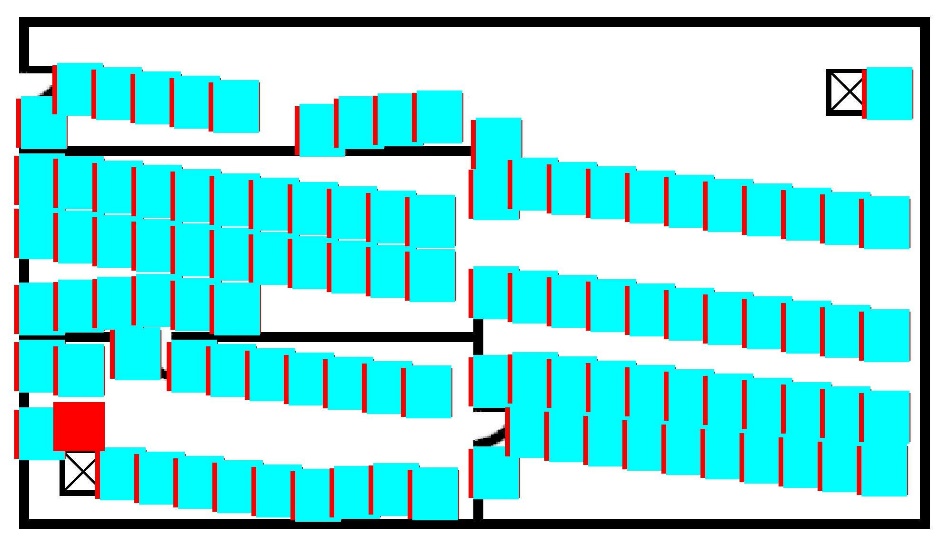
## Detectarea Obiectelor

Detectarea obiectelor are loc in clasa ***DetectObject***al pachetului ***com.imgprocessor.processor*** obiectele detectate in urma procesarii vor fi cele care au si un template asociat (imagine a obiectului respectiv) ex: scari, usi, hidranti, fereste, ascensoare, panouri electrice etc.

Primul pas in detectare acestora, a fost incercarea e le identifica printr-un algoritm de template matching oferit de biblioteca OpenCV care se baza pe raportul de pixeli colorati din imaginea Template si imaginea Scena. Dar acesta s-a dovedit a nu fi unul precis, avand marja de eroare foarte mare, din cauza ca lucram cu imagini bicolore (blueprint-uri)

Marja de eroare era destul de mare, o confirmare este urmatorul test esuat. Algoritmul are un comportament imprevizibil deoarece la modificarea dinamica a coeficientului de matching nu mai lucreaza template-Matching-ul :



Problema acestei metode de matching este urmatoarea : fiecare obiect are o ratie de matching si nu poate fi generalizata pentru toate template-urile, ceea ce creaza ingeuneaza adaugarea a noi template-uri pentru o eventuala noua imagine pentru procesare. Asadar am renuntat la aceasta metoda de detectare a obiectelor.

In urma unor cercetari am descoperit un alt algoritm de matching care sa bazeaza pe keyPoint-uri.

