Eliminació d'objectes en moviment a partir d'una seqüència d'imatges per a dispositius amb Android OS

Jairo Vadillo Martín

Resum— A vegades, quan es vol fer una fotografia d'un lloc que s'està visitant (un monument d'una ciutat, una platja etc.), hi ha persones, animals o altres objectes en moviment que impedeixen prendre una bona captura. La solució que es proposa en aquest projecte és prendre una seqüència de fotografies d'un mateix entorn per després detectar el que és comú durant les diferents captures. S'ha implementat aquesta solució en forma d'aplicació per a telèfons mòbils amb sistema operatiu Android. Per tal d'eliminar els objectes en moviment s'han estudiat diferents algorismes per a processar la imatge píxel a píxel fent servir les llibreries natives d'Android (NDK). El fet que el processament es faci píxel a píxel fa que les captures hagin de tenir el mateix enquadrament però amb els telèfons mòbils no es sol utilitzar un trípode. Per aquest motiu, s'ha hagut de dissenyar i desenvolupar també un algorisme d'alineació d'imatges fent servir la llibreria OpenCV4Android. El resultat ha sigut una aplicació que funciona com una càmera convencional i que mitjançant un algorisme de selecció, QuickSelect, calcula la mediana píxel a pixel de les diferents imatges. Aquest mètode és molt robust sempre que les diferents parts del fons siguin visibles a més de la meitat de les imatges.

Paraules clau—Alineació d'imatges, Android, fotografia computacional, mediana, NDK, OpenCV4Android, processament d'imatge.

Abstract—Sometimes, when you want to take a picture of a place (a monument in a city, a beach, etc.), there are people, animals or another moving objects that don't let us take a good picture. The solution proposed on this project is to take a sequence of pictures of the same place to check what is common along the different takes. This solution has been implemented for mobile phones with Android OS. In order to remove the moving objects different algorithms to process the image pixel-by-pixel using Android native libraries (NDK) have been studied. The fact that the image processing its done pixel-by-pixel forces the images to be equally aligned but it's very rare to use a tripod using mobile phones. For this reason, an algorithm to align the images using OpenCV4Android has been also designed and developed. The result it's an application that works as a common camera that calculates the median pixel-by-pixel using a selection algorithm: QuickSelect. This method is very good if the different background parts are visible at least at the half of the images.

Index Terms—Android, computacional photography, image align, image processing, median, NDK, OpenCV4Android.

1 Introducció

Moltes vegades, quan es vol prendre una fotografia d'un emplaçament, sobretot si aquest te algun atractiu turístic, resulta pràcticament impossible realitzar una captura sense persones creuant.

La solució que es presenta en aquest projecte és dissenyar i desenvolupar una aplicació que prenent diverses fotografies de la mateixa escena és pugui detectar el que és comú a la majoria d'imatges i generar així una nova imatge amb el que és estàtic.

L'article s'inicia amb els motius de la tria de la plataforma per a realitzar l'aplicació, els objectius i l'estat de l'art. Actualment ja hi ha alguns programes que ofereixen aquesta funcionalitat per això s'analitzaran abans les seves fortaleses i debilitats. S'explicarà com s'ha desenvolupat el projecte (Metodologia), en quant de temps (Planificació) i què s'ha desenvolupat (Desenvolupament). S'inclouran també els resultats del desenvolupament (Resultats) així com una valoració d'aquests (Conclusions) i els possibles treballs futurs (Treballs futurs).

1.1 Tria de la plataforma

Aquest programa es podria realitzar per a multitud de plataformes però tenint en compte la durada del projecte s'ha de triar només una. L'elecció ha sigut Android i els factors que han ajudat a la elecció han sigut:

- **El context tecnològic actual**: Cada cop més es prescindeix més d'utilitzar una càmera de fer fotos convencional per a utilitzar el mòbil, sobretot si les fotografies són turístiques i realitzades per persones sense grans coneixements de fotografia. Això és degut a que pràcticament tothom te un smartphone i a que les càmeres que incorporen aquests són iguals o a vegades millors que les dedicades.
- **Processament d'imatge**: Els smartphones permeten processar les imatges que es capturen al moment. Això estalvia a l'usuari haver d'esperar a tenir un ordinador per a poder copiar les imatges i després

- *E-mail de contacte: jairo.vadillo@gmail.com*
- *Menció realitzada: Enginyeria de Computació*
- *Treball tutoritzat per: Javier Sànchez (CVC)*
- *Curs 2013/14*

processar-les.

