinking or expanding over frames such as balloons, rubber-sheet etc.

(d) Why two staged optimization to solve the problem?

If ARAP optimization function is defined on per pixel level then the second step of our algorithm can be avoided. Nevertheless, doing so will ramp up the convergence time which is tough to realise on commodity desktop machine. Therefore, to realize the results in a reasonable computation time, we first perform ARAP optimization at superpixel level and then smooth the solution using message passing algorithm [4].



(a) (b) (c)

Figure 1: (a) Experimental setup for the first experiment (b) 3D reconstruction for the next frame after optimization (c) The 3D reconstruction error variations against the number of nearest neighbor in the experiment (topK variable in the code).

2. Synthetic Experiment Code and Explanation

We provide the code showing the utility of as rigid as possible constraint on two synthetic experimental setting of a dynamic scene. In these experiments, the background and the objects are shown in red and blue color respectively. The background undergoes a rigid motion and the object undergoes a non-rigid deformation in the scene. Given the depth of the reference frame and the image correspondences of the feature points, we can estimate the 3D reconstruction for both the foreground and the background in the next frame just by using the ARAP constraint without using any 3D motion parameters.

2.1. Experiment (1)

1. **Scene Setup**: A background and an object in the reference frame. The background undergoes a rigid motion and the single object deforms non-rigidly in the next frame (see Figure 1).
2. **Input**: 2D image feature correspondences, intrinsic camera parameters(K), depth of the points in the reference frame.
3. **Output**: 3D coordinates of the entire scene for the next frame.

