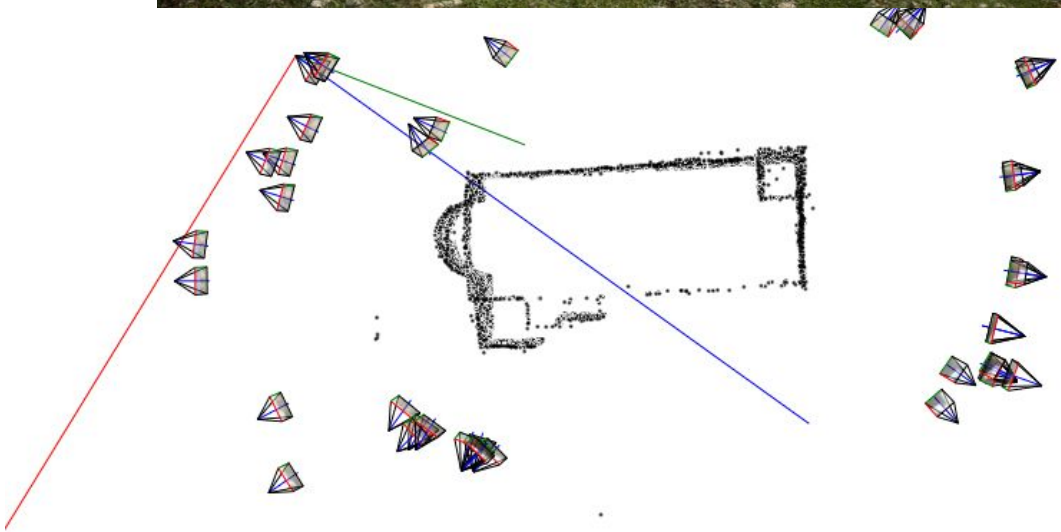
A Composable and Reusable Photogrammetric Reconstruction Library