- **Android**: El sistema operatiu Android compta actualment amb una quota de mercat creixent del 70%. A més, al contrari del segon sistema operatiu móvil més utilitzat: iOS, Android permet el desenvolupament d'aplicacions de forma gratuïta.

El fet de desenvolupar l'aplicació per a dispositius mòbils incorpora un problema addicional. Les imatges s'han de comparar píxel a píxel per tal de trobar la part estàtica de la seqüència i per això cal que les diferents imatges tinguin exactament la mateixa alineació. Les imatges capturades per un dispositiu mòbil mai es realitzen amb un trípode i per tant, per molt que s'intenti, diverses imatges no sortiran mai amb el mateix enquadrament. Aquest fet implica que s'haurà de desenvolupar una funcionalitat per a alinear aquestes imatges.

1.2 Objectius i restriccions

L'objectiu principal d'aquest projecte és realitzar una aplicació per a Android que funcioni com una càmera però que prengui diverses fotografies de la mateixa escena per després generar una sola imatge amb els elements que han estat estàtics. Les condicions o restriccions de funcionament seran les següents:

- Els objectes en moviment s'han de desplaçar de forma suficient com perquè les oclusions del fons no siguin majoritàries al conjunt d'images.
- El conjunt d'objectes en moviment han de permetre veure diferents parts del fons a cada imatge.
- Per tal d'alinear les imatges correctament els objectes en moviment no poden ser la part majoritària a la imatge ja que s'han de poder seleccionar punts de referencia entre les imatges.

S'ha de tenir en compte que treballant en un dispositiu mòbil no es disposa de tota la memòria ni tota la capacitat de processament que es desitjaria per això s'ha de trobar una proporció adequada entre la qualitat de les imatges a processar i el temps de processament.

1.3 Estat de l'art

Actualment hi ha diverses companyies que estan implementant sistemes amb un funcionament semblant però de formes molt variades:

- **Adobe Photoshop™**: Conegut com el millor software d'edició d'imatges ja fa temps que incorpora la opció *Auto-Align,* que fa una alineació d'un conjunt d'imatges, i la opció *Median* que selecciona els píxels més repetits a les imatges. El seu funcionament és molt bo però només funciona per a ordinadors i el seu preu es elevat.
- **Google+ Auto Awesome** [1]**:** Si es fa servir la copia de seguretat automàtica d'imatges des d'un dispositiu mòbil l'aplicació Auto Awesome pot afegir els efectes que trobi convenients de forma automàtica. En el cas de trobar més de 3 imatges aplicarà l'efecte *Eraser* que esborrarà els objectes en moviment. La desavantatge és que hem d'esperar a que les imatges s'hagin carregat als servidors de Google i a que l'efecte s'apliqui, ja que no es pot fer manualment. A més les imatges són de molt baixa qualitat.
- **HTC Zoe** [2] **& Samsung Galaxy S4 Camera** [3]**:** Ambdues aplicacions venen preinstalades als dispositius de les dues marques i funcionen de manera molt similar. El seu funcionament és bo tot i que en comptes d'esborrar els objectes en moviment de forma automàtica són els usuaris els qui han de decidir què esborrar.

A partir de les aplicacions que hi ha al mercat es poden afegir nous requeriments per tal de que aquest projecte sigui viable: l'aplicació ha de funcionar als dispositius Android sense cap restricció menys el fet que comptin amb una càmera i el processament d'imatge (eliminació d'objectes mòbils) ha de ser automàtic.

2 METODOLOGIA

És difícil trobar una metodologia de desenvolupament de software que tingui en compte només dos actors: client (professor) i programador (alumne). Per això, en comptes de triar una metodologia i seguir-la estrictament, es va triar Rapid Application Development (RAD) [4] i es van afegir algunes modificacions.

RAD és una metodologia que segueix l'SDLC (Software Development Life Cycle) i no una metodologia àgil, ja que en aquest cas els requeriments estan clars i són molt poc o gens variables. Aquesta metodologia es basa en reduir la planificació a favor d'un ràpid prototipatge i en generar la documentació mínima necessària per tal de poder dedicar el máxim temps al desenvolupament.

Tot i que no consta dins de la metodologia RAD el desenvolupament dels prototips es va dividir en quatre fases:

- Cerca de possibles solucions
- Implementació de la que sembla la millor solució al problema.
- Test de la solució.
- Validació dels resultats. En aquest punt s'ha de decidir si la solució es suficient bona o s'ha d'implementar una altre.

