Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document constitue une fraude caractrisée.
Nom, date et signature :

Recherche d'image sur Android

XU Chi
·
Supervisé par : Jean-Pierre Chevallet

Juin 2015

Résumé Ce rapport décrit une application sur Android, une application mobile de recherche d'image appliqué à la recherche de photographies de peintures. Cet application repose sur l'utilisation de OpenCV sur le plateforme Android : generation des caractéristiques qui représente les points clés d'une image et comparaison entre l'image requête et l'image originale. Cet application nous permet de reconnaitre une image photographié parmi les images stockées localement et d'ajouter une image dans la base de données.

Mots-clés Images Retrieval, OpenCV, Android

Abstract This document describes an application on Andorid, an application of searching image, which applied in search of paintings. This application is based on using OpenCV on the Android platform : generation of the features represented the key points of an image and comparaison between the required image and the original image. This application allow us to recognize an image photographed through the images stored locally and it is possible to add new image in the data base.

Keywords Images Retrieval, OpenCV, Android

Chi.XU
UJF IMAG
Tel. : +33 650247913
E-mail: xuchi9005@hotmail.com

1 Introduction

La recherche d'information visuelle consiste à indexer des images, les requêtes pouvant être du texte ou une image. Pour une requête visuelle, l'application doit retrouver les images les plus similaires à l'image requête en se basant sur ces caractéristiques visuelles. ces caractéristiques peuvent être détecté par ORB (Oriented BRIEF) de OpenCV. La recherche d'une image est simple, tout d'abord, nous indexons la collection des images et leur descripteurs, chaque image correspond à un ficher de descripteur, puis nous enregistrons tous ces corpus dans notre mobile lorsque la première fois nous démarchons notre application. Utilisateur peut prendre une photo comme une requête pour chercher. Et puis, l'application a reçu cette image de requête et commence à extraire les caractéristiques visuelles. Pour cela nous devons chercher les points clés d'une image avec FAST qui est stable et échelle. Pour chaque point clé, nous utilisons ORB qui est binaire et indépendant de l'échelle pour calculer les descripteurs binaires. En suite, pour l'interrogation, nous avons comparé la distance de Hamming entre les descripteurs de la requête et les descripteurs nous avons enregistré dans mobile. La distance de Hamming permet de quantifier la différence entre deux séquences de symboles. Nous avons défini un seuil pour cribler le nombre de paire de la distance de Hamming. Plus le nombre est élevée, plus la corrélation entre les deux images. En fin l'application retourne le resultat qui a le plus grand nombre. Ce resultat est le plus similaire image d'une requête. En situation de mobilité, une requête visuelle est plus adaptée qu'une description textuelle. Il suffit en effet de prendre un objet en photo, pour lancer une requête à propos de cet objet. Et en effet, l'image est une façon qui peut permet d'accéder à de l'information relative à l'objet. L'applicaiton peut affichier en detaillé les informations d'une image requête avec le nom d'image retourné.

Nous avons développé dans le cadre du projet CLICIDE, avec les sociétés Globe-VIP et Ophrys, une application de guidage et d'accès à de l'information à l'intérieur d'un musée. Le projet CLICIDE vise à permettre à un utilisateur d'accéder, en cours de visite, à des informations sur les oeuvres du musée. Il peut également assister à la visite et au guidage à l'intérieur du musée, en suggérant un parcours des oeuvres et des salles. Les contraintes fortes du musée (pas de borne WIFI, pas de codes, de marqueurs explicites, etc) ne laissent que l'image comme capteur d'information et de localisation.

2 Contexte

L'accès à de l'information en mobilité par l'image s'est développé avant la disponibilité de terminaux portables performants.

Il y a plusieurs algorithmes pour calculer les caractéristiques, par exemple BRIEF, SIFT, SURF, ORB, etc. Dans notre application, nous avons utilisé ORB présenté dans un article (Ethan et al., 2011). Comme le titre de cet article l'indique, ORB est une bonne alternative aux SIFT et SURF dans le coût de

calcul et la performance. ORB est essentiellement une fusion de détecteur de point-clé FAST et BRIEF descripteur avec de nombreuses modifications pour améliorer la performance : d'abord il utilise FAST pour trouver des points-clés, puis applique Harris mesure de coin pour trouver les N premiers points et utilise également la pyramide pour produire multi-échelle caractéristiques, en suite il calcule l'orientation de chaque coin avec l'algorithme de Intensity Centroid et la appliquer à BRIEF, en fin ORB exécute une recherche pour trouver ceux qui ont une forte variance. Nous avons utilisé BruteForce matcher pour comparer les deux ensembles de descripteur. Pour chaque descripteur dans le premier ensemble, ce matcher trouve le descripteur le plus proche dans le deuxième ensemble en essayant chacun.

