**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU ELEKTROTEHNIČKI FAKULTET**

**Sveučilišni studij**

**SIFT**

**Vježba 4**

**Antonio Lončar**

**Osijek, 2015.**

# Zadatak i rješenje

Omogućava se korisniku da mišem označi dio slike koji sadrži objekt od interesa, nakon čega se detektiraju SIFT značajke za označeni dio prve slike i SIFT značajke za cijelu drugu sliku. Treba označiti i prikazati značajke dobivene na obje slike te pravcima povezati slične značajke dobivene usporedbom. Klikom na četvrti gumb “**RANSAC**” pokreće se algoritam temeljen na RANSAC-pristupu koji na temelju geometrijskih ograničenja izbacuje krivo sparene značajke dobivene.

## Rješenje

Značajke na slici i modelu, odnosno roi-u su detektirane pomoću *SiftFeatureDtetector* objekta i spremljene u vektor *KeyPoint*-a. Nakon toga pomoću SiftDescriptorExtractor objekta su određeni desktriptori modela i scene koji za ulaz prima vektor *Keypoint*-a.  
Sparivanje deskriptora modela i scene je izvedeno sparivanjem deskriptora kojima je *udaljenost*.

Algoritam je sljedeći:

* Izaberi n-ti redak model deskriptora
* Prođi kroz scene deskriptor i napravi razliku svakog rednka sa n-tim redkom model deskrpitora
* Odredi L2-normu, redak koji ima najmanju vrijednost spari sa n-tim redkom model deskriptora
* Povećaj n-ti redak model deskriptora za jedan
* Ponavljaj sve dok ne dođeš do kraja model deskriptora.

U sklopu ovoga labosa napravio sam funkciju vector<DMatch> L2paring(Mat modelDesc, Mat sceneDesc) koja je izvršavala sparivanje, ali kasnije sma shvatio da OpenCV ima gotov objekt koji radi sparivanje pono brze i efikasnije. Klasa za sparivanje deskriptora je *BFMatcher* i treba u konsturktoru objekta naznačiti da radi *NORM\_L2* sparivanje*.*

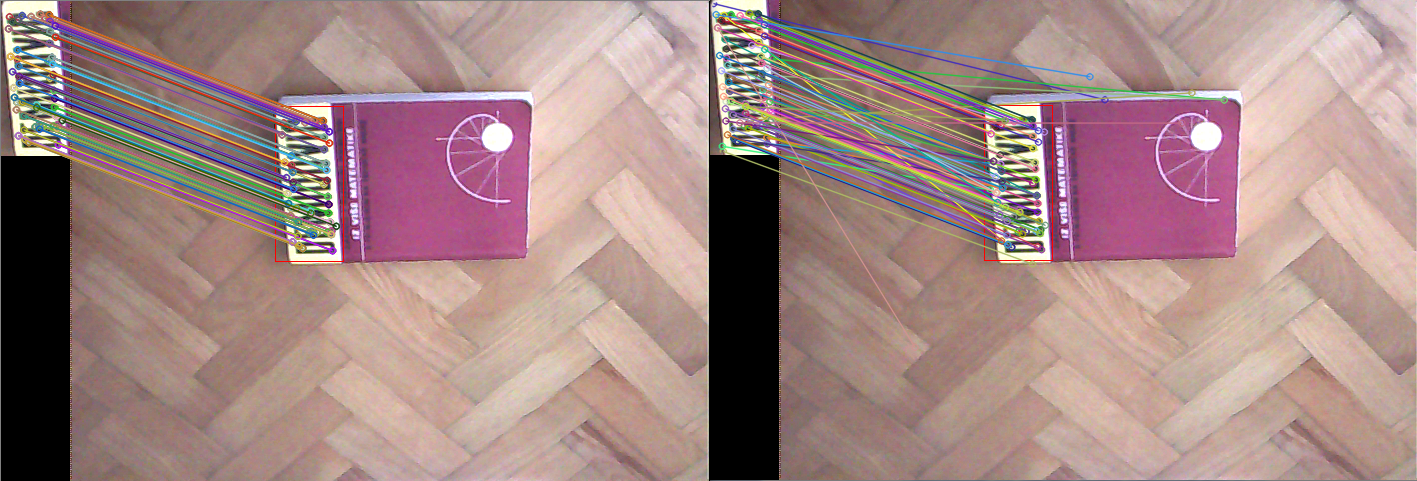
Ransac funckija koja izbacuje krivo sparene značajke ima sljedeći oblike:

void ransac(vector<DMatch> matches, vector<KeyPoint> modelKeypoints, vector<KeyPoint> sceneKeypoints, Mat\* ita, double\* alf, double\* sig, vector<DMatch> \*ransacMatches)

alrumenti funckije su sljedeći:

* matches – parovi koji imaju krivo sparenih značajki
* modelKeypoints – Keypoint-i modela
* sceneKeypoints – Keypoint-i
* ita – varijabla gdje se sprema translaciski vektor
* alfa – varijabla gdje se sprema kut za rotacijsku matricu
* sig – faktor skaliranja
* ransacMatches – varijabla gdje se sprema novi vektor sa izbačenim krivim parovima

Pošto *ransac* funkcija vraća vektor DMatch sparenih parova isto kao što vraća i OpenCV objekt za sparivanje, crtanje tih parova možemo izvesti OpenCV funkcijom *drawMatches*. Na sljedečoj slici možemo vidjeti sa lijeve strane sparivanje nakon ransac funkcije, a sad desne strane sparivanje bez ransac funkcije.



