CPE LYON

Année universitaire

2019 - 2020

Majeure IMI – Partie 3 – 5ETI

Acquisition Calibrage et Reconstruction 3D

TP d'acquisition

Eric Van Reeth

1 Organisation du TP

1.1 Objectifs

Ce TP est une introduction à l'acquisition et au traitement d'un flux vidéo en temps réel.
L'objectif premier est d'interfacer une caméra avec l'ordinateur, puis de réaliser certaines opérations qui seront utiles pour la suite du module.
Pour réaliser ce TP, la librairie python d'openCV sera utilisée.

1.2 Évaluation

Ce TP n'est pas noté, mais il est fortement conseillé de rédiger un compte-rendu. Les scripts implémentés ainsi que les résultats obtenus pourront servir lors de futurs projets ou stages.

1.3 Mise en route

Ce TP s'effectue en binôme par poste informatique (sous LINUX).
Le langage utilisé sera Python, en utilisant les librairies `numpy`, `matplotlib` et `opencv` (`import cv2`), pour les opérations sur les images.
L'utilisation d'IDE (PyCharm par exemple) est conseillée pour faciliter l'implémentation et le debug des codes. Veillez bien à utiliser Python 2.7 (et non pas Python 3).
Pour lancer la bonne version d'openCV (obligatoire) et PyCharm, entrer dans un terminal les commandes suivantes :

```
export PYTHONPATH=/sync/IMI/python_libs
/sync/Robotic/pycharm/bin/pycharm.sh &
```

2 Capture vidéo

2.1 Script de base de la capture vidéo

La première étape va consister à développer le script python pour la capture d'une vidéo à partir d'une webcam.

Rappel : en python, on accède à l'aide des fonctions grâce à la commande `help(`*`NomDeLaFonction`*`)`.

Pour réaliser la capture, on crée un objet de la classe `VideoCapture`. La création de cet objet requiert un argument qui spécifie l'indice de l'appareil réalisant la capture vidéo, où le nom du fichier d'une vidéo déjà existante. Dans notre cas, une seule caméra étant connectée, on passera simplement 0.
L'acquisition en tant que telle se fera dans une boucle `while`, qui appellera la méthode `read()` associée à l'objet `cap` créé précédemment. Cette fonction retourne deux arguments : un booléen valant `True` si la lecture est effectuée correctement, et la frame acquise (en couleur par défaut).
Afin de simplifier les traitements, chaque frame sera convertie en niveaux de gris avec la fonction avant l'affichage.
Pour sortir de la boucle, on attend que l'utilisation presse une touche du clavier (par exemple la lettre `'q'`). Ainsi, on

vérifie à chaque itération l'éventuel appui sur la lettre choisie.

Enfin, on pensera bien à libérer la capture et fermer toutes les fenêtres après la boucle `while`.

Le code d'acquisition et d'affichage sera donc :

```
cap = cv2.VideoCapture(0)
while(True):
        ret, frame = cap.read() #1 frame acquise a chaque iteration de la boucle
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #conversion en n&b
        cv2.imshow('Capture_Video', gray) #affichage

        key = cv2.waitKey(1) #on evalue la touche pressee
        if key & 0xFF == ord('q'): #si appui sur 'q'
                break #sortie de la boucle while

cap.release()
cv2.destroyAllWindows()
```

À ce stade, vous devez être capable d'afficher le flux vidéo provenant de la webcam.

2.2 Enregistrement d'images

Ajouter dans la boucle d'acquisition la possibilité d'enregistrer l'image courante du flux vidéo lors de l'appui sur la touche `'c'`. Veillez à ce que plusieurs images puissent être enregistrées sans écraser la précédente.

2.3 Traitements en temps réel

L'objectif de cette partie est d'appliquer des opérations sur chaque image acquise, et d'afficher le résultat en temps réel.

2.3.1 Affichage d'histogramme

Utiliser la fonction `cv2.calcHist` pour le calcul de l'histogramme. Soyez attentifs au format des variables d'entrée (pensez à regarder l'aide).