(1) firstExample.m Main file.

```
%% Evaluation of concept on sythetic dataset.
% 1. Given the 3D points for the background and the deforming object (foreground) for the reference frame.
% 2. Also, you are provided with camera intrinsic calibration matrix(K), 2D image correspondance between reference frame and next frame
% 3. Situation: The background is undergoing a rigid motion and the object is deforming non-rigidly.
%% Problem: % Get the 3D reconstruction of this dynamic scene for the next time frame without solving for motion.
%% 1. Generate a synthetic dataset for the reference frame
%Create a synthetic situation of the problem.
%generate 3D for the reference frame
%Background coordinate
ref_Xb = [1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5];
ref_Yb = [1, 1, 1, 1, 1; 2, 2, 2, 2, 2; 3, 3, 3, 3, 3; 4, 4, 4, 4, 4; 5, 5, 5, 5, 5];
ref_Zb = 2 * ones(5, 5);
%Object coordinate
ref_Xo = [2.5, 3.5, 4.5; 2.5, 3.5, 4.5; 2.5, 3.5, 4.5];
ref_Yo = [2.5, 2.5, 2.5; 3.5, 3.5, 3.5; 5.0, 5.0, 5.0];
ref_Zo = 3 * ones(3, 3);
%Arrange in the matrix form
ref_Xb = ref_Xb'; ref_Yb = ref_Yb'; ref_Zb = ref_Zb';
```

```
ref_Xo = ref_Xo'; ref_Yo = ref_Yo'; ref_Zo = ref_Zo';
ref_X = [ref_Xb(:)', ref_Xo(:)'];
ref_Y = [ref_Yb(:)', ref_Yo(:)'];
ref_Z = [ref_Zb(:)', ref_Zo(:)'];
%% 2. Generate the synthetic dataset for the next frame
%give some rigid motion to the background
angle = deg2rad(3);
R = [cos(angle), 0, sin(angle); 0, 1, 0; -sin(angle), 0, cos(angle)];
t = [0.2, 0.2, 0.2]';
next_b = R*[ref_Xb(:)'; ref_Yb(:)'; ref_Zb(:)'] + repmat(t, [1, 25]);
next_Xb = next_b(1, :); next_Yb = next_b(2, :); next_Zb = next_b(3, :);
%give some inconsistent changes to the object
next_Xo = [2.6, 3.7, 4.7; 2.8, 3.6, 4.5; 2.5, 3.5, 4.6];
next_Yo = [2.6, 2.7, 2.75; 3.4, 3.45, 3.5; 5.05, 5.10, 5.15];
next_Zo = [2.9, 2.9, 2.9; 2.9, 2.9, 2.9; 2.9, 2.9, 2.9];
%arrange in the matrix form
next_Xo = next_Xo'; next_Yo = next_Yo'; next_Zo = next_Zo';
next_X = [next_Xb, next_Xo(:)'];
next_Y = [next_Yb, next_Yo(:)'];
next_Z = [next_Zb, next_Zo(:)'];
%% 3. Generate synthetic image for the reference frame and the next frame.
%some K matrix
fx = 100; fy = 100; cx = 240; cy = 320;
K = [fx, 0, cx; 0, fy, cy; 0, 0, 1];
%image point for the reference image
ref_img = K*[ref_X;ref_Y; ref_Z];
ref_img = ref_img./repmat(ref_img(3, :), [3, 1]);
%image point for the next image
next_img = K*[next_X; next_Y; next_Z];
next_img = next_img./repmat(next_img(3, :), [3, 1]);
%plot the image points
figure, plot(ref_img(1, :), ref_img(2, :), 'k.'); hold on;
plot(ref_img(1, 26:34), ref_img(2, 26:34), 'ro'); title('Reference Image');
figure, plot(next_img(1, :), next_img(2, :), 'k.'); hold on;
plot(next_img(1, 26:34), next_img(2, 26:34), 'ro'); title('Next Image');
%% 4. Define the neighbors based on the reference image distance
%total number of anchor node.
N = 34; %K-NN to consider
topK = 15; %vary form 1 to N
%get the index of the neighbors
[persuperpixelKNNid, persuperpixelw1k] = givemeKNN(ref_img, N, topK);  %function call 1
%% 5. Use as rigid as possible optimization routine
%(Optional: You may provide explicit lower and upper bound for better convergence of a non-convex problem)
%(For large scale problems such bounds can be handy)
%dvariance = ones(N, 1);
%lb = ref_Z' - dvariance; %lower bound on the variables
%ub = ref_Z' + dvariance; %upper bound on the variables
%general upper and lower bound
lb = zeros(N, 1); ub = []; Aeq = []; Beq = []; A = []; B = []; d0 = ones(N, 1)/N;
%optimization options
%for MATLAB 2017 version uncomment
%options = optimoptions('fmincon', 'Algorithm', 'sqp', 'Display', 'iter-detailed', 'MaxIter', 1000, 'MaxFunctionEvaluations', 300000, 'PlotFcns', @optimplotfval);
```

```
%for MATLAB 2015 version
options = optimoptions('fmincon', 'Algorithm', 'sqp', 'Display', 'iter-detailed', 'MaxIter', 1000, 'MaxFunEvals', 300000,
'PlotFcns', @optimplotfval);
ref3D = [ref_X; ref_Y; ref_Z];
next3D = inv(K)*next_img;
disp('Optimizing....');
[depthVal, cost] = fmincon(@(d)objectiveFunctionARAP(d, ref3D, next3D, persuperpixelKNNid, persuperpixelw1k), d0, A,
B, Aeq, Beq, lb, ub,[], options); %function call 2
%% 6. Get the output depth and estimate the 3D.
output3D = zeros(3, N);
for i = 1:N
    output3D(:, i) = depthVal(i)*next3D(:, i);
end
%% 7. Plot the result
figure,
plot3(next_X(:), next_Y(:), next_Z(:), 'r.'); hold on;
plot3(output3D(1, :), output3D(2, :), output3D(3, :), 'go');
axis([0, 10, 0, 10, 0, 10]); grid on;
title('3D reconstruction for the next frame');
legend('Ground-Truth', 'Reconstructed Points')
%% 8. Perform error estimation (Relative Error)
gt_3D = [next_X(:)'; next_Y(:)'; next_Z(:)'];
es_3D = [output3D(1, :); output3D(2, :); output3D(3, :)];
error = norm(es_3D - gt_3D, 'fro')/norm(gt_3D, 'fro');
fprintf('Relative Error = %f \n', error);
```

