CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS
AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Programa de

Sistemas Autónomos de Navegación Aérea y Submarina

"Estabilización de vídeo en tiempo real para UAVs"

T E S I S

Que presenta

Karina Mayen González

Para obtener el grado de

Maestra en Ciencias

En

Sistemas Autónomos de Navegación Aérea y Submarina

Directores de la Tesis:

Dr. Sergio Rosario Salazar Cruz

Dr. Hugo Romero Trejo

México, D.F. **Diciembre, 2015**

“A mi familia por haberme brindado todo el apoyo,
y por ser siempre mi motor para seguir...”

El mayor riesgo en la vida, es no hacer nada.
“Anónimo”

AGRADECIMIENTOS

A mi familia por todo el amor, respeto, apoyo, dedicación y paciencia que me han brindado a lo largo de toda mi vida. Por todas las lecciones y consejos recibidos, sin ellos no habría sido posible cumplir otro más de mis objetivos. A mis hijos por ser parte fundamental de mi vida; mi apoyo y mis razones para seguir adelante. Y por mostrarme el significado de la inocencia y del verdadero amor.

A mis amigos por todas las aventuras que son tan necesarias en esta vida. En especial a Brian Ramirez y Sarahi Buendía ya que ustedes nunca me abandonaron cuando más lo necesitaba. A Alejandro Montiel y Carlos Espinosa por ser parte de esta aventura.

A los doctores Sergio Salazar, Hugo Romero, Mariano Lizarraga y Rogelio Lozano por brindarme todo el apoyo que requerí y por mostrarme el camino de la investigación. A mis sinodales los doctores Moisés Bonilla y Antonio Osorio por su valioso tiempo.

Al Concejo Nacional de Ciencia y Tecnología (CONACyT) por haberme apoyado con una beca para la realización de mis estudios de maestría en el CINVESTAV-IPN. Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV) por ser mi casa de estudios durante éstos dos años.

Al Instituto de Investigación y Desarrollo Tecnológico de la Armada de México (INIDETAM) por recibirme durante mi estancia en la Secretaría de Marina Armada de México (SEMAR). Y a todos los miembros del proyecto en el que estuve laborando en la SEMAR.

MUCHAS GRACIAS.

RESUMEN

En los últimos años hemos sido testigos del imparable crecimiento de los vehículos aéreos no tripulados, los cuales son aplicados en seguridad, búsqueda y rescate, monitoreo entre otras aplicaciones, ya que al hacer uso de estos vehículos se salvaguarda la seguridad de las personas en situaciones peligrosas. Motivo por el cual es de suma importancia obtener una visión clara de lo que está observando nuestro UAV.

Es por ello que esta tesis está enfocada en la estabilización del video obtenido de una cámara montada en un gimbal, el cual es un mecanismo de suspensión y puede ser de dos o tres grados de libertad, cuyos ejes forman un ángulo recto, este mecanismo es el encargado de mantener la orientación de un eje de rotación en un determinado espacio, sin importar que la plataforma que lo contenga sufra perturbaciones. La estabilización propuesta se logra mediante el uso de dos diferentes algoritmos de visión con la finalidad de compararlos entre ellos y obtener conclusiones.

En el primer capítulo se presenta un marco teórico, objetivos y justificación de la presente tesis. En el segundo capítulo vamos a obtener el modelado y el control de un gimbal, con el fin de observar gráficamente a nivel de simulación, como es que actúa el gimbal cuando la plataforma que lo contiene sufre perturbaciones. En nuestro tercer capítulo presentamos cómo es que funciona el primer algoritmo de estabilización, el cual está basado en la correlación de bloques en las imágenes. En el segundo algoritmo se utilizan algunas funciones de OpenCV, dicho algoritmo se presenta en el cuarto capítulo. Finalmente analizamos y comparamos los resultados obtenidos de esta tesis en el quinto capítulo, para validar los resultados de ambos algoritmos de estabilización utilizamos dos vídeos modificados manualmente, uno para la rotación y otro para la traslación.

ABSTRACT

In the last years we have been witnessed a tremendous growth in the use of Unmanned Aerial Vehicles (UAVs). These vehicles are applied in safety, search and rescue, monitoring and other applications. The use of these vehicles allows safeguar the safety of people in dangerous situations. So it is important to have a good vision about that our UAV is able to see.

So this thesis is focused on the video stabilization. This video is obtained from a camera mounted on a gimbal. Which is a suspension mechanism and may have two or three degrees of freedom and their axes form a right angle. This mechanism is responsible for maintaining the orientation of an axis of rotation in a given space, regardless of disturbances in the platform.

The first chapter presents a theoretical framework, objectives and justification of this thesis. In the second chapter we will get the modelling and control of a gimbal, in order to graphically observe it in the simulation, how the system acts when the gimbal platform is disturbed. In the third chapter we can see how the first stabilization algorithm works. This algorithm is based on block matching images. The second algorithm uses some OpenCV functions; this algorithm is presented in the fourth chapter. Finally we analyze and compare the results of this thesis in the fifth chapter. In order to validate the results of both stabilization algorithms we used two manually modified videos, one for rotation and other one for the translation.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1 Introducción general

Los vehículos aéreos no tripulados o mejor conocidos por sus siglas en inglés (Unmanned Aerial Vehicles - UAVs), son vehículos aéreos sin un piloto humano a bordo. Estos pueden ser autónomos, semi-autónomos o controlados por radio-control. Generalmente son empleados en misiones de reconocimiento, búsqueda y rescate, vigilancia, entre otros.

