

Laboratory 5: Rigid and Non-Rigid Structure from Motion

Jesus Miguel Adrian Matos

July 2, 2024

Abstract

Nothing for now.

1 Inputs:

Given an image in the frame k , calling p as the number of track point and f as the number of frames, then the image is described by a $2 \times p$ matrix I_k , defined

$$I_k = \begin{pmatrix} x_1 & x_2 & \cdots & x_p \\ y_1 & y_2 & \cdots & y_p \end{pmatrix} \quad (1)$$

where (x_k, y_k) are the coordinated of the bidimensional space. As a consequence, the input matrix is expressed

$$Input = \begin{pmatrix} I_1 \\ I_2 \\ \vdots \\ I_f \end{pmatrix} \quad (2)$$

2 Outputs:

We want to obtain a representation in the 3D plane of the points tracked in the 2D images. The resulting matrix is

$$Output = \begin{pmatrix} x_1 & x_2 & \cdots & x_p \\ y_1 & y_2 & \cdots & y_p \\ z_1 & z_2 & \cdots & z_p \end{pmatrix} \quad (3)$$

Where (x, y, z) represents the 3D coordinates in the \mathbb{R}^3 space.

3 First Part (Rigid structure from motion by factorization)

In this task, factorization for a rigid body must be implemented, as shown in the following slide figure 1. So, we should complete the orthogonal factorization function by means of the file **rigidfactorization_ortho.m**. The completed code runs with a provided test data composed by a synthetic sequence with an Ogre face, and the 3D reconstruction error is reported.

More on factorization Orthographic camera

Because \mathbf{R} is a 2×3 matrix and \mathbf{X} is a $3 \times P$ matrix, the rank of \mathbf{W} is 3. If we apply SVD to \mathbf{W} , we will have only **3 non-zero singular values**

However, measurements are normally noisy, and in practice the rank will not be 3. We have to impose it

Applying SVD factorization, we have:

$$\mathbf{W} = \mathbf{U} \mathbf{A} \mathbf{V}^T = [\mathbf{U} \sqrt{\mathbf{A}}] [\sqrt{\mathbf{A}} \mathbf{V}^T] = [\mathbf{U} \sqrt{\mathbf{A}} \mathbf{Q}] [\mathbf{Q}^{-1} \sqrt{\mathbf{A}} \mathbf{V}^T]$$

i.e., $\mathbf{R} = \mathbf{U} \sqrt{\mathbf{A}} \mathbf{Q}$ and $\mathbf{X} = \mathbf{Q}^{-1} \sqrt{\mathbf{A}} \mathbf{V}^T$ (the two factors we look for)

Many solutions can be achieved by modifying \mathbf{Q} . Of course, for all invertible 3×3 \mathbf{Q} matrices

Figure 1: Factoring slide

3.1 Code:

Now, we introduce the code. We will explain each line number, defining Zc as the difference between the input matrix and the average of all the points in each frame, and not as the input matrix. This way, we no longer have to consider translation in the computations.

Therefore, the average matrix would be the following, with dimensions $2f \times p$

$$Mean = \begin{pmatrix} Mean_1 & Mean_1 & \cdots & Mean_1 \\ Mean_2 & Mean_2 & \cdots & Mean_2 \\ \vdots & \vdots & \ddots & \vdots \\ Mean_f & Mean_f & \cdots & Mean_f \end{pmatrix} \quad (4)$$

where

$$Media_k = \begin{pmatrix} \frac{x_1 + x_2 + \cdots + x_p}{p} \\ \frac{y_1 + y_2 + \cdots + y_p}{p} \end{pmatrix} \quad (5)$$

for any k between 1 and f . Therefore from 4 we have:

$$Z_c = \text{Input} - \text{Mean} \quad (6)$$

Having said this, we continue with the application of the code figure 2.

In the first line, we calculate the factorization called Singular Value Decomposition (SVD).

In the second line, we approximate $R = U(:, 1 : 3)D(1 : 3, 1 : 3)^{1/2}$ and $S = D(1 : 3, 1 : 3)^{1/2}V(:, 1 : 3)'$ (where R represents rotation and S represents shape) in such a way that we only take the first three columns of U and, the first three columns of V and, the first three rows and the first three columns of D . This is because, being a rigid body, the rank of R and S is 3.