(2) givemeKNN.m First function file (K-nearest neighboring index)

```
function [persuperpixelKNNid, persuperpixelw1k] = givemeKNN(ref_img, N, topK)
persuperpixelKNNid = cell(1, N); persuperpixelw1k = cell(1, N); distanceMat = zeros(N, N);
for i = 1:N
   x_ai = ref_img(1:2, i);
   for j = 1:N
      x_ak = ref_img(1:2, j);
      distanceMat(i, j) = sqrt((x_ai(1, 1) - x_ak(1, 1))^2 + (x_ai(2, 1) - x_ak(2, 1))^2);
   end
end
[sortDistance, index] = sort(distanceMat, 2);
betad = 1;
for i = 1:N
   persuperpixelKNNidi.knnid = index(i, 2:topK); %1 id is always the same anchor (distance to itself = 0);
   persuperpixelw1ki.w1k = exp(-betad*sortDistance(i, 2:topK));
end
end
```

(3) objectiveFunctionARAP.m Second function file (As rigid as possible cost function definition).

```
function cost = objectiveFunctionARAP(d, ref3D, next3D, persuperpixelKNNid, persuperpixelw1k)
N = length(persuperpixelKNNid);
cost = 0;
for i = 1:N
   knnid = persuperpixelKNNidi.knnid;
   di = d(i);
   Xi = ref3D(:, i);
   Xip = next3D(:, i);
```

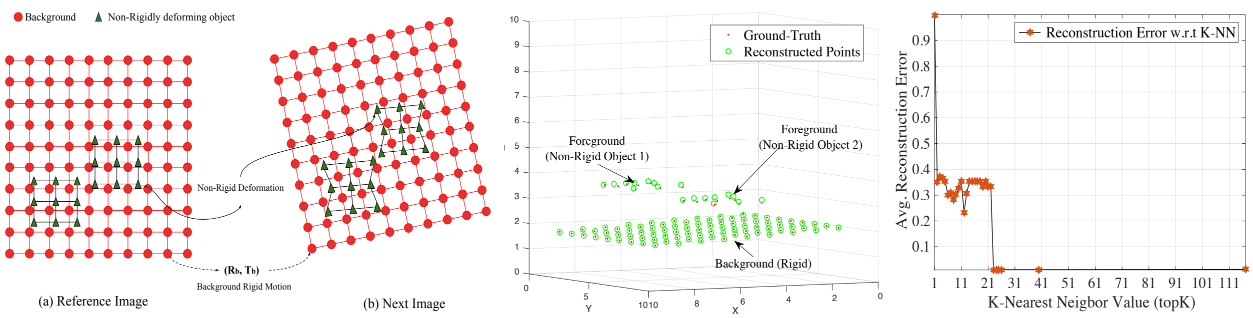
```
    for j = 1:length(knnid)
        dj = d(knnid(1, j));
        Xj = ref3D(:, knnid(1, j));
        Xjp = next3D(:, knnid(1, j));
        cost = cost + abs(norm(Xi-Xj)-norm(di*Xip - dj*Xjp));
    end
end
end
```

2.2. Experiment (2)

1. **Scene Setup**: A background with two objects in the reference frame scene. The background undergoes a rigid motion and both the objects deforms non-rigidly in the next frame (see Figure 2).
2. **Input**: 2D image feature correspondences, intrinsic camera parameters(K), depth of the points in the reference frame.
3. **Output**: 3D coordinates of the entire scene for the next frame.