Pour calculer et comparer les descripteurs, il faut compiler OpenCV sur Android. OpenCV est une bibliothèque graphique libre, spécialisée dans le traitement d'images en temps réel. Il fournit plusieurs façons pour représenter une image avec descripteur. Nous pouvons essayer JavaCV qui est une bibliothèque encapsulé qui fournit un accès efficace à l'intérieur de C++ natif via Java. Mais pour structurer plus claire et débuter plus facile, nous avons choisi d'écrire une bibliothèque qui peut appeler OpenCV en C++ natif via JNI. JNI (Java Native Interface),comme nous savons le programme d'Android est codé en Java, et la bibliothèque de OpenCV est code en C++, pour permet application en Java d'appeler la bibliothèque en C++, nous devons utiliser JNI.

Notre approche consiste alors à calculer les caractéristiques puis à indexer chaque image à l'aide de ces caractéristiques. Chaque image génère un document qui consiste à son ensemble de caractéristiques. Lorsque l'application a reçu une requête, il faut simplement comparer avec chaque document de l'image. Dans la suite nous déroulons la réalisation de notre application.

3 Déroulement

3.1 Architecture

Tout d'aboard, nous devons determiner en detaillé la conception de l'architecture afin de mener à terme les projets tous en respectant les délais annoncés au client et avec la qualité demandée. Plus, les choix spécifiques de conception de l'architecture d'une application mobile impact la conception de toute l'application informatique. Cela est reflété plus évident sur Android. Par exemple l'évolution entre thread de UI et de la tâche, la vie cycle entre les activités, l'ordre d'exécution de l'activité. Ce projet a divisé en trois parties :

EyeEye : ce qui traite tous les opérations de UI, reçoit les paramètres de capteur et affiche les résultats obtenu. Tous sont realisé en Java.
FonctionLib : un lien entre les deux autres parties, pour permet application en Java d'appeler la bibliothèque en C++ natif.

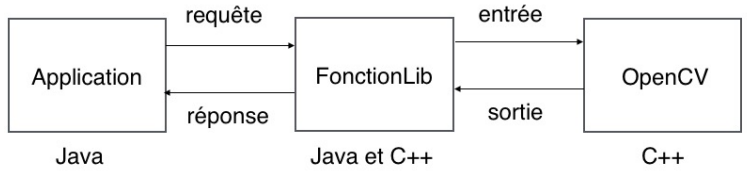

Figure 1 Relation entre trois parties

OpenCV : une bibliothèque graphique libre en C++ natif, spécialisée dans le traitement d'images en temps réel.

La classe de FonctionLib effectue le lien entre la classe EyeEye et la bibliothèque OpenCV. Elle reçoit une requête et la transmet. Après le traitement de la recherche, elle retourne le résultat obtenu. Tout les processus d'exécution peut être représenté par la figure1.

3.2 Classes détaillées et leur fonctions

Nous presentons en détaillées toutes les classes et leur fonctions. Il y a quatre classes pricipales dans notre projet :

Collection : Elle affiche catalogue de tous les images en collection et permet utilisateur de supprimer une image que il ne veut pas.
Gamera : Elle est l'accès principal à l'application. Elle prépare les caractéristiques lorsque la première fois nous démarchons notre application et génère l'interface de l'utilisateur. Et puis, elle permet utilisateur de prendre une photo comme une requête pour chercher ou pour ajouter. Pour la réponse, elle traite le resultat reçu depuis la Classe de FonctionLib et affiche en détaillé les informations concernant cette requête .
Présentation : Cela concerne l'accès à la présentation détaille d'une image, à fournir en réponse à une requête. Cette présentation est des informations de contenu en format XML stocké locale ou met en serveur(en cas de connecter l'internet). Elle est capable de analyser les XML fichier et afficher les informations sur mobile.
Fonction : C'est une classe consiste à deux interfaces de fonction native. Elle permet d'appeler la fonction de OpenCV en code C++. La première fonction est l'extraction des caractéristiques visuelles des images pendant la phase de préparation. Elle génère les caractéristiques avec ORB pour chaque image et les écrit dans un fichier nommé en format TXT. Chaque image correspond à un seul fichier avec le même nom. Cela est rapide et facile pour retourner le nom du résultat. La deuxième fonction est de chercher une requête dans la collection des images en comparant les caractéristiques des images avec BruteForceMatcher. Cette fonction retourne le plus similaire image après filtre les caractéristiques invalides.