Per poder treballar amb prototips amb més llibertat el projecte es va dividir en tres parts o tasques ben diferenciades: Interficie, processament de les imatges per esborrar els elements en moviment i alineació de les imatges. A partir d'aquestes tasques es van generar els següents projectes:

- **Moving Objects Eraser**: L'aplicació que conté la interfície i sobre la qual es van anar afegint les demés funcionalitats a mesura que passaven els tests i quedaven validades.
- **Test1**.m: Funció de MATLAB destinada a trobar la millor solució a l'eliminació d'objectes en moviment.
- **MeanTest**: Projecte per aprendre el funcionament de l'NDK d'Android [5] i testejar les solucions a l'eliminació d'objectes en moviment.
- **ImageAlign**: Amb aquest projecte es va treballar per trobar la millor forma d'aliniar les imatges.

Pel que fa la documentació es va generar un diari de

canvis del projecte que més tard es va substituir per un repositori de codi fent servir els comentaris als commit com a informe de canvis. A més, els informes parcials que s'adjunten al dossier ajuden al seguiment del projecte en diferents punts temporals.

3 PLANIFICACIÓ

Per tal de fer una planificació més acurada es va fer servir una estratègia bottom-up desglosant les tres tasques principals en tasques més petites. Amb questes tasques més petites es va poder calcular més fàcilment el temps de realització per després, sumant les hores, calcular els temps per a cada tasca. Aquest desglossament de tasques es pot veure a l'apèndix A1.

A la taula EvoPlanificacióShort 1 es pot veure un resum de l'evolució de la planificació per tasques.

Evolució de la planificació	*Inicial*	*Set. 10*	*Final*
Interfície	35h	24/29h	29h
Alineació de les imatges	40h	40h	86h
Processament d'imatges	50h	31h	46h
Unificació de tasques	20h	15h	24h
Documentació	35h	30h	50h
TOTAL	**180h**	**140-5h**	**235h**

EvoPlanificacióShort 1

La planificació inicial seria força encertada respecte la final de no ser perquè el temps de la tasca d'alineació d'imatges es va duplicar. Aquesta tasca, que el inicialment semblava trivial, és va convertir en una de les parts més complexes. Els principals factors que van fer variar més el temps d'aquestes tasques van ser: la cerca de llibreries, la configuració d'aquestes i el procés de cerca del millor algorisme (detallat a la secció desenvolupament).

Una altre variació es va produir pel fet que les subtasques fins a la setmana 10 es van produir amb fluïdesa i això va fer que la replanificació a aquella data fos massa optimista i es reduïssin les hores.

4 DESENVOLUPAMENT

A aquesta secció es descriuran els processos per a generar les dues funcions més importants: l'eliminació d'objectes en moviment i l'alineació d'imatges. No s'entrarà en detall del desenvolupament de l'interfície que s'explicarà directament a l'apartat de resultats.

4.1 Eliminació d'objectes en moviment

Per a la realització d'aquesta tasca, i com ja s'ha indicat a l'apartat metodologia, es van decidir realitzar dos projectes: un prototip en MATLAB i la versió definitiva, un cop triat l'algorisme, amb l'NDK d'Android.

Pel projecte en MATLAB la idea inicial va ser recórrer el conjunt d'imatges píxel a píxel i per cada píxel mirar quins tenien una distancia euclidiana més gran respecte els demés, el problema era que un cop fet això no hi havia cap manera de triar un dels valors que estaven més junts. També es produïa el problema que el mètode no era fàcilment escalable i que si s'afegien més imatges la complexitat s'incrementava de manera que si per 3 imatges s'havien de calcular 3 distancies amb 5 s'havien de calcular 10 i amb n es necessitarien $n!$.

Un cop vist que aquest mètode no donava cap resultat acceptable es va decidir imitar la manera de funcionar d'Adobe Photoshop i la seva funció estadística *Median* [6]. Per fer-ho, es fa un recorregut píxel a píxel de totes les imatges alhora, generant un array amb els valors de cada píxel a la mateixa posició de totes les imatges. A continuació, amb aquest array, es calcula la mediana (funció *median* a MATLAB), és a dir, s'ordenen el píxels pel seu valor i es tria el que ha quedat al mig. Aquest píxel central contindrà el valor més repetit i per tant el valor del que considerarem estàtic. Durant aquest procés la mediana actúa també com a filtre dissimulant les petites variacions que hi pot haver a l'hora d'alinear les imatges.