En navegación aérea no tripulada ya sea autónoma, semi-autónoma o a radio control, es de suma importancia obtener una alta calidad de video mediante una cámara de video abordo de un UAV, esto con el fin de tomar algunas decisiones, sin embargo no siempre es tan fácil como transmitir el vídeo y recibirlo en una estación terrestre, ya que como es muy conocido el UAV recibe muchas perturbaciones y algunas de ellas pueden afectar a la cámara, por lo que las imágenes que podemos recibir no solo presentarán el ruido a consecuencia de la transmisión del vídeo si no que también se obtiene un vídeo con muchas fluctuaciones el cual en ocasiones puede llegar hasta confundirnos con respecto a lo que observamos a consecuencia de los movimientos de la cámara.

Para ello se hace uso de los gimbals (Figura 1.1), siendo estos unos mecanismos de suspensión los cuales pueden ser de dos o tres grados de libertad, cuyos ejes forman un ángulo recto y son los encargados de mantener la orientación de un eje de rotación en una determinada posición, sin importar que la plataforma que lo contenga sufra perturbaciones. Sin embargo los gimbals no siempre nos brindan una estabilización óptima, ya que hay pequeñas vibraciones causadas por el mismo gimbal que afectan al vídeo.

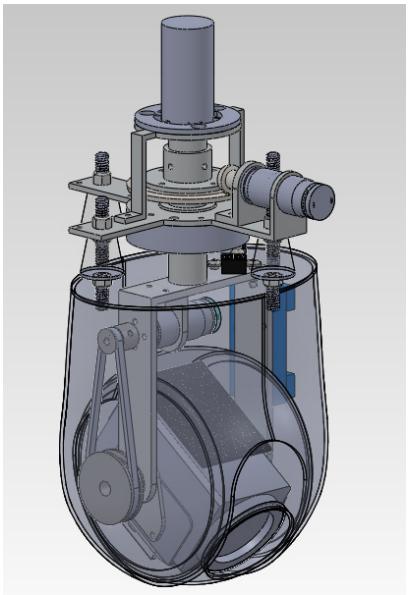


Figura 1.1: Gimbal.

Es por ello que esta tesis se enfoca en la resolución de dichas vibraciones ya sean las causadas por el gimbal o algunas perturbaciones que el mismo gimbal no sea capaz de compensar. En la presente tesis se hace el uso de dos algoritmos de estabilización con el fin de compararlos y obtener los resultados buscados. El primer algoritmo de estabilización se basa en el uso de planos de bits los cuales son correlacionados por bloques, y con ello obtener la estimación del movimiento local, para posteriormente estimar el movimiento global de la imagen actual con respecto a la anterior, para finalmente pasar a corregir dicho movimiento.

En el segundo algoritmo de estabilización primeramente se obtienen los puntos característicos de cada imagen, para después comparar la imagen actual con la anterior y con ello poder obtener el flujo óptico para calcular la transformación total.

Es importante mencionar que el Gimbal tiene diferentes aplicaciones, sin embargo generalmente éstas se pueden separar en dos, las de reconocimiento y las de seguimiento. En las aplicaciones de reconocimiento se tienen dos tipos de referencias, una representa la posición del objeto y la otra representa la trayectoria del vehículo. En nuestro caso vamos a utilizar la referencia por la trayectoria del vehículo.

Como ya se mencionó el gimbal consta de dos articulaciones, cuyos nombres en inglés son Pan y $Tilt$. Pan puede rotar continuamente 360°, sin embargo $Tilt$ se encuentra restringido. Estas articulaciones se encuentran distribuidas como se muestra en la Figura 1.2, donde podemos observar que Pan compensa la perturbación en el eje de rotación Yaw y de la misma manera $Tilt$ compensa las perturbaciones que afecten al eje de navegación en Pitch del vehículo.

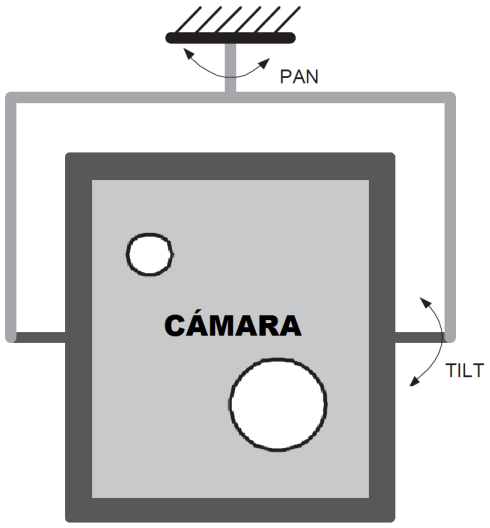


Figura 1.2: Gimbal dos grados de libertad.

1.2 Objetivo

1.2.1 Objetivo general

El principal objetivo que se persigue en la presente tesis se centra en el desarrolo de unos algoritmos de visión que estabilicen el vídeo en tiempo real obtenido de una cámara montada en un gimbal, el cual es un mecanismo de suspensión capaz de mantener en una posición específica a una cámara sin importar los movimientos que realice la plataforma en la que se encuentre (UAV), generalmente consta de 2 ejes que forman un ángulo recto (yaw y pitch).

