Practical Course

Multi-Camera Computer Vision and Algorithms

Jean Elsner

WS 17/18

Pipeline Architecture

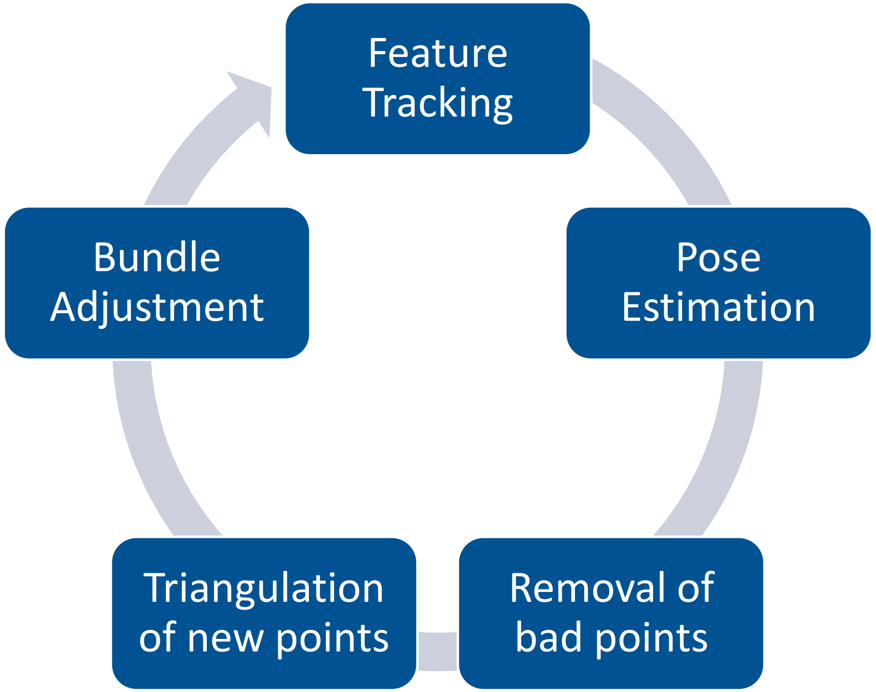
Odometry Pipeline					
2D Features		3D Point Cloud		Optimization	
Feature Extractor	Feature Tracker	Triangulation	Pose Estimation	Bundle Adjustment	Outlier Detection

Pipeline Architecture

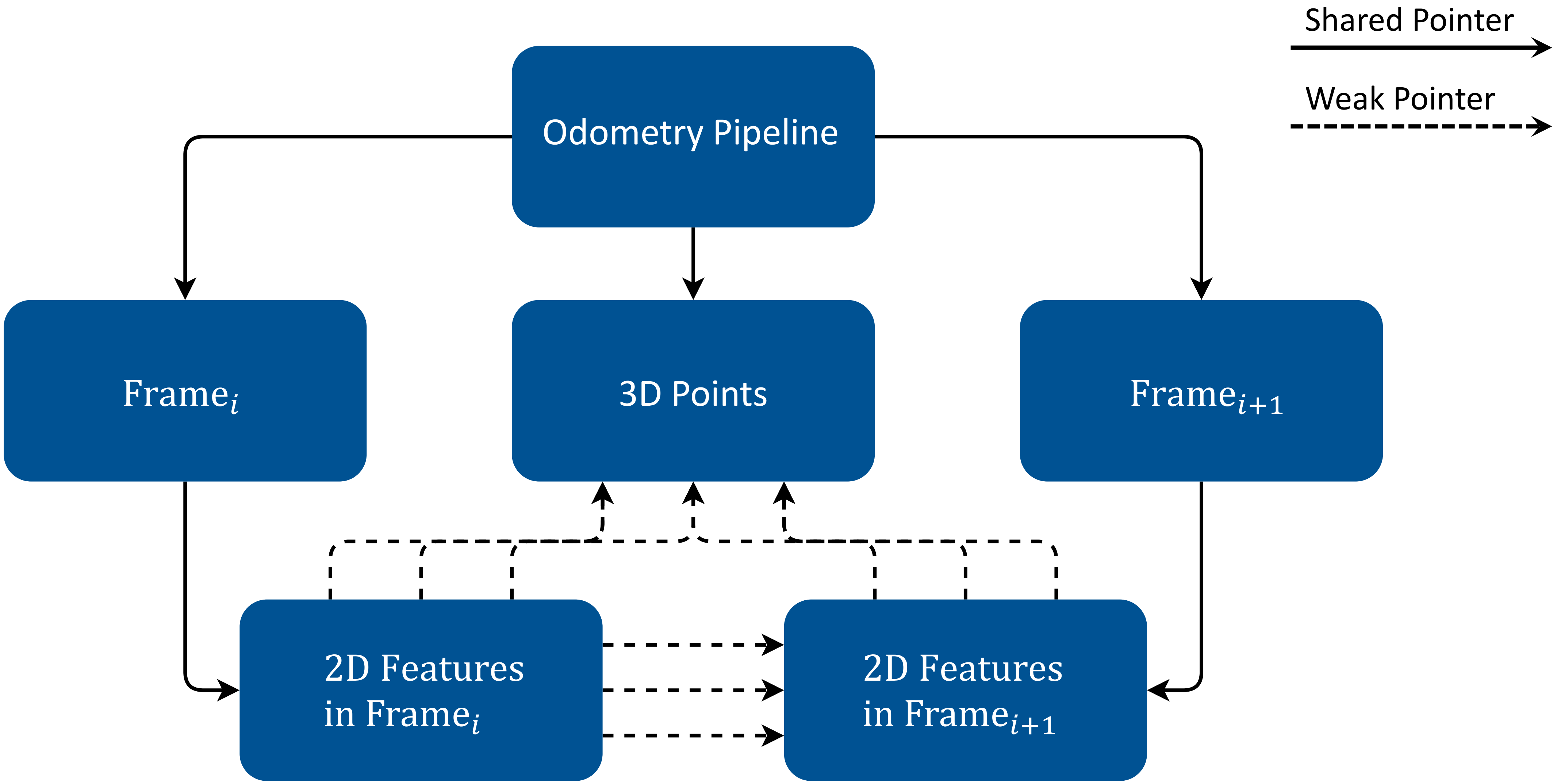
The Odometry Pipeline takes a series of calibrated images from the KITTI database and reconstructs the camera poses. To this end, a pair of images is used to triangulate a set of 3D points (initialization) and the main odometry loop is run.