Per exemple, en una situació on hi hagi cinc imatges d'un edifici però a les imatges 3 i 4 ha creuat una persona, en el punt on ha creuat la persona tindríem els següents valors de píxel:

[100 105 50 200 98]

Utilitzant el procés de mediana els valors s'ordenaríen de forma:

[50 98 100 105 200]

Obtenint al mig el valor *100* corresponent a l'edifici, és a dir, a la part estàtica de la imatge.

Un cop decidit l'algorisme a fer servir i abans de poder començar a programar la solució a Android es va haver de comprovar si realment valia la pena l'esforç d'utilitzar l'NDK (llenguatge C), ja que com s'esmenta a [5]: "*you should understand that* ***the NDK will not benefit most apps****. As a developer, you need to balance its benefits against its drawbacks.*" Per fer-ho, es va crear el projecte *MeanTest* que calcula la mitjana dels valors dels pixels d'un parell d'imatges basat en el projecte que es pot trobar a [7]. Es van desenvolupar dues funcions de mitjana una en Java i l'altre en C i es va cronometrar el temps per a cadascuna d'elles:

PID	TID	Application	Tag	Text
5252	5252	com.example	MEAN_TEST	Time calculateMean: 414
5252	5252	com.example	MEAN_TEST	Time calculateMeanJava: 8718

Els resultats van ser clarificadors: només el fet de recórrer la imatge utilitzant l'NDK va resultar 20 vegades més ràpid que fent-ho en Java.

El fet de trobar la mediana amb MATLAB va ser senzill ja que la funció *median* està implementada. Pel cas d'Android es va haver de buscar un algorisme que funcionés el més ràpid possible i amb el mínim consum addicional de memòria, ja que s'hauria d'executar tantes vegades com píxels d'una imatge.

Per tal de trobar la mediana es pot fer servir un algorisme d'ordenació i després agafar l'element que queda a la meitat. Fent sevir l'algorisme d'ordenació *QuickSort* es podría realizar amb una complexitat mitjana de $n \log n$. És una complexitat força bona però el problema de la mediana es pot resoldre en $O(n)$ fent servir un algorisme de selecció.

Els algorismes de selecció són aquells que busquen el *k-element* més petit d'una llista o array. Com es pot veure a [8] el millor algorisme d'aquest tipus és el *QuickSelect* i com fa referencia el seu nom està basat en la forma de funcionar del *QuickSort*.

Tot i així, abans de realizar la implementació utilitzant *QuickSelect* es va realizar una prova fent servir l'algorisme proposat a [9], *BinMedian*. Suposadament hauria de funcionar més ràpid però pel cas de tenir un màxim de 10 valors trigava més de dos minuts en processar 5 imatges *full-HD*. Això és degut a que es un algorisme molt complexe i ràpid quan es vol calcular la mediana de milions de valors.

Per tal de simplificar encara més el processament per píxel, en comptes de fer la mediana (*QuickSelect)* de cada color RGB es va decidir provar a sumar els tres valors abans de calcular-la. Això implica una reducció del temps a 1/3 obtenint els mateixos resultats. Amb aquesta optimització es van aconseguir processar 5 imatges *full-HD* en només un parell de segons.

Un cop triat l'algorisme es van haver de valorar les restriccions de memoria. El nombre d'imatges per a calcular la mediana ha de ser imparell per poder trobar el valor que queda al mig. Tinguent en compte això es va decidir donar les opcions de capturar 3, 5, 7 o 9 imatges, es podrien capturar més però també s'ha de tenir en compte que l'usuari no estarà molt de temps capturant imatges. Per tal de calcular la mediana s'han de mantenir totes les imatges capturades obertes a memoria principal ja que es necessiten tots els valors alhora. A més, Android, aplica una restricció a la memòria que poden utilitzar les aplicacions, fent que com a molt es puguin utilitzar 250Mb. Per això aquest prototip es veu limitat a 7 imatges si la resolució és de 6MP (3264x1836) o major.

Com aquests resultats ja són definitius es poden veure a l'apartat Resultats.