1.2.2 Objetivos particulares

- Selección de los algoritmos para procesamiento en tiempo real.
- Programación de los algoritmos.
- Evaluación de los algoritmos desarrollados previamente.
- Pruebas en vuelos reales.

1.3 Justificación

El principal motivo de esta tesis deriva a partir de la estabilización que se obtiene al utilizar un sistema Gimbal, el cual a grandes rasgos es una forma mecánica para equilibrar una cámara en una posición definida, sin embargo no siempre se puede obtener una idónea estabilización del vídeo al utilizar dicho sistema, por lo que la parte medular es la aplicación de dos algoritmos de visión que permitan aumentar y/o corregir la estabilización.

1.4 Estado del arte

La estabilizaciń de un vídeo es la tarea de eliminar los efectos de movimiento global irregulares (jitter), a fin de obtener un vídeo estable [1]. La estabilización de imagen básicamente se divide en dos grandes campos: la óptica y la digital. Un estabilizador de imagen óptico, OIS a menudo abreviado, es un mecanismo utilizado en una cámara, el cual estabiliza la imagen grabada variando la trayectoria óptica del sensor. Dicha tecnología es implementada en la propia lente, o moviendo el sensor como el elemento final en el camino óptico. El elemento clave de todos los sistemas ópticos de estabilización es que estabilizan la imagen proyectada en el sensor antes de que el sensor convierta la imagen en información digital. Videocámaras con estabilización óptica de imagen general cuentan con diminutos sensores giroscópicos dentro de la lente que cambian rápidamente piezas de vidrio en el lente para compensar el movimiento.

Se considera una estabilización de imagen óptica si se dispone de un elemento móvil dentro del objetivo de la cámara. A diferencia de los sistemas de estabilización óptica, los sistemas de estabilización digital utilizan la tecnología del software para reducir las vibraciones causadas por la plataforma que contiene la cámara. En la estabilización de imagen digital se desplaza la imagen electrónica de imagen a imagen en el vídeo, lo suficiente para contrarrestar el movimiento no deseado. Esta técnica reduce las vibraciones en los vídeos al suavizar la transición de una imagen a otra. Sin embargo en esta técnica no se pueden atacar los problemas con los desenfoques existentes durante el movimiento.

En la estabilización de imagen óptica, generalmente el sistema de la cámara requiere de dos componentes los cuales son: un detector de movimiento y un corrector de movimiento. Con respecto a la detección del movimiento, una técnica para medir el movimiento es mediante el uso de un sensor giroscópico [2], [3] y una técnica propuesta que se utiliza para la detección del movimiento en una imagen en un componente de

vector es por medio del procesamiento de la señal de la imagen. Cuando se utiliza un sensor giroscópico, la señal de ruido filtrado y la señal diferenciada entre el movimiento de la plataforma y el barrido de la cámara se convierten en partes muy importantes. Un corrector de movimiento, el cual usa la información detectada del movimiento, corrige la imagen usando un proceso ya sea óptico o electrónico.

La corrección electrónica de imagen generalmente se divide en dos técnicas: la primera es por medio del control de la posición en la entrada del sensor y en la otra se almacenan los datos de una imagen en una memoria y posteriormente se controla la dirección, la cual lee los datos de la imagen.

La corrección de imagen óptica se puede clasificar en dos tipos: la primera es una técnica de refracción de la luz en una dirección arbitraria usando un lente angular ajustable (o un prisma activo) [2], y la técnica de retorno de imagen desviada desde el eje óptico y de regreso a la posición original al desplazar una lente o un sensor de imagen tal como un dispositivo acoplado de carga (CCD) y un semiconductor complementario de óxido de metal (CMOS), utilizando un dispositivo piezoeléctrico o un motor de bobina de voz (VCM) [4]. Estos métodos de corrección óptica pueden dar resultados claros de corrección en tiempo real.

Mientras que los enfoque de sistemas ópticos de estabilización fueron muy populares en el pasado, las implementaciones digitales en tiempo real se están convirtiendo en las más comunes. Ya que uno de sus grandes inconvenientes al hacer uso de la estabilización de imagen óptica son los altos costos que se deben de cubrir ya que los elementos que se utilizan son de elevado costo, es por ello que generalmente se recurre a la estabilización de imagen digital.

Un sistema de estabilización de imagen digital primero estima el movimiento no deseado y después aplica dicha corrección a la secuencia de imágenes. Como se describe en [1] el movimiento de la imagen puede ser estimado mediante el uso de un espacio temporal o una región de correlación. El enfoque del espacio temporal abarca la correlación de bloques paramétricos [5], la estimación directa de flujo óptico [6], y también el enfoque de la matriz de inversión del error de mínimos cuadrados reportado en [7]. El método de correlación por regiones abarca la correlación de los planos de bits presentado en [8], la correspondencia punto a línea dada a conocer en [9], el seguimiento de características presentado en [10], aproximación piramidal dada a conocer en [11], y finalmente la correlación de bloques reportada en [12].