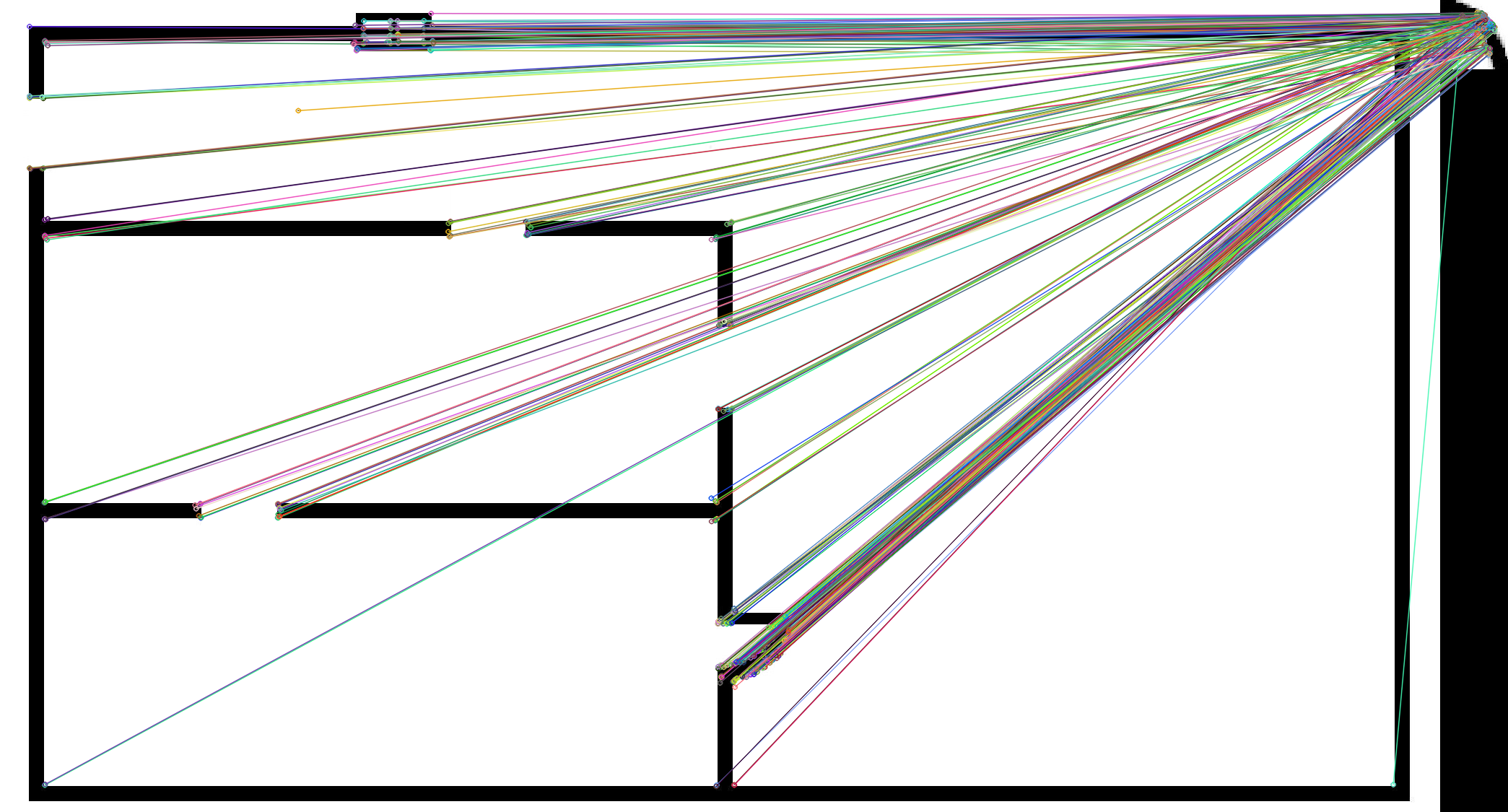
Avem 2 imagini ***Template***-ul si ***Scena***

1. La acest pas determinam keyPoint-uri in imaginea Template cu ajutorul FeautureDetector, apoi le extragem cu ajutorul DescriptorExtractor

featureDetector.detect(objectImage, objectKeyPoints);  
DescriptorExtractor descriptorExtractor = DescriptorExtractor.*create*(DescriptorExtractor.***SURF***);

descriptorExtractor.compute(objectImage, objectKeyPoints, objectDescriptors);

1. Determinam keyPoint-urile pentru imaginea Scena in aceeasi maniera .
2. Dupa apelam algoritmul knnMatch care va potrivi keyPoint-urile care fac match din acele 2 imagini



1. Algoritmul gaseste mai multe match-uri unele din fiind incorecte de aceea ele trebuiesc filtrate, conform unor calcule pentru “Clusterizare”. Le alegem pe cele mai apropiate intre ele.

float nndrRatio = 0.7f;

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

{

MatOfDMatch matofDMatch = (MatOfDMatch) matches.get(i);

DMatch[] dmatcharray = matofDMatch.toArray();

DMatch m1 = dmatcharray[0];

DMatch m2 = dmatcharray[1];

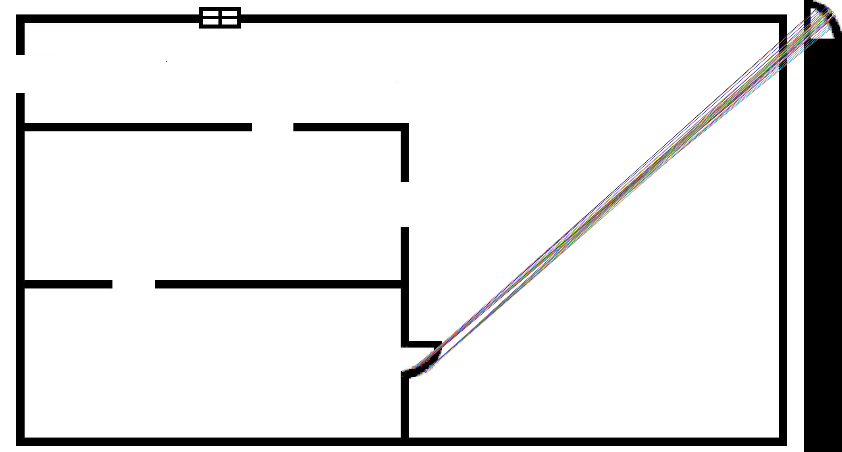
if (m1.distance <= m2.distance \* nndrRatio)

{

goodMatchesList.addLast(m1);

}

}



1. Avand aceste key-point-uri care se potrivesc putem determina coordonatele template-ului in cadrul scenei mari

(obs) Detectarea acestei zone e posibila daca au fost gasite cel putin 4 (pentru precizie am pus 7) KeyPoint-uri.

* 1. Daca imaginea Template este prea mica, algoritmul nu va da match deoarece nu vor fi gasite destule keyPoint-uri.

1. Avand coordonatele zonei care da match memoram obiectul gasit si il stergem din Scena