4.2 Alineació d'imatges

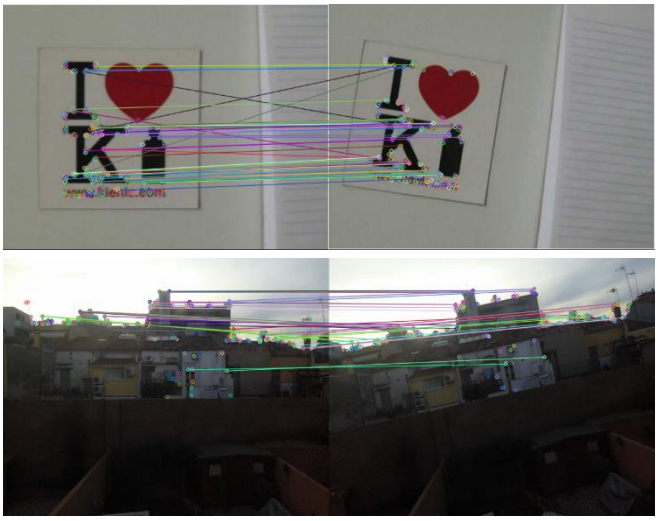
El procés d'alineació d'imatges s'ha d'executar per parells d'imatges prenent una imatge de referència o destinació i una imatge a transformar o origen. Es va decidir seguir un procediment a partir del que es pot veure a [10]:

1. *Feature detection*: Cerca de punts clau a ambdues imatges.
2. *Matching* d'aquests punts clau segons els seus descriptors.
3. Filtratge dels millors punts clau.
4. Generació de la matriu d'afinitat entre les dues imatges a partir dels millors punts clau.
5. Transformació de la imatge origen amb la matriu d'afinitat.

La complexitat dels algorismes que conformen cadascun dels passos es força alta i podrien comportar un projecte sencer estudiar-los un a un. Per això es va decidir utilitzar una llibreria de processament d'imatges.

La primera idea va ser utilitzar BoofCV [11], una llibreria escrita integrament en Java que segons la seva web funciona inclús millor que les natives i que al ser en Java s'adapta perfectament a Android. Després d'instalar i configurar aquesta llibreria es va trobar que la documentació era molt escassa i que encara tenia molts errors ja que estava en fase *Alpha*. Més tard es va decidir utilitzar una de les llibreries més conegudes per aquests propòsits: OpenCV, amb la seva vessant OpenCV4Android [12]. La forma recomanada d'utilitzar OpenCV4Android és inicialitzant la llibreria de forma asíncrona instal·lant una aplicació a banda del projecte anomenada OpenCV Manager. Aquesta forma està recomanada ja que així la llibreria sempre està actualitzada. Tot i així, es va decidir integrar la llibreria al propi projecte per tal de no haver d'intalar cap aplicació de tercers i per tenir controlada la versió de la llibreria que s'estava fent servir.

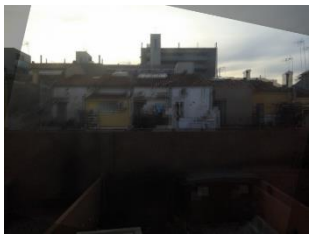
Un cop triada la llibreria els passos 1, 2 i 5 de l'algorisme esmentat anteriorment passaven a ser trivials. Per la detecció de punts clau i l'extracció de característiques es va fer servir l'algorisme *ORB* ja implementat a la llibreria. Es va triar aquest algorisme degut a la seva eficiència [13] i que a banda d'altres com *SIFT* o *SURF* és *opensource*. Pel *matching* es va fer servir l'algorisme de *bruteforce Hamming*. A continuació es pot veure un exemple de *matching*:



La part de triar els punts clau més bons per alinear va ser la més complicada. Els prototips que es van generar van ser:

- **Aleatori**: Per tal de generar una matriu d'afinitat es van triar tres relacions aleatòries entre punts clau de les dues imatges [14]. A partir de les coordenades d'aquests sis punts es generava la matriu d'afinitat fent servir *getAffineTransform*. Aquesta solució donava resultats massa incerts i dolents ja que depenent de quins punts s'agaféssin la imatge es podia alinear bé o malament. Generalment es feia malament ja que *ORB* retorna 500 punts clau i més de la meitat són relacions entre punts clau que no són iguals (e.g. una cantonada d'una finestra amb un marc de fotos). Es va afegir també una modificació per tal de reduir els 500 punts a uns pocs fent servir la distancia de *match*. En una primera iteració es trobava la distancia mínima per després filtrar les relacions segons un factor de la forma:

$$4 * minMatchDistance$$



Com es pot veure a les figures anteriors aquesta solució tampoc donava bons resultats ja que el factor podia variar entre imatges agafant punts de més o de menys.