Método de detección	Resolución	Transformación	Referencia
Correlación de bloques paramétricos	Sub-pixel	Traslación y Rotación	[5]
Estimación flujo óptico	Sub-pixel	Traslación y Rotación	[6]
Correlación de regiones lineales	Sub-pixel	Traslación, Rotación y distorsión	[7]
Correlación de planos de bits	Pixel	Traslación	[8]
Correspondencia punto a línea	Pixel	Distorsión	[9]
Seguimiento de caracteristicas	Pixel	Distorsión	[10]
Piramidal	Pixel	Distorsión	[11] y [12]
Correlación de bloques	Pixel	Traslación	[13]

Tabla 1.1: Comparación de correlación.

Mucho se puede inferir acerca de la complejidad computacional de los algoritmos como se observa en la tabla 1.1 y teniendo en cuenta la corrección de movimiento a transformar que cada enfoque es capaz de soportar. Cuando comparamos la eficiencia de alternar soluciones a cada problema, es importante considerar diferentes opciones de

implementación.

Las implementaciones de software pueden tener acceso a librerías que optimizan las funciones de un alto nivel matemático, sin embargo en la implementación del hardware en tiempo real puede implementarse con elementos básicos como operadores Booleanos. Esta tesis se centra en la simulación e implementación de dos algoritmos que realicen eficientemente la estabilización del vídeo en tiempo real. Además a estos algoritmos de estabilización en tiempo real les implementamos un filtro de Kalman para eliminar las fluctuaciones bruscas a baja frecuencia. Ya que el movimiento global de la cámara está definido en términos del movimiento de la aceleración constante y de los modelos de movimiento de velocidad constante, se utiliza el filtro de Kalman.

CAPÍTULO 2

MODELADO Y CONTROL DEL GIMBAL

Con el fin de comprender como es que funciona el sistema Gimbal, en el presente capítulo vamos a obtener el modelo del sistema Gimbal, asi como diseñar un control PD para dicho sistema. Para el modelado haremos uso de la convención de Denavit- Hartenberg (D-H) la cual se encuentra a detalle en el Anexo A.

2.1 Aproximación de la cinemática del Gimbal mediante D-H

La estructura del Gimbal a simple vista parece ser muy simple, sin embargo la realidad es otra. Las cámaras generalmente no se encuentran localizadas sobre los dos ejes rotacionales (*Pan* & *Tilt*), tal como se puede observar en la Figura 1.2; por lo que dichos ejes deben de transladarse cuando alguno de los dos ejes es rotado. Generalmente cuando el Gimbal es utilizado en distancias cortas entre el objetivo a visualizar y la plataforma que contiene el Gimbal para la vigilacia, el punto de inicio es muy

significativo. Por otra parte, no podemos suponer que la posición de la cámara se encuentra directamente fija hacia el objetivo a observar. Sin embargo, con el propósito de entender a groso modo el funcionamiento del Gimbal, supondremos que los dos ejes de rotación son perpendiculares. El sistema mecánico de amortiguamiento en este análisis se despreciará.

Llevando a cabo el algoritmo de Denavit Hartenberg, obtenemos la siguiente tabla con los parámetros $\alpha_i, a_i, \theta_i y d_i$:

Eslabón i	α_i $[rad]$	a_i $[m]$	θ_i $[rad]$	d_i $[m]$	R/P
PAN	$-\pi\backslash 2$	0	θ_1	0	R
TILT	$\pi\backslash 2$	0	θ_2	0	R

Tabla 2.1: Parámetros de D-H.

Por lo que finalmente después de algunos cálculos nuestra matriz de transformación homogénea para el Gimbal mediante la convención de Denavit Hartenberg queda de la siguiente manera:

$$T = T_{PAN} * T_{TILT} = \begin{bmatrix} C_{\theta 1}C_{\theta 2} & -S_{\theta 1} & C_{\theta 1}S_{\theta 2} & 0 \\ C_{\theta 2}S_{\theta 1} & C_{\theta 1} & S_{\theta 1}S_{\theta 2} & 0 \\ -S_{\theta 2} & 0 & C_{\theta 2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

2.2 Cinemática diferencial.

El objetivo principal de la cinemática diferencial, es encontrar una relación entre las velocidades de cada uno de los eslabones, el efector final y las velocidades angulares. Esta relación que se busca obtener es de suma importancia ya que se utiliza en el cómputo de la dinámica del sistema del Gimbal y para ello hacemos uso del Jacobiano geométricoA.5.

2.2.1 Jacobiano geométrico para el sistema Gimbal.

Ahora proseguimos a calcular el Jacobiano basado en los parámetros de Denavit-Hartenberg. Siguiendo todos los términos obtenidos en el Anexo anteriormente, ahora es relativamente sencillo calcular la parte rotacional del Jacobiano como:

$$J_{\omega 1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad J_{\omega 2} = \begin{bmatrix} 0 & -sin_{\theta 1} \\ 0 & cos_{\theta 1} \\ 1 & 0 \end{bmatrix} \tag{2.2}$$

En este análisis se asume que el origen de cada marco se encuentra en el centro de masa de cada eslabón. Posteriormente asumiremos que el centro de masa para cada eslabón se encuentra en la intersección de todos los ejes rotacionales. Esto significa que los marcos no tienen ningún movimiento traslacional y por lo tanto:

$$J_{v1} = 0 \qquad J_{v2} = 0 \tag{2.3}$$

2.2.2 Ecuaciones de movimiento del sistema Gimbal.

Para el cálculo de las ecuaciones de movimiento del sistema Gimbal, asumimos que la estructura es rígida. Por lo tanto seguimos los siguientes pasos para un robot manipulador de cinemática abierta:

1. Cálculo de $D(q)$

   (a) Calcular la cinemática y las matrices rotacionales R_0^i.