secondExample.m Main file.

```
% 1. Given the 3D points for the background and the two foreground object for the reference frame.
% 2. Also, you are provided with 2D image correspondance between reference frame and next frame.
% The 3D background is undergoing rigid motion and the two foreground are undergoing non-rigid deformation.
% 3. use ARAP constraint to estimate the 3D output for the next frame.

%% 1. Generate a synthetic dataset for the reference frame

%3D in the reference frame.
ref_Xb = repmat(1 : 10, [10, 1]);
ref_Yb = ones(10, 10). * repmat((1 : 10)', [1, 10]);
ref_Zb = 2 * ones(10, 10);

ref_Xo1 = [2.5, 3.5, 4.5; 2.5, 3.5, 4.5; 2.5, 3.5, 4.5];
ref_Yo1 = [2.5, 2.5, 2.5; 3.5, 3.5, 3.5; 5.0, 5.0, 5.0];
ref_Zo1 = 3 * ones(3, 3);

ref_Xo2 = [7.5, 8.5, 9.5; 7.5, 8.5, 9.5; 7.5, 8.5, 9.5];
ref_Yo2 = [5.5, 5.5, 5.5; 6.5, 6.5, 6.5; 8.0, 8.0, 8.0];
ref_Zo2 = 4 * ones(3, 3);

% figure, plot3(ref_Xb(:), ref_Yb(:), ref_Zb(:), 'r*'); hold on;
% plot3(ref_Xo1(:), ref_Yo1(:), ref_Zo1(:), 'g.'); hold on;
% plot3(ref_Xo2(:), ref_Yo2(:), ref_Zo2(:), 'g.'); hold on;

ref_Xb = ref_Xb'; ref_Yb = ref_Yb'; ref_Zb = ref_Zb';
ref_Xo1 = ref_Xo1'; ref_Yo1 = ref_Yo1'; ref_Zo1 = ref_Zo1';
ref_Xo2 = ref_Xo2'; ref_Yo2 = ref_Yo2'; ref_Zo2 = ref_Zo2';

ref_X = [ref_Xb(:)', ref_Xo1(:)', ref_Xo2(:)'];
ref_Y = [ref_Yb(:)', ref_Yo1(:)', ref_Yo2(:)'];
ref_Z = [ref_Zb(:)', ref_Zo1(:)', ref_Zo2(:)'];
plot3(ref_X(:), ref_Y(:), ref_Z(:), 'ro'); hold on;

%% 2. Generate the synthetic dataset for next frame
angle = deg2rad(3);
R = [cos(angle), 0, sin(angle); 0, 1, 0; -sin(angle), 0, cos(angle)];
```

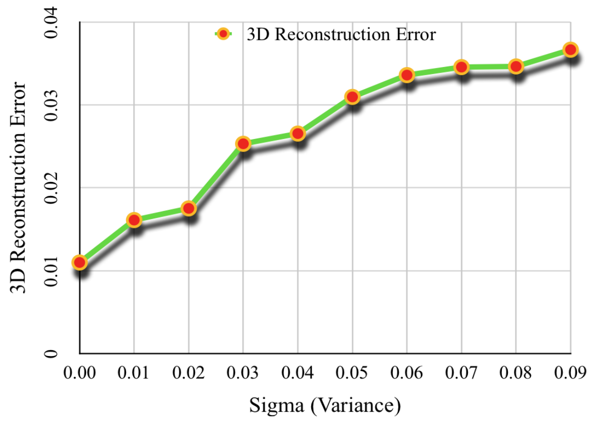


(a) (b) (c)

