# Odometría visual inercial con OPENCV/C++

Para curso de PDI - Dictado por MSc. Ing. Félix G. Safar y Ing. Jorge Rafael Osio

CONCIA Bernardo - MITIDIERI Pedro - ZUMARRAGA Agusto

17 de Mayo de 2018

# Contenido

**Resumen**

El siguiente informe explica algoritmos utilizados para realizar Odometría visual inercial. Odometría visual se refiere a la estimación de de movimiento por medios visuales, tanto la translación como la actitud. En este caso es incercial ya que para obtener la posición de la cámara a partir de la filmación obtenida por la misma se utilizan sensores inerciales. Por un lado se ha calculado la actitud, sin necesidad de sensores inerciales y por el otro se desarrollo el codigo para utilizar junto con los sensores inerciales para estimación de posición. El mismo fue realizado como trabajo final para el Curso de postgrado de Procesamiento Digital de Imágenes con OPENCV/C++ dictado en la Universidad Nacional de La Plata dictado por los docentes MSc. Ing. Félix G. Safar y Ing. Jorge Rafael Osio. Se aprovechó la oportunidad para colaborar con los trabajos realizados en el área de Control y Guiado del Departamento de Aeronáutica de la Univerdad Nacional de La Plata.

# 1 Introducción

Con la odometría visual buscamos obtener la posición de la cámara en función de los cambios que se producen en la imagen tomada. Para ello se utiliza el seguimiento de características especificas de la imagen, que deben ser invariantes a los movimientos de la cámara y, en lo posible, a cambios no uniformes en la intensidad de los píxeles. Se realizaron dos programas de los cuales se presenta el código en el repositorio `https://github.com/AControlUNLP/VOPDI-UNLP`. Uno es la aproximación de actitud, un método de odometría visual y luego un algoritmo de odometría visual inercial, para que junto con las mediciones obtenidas por una IMU se puede aproximar la posición. Además el segundo cuenta con una interfaz donde se grafica el movimiento promedio de la imagen. No se tuvieron en cuenta en este trabajo las distorciones generadas por la cámara.

# 2 Algoritmo

## 2.1 Actitud

Para obtener la aproximación de la actitud, es necesario tomar cuadro a cuadro el movimientos de diferentes puntos característicos de la imagen que en este trabajo se eligieron esquinas gracias a la velocidad de los algoritmos que los obtienen. Con el movimiento de los puntos, que hipotéticamente están distribuidos por toda la imagen utilizamos el algoritmo de Niester de 5 puntos[1, 2] con RANSAC para computar la matriz esencial relaciona los puntos correspondientes en imágenes estéreo. De esta matriz podemos obtener R y t, la matriz de rotacion y el vector de traslación. Cabe mencionar que es un método aproximado y se obtiene diferentes soluciones, se toma la de mayor confianza. Con la matriz R que es de 3x3 y con la cual pueden obtenerse los ángulos de Euler. El vector de translación que se obtiene es de módulo 1 por lo que solo nos da la dirección de la traslación y es necesario una medición externa al método.

1. Tomar imágenes en $t_0$ y $t_{-1}$.
2. Detectar los puntos claves en la imagen en $t_0$.
3. Tracking de los puntos de una imagen a la siguiente con algoritmo Kanade-Lucas-Tomasi
4. Si se pierden mas de una determinada cantidad de puntos volver a 2.
5. Calcular matriz esencial con Niester 5 puntos y RANSAC.
6. Estimar R y t a partir de la matriz esencial.
7. Calcular los ángulos de Euler.
8. Sumar los ángulos de Euler al valor inicial (Dead Reckoning).

### 2.2 Traslación (actitud fija)

Este algoritmo funciona si la cámara mantiene su actitud y el valor obtenido sirve de estimación. Si se mide la distacia promedio desde la cámara a la imagen y se estima la actitud también puede calcularse con variaciones de la actitud de la cámara, como en el caso de un drone con la cámara apuntando hacia el suelo. Se coloca en el repositorio ademas una variante en la que se puede utilizar el algoritmo con videos previamente grabados.