Attila Szabo

Photogrammetry

- Take metric measurements in a photo



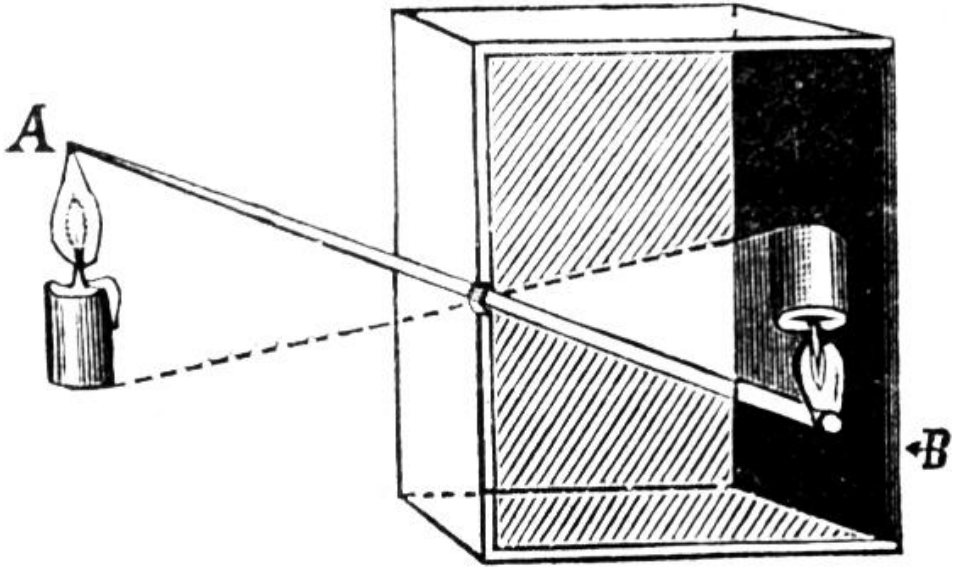
Motivation

- Existing software lacks transparency
  - highly specialised
  - low-level hackery
  - closed source

```
std::vector<IndMatch> vec_PutativeMatches;
//-- Perform matching -> find Nearest neighbor, filtered with Distance ratio
{
  // Find corresponding points
  matching::DistanceRatioMatch(
    0.8, matching::BRUTE_FORCE_L2,
    *regions_perImage.at(0).get(),
    *regions_perImage.at(1).get(),
    vec_PutativeMatches);

  // Draw correspondences after Nearest Neighbor ratio filter
  const bool bVertical = true;
  Matches2SVG
  (
    jpg_filenameL,
    {imageL.Width(), imageL.Height()},
    regionsL->GetRegionsPositions(),
    jpg_filenameR,
    {imageR.Width(), imageR.Height()},
    regionsR->GetRegionsPositions(),
    vec_PutativeMatches,
    "03_Matches.svg",
    bVertical
  );
}

//---
// Homography geometry filtering of putative matches
// - Show how to use the robust_estimation framework with different robust_estimat
//---

// First we list the SIFT photometric corresponding points to Mat arrays (The datu
Mat xL(2, vec_PutativeMatches.size());
Mat xR(2, vec_PutativeMatches.size());

for (size_t k = 0; k < vec_PutativeMatches.size(); ++k)
{
  // For each correspondence, add the Right & Left feature point positions
  const PointFeature & imaL = featsL[vec_PutativeMatches[k].i_];
  const PointFeature & imaR = featsR[vec_PutativeMatches[k].j_];
  xL.col(k) = imaL.coords().cast<double>();
  xR.col(k) = imaR.coords().cast<double>();
}

// Then we use a robust_estimator to find if a model can be fitted in the defined

//--
//-- Max Consensus
//- Return the Model that have the most of inliers
//- Perform all the iterations (no early exit)
{
  std::cout
    << "-----------------------------------\n"
    << "MAXConsensus -- Robust estimation \n"
    << "-----------------------------------\n";

  // Define the Homography Kernel
  using KernelType = homography::kernel::UnnormalizedKernel;
  KernelType kernel(xL, xR);

  // The Model type
  Mat3 H;
  // The inlier list
  std::vector<uint32_t> vec_inliers;
  H =
    MaxConsensus
    (
      kernel, // The Kernel (embed the correspondences, the Model Solver & the fit
      ScorerEvaluator<KernelType>(4.0), // Upper bound of the tolerated error for
      &vec_inliers, // Inlier list
      1024 // max_iteration count
    );
```

Camera Model

- Pinhole Camera
- Shot
- Photo Undistortion



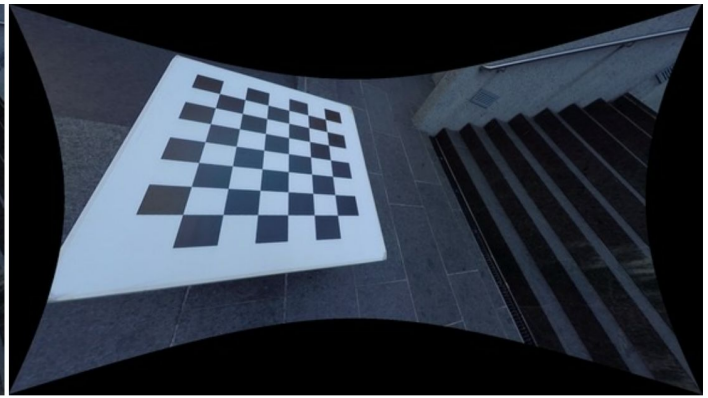
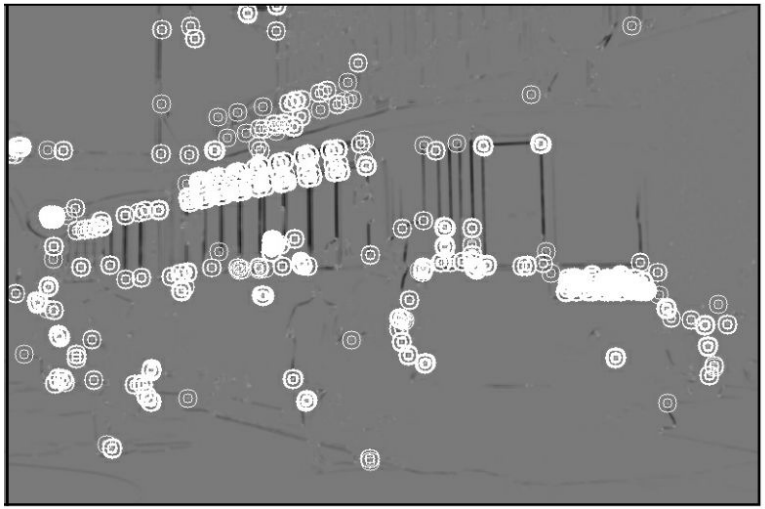