   (b) Calcular la cinemática diferencial y las matrices del Jacobiano J_{vi} y $J_{\omega i}$.

   (c) Calcular la suma A.40 p. ej. $D(q)$. Esto también requiere de la masa y la matriz de inercia de cada eslabón.

2. Cálculo de $C(q, \dot{q})$

   (a) Cálculo de los coeficientes de Christoffel A.49.

   (b) Encontrar $C(q, \dot{q})$ por medio del cálculo de los elementos como se indica en A.52.

3. Cálculo de $g(q)$.

   (a) Cálculo de $V(q)$ en A.43. También se requiere del conocimiento del centro de masa de cada eslabón.

   (b) Derivar V como se indica en A.50.

   (c) Siendo $g(q) = [g_1, ..., g_n]^T$.

4. Cálculo de $F_v(\dot{q})$.

5. Finalmente escribimos las ecuaciones de movimiento A.51.

$$D(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + F_v\dot{q} = \tau \tag{2.4}$$

Ahora debemos de hacer algunas aproximaciones y así simplificar los cálculos y para ello primero asumimos que el marco de la base se encuentra en reposo (él UAV no se está moviendo). Considerando la estructura de cada eslabón suponemos que el centro de masa de cada eslabón se encuentra cerca de su eje de rotación. Y si asumimos que I_i es diagonal, así podemos reducir el número de parámetros desconocidos.

Además realizamos un cambio de variable de la siguiente manera:

$$\begin{aligned} q &= \theta \\ \dot{q} &= \dot{\theta} = \omega \\ \ddot{q} &= \ddot{\theta} = \dot{\omega} \end{aligned} \tag{2.5}$$

Por lo que ahora A.53 queda de la siguiente manera:

$$D(\theta)\dot{\omega} + C(\theta, \omega)\omega + F_v\omega + g(\theta) = \tau \tag{2.6}$$

Para el cálculo de $D(\theta)$ asumimos que las matrices de inercia son diagonales ya que la estructura es simétrica y por lo tanto el centro de masa coincide con el centro geométrico del eslabón, por lo que:

$$I_1 = \begin{bmatrix} I_{111} & 0 & 0 \\ 0 & I_{122} & 0 \\ 0 & 0 & I_{133} \end{bmatrix} \qquad I_2 = \begin{bmatrix} I_{211} & 0 & 0 \\ 0 & I_{222} & 0 \\ 0 & 0 & I_{233} \end{bmatrix} \tag{2.7}$$

Ahora podemos calcular a $D(\theta)$ como la suma de:

$$D(\theta) = \sum_{i=1}^{n} [m_i J_{vi}(\theta)^T J_{vi}(\theta) + J_{\omega i}(\theta)^T R_i(\theta) I_i R_i(\theta)^T J_{\omega i}(\theta)] \tag{2.8}$$

Cómo ya se mencionó anteriormente en este análisis se asume que el centro de masa de cada eslabón es localizado en la intersección del eje rotacional de todas las articulaciones. Lo que significa que ningún marco tiene movimiento traslacional por lo que:

$$J_{vi} = J_{v2} = 0 \tag{2.9}$$

Considerando lo anterior tenemos que la ecuación 2.8 queda de la siguiente manera:

$$D(\theta) = \sum_{i=1}^{n} J_{\omega i}(\theta)^T R_i(\theta) I_i R_i(\theta)^T J_{\omega i}(\theta) \tag{2.10}$$

Siguiendo los pasos mencionados anteriormente:

Paso 1.- Cálculo de D($\boldsymbol{\theta}$):

Obtenemos los términos en $D(\theta)$, los cuales son:

$$
\begin{aligned}
d_{11} &= I_{233}C_{\theta_2}^2 + I_{211}S_{\theta_2}^2 && (2.11)\\
d_{12} &= -S_{\theta_1}(-I_{211}C_{\theta_1}C_{\theta_2}S_{\theta_2} + I_{233}C_{\theta_1}C_{\theta_2}S_{\theta_2}) \\
&\quad +C_{\theta_1}(-I_{211}C_{\theta_2}S_{\theta_1}S_{\theta_2} + I_{233}C_{\theta_2}S_{\theta_1}S_{\theta_2}) && (2.12)\\
d_{21} &= 0 && (2.13)\\
d_{22} &= I_{222}C_{\theta_1}^2(C_{\theta_1}^2 + S_{\theta_1}^2) + I_{222}S_{\theta_1}^2(C_{\theta_1}^2 + S_{\theta_1}^2) && (2.14)
\end{aligned}
$$

donde nuestra matriz $D(\theta)$ tiene la siguiente forma:

$$
D(\theta) = \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \tag{2.15}
$$

Paso 2.- Cálculo de C($\boldsymbol{\theta}, \boldsymbol{\omega}$):