1. Tomar imágenes en $t_0$ y $t_{-1}$.
2. Detectar los puntos claves en la imagen en $t_0$.
3. Tracking de los puntos de una imagen a la siguiente con algoritmo Kanade-Lucas-Tomasi.
4. Si se pierden mas de una determinada cantidad de puntos volver a 2.
5. Promedio del movimiento de los puntos en unidad de pixeles.
6. Sumar el desplazamiento al valor inicial (Dead Reckoning).

## 3 Métodos de detección de puntos claves

Existen diversos métodos para encontrar diversas características de las imágenes. Algunas de ellas son líneas, circunferencias, formas geométricas específicas, objetos etiquetados y esquinas. De todas ellas se hizo uso de la última, donde se buscan las esquinas estables dentro de la imagen que permitan su seguimiento entre cuadros. Realizando una busqueda bibliografica de trabajos previos sobre el tema, se encuentra que existe una gran variedad de

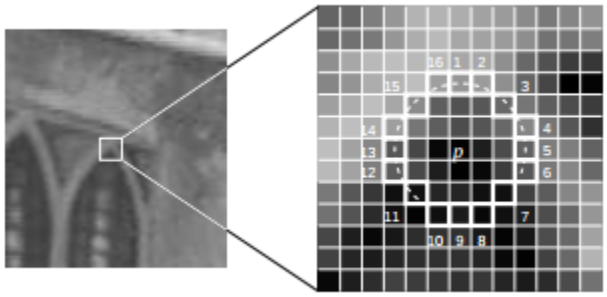


Figure 1: Fast Detector

métodos para encontrar dichas características en una imagen. Algunos de ellos son Harris, SURF, SIFT, FAST[3] y ORBORB[4], entre otros. De todos ellos se eligió el último para realizar la detección de puntos claves. Este detector utiliza las esquinas como puntos claves, las encuentra mediante el algoritmo FAST[3] y las ordena gracias a Harris[5].

### 3.1 FAST

El método de FAST [3] para encontrar esquinas analiza los pixeles alrededor del punto a evaluar. Para discernir entre un punto que es una esquina y otro que no, el algoritmo busca punto a punto en toda la imagen. En cada pixel de estudio crea un círculo de otros píxeles que tienen al primero como centro y realiza una comparación de intensidades. Si existe un arco ininterrumpido de N píxeles cuyas intensidades se encuentran por arriba o abajo de la intensidad del centro más un umbral, en el primer caso, o menos un umbral, para el segundo, entonces dicho pixel es una esquina. El algoritmo es muy eficiente debido a que utiliza un algoritmo de machine learning como es árbol de decisión para evaluar de manera eficiente cada uno de los puntos y descartar rápidamente con menos pasos los puntos que no son esquinas.

La siguiente dirección lleva a la ayuda de OpenCV para este método.

`https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html`

### 3.2 ORB

El algoritmo de ORB[4] utiliza FAST[3] para encontrar los puntos que son esquina dependiendo del umbral elegido y aplica un descriptor BRIEF [6]. Luego, utiliza Harris[5] sobre los puntos encontrados y los ordena según la probabilidad de que sean una esquina. Gracias a ello permite la elección de la cantidad de puntos a mostrar y solo devuelve los N mejores puntos que es de gran utilidad cuando la potencia de cálculo es baja y se necesita

reducir la cantidad de puntos a utilizar.

`https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html`

# 4 Tracking de puntos

El algoritmo que se utilizo para el tracking de los puntos es el de Kanade-Lucas-Tomasi[7, 8]. Fue propuesto con la idea de lidiar contra el problema de alto coste que tenían las técnicas tradicionales. Este algoritmo hace uso de la información de intensidad espacial. Utiliza el gradiente de la imagen para encontrar el mejor posible movimiento que tuvo el punto bajo observación.