The main loop consists of these steps:

- 2D Features are tracked to the next frame
- 2D-3D correspondences are used to solve the Perspective-n-Point (PnP) problem
- New 3D points are triangulated when necessary, while outliers are removed
- Bundle Adjustment may be performed to further optimize camera poses and 3D points



Pipeline Architecture



Pipeline Architecture

The reconstructed trajectory is rendered next to the ground truth and written to a video file. Additionally the 3D point cloud can be viewed within the application.

Notable aspects of the architecture include:

- Configurability: many of the pipeline's settings can be fine tuned in an external configuration file
- Modularity: several main components may be switched out, alternative implementations are available
- Multithreading: Feature extraction and matching is separated from pose estimation and optimization

Screenshot of video produced by Odometry Pipeline

Pipeline Architecture

The Odometry Pipeline is implemented in C++ and uses several open source libraries. OpenCV is used for general image processing, 2D feature extraction and tracking, pose estimation and video production. Optimization through bundle adjustment is achieved with Ceres Solver. Finally, Dlib is utilized for its 3D point cloud viewer as well as multithreading tools.

Feature Extraction and Tracking

The robust extraction of invariant points of interest is the corner stone of the Odometry pipeline. Within the architecture, different modules for 2D feature extraction are available.

ShiTomasiFeatureExtractor

Corner detector that selects local maxima in variation in all directions. Candidates are calculated using the Eigenvalues of the pixel‘s structural tensor [1].

OpenCVGoodFeatureExtractor

OpenCV based corner detector similar to the implementation above. Uses heuristic corner response measure instead of calculating Eigenvalues [2].

OpenCVFASTFeatureExtractor

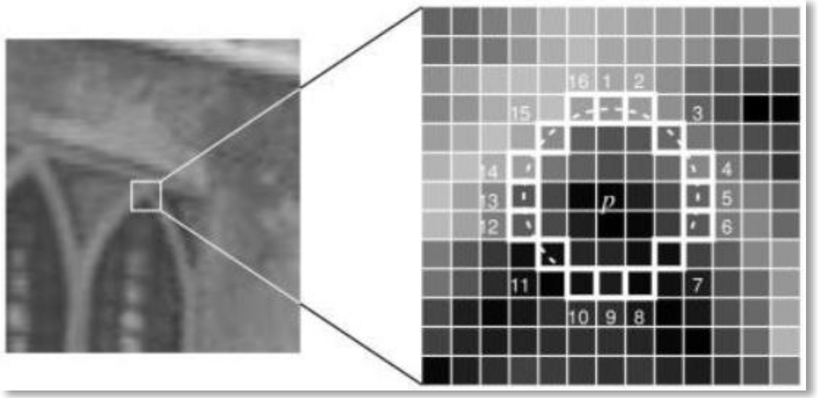
Machine learning based. Checks pixels on a circle around the point of interest [3].