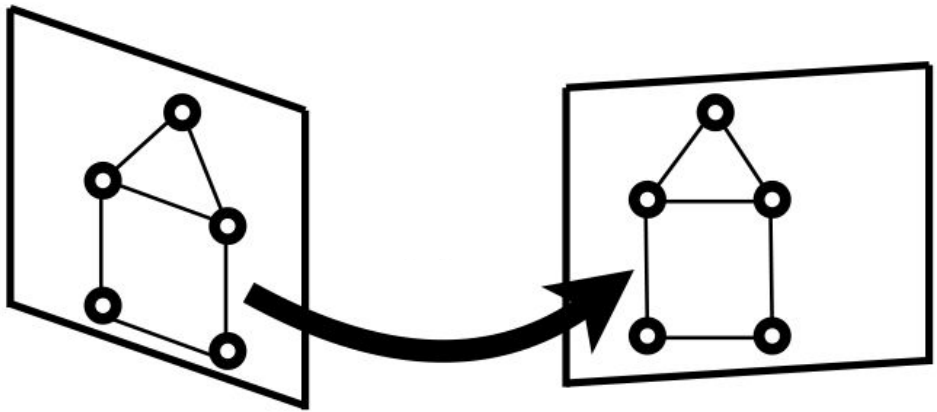
Image Features

- Identifying important things in photos
- For example Keypoints
  - SIFT
  - BRISK
  - AKAZE
  - ...



Feature Matching

- Identify the same feature in multiple images
- Respect the baseline



```
val matcher : list<Feature> -> list<Feature> -> list<Feature * Feature>
```