# 5 Estimación de la matriz esencial

Esta se calcula a partir de la correspondencia de los punto en las imágenes consecutivas. Esta matriz da la relación entre imágenes obtenidas por una cámara estéreo. En este caso se toman las imágenes consecutivas como si fueran las obtenidas por una cámara estereo para obtener la una aproximación del movimiento de la cámara, que en una cámara estéreo vendría a ser la posición relativa entre una cámara y la otra. La matriz cumple que:

$$(y')^T E y = 0$$

Siendo $y$ y $y'$ los puntos en 3D de la misma escena en coordenadas homogéneas normalizadas en las dos imágenes consecutivas. El camino mas directo para calularlo es utilizar el método de mínimos cuadrados llamado algoritmo de 8 puntos[9]. Un método mas actual y con mejores resultados lo resuelve utilizando solo 5 puntos, este es el método Niester de 5 puntos[1]. Debido a que los métodos de detección y tracking no son perfectos no basta con calcular la matriz esencial con solo 5 puntos sino que el algoritmo utiliza el método RANSAC. Por lo que primero se toman 5 puntos de manera aleatoria y calcula la matriz esencial y verifica si los demás puntos se corresponden(inliers) utilizando la matriz esencial obtenida. Iterativamente realiza esto luego de un número fijo de iteraciones obteniendo la matriz esencial con menor cantidad de puntos sin correspondencia (outliers).

# 6 Estimación de R y t

La matriz esencial puede ser también escrita como:

$$E = R[t]_x$$

Siendo $R$ la matriz rotación y $[t]_x$ la representación matricial del producto cruzado con $t$. A partir de esta ecuación se puede realizar la descomposicion por valores singulares y estimar diferentes posibilidades de las matrices de rotación y traslación sin escala.

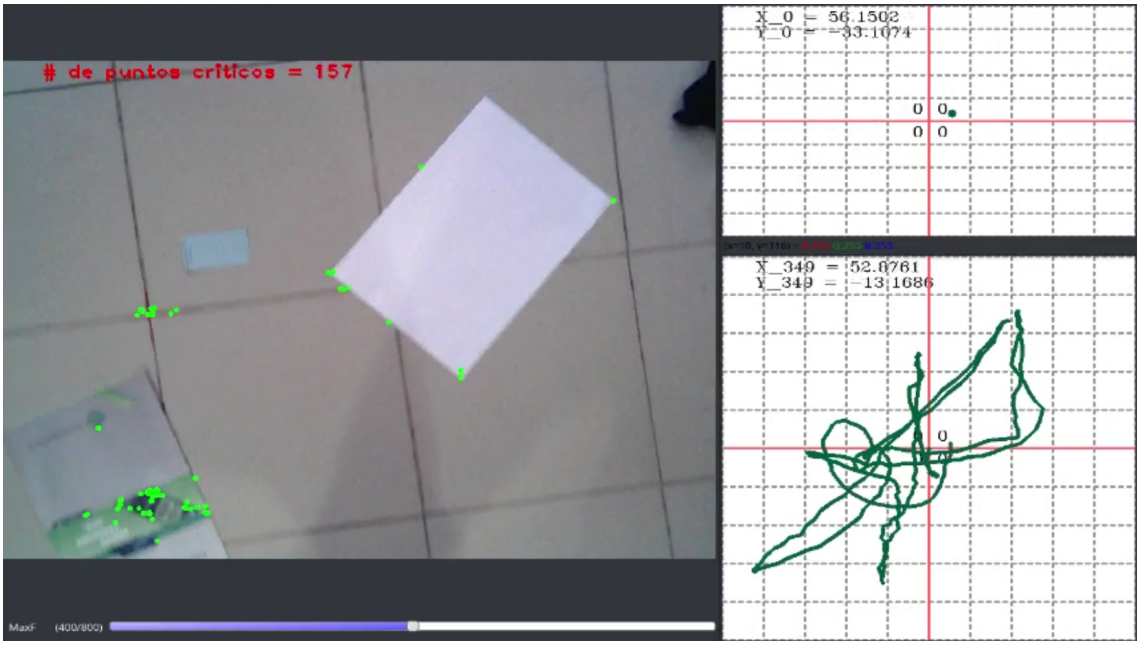


Figure 2: Ensayo 1

## 7 Cálculo de traslación o actitud

En ambos algoritmos desarrollados se calcula la traslación o actitud entre dos imágenes consecutivas, por lo que para obtener la posición actual a partir de la posicion inicial conocida basta con sumar los vectores actitud y traslación que se obtienen en cada iteración.