Figure 2: (a) Experimental setup for the second experiment (b) 3D reconstruction of the points in the next frame after optimization (c) The 3D reconstruction error variations against the number of nearest neighbor in the experiment (topK variable in the code)

```
t = [0.2, 0.2, 0.2]’;
next_b = R*[ref_Xb(:)’; ref_Yb(:)’; ref_Zb(:)’] + repmat(t, [1, 100]);

next_Xb = next_b(1, :);
next_Yb = next_b(2, :);
next_Zb = next_b(3, :);

next_Xo1 = [2.6, 3.7, 4.7; 2.8, 3.6, 4.5; 2.5, 3.5, 4.6];
next_Yo1 = [2.6, 2.7, 2.75; 3.4, 3.45, 3.5; 5.05, 5.10, 5.15];
next_Zo1 = [2.9, 2.9, 2.9; 2.9, 2.9, 2.9; 2.9, 2.9, 2.9];

next_Xo2 = [7.6, 8.7, 9.7; 7.8, 8.6, 9.5; 7.5, 8.5, 9.6];
next_Yo2 = [5.6, 5.7, 5.75; 6.4, 6.45, 6.5; 8.05, 8.10, 8.15];
next_Zo2 = [3.9, 3.9, 3.9; 3.9, 3.9, 3.9; 3.9, 3.9, 3.9];

% figure, hold on;
% plot3(next_Xb(:), next_Yb(:), next_Zb(:), ’ro’); hold on;
% plot3(next_Xo1(:), next_Yo1(:), next_Zo1(:), ’go’); hold on;
% plot3(next_Xo2(:), next_Yo2(:), next_Zo2(:), ’go’); hold on;

next_Xo1 = next_Xo1’; next_Yo1 = next_Yo1’; next_Zo1 = next_Zo1’;
next_Xo2 = next_Xo2’; next_Yo2 = next_Yo2’; next_Zo2 = next_Zo2’;

next_X = [next_Xb, next_Xo1(:)’, next_Xo2(:)’];
next_Y = [next_Yb, next_Yo1(:)’, next_Yo2(:)’];
next_Z = [next_Zb, next_Zo1(:)’, next_Zo2(:)’];
%% 3. generate a synthetic image for the reference frame and next frame.
%some K matrix
fx = 100; fy = 100; cx = 240; cy = 320;
K = [fx, 0, cx; 0, fy, cy; 0, 0, 1];

% image point for the reference image
ref_img = K*[ref_X;ref_Y; ref_Z];
ref_img = ref_img./repmat(ref_img(3, :), [3, 1]);

% image point for the next image
next_img = K*[next_X; next_Y; next_Z];
```

```
next_img = next_img./repmat(next_img(3, :), [3, 1]);

%plot the image points
figure, plot(ref_img(1, :), ref_img(2, :), 'k.'); hold on;
plot(ref_img(1, 101:118), ref_img(2, 101:118), 'ro');

figure, plot(next_img(1, :), next_img(2, :), 'k.'); hold on;
plot(next_img(1, 101:118), next_img(2, 101:118), 'ro');

%% 4. Now define the neighbors based on the reference image distance
N = 118; %total number of anchor node.
topK = 22; %vary form 1 to N
[persuperpixelKNNid, persuperpixelw1k] = givemeKNNforConcept(ref_img, N, topK);

%% 5. Perform ARAP optimization
%dvariance = ones(N, 1);
%lb = ref_Z' - dvariance; % lower bound on the variables, this works
%ub = ref_Z' + dvariance; % upper bound on the variables
lb = zeros(N, 1); %this also works
ub = []; %this also works
Aeq = []; % equality constraint
Beq = [];
A = []; % inequality constraint
B = [];

d0 = ones(N, 1)/N; %variable initialization

%optimization options
options = optimoptions('fmincon', 'Algorithm', 'sqp', 'Display', 'iter-detailed', 'MaxIter', 400, 'MaxFunEvals', 300000,
'PlotFcns', @optimplotfval);
ref3D = [ref_X; ref_Y; ref_Z];
next3D = inv(K)*next_img;

disp('Optim');
[depthVal, cost] = fmincon(@(d)objectiveFunctionConceptARAP(d, ref3D, next3D, persuperpixelKNNid, persuperpixelw1k),
d0, A, B, Aeq, Beq, lb, ub, [], options);

output3D = zeros(3, N);
for i = 1:N
    output3D(:, i) = depthVal(i)*next3D(:, i);
end

figure,
plot3(next_X(:), next_Y(:), next_Z(:), 'r.'); hold on;
plot3(output3D(1, :), output3D(2, :), output3D(3, :), 'go');

 %% error estimation
gt_3D = [next_X(:)'; next_Y(:)'; next_Z(:)'];
es_3D = [output3D(1, :); output3D(2, :); output3D(3, :)];
error = norm(es_3D - gt_3D, 'fro')/norm(gt_3D, 'fro');
fprintf('Relative Error = %f \n', error)
```