Para obtener la matriz $C(\theta, \omega)$ hacemos uso de las ecuaciones A.49 y A.52 por lo que los términos de nuestra matriz y nuestra matriz $C(\theta, \omega)$ quedan de la siguiente manera:

$$
\begin{aligned}
c_{11} &= -\omega_2(I_{211}C_{\theta_2}S_{\theta_2} - I_{233}C_{\theta_2}S_{\theta_2}) && (2.16)\\
c_{12} &= \omega_1(I_{211}C_{\theta_2}S_{\theta_2} - I_{233}C_{\theta_2}S_{\theta_2}) \\
&\quad -\frac{\omega_2}{2}(S_{\theta_1}(I_{211}C_{\theta_1}C_{\theta_2}^2 - I_{233}C_{\theta_1}C_{\theta_2}^2 - I_{211}C_{\theta_1}S_{\theta_2}^2 + I_{233}C_{\theta_1}S_{\theta_2}^2)) \\
&\quad -\frac{1}{2}(C_{\theta_1}(I_{211}C_{\theta_2}^2S_{\theta_1} - I_{233}C_{\theta_2}^2S_{\theta_1} - I_{211}S_{\theta_1}S_{\theta_2}^2 + I_{233}S_{\theta_1}S_{\theta_2}^2)) && (2.17)\\
c_{21} &= \omega_1(I_{211}C_{\theta_2}S_{\theta_2} - I_{233}C_{\theta_2}S_{\theta_2}) && (2.18)\\
c_{22} &= \omega_1(S_{\theta_1}(I_{211}C_{\theta_1}C_{\theta_2}^2 - I_{233}C_{\theta_1}C_{\theta_2}^2 - I_{211}C_{\theta_1}S_{\theta_2}^2 + I_{233}C_{\theta_1}S_{\theta_2}^2) \\
&\quad -C_{\theta_1}(I_{211}C_{\theta_2}^2S_{\theta_1} - I_{233}C_{\theta_2}^2S_{\theta_1} - I_{211}S_{\theta_1}S_{\theta_2}^2 + I_{233}S_{\theta_1}S_{\theta_2}^2)) && (2.19)
\end{aligned}
$$

$$C(\theta,\omega) = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \tag{2.20}$$

Paso 3.- Cálculo de g($\boldsymbol{\theta}$):

La gravedad no afecta la dinámica, ya que la posición del centro de masa es constante, por lo que:

$$g(\theta) = 0 \tag{2.21}$$

Paso 4.- Cálculo de $F_v\boldsymbol{\omega}$:

La matriz de fricción se define como una diagonal de las fricciones de viscosidad ya hablamos de una estructura desacoplada, y se deprecia la fricción estática porque se considera que el sistema se encuentra lubricado. Por lo que tenemos:

$$F_v = \begin{bmatrix} f_{v1} & 0 \\ 0 & f_{v2} \end{bmatrix} \tag{2.22}$$

Paso 5.- Ecuaciones de movimiento:

Nuestro modelo dinámico representado en espacio de estados queda de la siguiente manera:

$$\dot{\theta} = \omega \tag{2.23}$$

$$\dot{\omega} = D(\theta)^{-1}[\tau - C(\theta,\omega)\omega - F_v\omega - g(\theta)] \tag{2.24}$$

2.3 Control PD del Gimbal.

En el control proporcional-derivativo (PD), como su nombre lo indica, la ley de control está formada no solo por un término proporcional al error de posición $\tilde{q}$ como el controlador Proporcional con retroalimentación de velocidad, sino también por otro término

proporcional a su derivada, p. ej. al error de velocidad $\tilde{\dot{q}}$. La ley de control PD viene dada por:

$$\tau = K_p\tilde{q} + K_v\dot{\tilde{q}} \tag{2.25}$$

donde $K_p, K_v \in \mathbb{R}^{n\times n}$ son matrices simétricas definidas positivas seleccionadas por el diseñador. $\tilde{q} = q_d - q$ y $\dot{\tilde{q}} = \dot{q}_d - \dot{q}$, donde q_d y $\dot{q}_d$ son la posición y velocidad articular deseadas respectivamente. En la figura 2.1 se observa el diagrama a bloques de un sistema con un controlador PD.

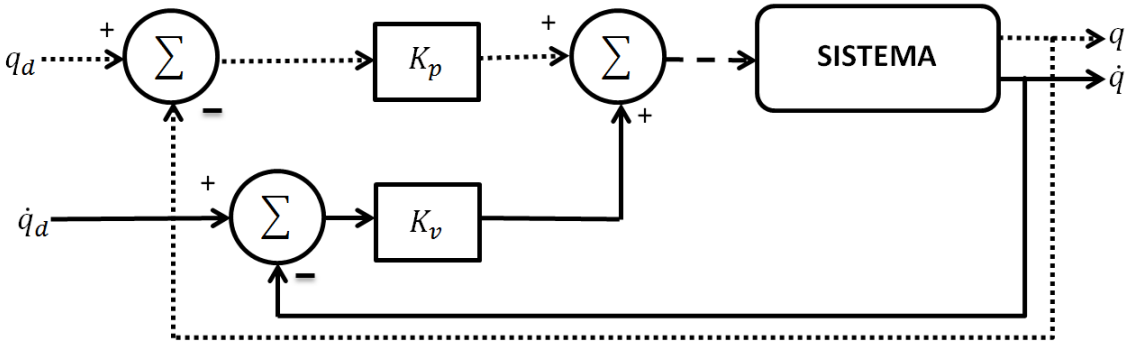


Figura 2.1: Diagrama a bloques control PD.