Motion Consistency

- Adjacent features exhibit similar 2D motion

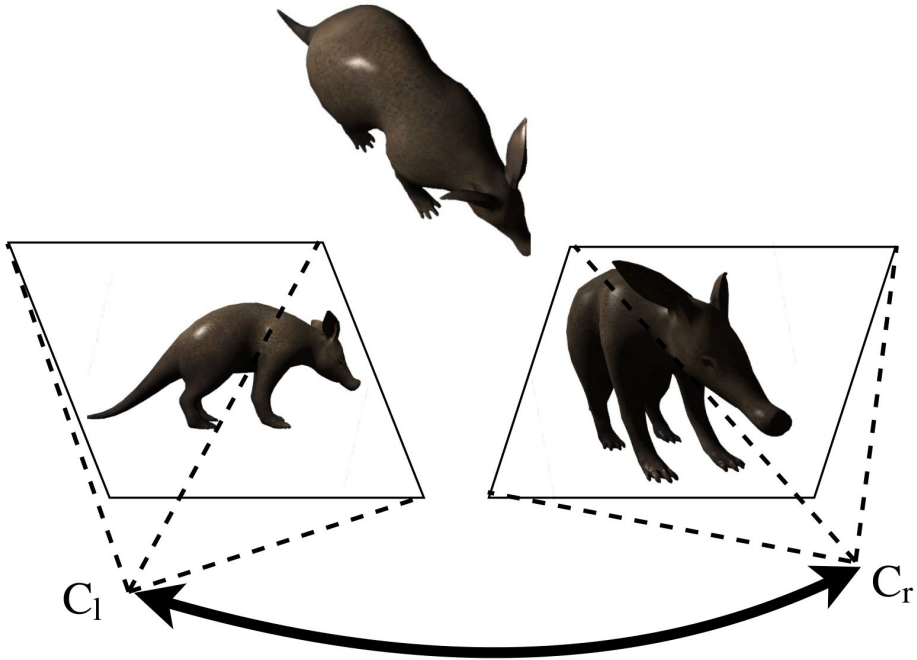
$$m_i = [x_i; v_i; o_i]$$

$$prob(m) = \sum_{i=1}^{N} w_i * exp^{-\frac{\|m - m_i\|^2}{\sigma}}$$

```
let consistency s p l r =
     List.cross l r
          |> List.filter ( fun m -> prob m s > p )

let match = consistency 0.5 0.9
```

Pose Recovery

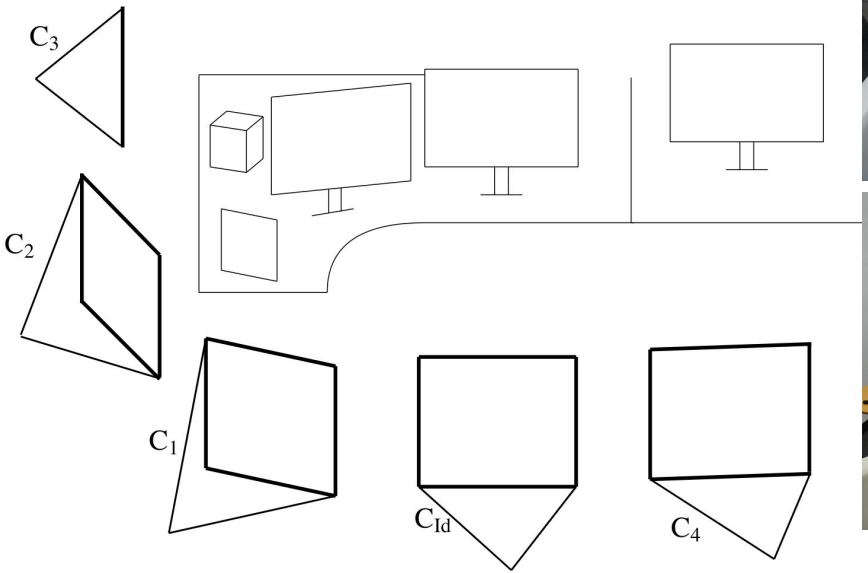
- Find two shots for matches
- Explain away ambiguities

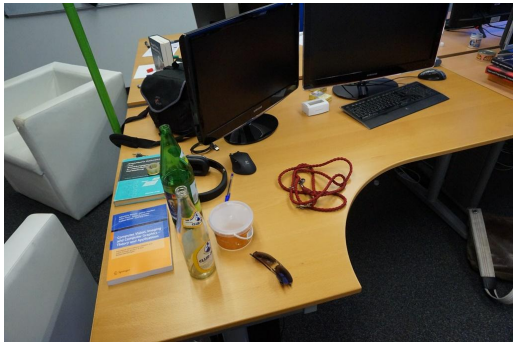


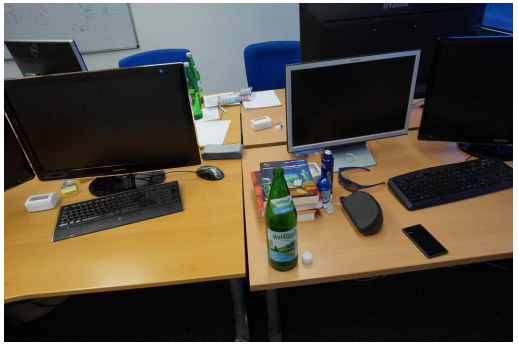

```
val recoverPose : list<Feature * Feature> -> Trafo
```

