Locally-Rigid Motion v0.2
Release Notes

Jonathan Taylor

This updated release is a signficantly improved implementation of the Local Rigidity algorithm described in [**?**]. It is roughly two orders of magnitude faster than version 0.1 while generally considering a much larger set of triangles (see below). In general, the added triangles adds to the robustness of the method, and allows for simplification of some of the parameters as detailed below. Please consult the included README.txt for instructions on running the code. The set of changes and deviances from [**?**] include:

- The 3-SFM fitting of triangles and the reconstruction of different frames is now mapped across a set of worker processes allowing for the use of multi-core processors (see `python lrigid.py --help`).
- Nearly all of the code underlying the 3-SFM bundle adjustment is now written in C (via Cython) or in vectorized python (via NumPy).
- The costly per-frame least squares initialization of rotation is replaced with a closed form rotation that first optimizes the 2D xy-planar rotation followed by a slight out of plane tilting.
- Many sequences contain feature triplets that are not well constrained leaving a long flat valley in the bundle-adjustment objective function where "long" triangles extended deeply into depth can still use small rotations to model image projections. When this occurs, it is difficult for gradient based optimizers to find a well defined minimum. Thus a weak prior is added to discourage these large triangles and to create a well defined minimum. The objective is then

$$f(\theta;\lambda) = \sum_{n=1}^{N}\sum_{i=1}^{3} || \left[\begin{smallmatrix}1&0&0\\0&1&0\end{smallmatrix}\right] \mathbb{R}_f(\theta)\hat{\mathbf{p}}_i(\theta) + \mathbf{t}(\theta) - \mathbf{q}_{in}||^2 + \lambda \sum_{i=1}^{3} L_i^2(\theta)$$

where the first term corresponds to the sum of squared reprojection errors and the second term, the sum of squared triangle edge lengths. As the valleys described are so flat, the choice of λ is not crucial and thus we simply set it to 0.01 throughout.

- The threshold for interpreting a triangle to be rigid based on it's RMS reprojection error $\sqrt{f(\theta;0.0)/3N}$ is now by default 1.5 times the median of the set of these values for all triangles considered. This is a reasonable setting, but one should consider changing it depending on how many rigid triangles are expected in the sequence (see `python lrigid.py --help`).

- L-BFGS is now used instead of conjugate gradient since it is better at learning the shape of this objective function and converging quickly.

- Using triangle proposals from a delaunay triangulation in a single frame is not robust for two reasons. First, if a track slips, then there will be a hole in the reconstruction. Thus, the implementation now starts by taking the union of all delaunay triangulations of the tracks in all frames. Second, for datasets in which the locations of the tracks are structured (e.g. a grid), it is very likely to find long runs of fronto-parallel edges, making flip resolution difficult or ambiguous. Therefore, the code now subsamples 25% of the tracks (corresponding to keeping roughly every fourth point in a grid) and adds the union of all triangulations of this set in all frames to the set of proposal triangles.

- With the addition of so many triangles, it now appears that a much simpler binary flip MRF can be used. In particular, we define the energy of two spatial neighbours having their common edge at an angle of θ in degrees as

$$E_s(\theta) = \frac{\theta^2}{\theta^2 + \sigma_s^2}$$

where the above Geman-McClure [**?**] robust function asymptotes to 1 as θ increases forcing the difference between two "bad" alignments (e.g.

greater than 20 degrees) to be small. The energy of an angle θ between a triangle's normal and that of it's temporal neighbour is simply scaled linearly.

$$E_t(\theta) = c_s\theta$$

It is not necessary to force E_t to be robust like E_s as one can expect one of the two flip configurations to provide an angle near 0. Here σ_s is set to 10 and c_s is set to 50 to move the energies to a similar scale.

The greedy algorithm then finds a minimum spanning tree on the graph with spatial edges weighted by $|E_s(\theta_{EQ}) - E_s(\theta_{OPP})|$ and temporal edges weighted by $|E_t(\theta_{EQ}) - E_t(\theta_{OPP})|$. The lack of any infinite edges means that each "body" is completely reconstructed.

- The greedy flip resolution algorithm is now much faster as it is written in C (via Cython) despite being a naive minimum spanning tree implementation. A future version will include a more efficient union-find implementation.

- The use of QPBO-I [**?**] to improve the greedy solution to the binary MRF is disabled by default since it makes compiling this code substantially more involved and it doubles the running time with very little increase (and occasional decrease) in reconstruction accuracy. See the `README.txt` for compiling the pyqpbo wrapper and then enable it on the command line (see `python lrigid.py --help`).

- In the linear system $Az = b$ used for resolving the absolute triangle depths z, the matrix A can be extremely tall making A^TA comparitively small. The sparsity of A makes forming A^TA particulary easy to form and thus the normal equations $A^TAz = Ab$ are used to solve for z. A standard sparse solver is not used, since the ones available in numpy/scipy do not allow for many solutions (one for each frame) to be computed simultaneously. If there is concern over the numerical stability a standard dense solver can be used (see `python lrigid.py --help`).

- As in [**?**], for each component (or body), the reconstructed 3D point p_{in}^{rec} of track i in frame n are taken to be the average of all triangle vertices that match that track.

- When ground truth is available, the reconstruction error for each component can be examined independently. If the component includes the track indices $\mathcal{I}$, then the RMS reconstruction error can be considered.

$$\sqrt{\frac{1}{N|\mathcal{I}|}\sum_{n=1}^{N}\sum_{i\in\mathcal{I}}||p_{in}^{rec}-p_{in}^{gt}||^2}$$

  Due to the inherent orthographic ambiguities, a per-frame depth flip and translation is applied to the reconstructed point set to minimize this number. This is the number reported via `show.py` for each component.