In the “while loop” we try to reduce $\|Z_c - RU\|$ to the threshold, which in the code is called epsilon. The idea to optimize the rotation matrix is use the rotation condition $R_i R_i' = I$, where I corresponds to the identity matrix. This rotation condition is used to calculate T in the code and thus be able to calculate the new matrices S and R .

```

39      [U, D, V] = svd(Zc);
40      R = U(:, 1:3)*(D(1:3, 1:3) ^ 1/2);
41      S = (D(1:3, 1:3) ^ 1/2)*V(:, 1:3)';
42
43      %-----
44      % Metric Upgrade Step
45      F=size(Zc,1);
46      k=0;
47      thenorm=epsilon+1;
48      while thenorm>epsilon && k<n_iter
49          Rp=R;
50          M=[];
51          for f=1:2:F
52              Rf=R(f:f+1,:);
53              [U2,useless,V2]=svd(Rf,'econ');
54              T=U2*V2';
55              M=[M;T(1:2,:)];
56          end
57          S=pinv(M)*Zc;
58          R=Zc*pinv(S);
59          k=k+1;
60          thenorm=norm(R-Rp,'fro')/numel(M);
61      end

```

Figure 2: source code

3.2 Examples:

Here we will show how to reconstruct the face of an Ogre with the given frames and using the following input parameters:

- threshold: 10^{-7}
- maximum number of iterations: 200

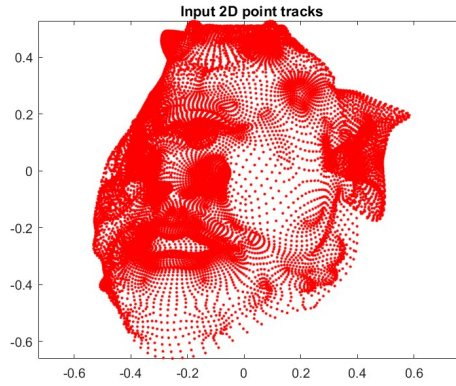


Figure 3: The original face of the Ogre.

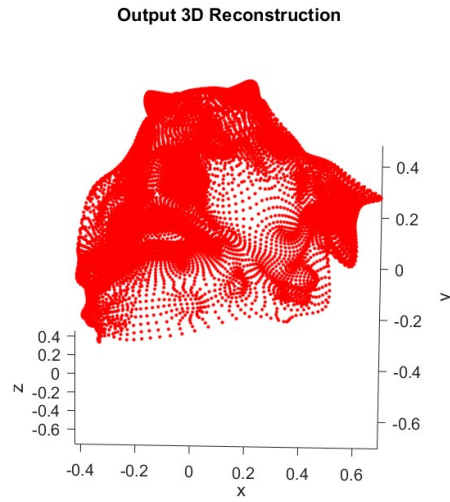


Figure 4: The Ogre's face reconstructed by the algorithm.

As can be seen in Figure 5, the reconstruction of the Ogre's face is quite good. This is demonstrated by the fact that the error between the original Ogre's face and its face reconstructed by the algorithm is 0,0026993% it means quite low, as depicted Figure 6.

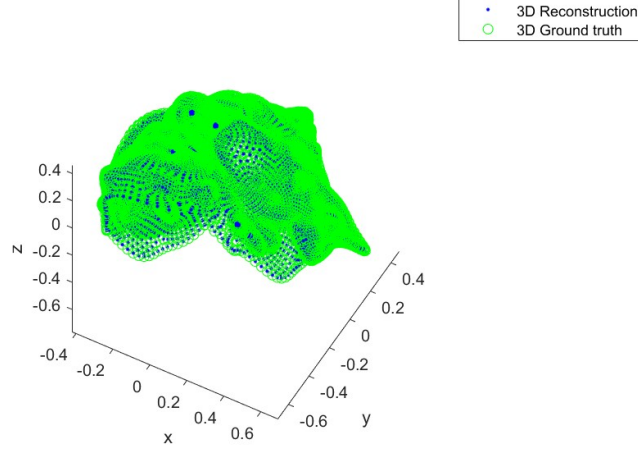


Figure 5: Comparison between the 3D representation of the Ogre’s original face and its face reconstructed by the algorithm.

```
Running...
3D error: 0.0026993 %
>>
```

Figure 6: Error between the ogre’s face and the reconstructed face.