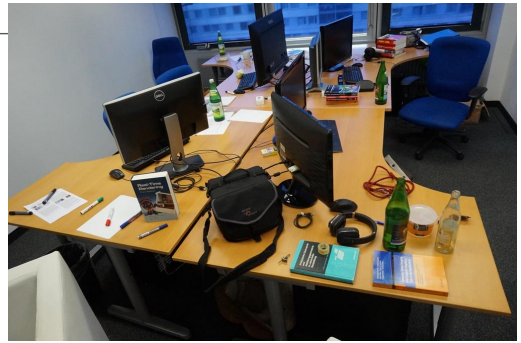
Structure From Motion

- Iterative Pose Recoveries









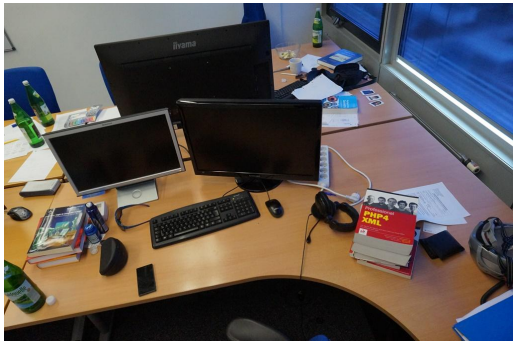
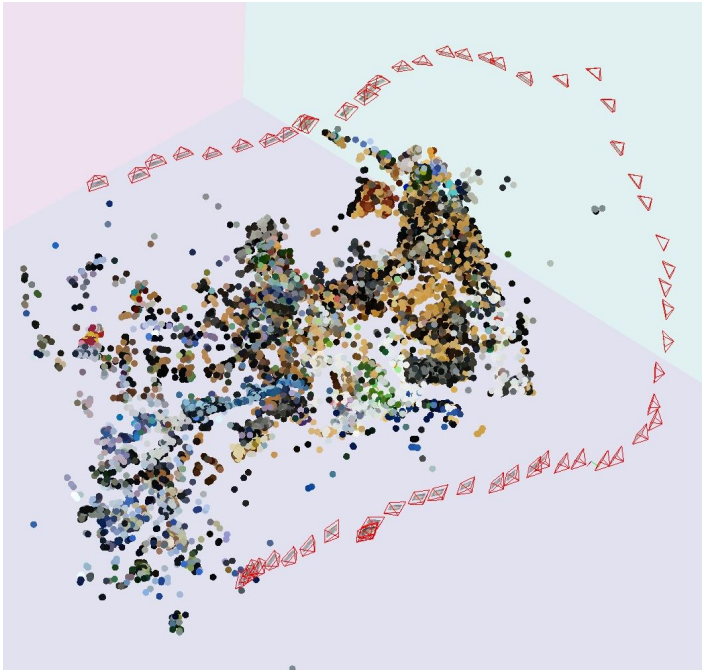