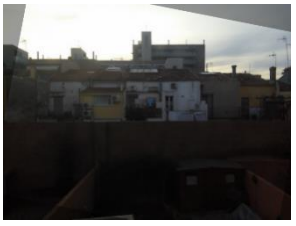
- **Millors 3 punts**: S'agafaven totes les combinacions de 3 relacions, es generava la matriu d'afinitat, s'aplicava sobre la resta de relacions i mitjançant la distancia euclidiana entre les cordenades origen i destí es determinava quina combinació era la millor (amb la mínima distancia euclidiana mitjana). El problema d'aquesta solució residia en el fet que tres relacions "dolentes" generarien una matriu d'afinitat que s'adaptaria a aquestes i reduiria la distancia euclidiana mitjana, inclús més que amb els tres punts millors. Per aquest motiu la opció va quedar desestimada abans de completar-se.
- **Solució final**: El problema principal amb les solucions anteriors era que no es podia determinar quines relacions eren correctes i per tant no es podia comprovar si la matriu d'afinitat s'estava adaptant bé a les cordenades de les relacions correctes o incorrectes. Els passos que segueix la solució final són:

```
M = trobarPrimersNMatches(matches)
Per i=N:4 fer
 Matches = Matches[0:i]
 Aff = generarAfinitat(matches)
 pOrigenTransf = matches.origen*Aff
 dist = calcularDistanciaMitjana(pOrigenTransf, destinació)
 si dist < 1.0
  sortir()
 fisi
fiper
```

El primer pas ordena les relacions per la seva distancia fent servir l'algoritme *Quicksort* [15]. Al ser pocs elements (com a màxim 500) no cal que l'algorisme sigui el millor i per això no es va fer una gran investigació sobre la tria de l'algorisme. Després d'ordenar les relacions s'agafen les primeres 50. Aquest pas ja fa un gran filtratge eliminant les pitjors relacions.

A continuació es van reduint el número de relacions que s'agafen des de les 50 inicials fins a quatre. Es genera la matriu d'afinitat a partir de les cordenades origen i destinació de cada relació fent servir la fórmula dels mínims cuadrats: $a = (X^T * X)^{-1} * X^T * y$ on a es la matriu d'afinitat, X es la matriu amb els punts origen i y són els punts de referència o destinació [16].

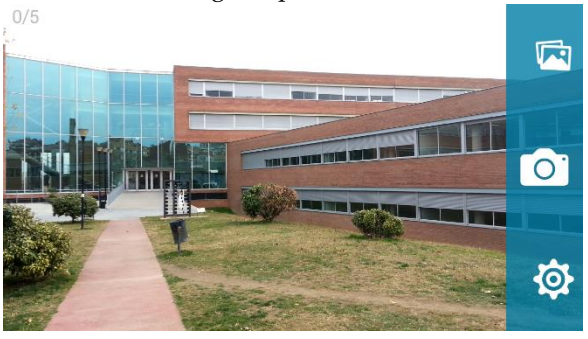
S'aplica la matriu afinitat als punts origen i es calcula la distancia euclidiana mitjana entre els punts origen transformats i els punts destinació o referència (no tots els punts si no els de les relacions seleccionats). La idea inicial era trobar la mínima distancia euclidiana mitjana però es va comprovar que aquest mínim es produïa sempre quan es tenien només 4 relacions. Lògicament la distancia euclidiana mitjana es reduirà quants menys punts hi hagi ja que la matriu d'afinitat s'adapta millor al punts fets servir per generar-la. Per això, es va decidir buscar un llindar (finalment < 1.0) que s'adaptès a tots els casos. La diferència d'un píxel es prou petita com per considerar la imatge alineada i a més s'ha comprovat que funciona bé en casos on es necessiten 4 punts com en altres que en requereixen 50:



5 RESULTATS

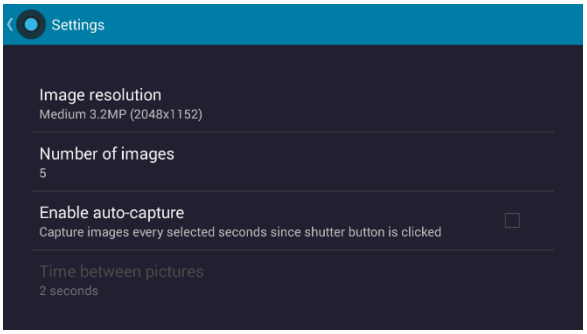
5.1 Interfície

- **Captura d'imatges**: Mostra una vista prèvia del que veu la càmera, el número de fotografies capturades i els botons de capturar, preferències i galeria. Aquesta és la vista principal a la qual l'usuari accedeix a l'iniciar l'aplicació. Des d'aquesta vista l'usuari pot capturar les fotografies que ha indicat a l'apartat preferències. Un cop capturada la primera fotografia aquesta es mostra per sobre de la vista prèvia de la càmera fent que la següent imatge es pugui capturar intentant que el pla es mogui el menys possible. A partir d'aquesta primera fotografia les següents es van alineant respecte a la primera de forma concurrent a la captura. Un cop capturades totes les fotografies l'interficie espera que s'hagin completat les alineacions d'imatges que puguin quedar pendents i tot seguit s'inicia la vista de visualització d'imatges capturades.