Structural tensor

$$S(x,y) = \sum_p w(p) \begin{bmatrix} I_x(p)^2 & I_x(p)I_y(p) \\ I_x(p)I_y(p) & I_y(p)^2 \end{bmatrix}$$

Corner response measure

$$R = \det M - k(\operatorname{trace} M)^2$$



FAST (Features from Accelerated Segment Test) algorithm, from Rosten 2006

Feature Extraction and Tracking

Features need to be tracked between frames in order to perform pose estimation or triangulation of new 3D points.

Video of Odometry Pipeline tracking 2D features with the OpenCVLucasKanadeFM module

Feature Extraction and Tracking

Features need to be tracked between frames in order to perform pose estimation or triangulation of new 3D points.

OpenCVLucasKanadeFM

OpenCV‘s implementation of a pyramidal Lucas Kanade feature tracker [4]. On each level of the pyramid, the residual function $\sum_{x \in w_x} \sum_{y \in w_y} \left(I_i(\vec{x} + \vec{u}) - I_{i+1}(A\vec{x} + \vec{d} + \vec{u}) \right)^2$ is minimized for an image patch around each feature in regard to image velocity $\vec{u}$ and the affine transformation matrix A.

kNNFeatureMatcher

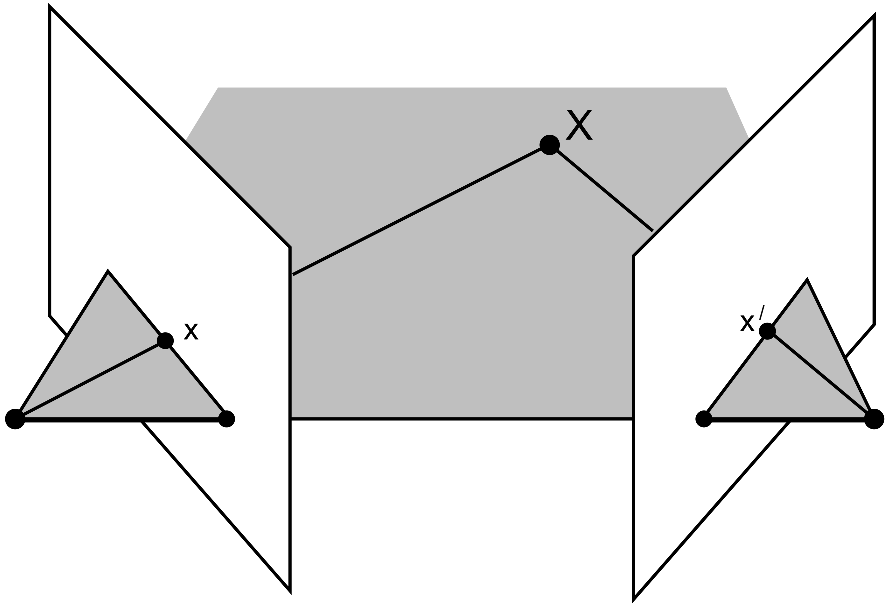
Selects a local neighborhood of 2D features in the target image and compares them to a feature in the source image. Relatively slow, as features have to be extracted from every frame.

Triangulation and Pose Estimation

Both during initialization and whenever the number of seen 3D points drops below a threshold, new 3D points need to be triangulated given 2D-2D correspondences between frames.

OpenCVFivePointTri

This module uses OpenCV's implementation of a five-point algorithm [6] to determine the essential matrix for 2D-2D correspondences between two frames. Given the relative poses of the cameras, 3D points can be triangulated as an intersection of two lines, as seen to the right.