## 8 Resultados y conclusiones

Ambos algoritmos funcionan y realizan lo indicado. En el algoritmo de actitud se encontraron muchas limitaciones por el costo computacional y la sensibilidad al ruido, cabe mencionar que fue el algoritmo al que menor cantidad de horas fueron dedicadas al encontrarnos con esas limitaciones. El segundo algoritmo es mas robusto y veloz, además cuenta con la ventaja de poder adaptarse a diferentes casos. En las figuras 2 y 3 se visualiza la interfaz del programa con el algoritmo de traslación bajo funcionamiento con dos videos de prueba (no en vivo), los cuales se utilizaron para comparar y realizar modificaciones al algoritmo hasta lograr los mejores resultados, los cuales son compartidos en el apéndice A. El algoritmo presenta un problema crítico cuando la cantidad de puntos que se trackean desciende por debajo del umbral, donde se vuelve a calcular los puntos claves y si el movimiento es considerable en este momento entonces se pierde información y la estimación de posición da resultados erróneos.

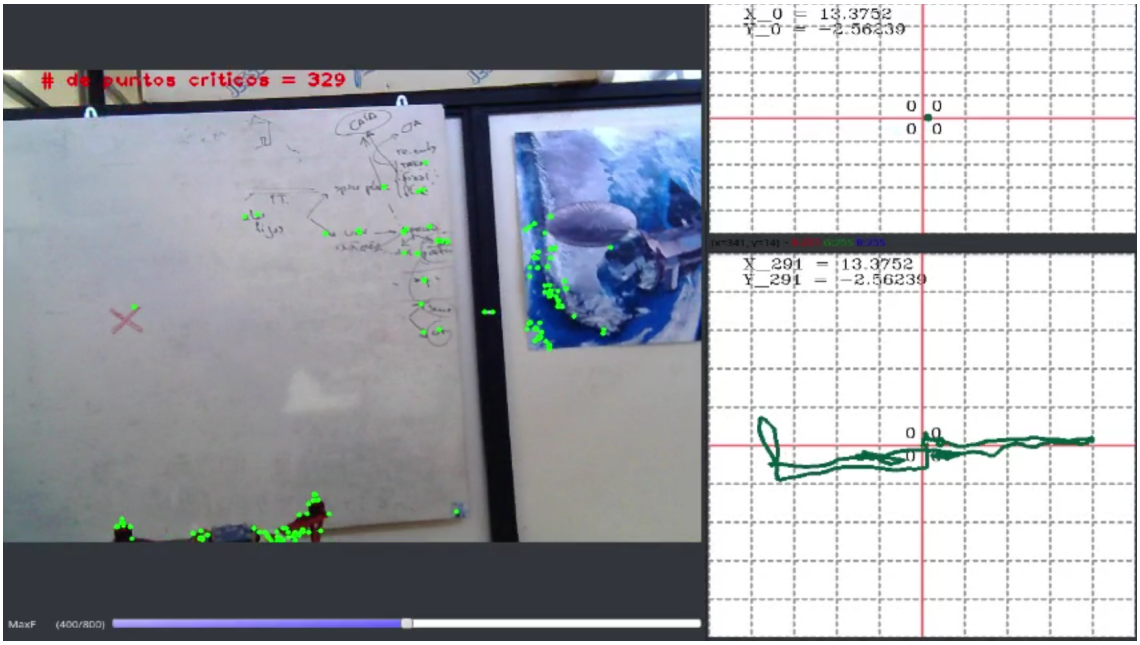


Figure 3: Ensayo 2

## 9 Trabajo futuro

Se trabajará en la subdivisión de la imágenes en la etapa de detección para tener en cuenta la falta de puntos claves por sección. De esta manera la imagen siempre contará con puntos claves en al menos un cuarto de la imagen y se correra la detección por sección. Para realizar esto se deberá trabajar con threads(hilos) y sincronizar el trabajo de detección y tracking por que así se aumentará la velocidad del programa y siempre habrá puntos claves en la imagen. Se implementará el código para realizar la estimación de posición on-board en un cuadricóptero combinando las mediciones inerciales obtenidas por la IMU, barómetro, sensor de altura ultrasonico y la estima de posición del algoritmo de traslación plana.