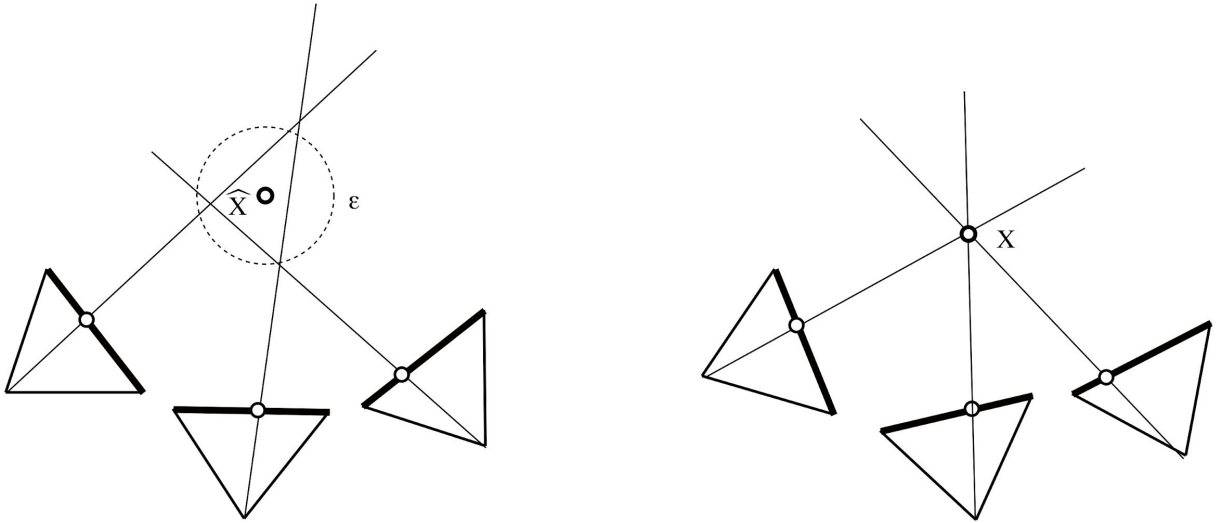
Photo Network

```
type PhotoNetwork = list<Shot>
val createPhotoNetwork : list<Image> -> PhotoNetwork


let createPhotoNetwork =
  List.fold ( fun left net ->
    let right = net.Head
    let newShot =
      left.transformed recoverPose match left right
    newShot :: net
  ) [Shot.Identity]
```



Bundle Adjustment

- Global error correction

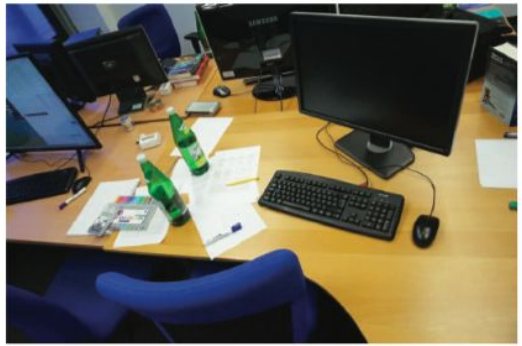


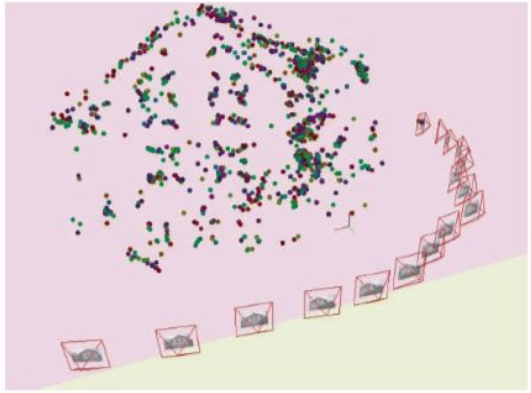

```
val bundleAdjust : PhotoNetwork -> PhotoNetwork
```

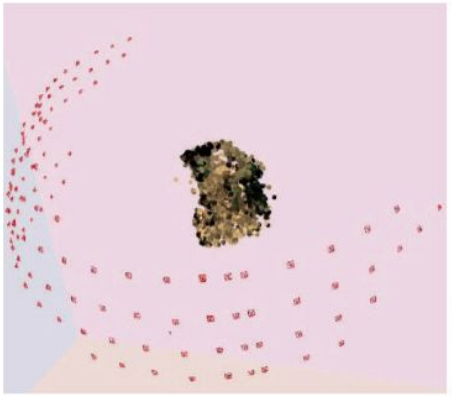
Computation through Composition

```
let extractFeaturesFast =
    Array.Parallel.map Akaze.extract
```

```
let photogrammetryPipeline files =
    "C:\undistortedPhotos\"
        |> readImgs
        |> extractFeaturesFast
        |> createPhotoNetwork
        |> bundleAdjust
        |> render
```







Contact me: szabo@vrvis.at

https://github.com/aardvark-platform/aardvark.mondo

AARDVARK