Lab 4 –
Image mosaics from feature matching

Topics

- Experiment with features
- Estimate homographies
- Create mosaics




Features in OpenCV

- Several methods
  - Some are detectors
  - Some are descriptors
  - Some are both detectors *and* descriptors

- Module *features2d*

- Module *xfeatures2d*

Features in OpenCV

- Construction

```
cv::Ptr<cv::Feature2D> detector = cv::xfeatures2d::SURF::create();
cv::Ptr<cv::Feature2D> desc_extractor = cv::xfeatures2d::LATCH::create();
cv::BFMatcher matcher{desc_extractor->defaultNorm()};
```

- Use

```
// Detect keypoints
std::vector<cv::KeyPoint> frame_keypoints;
detector->detect(gray_frame, frame_keypoints);
cv::KeyPointsFilter::retainBest(frame_keypoints, 500);

// Extract descriptors.
cv::Mat frame_descriptors;
desc_extractor->compute(gray_frame, frame_keypoints, frame_descriptors);

// Match descriptors.
std::vector<std::vector<cv::DMatch>> matches;
matcher.knnMatch(frame_descriptors, ref_descriptors, matches, 2);
std::vector<cv::DMatch> good_matches = extractGoodRatioMatches(matches, 0.8);
```

Step 1: Experiment with features

- Test different detectors
- Play around with the parameters
- What does `cv::KeyPoint` contain?

Step 2: Feature matching

- Implement in `feature_utils.cpp`:

```
std::vector<cv::DMatch> extractGoodRatioMatches(
  const std::vector<std::vector<cv::DMatch>>& matches, float max_ratio)
```

- Experiment with description and matching
  - Different detectors
  - Different descriptors
  - Different parameters

- Try different camera poses, different scene structures

Step 3: Compute the reprojection error

```
float HomographyEstimator::computeReprojectionError(const Eigen::Vector2f& pt1, const Eigen::Vector2f& pt2,
                                                    const Eigen::Matrix3f& H, const Eigen::Matrix3f& H_inv)
const
{
  // Todo: Step 3: Compute the two-sided reprojection error.
  // Map points onto each other using the homography.
  Eigen::Vector2f pt_1_in_2;
  Eigen::Vector2f pt_2_in_1;

  // Compute the two-sided reprojection error.
  return 2*distance_threshold_;
}
```

$$\varepsilon_i = d(H\boldsymbol{u}_i, \boldsymbol{u}'_i) + d(\boldsymbol{u}_i, H^{-1}\boldsymbol{u}'_i)$$ **(Reprojection error)**

Notation	
Euclidean distance	$d(\cdot,\cdot)$
Inhomogenous $H\tilde{\boldsymbol{u}}_i$	$H\boldsymbol{u}_i$
Inhomogeneous $H^{-1}\tilde{\boldsymbol{u}}'_i$	$H^{-1}\boldsymbol{u}'_i$

Step 4: Mosaicking

- What does *S* do?

```
cv::Matx33d S{
0.5, 0.0, 0.25 * frame_cols,
0.0, 0.5, 0.25 * frame_rows,
0.0, 0.0, 1.0};
```

- Coregister the images by applying the homography *H,* and move and scale them by using the similarity transform *S*

```
cv::warpPerspective(ref_image, mosaic, /* ? */, frame.size());
cv::warpPerspective(mask, mask_warp, /* ? */, frame.size());
cv::warpPerspective(frame, frame_warp, /* ? */, frame.size());
```

- Combine the image into the mosaic

```
frame_warp.copyTo(mosaic, mask_warp);
```

- How can we avoid the edge effects? (`cv::erode()`?)

Step 4: Mosaicking

Transformation of $\mathbb{P}^2$	Matrix	#DoF	Preserves	Visualization
Translation	$\begin{bmatrix} I & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	2	Orientation + all below	
Euclidean	$\begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	3	Lengths + all below	
Similarity	$\begin{bmatrix} sR & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	4	Angles + all below	
Affine	$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$	6	Parallelism, line at infinity + all below	
Homography /projective	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$	8	Straight lines	

Then…

- Read through `HomographyEstimator` and understand what it does!
- Try `cv::findHomography(…)` instead of our method
- Combine several detectors and descriptors
- Apply blending to the mosaic
  - Alpha blending
  - Laplace blending
- Expand the program to make a mosaic of more than two images