- **Visualització d'imatges capturades**: Es mostren totes les fotografies capturades a més dels botons per editar-les i guardar-les. Un cop inciada aquesta vista es fa una crida a la funció de mediana la qual a partir de totes les imatges genera la resultant. El botó d'editar obre la vista de captura d'imatges per tal de poder tornar a capturar una imatge que ha sortit malament, ja sigui per una mala alineació o perquè hi ha masses objectes en moviment.

- **Preferències**: En aquesta vista es poden modificar opcions com el número d'imatges, la qualitat, si es volen capturar imatges de forma automàtica i si és així cada quants segons.



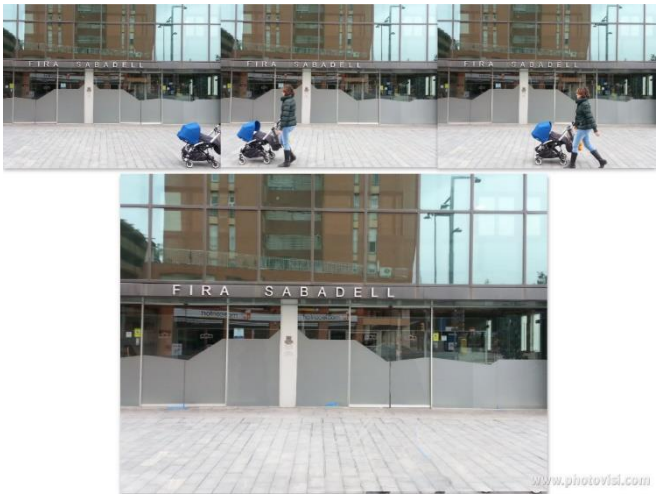
5.2 Alineació d'imatges

S'ha comprovat la fiabilitat de l'algorisme en situacions molt diferents. A banda de les ja vistes a l'apartat de desenvolupament també s'ha inclòs la possibilitat de que un objecte ocupi la major part de la imatge.

Fins i tot en aquest cas els punts que s'han triat al primer filtratge (de 500 a 50) són correctes alineant la imatge segons el fons i no segons l'objecte que s'ha rotat.

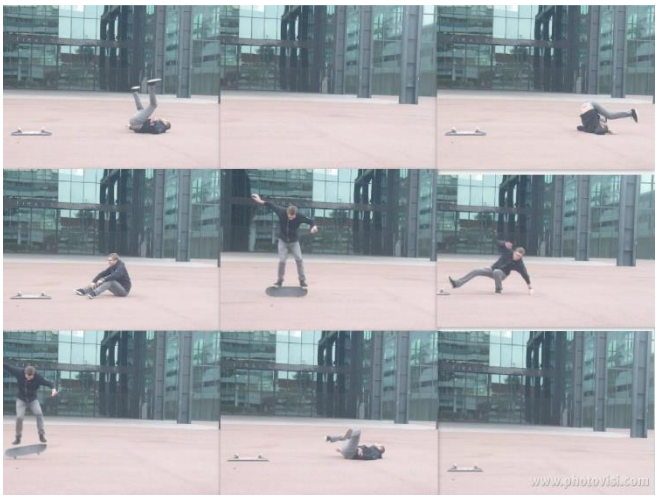
5.3 Eliminació d'objectes en moviment

Els resultats han sigut molts i la majoria correctes per això no s'han afegit totes aquí. A més, resultaria del tot inútil posar només les imatges finals ja que no es pot apreciar si realment s'ha esborrat alguna cosa.



Al continuació veu un exemple d'error on a causa d'un punt de la imatge constantment ocupat per vehicles el resultat deixa una ombra vermella:

Més resultats:



(la imatge resultant es troba a baix a la dreta)

6 Conclusió

Com es pot veure a totes les imatges el resultats són molt bons. S'han aconseguit un mètode d'alineació d'imatges molt robust i resistent a errors. També s'ha tingut en compte que aquests errors serien petits ja que quan s'estan capturant imatges hi pot haver un moviment però no molt gran. També s'ha aconseguit dissimular el temps d'execució fent que l'alineació s'executi concurrentment a mesura que es capturen d'imatges.