4 Second Part (Non-Rigid structure from motion by non-linear optimization and assuming a low-rank shape model)

In this task we should perform a nonlinear optimization for a non-rigid body, assuming a low rank. This code uses Levenberg-Marquardt algorithm to optimize the non-linear equation that it is shown to next:

$$\arg \min_{R^i, B_k, l_i^k, t^i} \sum_{i=1}^i \|\widehat{W}^i - R^i \sum_{k=1}^k l_i^k B_k - t^i\|^2 + \gamma \sum_{i=1}^{i-1} \|L^i - L^{i+1}\|^2 + \phi \sum_{i=1}^{i-1} \|R^i - R^{i+1}\|^2 \quad (7)$$

Where: B_k is a row of bases matrix. R^i is a rotation for a point in matrix column i of X , which is the 3D shape vector. l_i are the constants such that $\sum_{k=1}^k l_i^k B_k = X$. t^i is the translation. L^i is the vector of all constants in frame i . The norm used is the Frobenius norm.

In order to calculate the Levenberg-Marquardt optimization, we need to calculate the Jacobian form of the equation 7. We do this as shown in the following

slide Figure 7

Exercise

Let us assume a monocular video of 3 images, where 6 points are observed. Considering the map is non-rigid and the visibility is full, define the corresponding Jacobian matrix. A low-rank shape model of rank 2 can be considered

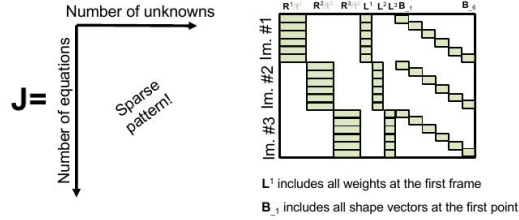


Figure 7: Jacobian form

4.1 Code:

Now we will show how this Jacobian form is calculated in the code.

So first we will define the dimensions of Jacobian matrix. For this code we are using quaternions for the rotations, therefore R^i is 2×4 .

Then the dimension of Jacobian matrix is defined in the following way

- number of rows of $Dim(J^{rows}) = 2 \sum_{i=1}^f (number\ of\ points\ visible\ in\ frame\ i)$
- number of columns of $Dim(J_{columns}) = 4f + 2f + kf + 3kp$

Where f is the number of frames, p is the total number of points tracked and k is the total number of bases.

Explaining row sizes:

Green: Represents the visible points per frame. When we multiply the sum of the visible points by two it is because the rotations are in quaternions. The quaternions are 2×4 and then they are two vectors per each point and that is why it is multiplied by two.

Explaining column sizes:

Red: Represents the value of the columns of the rotation matrix, which is 2×4 quaternions. We have a rotation matrix R^i for each frame i . Therefore, the

dimension is $4f$, where f is the number of frames.

Blue: Represents the translation matrix t^i , whose dimension is 1×2 , for each frame i , therefore $2f$ where f is the number of frames.

Orange: Represents the matrix L_i whose dimension is $1 \times k$ for each frame i , therefore kf where f is the number of frames.

Yellow: Represents the matrix B_b whose dimension is $3 \times p$ for each base b , therefore $3pk$ where k is the number of bases.

Well then we need to define the shape of our Jacobian matrix by putting 1 where there are values and leaving 0 in the spaces that do not exist, as shown in the following code.

```

1  for i=1:n_frames
2      for j=1:nnz(vij(i,:))
3          J(2*j-1+computed_points:2*j+
              computed_points,6*j-5:6*j)=ones(2,6);
4          J(2*j-1+computed_points:2*j+
              computed_points,K*j-K+1+6*n_frames:K*j
              +6*n_frames)=ones(2,K);
5          J(2*j-1+computed_points:2*j+
              computed_points,3*K*j-3*K+1+(6+K)*
              n_frames:3*K*j+(6+K)*n_frames)=ones
              (2,3*K);
6      end
7      computed_points=computed_points+2*nnz(vij(i,:))
8  end

```

In the first for loop each frame is iterated, in the second for loop only the points visible in that frame are iterated.

```

J(2*j-1+computed_points:2*j+computed_points,6*j-5:6*j)
=ones(2,6);

```

Here, the Jacobian is calculated for the rotation R part (in quaternions) and the translation part t , having this

$$(R_{2 \times 4} \mid t_{2 \times 2})_{2 \times 6} \quad (8)$$

we can add a matrix of ones of 2×6

Now, the Jacobian is calculated for the constants L^i part, where $L^i = [l_1^i, l_2^i, \dots, l_k^i]$ such that i is a frame.

```

J(2*j-1+computed_points:2*j+computed_points,K*j-K+1+6*
  n_frames:K*j+6*n_frames)=ones(2,K);

```

and for that, in the code is added a matrix of ones from $2 \times k$.