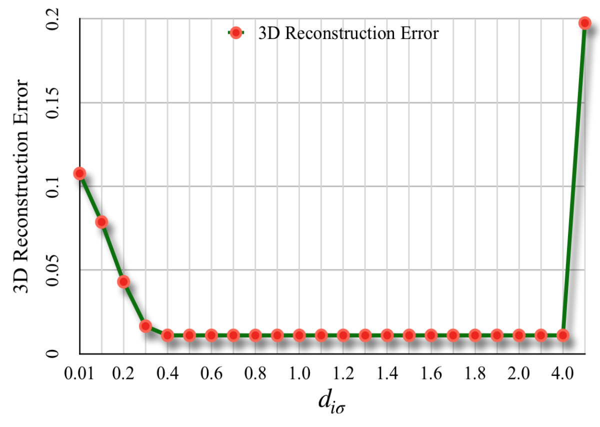


Figure 3: (a) 3D reconstruction results for the next frame with different levels of Gaussian noise in the reference frame coordinate initialization. The curve is generated using the second synthetic experiment with K-NN as 117 (topK = 117) *i.e*. fully connected graph. (b) Variation in the performance with the change in the $d_{i\sigma}$ values for synthetic example 2.

3. Statistical Evaluation

We performed few more experiments to better understand the behavior of the algorithm under different input condition and variable initialization.

(a) Performance of the algorithm under noisy 3D initialization for the reference frame: This experiment is conducted to study the sensitivity of the method to noisy initialization. Fig. (3(a)) show the change in the 3D reconstruction accuracy with the variation in the level of noise from 1% to 9%. The Gaussian noise is introduced using randn() function of MATLAB and the result is documented for example(2.2) after repeating the experiment 10 times and taking its average value. We observe that algorithm can provide unsettling results when the noise becomes very large

(b) Performance of the algorithm under restricted isometry constraint $(d_{i\sigma})$ with Φ^{arap} objective function: While minimizing the as rigid as possible objective function under the $|\tilde{d}_i - d_i| < d_{i\sigma}$ constraint, we restrict the convergence trust region of the optimization. This constraint makes the algorithm works extremely well —both in terms of timing and accuracy, if the prior knowledge about the deformation that the scene may undergo is known a priori. Fig. (3(b)) show the reconstruction accuracy as a function of $d_{i\sigma}$. Clearly, if we have the the approximate knowledge about the scene scene transformation, we can get high accuracy in less computation time. See Fig: (4(b)) which illustrates the quick convergence by using this constraint under proper the values of $d_{i\sigma}$.

(c) Nature of convergence of the proposed as rigid as possible optimization

- *Without restricted isometry constraint*: As rigid as possible minimization Φ^{arap} under the constraint $\tilde{d}_i > 0$ is a good enough constraint to provide acceptable results. However, it may take considerable number of iteration to do so. Fig. (4(a)) show the convergence curve
- *With restricted isometry constraint*: Employing the approximate bound on the deformation that the scene may undergo in the next time instance can help fast convergence with similar accuracy. Fig. (4(b)) show that the same accuracy can be achieved in 60 iteration.

References

[1] David Crandall, Andrew Owens, Noah Snavely, and Dan Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *CVPR 2011*, pages 3001–3008. IEEE, 2011.
[2] Venu Madhav Govindu. Robustness in motion averaging. In *Asian Conference on Computer Vision*, pages 457–466. Springer, 2006.