3.3 FAST et ORB

ORB commence par la détection des points clés avec FAST dans une image. FAST se base sur la valeur de gris autour un pixel. La formule est comme suivant :

$$N = \sum_{x \forall (circle(P))} |I(x) - I(P)| > \varepsilon_d \tag{1}$$

ou I(x)est une valeur de gris d'un pixel, I(p) les valeurs de gris autour d'un pixel, Ed est un seuil.

Et puis, ORB cherche les N premiers points utilisant Harris mesure. En raison du manque d'orientation des points clés de FAST, ORB utilise une mesure simple mais efficace pour calculer l'orientation des points : Intensité Centroid qui suppose que l'intensité d'un point est décalé par rapport à son centre, et ce vecteur peut être utilisé pour imputer une orientation. Les moments sont définis par :

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{2}$$

et avec ces moments nous pouvons trouver le centre de gravité :

$$C = \left(\frac{m10}{m00} \ , \ \frac{m01}{m00} \right) \tag{3}$$

Nous pouvons construire un vecteur du centre de coin au centre de gravité. L'orientation du pack est alors simplement comme le suivant :

$$\theta = atan2(m_{01}, m_{10}) \tag{4}$$

ORB remédie à l'inconvénient de BRIEF avec le matrice de la rotation :

$$g_n(p, \theta) = f_n(P) | (x_i, y_i) \in S_\theta \tag{5}$$

ou p(x) est intensité d'un point x, S est le matrice après la rotation. fn définie les caractéristiques comme un vecteur de tests binaires en taille n.

Mais après la rotation de BRIEF, il apparait le problème de variance. Pour résoudre cela, ORB cherche tout les BRIEF de haute variance en utilisant l'algorithme de glouton,

3.4 Interface d'application

Cet application est composé de trois parties principales :

La première partie est pour prendre une photo comme une image de requête. Cette partie permet utilisateur de choisir la fonction (interrigation ou ajouter) et de prendre une photo en touchant l'écran de mobile(figure2).

La deuxième partie est pour afficher la collection des images locales. Cette partie permet utilisateur de parcourir la collection des images ou supprimer ce que il n'aime pas(figure3).

La troisième partie est pour afficher la prestation d'une image de résultat. Cette partie permet utilisateur de lire tous les informations concernant l'image de requête(figure3).