Now, the Jacobian is calculated for the bases B_i part, where

$$B_i = ([b_1^i]_{3 \times 1} \quad [b_2^i]_{3 \times 1} \quad \cdots \quad [b_k^i]_{3 \times 1})_{3 \times k} \quad (9)$$

```
J(2*j-1+computed_points:2*j+computed_points,3*K*j-3*K
+1+(6+K)*n_frames:3*K*j+(6+K)*n_frames)=ones(2,3*K)
;
```

and for that, in the code is added a matrix of ones from $2 \times 3k$.

Now we calculate the Jacobian part of the priors.

$$\arg \min_{R^i, B_k, l_i^k, t^i} \gamma \sum_{i=1}^{i-1} \|L^i - L^{i+1}\|^2 + \phi \sum_{i=1}^{i-1} \|R^i - R^{i+1}\|^2 \quad (10)$$

from the equation we have

- **The prior with respect to the rotation R^i :** This means that the rotation between two consecutive frames $\sum_{i=1}^{i-1} \|L^i - L^{i+1}\|^2$ do not have abrupt changes, it is a smoothing of the rotations.
- **The prior with respect to the constants L^i :** This means that the costates that multiply the bases, such that $X^i = \sum_{j=1}^k l_j^i B_j$ form the shape matrix. Therefore, if the constans for two consecutive frames $\sum_{i=1}^{i-1} \|L^i - L^{i+1}\|^2$ does not have abrupt changes then the shapes do not have abrupt changes, it is a smoothing of the shapes.

Well, we use the expressions in equation 10 to calculate the dimension of the Jacobian for the priors.

So from this equation 10 and the orange color of the equation 7 we deduce that the Jacobian dimension for the prior L would be

$$Dim(J_{rows}) = f - 1Dim(J_{columns}) = kf \quad (11)$$

and the blue and red color of the equation 7 also we can deduce the Jacobian dimension for the prior R as

$$Dim(J_{rows}) = f - 1Dim(J_{columns}) = 6f \quad (12)$$

Now we will show how we fill the indices that have a value in the Jacobian Matrix of the L prior with 1 and otherwise hold with zero.

```
if (priors.coeff_prior == 1)
    % prior terms on L
```



```

L=spalloc(n_frames-1,(K+6)*n_frames + K*3*
          n_points,2*K*(n_frames-1));
for i=1:n_frames-1
    L(i,K*i-K+1+6*n_frames:K*i+K+6*n_frames)=ones
        (1,2*K);
end
if K==1
    J(2*nnz(vij)+1:size(J,1), :)=L;
else
    J(2*nnz(vij)+1:2*nnz(vij)+n_frames-1, :)=L;
end
end
end

```

We do the following, we take the L_i and L_{i+1} and join them as shown in the equation 13

$$\left([L_i]_{1 \times k} \mid [L_{i+1}]_{1 \times k} \right)_{1 \times 2k} \quad (13)$$

and because of this in the code we add a matrix of ones $1 \times 2K$.

For the Jacobian Matrix of the R prior is the similar way, in this case we joint R^i and R^{i+1} like show in the equation 14

$$\left([R^i]_{1 \times 6} \mid [R_{i+1}]_{1 \times 6} \right)_{1 \times 12} \quad (14)$$

and because of this in the code we add a matrix of ones 1×12 .

Joining the Jacobian and the Jacobian of the priors we have the following graph figure 8.

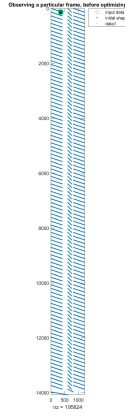


Figure 8: Jacobian result

4.2 Examples:

Here we will show how to reconstruct the human face with the given frames and using the following input parameters:

- low-rank K (it is the number of bases): 2
- $camera_{prior}$ (it mean that the R prior is active): 1
- $coeff_{prior}$ (it mean that the L prior is active): 1
- number of optimizer iterations: 50

In Figure 9 we can see the comparison of the input points of a frame in green against the estimated points of that frame in blue.