Afin de mettre à jour le tracé de l'histogramme à chaque itération de la boucle d'acquisition, l'affichage doit être fait avec les commandes suivantes :

```
plt.plot(hist) #ou hist est la sortie de cv2.calcHist
plt.draw() #execute l'affichage
plt.pause(0.0001) #delai necessaire a l'affichage
plt.cla() #evite la superposition des courbes
```

2.3.2 Affichage des contours

Utiliser la fonction `cv2.canny` pour effectuer une détection de contours, et afficher le résultat à chaque itération de la boucle d'acquisition.

3 Application d'une transformation affine

Il s'agit maintenant d'appliquer en temps réel des déformations géométriques sur l'image acquise. OpenCV permet à la fois l'implémentation de transformations linéaires, affines et non linéaires.
Pour le cas de transformations affines, on spécifiera la matrice M (2×3), contient les paramètres de transformation :

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1}$$

M contient à la fois les paramètre de translation, d'échelle, de rotation (angle et centre) :

$$M = \begin{pmatrix} \alpha & \beta & (1-\alpha)c_x - \beta c_y + t_x \\ -\beta & \alpha & \beta c_x + (1-\alpha)c_y + t_y \end{pmatrix} \tag{2}$$

avec $(c_x,\, c_y)$ les coordonnées du centre de rotation, $(t_x,\, t_y)$ le vecteur translation, s le facteur d'échelle, et :

$$\begin{aligned} \alpha &= s.\cos\theta \\ \beta &= s.\sin\theta \end{aligned}$$

En utilisant la fonction `cv2.warpAffine`, implémenter les transformations permettant de réaliser sur chaque image du flux vidéo acquis :

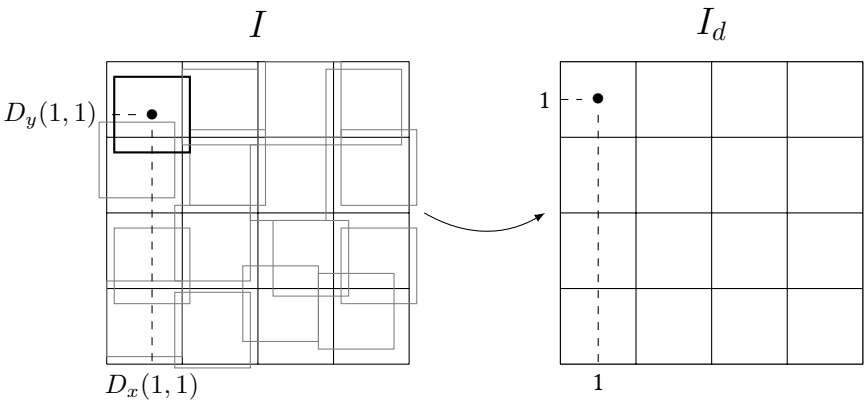
- une translation de 10 pixels en abscisse et 15 pixels en ordonnées
- une rotation de 180 degrés autour du centre de l'image
- une diminution de moitié de la taille de l'image (dans les deux dimensions)

4 Application d'une transformation non linéaire

Par définition, l'application d'une transformation non linéaire ne peut se faire via une opération matricielle. On définit dans ce cas deux matrices de déformation, appelées D_x et D_y, de mêmes dimensions que l'image à déformer. Elles contiennent les nouvelles positions (abscisses pour D_x et ordonnées pour D_y) de chaque pixel de l'image.
Soit I l'image originale et I_d l'image déformée, on aura donc :

$$I_d(x, y) = I(D_x(x, y), D_y(x, y))$$

Notons que I et I_d sont toutes les deux définies sur la même grille de pixels. Autrement dit, la transformation affecte seulement le contenu de l'image, et non son domaine de définition. Les valeurs des pixels de I_d sont obtenues par interpolation des valeurs de I, aux emplacements indiqués par les matrices de déformation :



En utilisant la fonction `cv2.remap()`, appliquer sur chaque image du flux vidéo une déformation Gaussienne à deux dimensions. On prendra le centre de l'image comme point central de la Gaussienne 2D.

Rappel : formule d'une Gaussienne 2D

$$g(x,y) = Ae^{-\left(\frac{(x-c_x)^2}{2\sigma_x^2}+\frac{(y-c_y)^2}{2\sigma_y^2}\right)}$$

Les valeurs de A, σ_x et σ_y doivent être adaptées au format de l'image acquise.