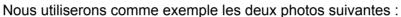
Vision Par Ordinateur TP noté

Le but de ce TP est de calculer la carte de disparité à partir de deux images non calibrées, correspondant a deux points de vue décalées. On peut imaginer ces deux images comme étant celles vues par les deux yeux.

IL EST FORTEMENT RECOMMANDE DE LIRE TOUT L'ENNONCE AVANT DE COMMENCER.





On note I_L et I_R les images correspondant aux yeux gauche et droit, respectivement.

Les étapes sont les suivantes :

- 1. trouver des points de correspondance $\mathbf{P_L}$ et $\mathbf{P_R}$ dans $\mathbf{I_L}$ et $\mathbf{I_R}$
- 2. rectifier les images :
 - a. calculer la matrice fondamentale ${\bf F}$ a partir de ${\bf P}_{{\bf L}}$ et ${\bf P}_{{\bf R}}$
 - b. calculer les matrices de rectification \mathbf{H}_{L} et \mathbf{H}_{R} a partir de \mathbf{F} , \mathbf{P}_{L} et \mathbf{P}_{R}
 - c. rectifier les deux images I_L et I_R en I'_L et I'_R
- 3. calculer la carte de disparité **D** à partir de **l'**_L et **l'**_R

1 Trouver les points de correspondance (5 pts)

Pour cette étape, vous allez créer une fonction *findMatchings*, qui prend en paramètres par référence deux images I_A et I_B , et deux vecteurs de points (std::vector < cv::Point2f >) P_A et P_B .

Utilisez la fonction goodFeaturesToTrack pour trouver une liste de points facilement identifiables dans l'image I_A et les stocker dans une liste tmp_A .

Utilisez ensuite la fonction *calcOpticalFlowPyrLK* pour trouver les correspondances de tmp_A dans l'image I_B et les stocker dans tmp_B .

En vous aidant de status (paramètre de la fonction), ne mettez dans P_A et P_B que les points correctement trouvés.

Faites une fonction *displayMatchings*, qui prend en paramètres vos deux images et leur liste de points, et fait un affichage comme le suivant :



Ou encore mieux (points bonus):



2 Rectifier les images (5 pts)

Pour cette étape, créez une fonction rectify qui prend en paramètres (passés par référence)

les deux images I_L et I_R

- les listes de points de correspondances P_L et P_R
- les images rectifiées l'_L et l'_R

Utilisez la fonction *findFundamentalMat* pour calculer la matrice fondamentale F à partir de P_L et P_R .

Utilisez ensuite la fonction **stereoRectifyUncalibrated** pour calculer les matrices de rectification $\mathbf{H_L}$ et $\mathbf{H_R}$ a partir de \mathbf{F} , $\mathbf{P_L}$ et $\mathbf{P_R}$.

Utilisez enfin la fonction *warpPerspective* pour appliquer les matrices de rectification \mathbf{H}_{L} et \mathbf{H}_{R} sur \mathbf{I}_{L} et \mathbf{I}_{R} et stocker le résultat dans $\mathbf{I'}_{L}$ et $\mathbf{I'}_{R}$.

Les images rectifiées doivent ressembler à celles ci-dessous :





3 Calcul de la carte de disparité

Pour cette étape, créez une fonction *computeDisparity* qui prend en paramètres vos deux images rectifiées, et retourne la carte de disparité.

Utilisez d'abord la classe **StereoBM** pour calculer la carte de disparité dans une matrice de type 16S.

Récupérez les valeurs min et max de la carte de disparité au moyen de la fonction *minmaxLoc*.

Convertissez enfin la carte de disparité en type 8U :

coefficient : 255.0/(maxVal-minVal)offset : -minVal*255.0/(maxVal-minVal)

Le résultat retourné doit ressembler à l'image ci-dessous :



4 Bonus:

Le reste des points sera attribué en fonction de la qualité du code, de l'affichage, et des optimisations pour obtenir une carte de disparité de meilleure qualité.

Informations:

le type des paramètres de calcOpticalFlowPyrLK :

prevlmg, nextlmg : cv::Mat

prevPts, nextImg : std::vector<cv::Point2f>

status : std::vector<uchar>

err : std::vector<float>

Les paramètres par défaut des différentes fonctions donnent de bons résultats. Pour les paramètres à définir soit même, la documentation donne tous les détails nécessaires.

Le main doit ressembler à cela :

```
int main(int argc, char* argv[])
 if(argc < 3)
    std::cerr << "Required arguments: left.jpg right.jpg" << std::endl;</pre>
    return -1;
  cv::Mat image1 = cv::imread(argv[1], CV LOAD IMAGE GRAYSCALE);
  cv::Mat image2 = cv::imread(argv[2], CV LOAD IMAGE GRAYSCALE);
  std::vector<cv::Point2f> points1;
  std::vector<cv::Point2f> points2;
  findMatchings(image1, image2, points1, points2);
 findMatchings(image2, image1, points2, points1);
  showMatchings(image1, image2, points1, points2);
 cv::Mat rectified1(image1.size(), image1.type());
 cv::Mat rectified2(image2.size(), image2.type());
 rectify(image1, image2, points1, points2, rectified1, rectified2);
 cv::imshow("rectified L", rectified1);
 cv::imshow("rectified R", rectified2);
  cv::waitKey();
  cv::Mat disparity = computeDisparity(rectified1, rectified2);
 cv::imshow("disparity", disparity);
 cv::waitKey();
 return 0;
}
```