Slika 1.1. Sparivanje značajki modela i scene.

# Ransac funkcija

void ransac(vector<DMatch> matches, vector<KeyPoint> modelKeypoints, vector<KeyPoint> sceneKeypoints, Mat\* ita, double\* alf, double\* sig, vector<DMatch> \*ransacMatches)

{

vector<DMatch> bestMatch, tempBestMatches;

int size = matches.size(), r1, r2;

int nOfIteration = round(log(1 - 0.99)/log(1 - pow( 1 - 0.7, 2))); //0.99 - željena vjerojatnost odabira ispravnih parova; 0.7 - procjenjeni postotak neispravnih parova u matches

double epsilon = 2;

Mat mj(2, 1, DataType<double>::type), mi(2, 1, DataType<double>::type), mk(2, 1, DataType<double>::type), ml(2, 1, DataType<double>::type);

double alfa, sigma, sigmaStar;

Mat cd2(2, 1, DataType<double>::type), cd4(2, 2, DataType<double>::type);

Mat matDeltaUV(2, 2, DataType<double>::type);

Mat vecDeltaUV(2, 1, DataType<double>::type);

Mat ItA(1, 2, DataType<double>::type);

Mat Ralfa(2, 2, DataType<double>::type);

Mat wstar(1, 3, DataType<double>::type);

int model1, model2;

int scena1, scena2;

if (size > 0)

{

for (int i = 0; i < nOfIteration; i++)

{

do{

r1 = rand() % size;

r2 = rand() % size;

} while (r1 == r2);

model1 = matches.at(r1).queryIdx;

scena1 = matches.at(r1).trainIdx;

model2 = matches.at(r2).queryIdx;

scena2 = matches.at(r2).trainIdx;

mi.at<double>(0, 0) = modelKeypoints.at(model1).pt.x;

mi.at<double>(1, 0) = modelKeypoints.at(model1).pt.y;

mj.at<double>(0, 0) = sceneKeypoints.at(scena1).pt.x;

mj.at<double>(1, 0) = sceneKeypoints.at(scena1).pt.y;

mk.at<double>(0, 0) = modelKeypoints.at(model2).pt.x;

mk.at<double>(1, 0) = modelKeypoints.at(model2).pt.y;

ml.at<double>(0, 0) = sceneKeypoints.at(scena2).pt.x;

ml.at<double>(1, 0) = sceneKeypoints.at(scena2).pt.y;

if (abs(norm(mi - mk) - norm(mj - ml)) <= epsilon)

{

vecDeltaUV = ml - mj;

matDeltaUV.col(0) = mk - mi;

matDeltaUV.at<double>(0, 1) = -matDeltaUV.at<double>(1, 0);

matDeltaUV.at<double>(1, 1) = matDeltaUV.at<double>(0, 0);

cd4.col(0) = matDeltaUV.inv()\*vecDeltaUV;

cd4.at<double>(0, 1) = -cd4.at<double>(1, 0);

cd4.at<double>(1, 1) = cd4.at<double>(0, 0);

cd2 = cd4.col(0);

alfa = atan2(cd2.at<double>(1, 0), cd2.at<double>(0, 0));

sigma = sqrtl(powl(cd2.at<double>(1, 0), 2) + powl(cd2.at<double>(0, 0), 2));

ItA = mj - cd4\*mi;

Ralfa.at<double>(0, 0) = cosh(alfa);

Ralfa.at<double>(1, 0) = sinh(alfa);

Ralfa.at<double>(0, 1) = -Ralfa.at<double>(1, 0);

Ralfa.at<double>(1, 1) = Ralfa.at<double>(0, 0);

for (int i = 0; i < size; i++)

{

int modeli = matches.at(i).queryIdx;

int scenei = matches.at(i).trainIdx;

mi.at<double>(0, 0) = modelKeypoints.at(modeli).pt.x;

mi.at<double>(1, 0) = modelKeypoints.at(modeli).pt.y;

mj.at<double>(0, 0) = sceneKeypoints.at(scenei).pt.x;

mj.at<double>(1, 0) = sceneKeypoints.at(scenei).pt.y;

if (norm(mj - (sigma\*Ralfa\*mi + ItA)) <= epsilon)

{

tempBestMatches.push\_back(matches.at(i));

}

}

if (tempBestMatches.size() > bestMatch.size())

{

bestMatch = tempBestMatches;

tempBestMatches.clear();

\*ita = ItA;

\*alf = alfa;

\*sig = sigma;

}

}

}

}

\*ransacMatches = bestMatch;

cd2.release();

cd4.release();

matDeltaUV.release();

vecDeltaUV.release();

ItA.release();

Ralfa.release();

wstar.release();

mi.release();

mj.release();

ml.release();

mk.release();

}