Computer Vision

Professor William Hoff
Dept of Electrical Engineering & Computer Science
http://inside.mines.edu/~whoff/
Using RANSAC to fit a Homography in OpenCV
SURF

• SURF: Speeded Up Robust Features
• Similar to SIFT – scale and rotation invariant
• We create a “SURF” object and initialize its parameters (see http://docs.opencv.org)

```cpp
cv::Ptr<cv::Feature2D> f2d = cv::xfeatures2d::SURF::create(
    1000.0, // threshold (default = 100.0)
    4, // number of octaves (default=4)
    2, // number of octave layers within each octave (default=2)
    true, // true=use 128 element descriptors, false=use 64 element
    false); // true=don't compute orientation, false=compute orientation
```

Keypoints

• We detect “keypoints”
  – Each keypoint has location, size, angle, etc

```c++
// Detect keypoints in the input image using the SURF detector.
std::vector<cv::KeyPoint> keypoints;
mySurf.detect(imageInput, keypoints);
printf("Detected %d points in image.\n", keypoints.size());
```

• We can draw keypoints on the image:

```c++
// Draw keypoints on the image and display it.
drawKeypoints(imageInput, keypoints, imageOutput,
              cv::Scalar(0, 0, 255), cv::DrawMatchesFlags::DEFAULT);
cv::imshow("image", imageOutput);
```

• Next, extract descriptors for all keypoints
Matching points to new image

• “Brute force” matcher – computes the distance between each pair of descriptors
• We find the closest k=2 matches, so that we can apply the “ratio” test (i.e., keep a match only if it is significantly better than the 2\textsuperscript{nd} best match)

```cpp
// Match descriptors between reference and new image.
// For each, find the k nearest neighbors.
cv::BFMatcher matcher(cv::NORM_L2);
std::vector<std::vector<cv::DMatch>> knnMatches;
matcher.knnMatch(
    descriptors2, // These are the "query" descriptors, in the new image
descriptors1, // These are the "training" descriptors, from reference image
knnMatches, // Output matches
2); // Value of k (we will find the best k matches)
```
Fitting a homography

- Fit a homography transform to the tentative matches
- Use RANSAC to eliminate outliers
- Return a mask to show which points are inliers
  - i.e., inliersMask[i] is true if point i is an inlier

```cpp
// Find homography matrix and get the inliers mask.
std::vector<unsigned char> inliersMask(pts1.size());
cv::Mat homography = cv::findHomography(pts1, pts2,
    cv::FM_RANSAC,
    5, // Allowed reprojection error in pixels (default=3)
inliersMask);
```