# A Comandos OpenCV 3.0

## A.1 ORB

En primera instancia se debe crear un objeto *Ptr* del tipo ORB. Este posee diferentes parámetros de entrada y la sintaxis es la siguiente.

$$cv :: Ptr < cv :: ORB > NombrePuntero = cv :: ORB :: create(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$$

Los parámetros utilizados son 8 y se explican a continuación.

- $p_1$ es la cantidad máxima de puntos claves que devuelve el programa
- $p_2$ Factor de escala. Tiene que ser mayor que 1 y marca la reducción de píxeles sufrida en altura y ancho al realizar los distintos niveles de la pirámide. La relación es $NPixelesAncho_NuevoNivel = \frac{NPixelesAncho_ViejoNivel}{p_2} y NPixelesAlto_NuevoNivel = \frac{NPixelesAlto_ViejoNivel}{p_2}$.
- $p_3$
- $p_4$
- $p_5$
- $p_6$
- $p_7$
- $p_7$

## A.2 CalcOpticalFlowPyrLK

En el comando : "calcOpticalFlowPyrLK". Se tienen 11 parámetros a pasarle al comando. Este programa realiza iteraciones hasta que se cumple alguna condición pre-establecida. Se utilizan los siguientes valores para hacer modificaciones sobre la estabilidad de los puntos. Es decir, para hacer que el programa siga los puntos de manera adecuada.

El $7^{mo}$ parámetro es el tamaño de ventana. Al modificar este valor se agranda o achica el tamaño de la ventana que usa el algoritmo para buscar la nueva posicion del punto. Al agrandar la ventana permite mayor movimiento entre cuadro y cuadro, por lo que es más factible hallar el punto cuando se tiene una gran velocidad relativa a corta distancia. Sin embargo, hace el programa más lento porque debe aplicar el algoritmo a una mayor cantidad de pixeles.

El $8^{vo}$ parámetro marca la cantidad de niveles de pirámide utilizados. En el caso de 0, no se usa. Para un valor de 1 se usan 2 niveles, para 2 se usan 3 niveles y así siguiendo.

El $9^{no}$ parámetro es el criterio de finalización de las iteraciones. Se usa una clase particular de OpenCV que tiene en cuenta 3 situaciones. La primera es una cantidad de iteraciones maximas, la segunda es iterar hasta que el error sea menor que un valor umbral y el tercero es aplicar los 2 anteriores juntos con la logica de terminar cuando suceda lo primero.

El $10^{mo}$ parámetro marca como el algoritmo busca las nuevas posiciones esperadas. Hay dos opciones. La primera usa los guardados en el vector de NextKeyPoints como referencia, hay que ponerlos ordenados. La segunda usa el método de mínimos autovalores.

El $11^{vo}$ valor marca el valor mínimo de autovalores que debe tener una zona para ser considerada dentro del algoritmo.

### A.3 buildOpticalFlowPyramid

Este comando espera como entrada una variable "Mat", con la imagen, y una variable "vector¡Mat¿" para construir la pirámide.

## Referencias


[1] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 756–770, June 2004.

[2] H. Li and R. Hartley, "Five-point motion estimation made easy," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 1, pp. 630–633, 2006.

[3] D. Viswanathan, "Features from accelerated segment test (fast)," 2011.

[4] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf," *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[5] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, 1988.

[6] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "Brief: Computing a local binary descriptor very fast," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, pp. 1281–1298, July 2012.

[7] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[8] C. Tomasi and T. Kanade, "Detection and tracking of point features," tech. rep., International Journal of Computer Vision, 1991.

[9] R. I. Hartley, "In defense of the eight-point algorithm," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, pp. 580–593, June 1997.