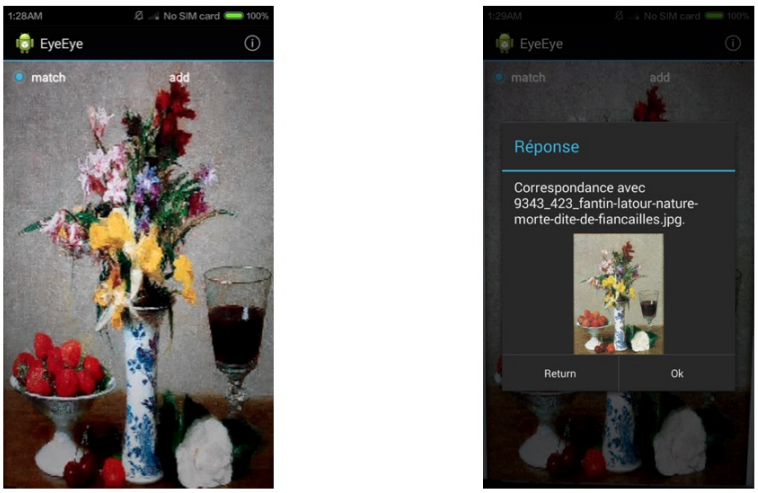

Figure 2 Prendre photo et Réponse d'une requête

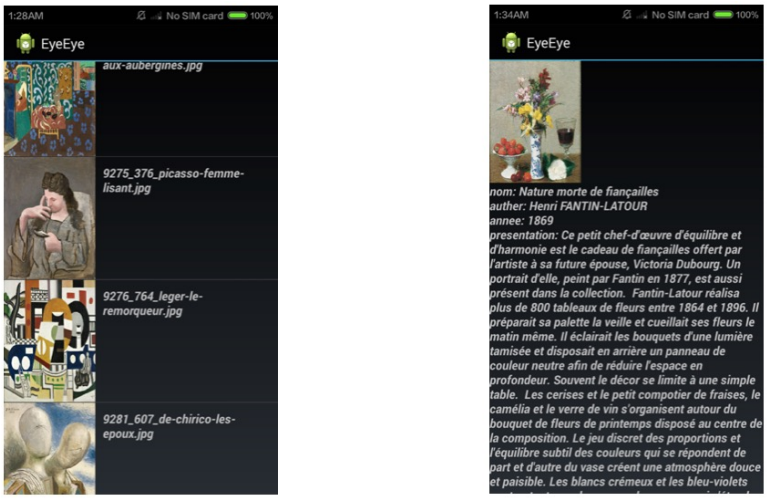

Figure 3 Collection des images et Présentation d'une requête

4 Expérimentation et résultat

4.1 Collection de test

Nous effectuons nos expérimentations sur la collection téléchargé sur la site du musée de Grenoble. Il y a 500 oeuvres de la différente période et du different artiste. Nous avons fait l'expérimentation sous different cas : éclairage normal, éclairage intérieur, rotation de 30 degré. Afin d'évider le cas accidentel, nous

Table 1 Resultat obtenu sous different cas

	Eclairage normal		Eclairage intérieur		Rotation de 30 degré	
Fois	Nb.R.	Taux R.	Nb.R	Taux R.	Nb.R.	Taux R.
1	60	78.33%	50	80.00%	62	80.64%
2	55	85.45%	52	84.61%	56	85.71%
3	53	83.01%	48	75.00%	47	74.46%
TOTAL	168	82.26%	150	79.87%	165	80.27%

expérimentons trois fois pour chaque cas. En fin, la moyenne comme le résultat final.

4.2 Résultats obtenus

Le résulta affiché dans le tableau. Nous pouvons observer les résultats de different cas sont très proche. Cela signifie que ORB surmonte l'inconvénient de sensibilité de noise et l'imperfection de l'invariance de rotation.

5 Conclusion

Ce rapport décrit une application sur plateforme Android pour la recherche d'images en mobilité. Nous avons présenté en détaille la réalisation de cet application, les méthodes que nous avons utilisé dans le processus du développement et l'interface de notre application. Nous avons également expérimenté dans la pratique et analysé le résultat obtenu. Après ce stage, je suis plus familier avec le processus du développement d'applications sur plateforme Android. J'ai eu une meilleure compréhension de reconnaissance d'image avec OpenCV et de utilisation OpenCV en mobile. Un peu difficile pour comprendre l'algorithme concernant la calculation de caractéristique de l'image. Mais cela est facile pour utiliser. Ce stage me semble une bonne opportunité pour allier enseignement théorique et formation pratique.

6 Remerciement

Je remercie Monsieur Jean-Pierre Chevallet, Philippe Mulhem, Nicolas Cubaud pour l'aide et les conseils concernant le mission évoquées dans ce article, qu'ils m'ont apporté lors des différents suivis.

Références

1. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski, "ORB an efficient alternative to SIFT or SURF", Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011.
2. C. Harris and M. Stephens, "A combined corner and edge detector", Proceedings of the 4th Alvey Vision Conference, pp. 147–151, 1998
3. J. Shi and C. Tomasi, "Good Features to Track", 9th IEEE Conference on Computer Vision and Pattern Recognition, Springer, 1994
4. Andrew Willis and Yunfeng Sui, "An Algebraic Model for fast Corner Detection", 2009 IEEE 12th International Conference on Computer Vision. IEEE. pp. 2296–2302, 2009
5. Rosten, Edward, Reid Porter, Tom Drummond, "FASTER and better : A machine learning approach to corner detection", IEEE Trans, Pattern Analysis and Machine Intelligence 32 : 105–119, 2010
6. Rosten, Edward ; Tom Drummond, "Fusing points and lines for high performance tracking", IEEE International Conference on Computer Vision 2 : 1508–1511, 2005
7. Rosten, Edward ; Tom Drummond, "Machine learning for high-speed corner detection", European Conference on Computer Vision 1 : 430–443, 2006