Para la definición del controlador PD, no se impone ninguna restricción al vector de posiciones articulares deseadas q_d. Esto se debe a que un controlador debe caracterizar únicamente a su estructura y no debe de depender de la clase de referencias.

El control PD es local en el sentido que el par o fuerza determinado por dicho controlador, y a ser aplicado en una articulación, sólo depende de la posición y velocidad de dicha articulación y no de las demás articulaciones. Esto es una selección diagonal de matrices de diseño K_p y K_v.

El controlador PD dado por la ecuación 2.25 requiere de la medición de las posiciones q y velocidades $\dot{q}$, así como la especificación de la posición articular deseada q_d como se observa en la figura 2.1 donde observamos que no es necesario especificar la velocidad ni la aceleración deseadas $\dot{q}_d$ y $\ddot{q}_d$.

El comportamiento en malla cerrada de un robot de n g.d.l bajo control PD se obtiene combinando el modelo con la ley de control 2.25.

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = K_p\tilde{q} + K_v\dot{\tilde{q}} \tag{2.26}$$

O equivalentemente en términos del vector de estado $\begin{bmatrix}\tilde{q}^T & \dot{\tilde{q}}^T\end{bmatrix}^T$:

$$\frac{d}{dt}\begin{bmatrix}\tilde{q}\\ \dot{\tilde{q}}\end{bmatrix} = \begin{bmatrix}\dot{\tilde{q}}\\ \ddot{q}_d - M(q)^{-1}[K_p\tilde{q} + K_v\dot{\tilde{q}} - C(q,\dot{q})\dot{q} - g(q)\end{bmatrix} \tag{2.27}$$

que es una ecuación diferencial no lineal y no autónoma. Si suponemos que el vector de posiciones articulares deseadas q_d es constante, la ecuación de malla cerrada puede escribirse en términos del nuevo vector de estado $\begin{bmatrix}\tilde{q}^T & \dot{q}^T\end{bmatrix}^T$ como:

$$\frac{d}{dt}\begin{bmatrix}\tilde{q}\\ \dot{q}\end{bmatrix} = \begin{bmatrix}-\dot{q}\\ M(q)^{-1}[K_p\tilde{q} - K_v\dot{q} - C(q,\dot{q})\dot{q} - g(q)]\end{bmatrix} \tag{2.28}$$

la ecuación anterior podrá tener múltiples estados de equilibrio. Ya que $\dot{q} = 0$, entonces tenemos que de la ecuación 2.28, queda de la siguiente manera:

$$\frac{d}{dt}\begin{bmatrix}\tilde{q}\\ \dot{q}\end{bmatrix} = \begin{bmatrix}0\\ M(q)^{-1}[K_p\tilde{q} - g(q)]\end{bmatrix} \tag{2.29}$$

Y sabemos que $q = q_d - \tilde{q}$, entonces de la ecuación 2.29 obtenemos:

$$K_p\tilde{q} - g(q_d - \tilde{q}) \tag{2.30}$$

Los equilibrios están dados por $\begin{bmatrix}\tilde{q}^T & \dot{q}^T\end{bmatrix}^T = \begin{bmatrix}s^T & 0^T\end{bmatrix}^T$ donde $s \in \mathbb{R}^n$ es solución de:

$$K_p s - g(q_d - s) = 0 \tag{2.31}$$

Ya que el modelo no posee el término de pares gravitacionales $g(q)$, entonces el único equilibrio será el origen. La simulación y los resultados de dicha simulación se observarán en el capítulo correspondiente a “Simulaciones y resultados” , los cuales discutiremos a más detalle.

CAPÍTULO 3

PRIMER ALGORITMO DE ESTABILIZACIÓN

El primer algoritmo de estabilización que se utilizó está basado en la correlación de los planos de bits en código gray, él cual es un tipo de código binario no ponderado (los dígitos que lo contienen no tienen un peso asignado). Su principal característica es que entre la combinación de dígitos y la siguiente, sea ésta su anterior o posterior combinación, sólo hay una diferencia de un dígito. La estimación del movimiento basado en el algoritmo de la correlación de bloques es de suma importancia en la estabilización de los videos en tiempo real, ya que su tiempo de procesamiento se ve disminuido al hacer uso de dicho algoritmo.

Este algoritmo de estabilización de video se divide en cinco bloques principales (figura 3.1), el primero es el (PP) pre-procesamiento en el cual se extraen los planos de bits de la región de interés para posteriormente aplicarle código gray a estos planos de bits, el segundo bloque es el (EM) de estimación del movimiento el cual se subdivide en dos, el primer sub-bloque es (EL) estimación local donde mediante el uso de los vectores de movimiento se obtiene el estimado del flujo óptico entre la imagen actual y

la anterior, el segundo sub-bloque es (EG) la estimación global donde se determina una transformación apropiada de forma que sea la mejor caracterización del movimiento descrito por la estimación de movimiento local. El tercer bloque es (DM) la decisión de movimiento, aquí se hace uso de la EG y de un coeficiente de amortiguamiento para finalmente obtener una decisión del movimiento calculado, ahora pasamos al cuarto bloque (SM) de suavizado del movimiento, en este bloque se utiliza un filtro de Kalman para lograr suavizar la corrección del movimiento previamente calculado. Finalmente el último bloque es (CM) el de compensación del movimiento en el cual como su nombre lo dice se realiza la compensación del movimiento previamente calculado y suavizado, por lo que se realiza la transformación a la imagen actual y se coloca sobre la imagen anterior logrando así la estabilización del video [14].