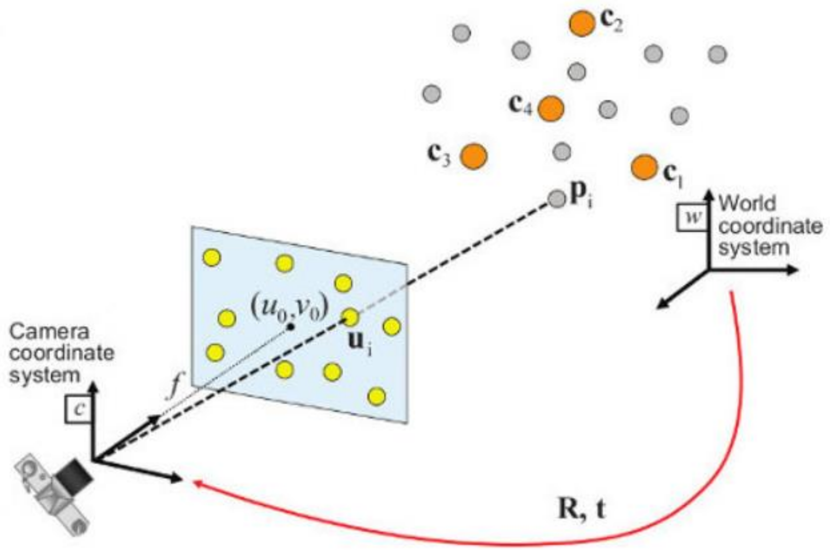
Triangulation of image point correspondences [5]

Triangulation and Pose Estimation

After initialization, feature tracking will provide 2D-3D correspondences for each frame, that can be used to estimate the pose of the camera (Perspective-n-Point problem).

OpenCVEPnPSolver

Based on OpenCV's implementation of the Efficient PnP method (EPnP) described in [7]. Given $n \geq 4$ reference point and a calibrated camera, the algorithm calculates an estimate for the camera pose in linear time.



Perspective-n-Point problem, as illustrated in the OpenCV documentation [8]

Bundle Adjustment and Optimizations

Outlier detection

During pose estimation, RANSAC is used to select a consensus set of reference points. 3D points not within this set are considered outliers and removed from the global shared list.

CeresBundleAdjustment

Ceres Solver is used to minimize the reprojection error of the last n frames. For each frame, a residual function varying the 3D points seen in the frame, as well as the pose of the camera are added to a cost function. Ceres Solver simultaneously optimizes the shared 3D point coordinates as well as the camera poses.

Bundle Adjustment and Optimizations

Video of Odometry Pipeline comparing performance without and with bundle adjustment

Conclusion

Bundle/it.	Δt_{total}	Δt_{min}	Δt_{max}	Δt_{std}	ΔR_{total}	ΔR_{min}	ΔR_{max}	ΔR_{std}	Runtime
0/0	37977.8	0.013902	179.65	50.0298	457.367	0.0019524	1.49654	0.013902	16.0372
5/5	13288.2	0.0538121	50.6598	15.1565	223.147	0.0106561	0.689664	0.181154	24.1479
5/50	12337.4	0.0817539	49.948	12.2226	218.43	0.0174908	0.75612	0.189256	78.1034

Finally, Sequence 07 of the KITTI benchmark is run for 600 frames using different combinations of Bundle size and iterations (Bundle/it.), as seen on the previous slide. For each camera pose, the absolute deviation from the ground truth is calculated using the L_2 norm. The total error, as well as the minimum, maximum and standard deviation of the error for the 600 frames are given for both translation and rotation in the table above.

Observed trend: Using Bundle Adjustment can significantly improve the reconstructed trajectory, albeit at the cost of increased runtimes. Heuristically, a Bundle size of 5 frames has proved most advantageous. While significantly increasing the thresholds for seen 3D points and tracked 2D features (beyond the standard 150 and 400 respectively) does little to improve the result.

Conclusion

YouTube Channel

Including videos running with different settings and during different stages of development

https://www.youtube.com/channel/UC1qMUwRQFM96uWErU2Avc1g

GitHub Repository

Full source code and instruction for compilation available on GitHub

https://github.com/JeanElsner/practical-multi-view

Conclusion

References

[1] J. Shi and C. Tomasi. Good Features to Track. 9th IEEE Conference on Computer Vision and Pattern Recognition, 1994.

[2] C. Harris and M. Stephens. A Combined Corner and Edge Detector. Proceedings of the 4th Alvey Vision Conference, 147–151, 1988.

[3] E. Rosten and T. Drummond. Machine Learning for High-Speed Corner Detection. Computer Vision–ECCV 2006, 430–443, 2006.

[4] Jean-Yves Bouguet. Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker. Intel Corporation, 5, 2001.

[5] R.Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, 2004.

[6] David Nistér. An Efficient Solution to the Five-Point Relative Pose Problem. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 26(6):756–770, 2004.

[7] V. Lepetit and F. Moreno-Noguer and P. Fua. EPnP: An Accurate O(n) Solution to the PnP Problem. International journal of computer vision, 81(2):155–166, 2009.

[8] OpenCV: Camera Calibration and 3D Reconstruction. Retrieved from https://docs.opencv.org/3.4.1/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d.