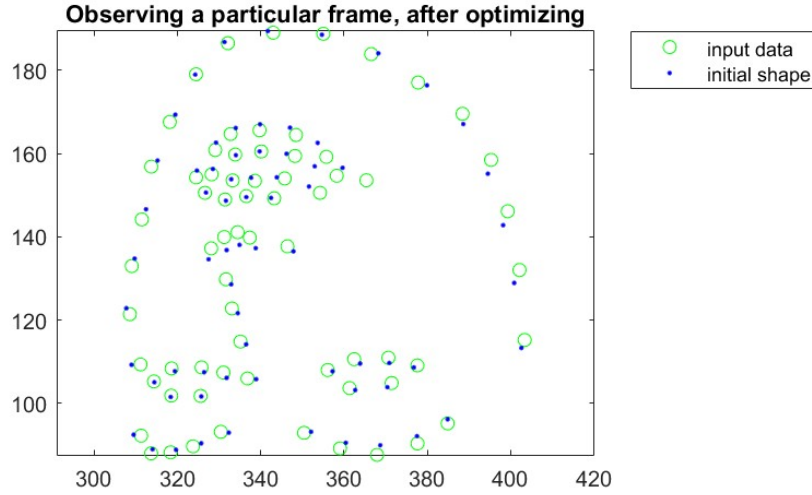


Figure 9: Input points compared to optimized points

In Figure 10 we can see the estimated 3D points for a frame.

In Figure 11 we see the calculation of each of the 50 optimizations, this method has a high cost in performance, but a good result is achieved as shown by the low error at the end.

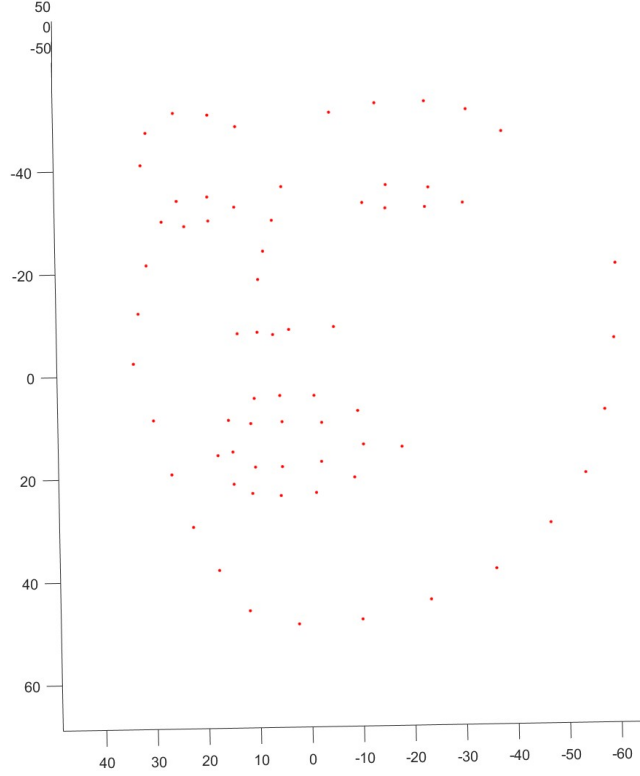


Figure 10: 3D reconstruction

5 Second Part (Non-Rigid structure from motion by factorization and assuming a low-rank trajectory model)

In this section, we calculate the reconstruction with the low-rank trajectory method. This method is a little simpler because we already know in advance the matrix of base trajectories since we defined them with the value of k , we will show the structure in Equation 3.

$$\theta = \begin{pmatrix} \theta_1^1 & \cdots & \theta_k^1 \\ \vdots & \ddots & \vdots \\ \theta_1^f & \cdots & \theta_k^f \end{pmatrix}_{f \times k}$$

$$[a^x] = \begin{pmatrix} a_{11}^x & \cdots & a_{1p}^x \\ \vdots & \ddots & \vdots \\ a_{k1}^x & \cdots & a_{kp}^x \end{pmatrix}_{k \times p}, [a^y] = \begin{pmatrix} a_{11}^y & \cdots & a_{1p}^y \\ \vdots & \ddots & \vdots \\ a_{k1}^y & \cdots & a_{kp}^y \end{pmatrix}_{k \times p} \quad \text{and} \quad [a^z] = \begin{pmatrix} a_{11}^z & \cdots & a_{1p}^z \\ \vdots & \ddots & \vdots \\ a_{k1}^z & \cdots & a_{kp}^z \end{pmatrix}_{k \times p}$$