En quant a l'eliminació d'objectes en moviment funciona bé sempre i quant hi hagi un desplaçament suficient dels objectes en moviment, com ja s'havia proposat inicialment. Com s'ha vist a l'exemple del carrer si hi ha una part de la imatge coberta per vehicles durant més del 50% de les imatges aleshores crea una petita ombra de diferents colors a aquell punt. A la interfície s'ha donat solució a aquest problema fent que es puguin tornar a realitzar certes imatges si es veu que aquestes, al final, generen algun tipus d'error.

6.1 Treballs futurs

L'aplicació es troba totalment acabada però per tal de posar-la a la botiga d'aplicacións d'Android s'hauria d'obrir una fase de testing en diferents dispositius, ja que fins al moment només s'ha comprovat el funcionament a dos dispositius. També es podria millorar la interfície amb opcions per a compartir les imatges generades. En quant al temps d'alineació d'imatges potser es podria millorar fent servir OpenCV de forma nativa la qual cosa no s'ha fet ja que és incert si milloraria el rendiment i el cost de fer-ho seria molt alt.

Agraïments

A en Javier Sànchez per l'ajuda constant cada setmana.

A en Cristian Rodríguez per la col·laboració que hem tingut al crear la galeria d'imatges així com a la cerca del millor algorisme per alinear les imatges.

A en Daniel Jimenez pels seus consells sobre com desenvolupar la interfície.

I per acabar a tots els que han accedit a deixar ser fotografiats per realitzar les proves.

Bibliografia

[1] Google Inc., «Auto Awesome photos & movies - Google+ help,» 2013. [En línea]. Available: https://support.google.com/plus/answer/3113884?p=photos_auto_awesome&hl=en&authuser=0&rd=1.

[2] HTC, «HTC Zoe™: HD Photos that Move,» HTC, 2014. [En línea]. Available: http://www.htc.com/us/why-htc/zoe/.

[3] E. Lodi, «Samsung Galaxy S4 camera first look: Imaging features,» 10 May 2013. [En línea]. Available: http://connect.dpreview.com/post/2880423642/samsung-galaxy-s4-camera-features.

[4] Projectmanagement.com, «Process/Project RAD - RAD - Rapid Application Development Process,» 2014. [En línea]. Available: http://www.projectmanagement.com/content/processes/11306.cfm.

[5] Android, «Android NDK | Android Developers,» [En línea]. Available: https://developer.android.com/tools/sdk/ndk/index.html.

[6] M. Johnson, «Photoshop Workbench 144: Removing Undesirable Subjects Using Median Stack Mode - Youtube,» 25 February 2013. [En línea]. Available: https://www.youtube.com/watch?v=gomwGghrb_w.

[7] ruckus, «Ruckus notes - Writing a basic image filter in Android using NDK,» 21 February 2012. [En línea]. Available: http://ruckus.tumblr.com/post/18055652108/writing-a-basic-image-filter-in-android-using-ndk.

[8] Comunity, «Selection algorithm - Wikipedia,» 21 January 2014. [En línea]. Available: https://en.wikipedia.org/wiki/Selection_algorithm.

[9] R. J. Tibshirani, «Fast Computation of the Median by Successive Binning,» Dept. of Statistics, Stanford University, Stanford, CA 94305, 2008.

[10] TonySJH, «OpenCV Answers,» 19 November 2013. [En línea]. Available: http://answers.opencv.org/question/24127/findhomography-the-line-come-out-not-around-object/.

[11] P. Abeles, «BoofCV,» 27 December 2013. [En línea]. Available: http://boofcv.org/index.php?title=Main_Page.

[12] OpenCV, «OpenCV4Android SDK,» [En línea]. Available: http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html.

[13] E. R. e. al., «ORB: an efficient alternative to SIFT or SURF,» Willow Garage, Menlo Park, 2011.

[14] OpenCV, «Affine transformations - OpenCV Docs,» December 2013. [En línea]. Available: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/warp_affine/warp_affine.html.

[15] E. G. Hernández, «Programación en Java: Java QuickSort,» 2012. [En línea]. Available: http://puntocomnoesunlenguaje.blogspot.com.es/2012/12/java-quicksort.html.

[16] E. W. Weisstein, «Least Squares Fitting--Polynomial,» Wolfram MathWorld, [En línea]. Available: http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html.

[17] Google Inc., «Android developers: Themes,» [En línea]. Available: https://developer.android.com/design/style/themes.html.

APÈNDIX
A1. Work Breakdown Structure