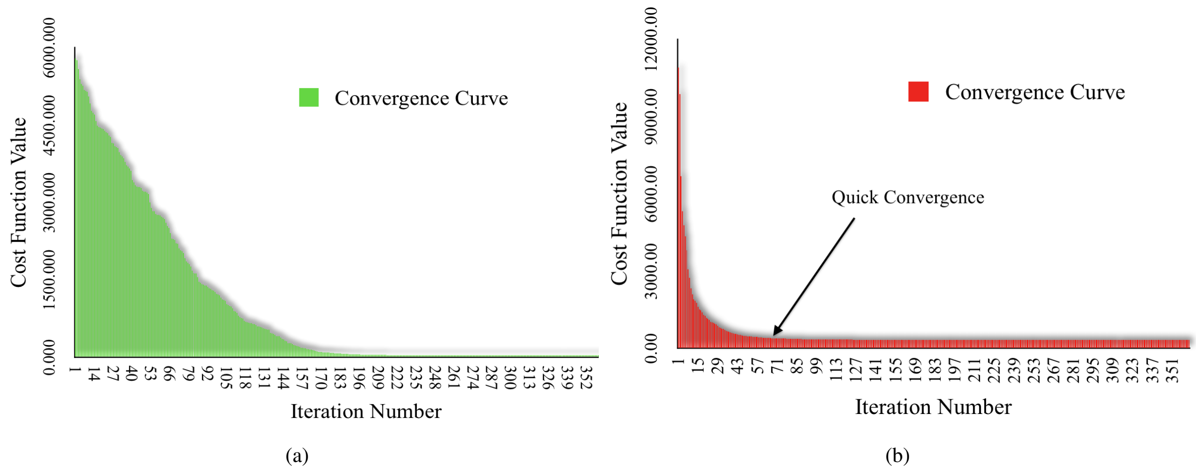


Figure 4: (a) Convergence curve of the cost function using SQP implementation of MATLAB toolbox for the second example. (b) Quick convergence with similar accuracy on the same example can be achieved by using isometry constraint.

[3] Venu Madhav Govindu. Motion averaging in 3d reconstruction problems. In *Riemannian Computing in Computer Vision. To Appear*. Springer, 2015.

[4] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006.

[5] Suryansh Kumar. Jumping manifolds: Geometry aware dense non-rigid structure from motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5346–5355, 2019.

[6] Suryansh Kumar. Non-rigid structure from motion: Prior-free factorization method revisited. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 51–60, 2020.

[7] Suryansh Kumar, Anoop Cherian, Yuchao Dai, and Hongdong Li. Scalable dense non-rigid structure-from-motion: A grassmannian perspective. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[8] Suryansh Kumar, Yuchao Dai, and H.Li. Spatio-temporal union of subspaces for multi-body non-rigid structure-from-motion. In *Pattern Recognition*, volume 71, pages 428–443. Elsevier, May 2017.

[9] Suryansh Kumar, Yuchao Dai, and Hongdong Li. Multi-body non-rigid structure-from-motion. In *International Conference on 3D Vision (3DV)*, pages 148–156. IEEE, 2016.

[10] Suryansh Kumar, Yuchao Dai, and Hongdong Li. Monocular dense 3d reconstruction of a complex dynamic scene from two perspective frames. In *IEEE International Conference on Computer Vision*, pages 4649–4657, Oct 2017.

[11] Suryansh Kumar, Yuchao Dai, and Hongdong Li. Superpixel soup: Monocular dense 3d reconstruction of a complex dynamic scene. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI), 2019*, 2019.

[12] Suryansh Kumar, Ram Srivatsav Ghorakavi, Yuchao Dai, and Hongdong Li. Dense depth estimation of a complex dynamic scene without explicit 3d motion estimation. *arXiv preprint arXiv:1902.03791*, 2019.

[13] Philip HS Torr and David W Murray. The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*, 24(3):271–300, 1997.