$$X = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_p^1 \\ y_1^1 & y_2^1 & \cdots & y_p^1 \\ z_1^1 & z_2^1 & \cdots & z_p^1 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^f & x_2^f & \cdots & x_p^f \\ y_1^f & y_2^f & \cdots & y_p^f \\ z_1^f & z_2^f & \cdots & z_p^f \end{pmatrix} = \begin{pmatrix} \theta_{f \times k} & 0_{f \times k} & \cdots & 0_{f \times k} \\ 0_{f \times k} & \theta_{f \times k} & \vdots & 0_{f \times k} \\ \vdots & \vdots & \ddots & \vdots \\ 0_{f \times k} & 0_{f \times k} & \cdots & \theta_{f \times k} \end{pmatrix} \begin{pmatrix} [a^x]_{k \times p} \\ [a^y]_{k \times p} \\ [a^z]_{k \times p} \end{pmatrix} \quad (15)$$

Iteration	Func-count	Resnorm	First-order optimality	Lambda	Norm of step
0	1225	1.84649e+06	9.51e+04	0.01	
1	2450	26864.9	2.09e+03	0.001	25.0136
2	3678	16728	1.07e+03	1	15.9044
3	4903	12361	487	0.1	24.5625
4	6129	11207.5	837	1	15.9301
5	7354	10545.2	661	0.1	12.7065
6	8580	10245.9	717	1	7.64863
7	9805	10085.3	465	0.1	5.64436
8	11031	9981.24	272	1	4.18336
9	12256	9910.45	164	0.1	3.76857
10	13482	9851.74	138	1	3.28008
11	14707	9806.24	108	0.1	2.81746
12	15932	9769.25	165	0.01	5.31267
13	17158	9728.06	114	0.1	3.42729
14	18383	9704.76	96.1	0.01	3.27792
15	19609	9687.54	81.7	0.1	2.7978
16	20834	9675.29	48.5	0.01	2.4206
17	22060	9666.78	42.2	0.1	2.06087
18	23285	9660.79	36.6	0.01	1.83145
19	24511	9656.37	30.5	0.1	1.78
20	25736	9652.86	25.5	0.01	1.81386
21	26962	9649.82	22.3	0.1	1.9704
22	28187	9646.98	20.2	0.01	2.11821
23	29413	9644.15	19.7	0.1	2.32606
24	30638	9641.23	19.1	0.01	2.48918
25	31864	9638.15	19.8	0.1	2.69185
26	33089	9634.88	19.7	0.01	2.83915
27	34315	9631.4	21	0.1	3.02158
28	35540	9627.71	20.9	0.01	3.14215
29	36766	9623.8	22.1	0.1	3.29737
30	37991	9619.73	22.9	0.01	3.38541
31	39217	9615.48	22.4	0.1	3.50923
32	40442	9611.11	24.7	0.01	3.56288
33	41668	9606.61	22.1	0.1	3.65309
34	42893	9602.06	26.4	0.01	3.67613
35	44119	9597.42	21.1	0.1	3.73425
36	45344	9592.76	28	0.01	3.73824
37	46570	9588.01	20.6	0.1	3.76665
38	47795	9583.3	29.4	0.01	3.76896
39	49021	9578.49	23.2	0.1	3.76972
40	50246	9573.86	40.4	0.01	3.79026
41	51472	9569.16	30.5	0.1	3.76321
42	52697	9565.07	73.9	0.01	3.81738
43	53923	9560.89	73.9	0.1	3.76668
44	55148	9558.3	153	0.01	3.86225
45	56374	9555.13	168	0.1	3.80734
46	57601	9550.17	104	10	0.546409
47	58826	9547.64	80.2	1	0.302035
48	60051	9546.47	34.9	0.1	0.763707
49	61276	9542.47	44.1	0.01	3.79643
50	62502	9539.54	86.7	0.1	3.62819

[Solver stopped prematurely.](#)

lsqnonlin stopped because it exceeded the iteration limit,
[options.MaxIterations](#) = 5.000000e+01.

Final reprojection (error: 0.0030556%)

Figure 11: Optimizer iterations