Figura 3.1: Diagrama a bloques.

Como se puede observar en este algoritmo de estabilización, el componente crucial es la estimación del movimiento, pero esto no quiere decir que los otros bloques no sean importantes ya que si aplicamos la pura estimación del movimiento a la transformación de la imagen actual con respecto a la anterior, nuestro resultado sería una estabilización muy burda, ya que no se están tomando en cuenta las mediciones anteriores.

Antes de introducirnos en la descripción del método propuesto, primero debemos de conocer el primer bloque PP, donde en cada imagen obtenemos los planos de bits de las regiones de interés para posteriormente aplicarle Código-Gray a este conjunto de planos de bits.

3.1 PRE-PROCESAMIENTO (PLANOS DE BITS Y CÓDIGO GRAY)

Sabemos que la parte fundamental del procesamiento en tiempo real es el tiempo, por lo que al realizar menos cálculos nuestro tiempo de procesamiento va disminuyendo, es por lo que este algoritmo no se aplica a toda la imagen, y por ello definimos las regiones de interés de la siguiente manera, primero dividimos cada imagen en 4, si suponemos que la imagen es de 640X480 pixeles, la división antes mencionada es mostrada en la Figura 3.2.

Figura 3.2: División de la imagen en 4 regiones (S_1,S_2,S_3,S_4).

En el presente algoritmo asumimos que todos los pixeles de cada sub-imagen tienen movimiento uniforme y a su vez el rango del vector de movimiento se encuentra restringido por la ventana de búsqueda, el tamaño de bloque será de MXN y el tamaño de la ventana de búsqueda es de $(M + 2p)X(N + 2p)$ donde p es el máximo desplazamiento que se espera entre la imagen actual y la anterior, en la (figura 3.3) observamos

que la imagen es un cuarto de la imagen original y el cuadrado de linea continua es la ventana de búsqueda y la ventana punteada es el bloque que se va a buscar en la ventana de 37X37 y así obtener el movimiento entre la imagen anterior y la actual.

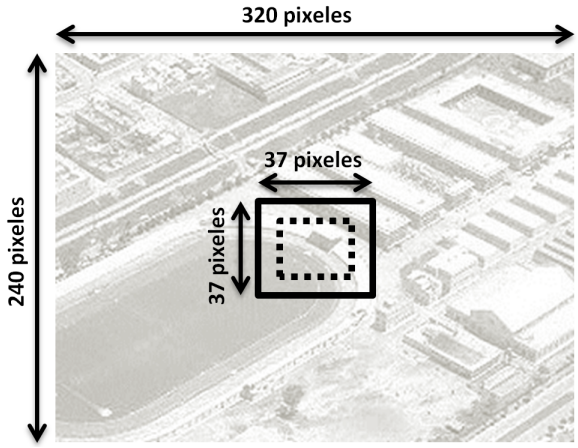


Figura 3.3: Ventana de búsqueda.

En este caso se seleccionó un bloque de 25 pixeles por 25 pixeles, y esperamos un máximo desplazamiento de 6 pixeles en cualquier dirección por lo que la ventana de búsqueda tendrá un tamaño final de 37 pixeles X 37 pixeles.

Una vez que ya tenemos nuestra ventana de búsqueda seleccionada el procesamiento se realizará única y exclusivamente a las cuatro ventanas de búsqueda de cada imagen $(S_1, S_2, S_3 y S_4)$, estas imágenes las obtenemos a color, por lo que tenemos 3 vectores en cada pixel (RGB), y a partir de estos obtenemos un vector a escala de grises, esto lo obtenemos de la siguiente manera:

$$Y = (0.299 * R) + (0.587 * G) + (0.114 * B) \tag{3.1}$$

Al tener la ventana de búsqueda en escala de grises, significa que en cada pixel estamos haciendo uso de 8 bits, esto afecta directamente en el tiempo de procesamiento por lo que para evitar hacer operaciones de 8 bits a nuestra ventana de búsqueda le extraemos los planos de bits para hacer únicamente operaciones Booleanas y con ello disminuye el tiempo del procesamiento en cada imagen. Para una mayor comprensión de lo que ocurre en la transformación de la ventana de búsqueda nos vamos a centrar en un pixel de la imagen. Por lo que tenemos el pixel en escala de grises, el cual se encuentra localizado en (x, y), en la $t-ésima$ imagen con 2^K niveles de gris representado de la siguiente manera:

$$f^t(x,y) = a_{K-1}2^{K-1} + a_{K-2}2^{K-2} + ... + a_1 2^1 + a_0 2^0 \tag{3.2}$$

donde $a_k, 0 \leq k \leq K-1$, siendo 0 o 1. Ahora tenemos nuestros planos de bits (8 para un nivel de gris de 256) denotados por $b_k(x, y)$, este plano contiene todos los $k-ésimos$ bits (a_k). Que en nuestro caso de 8 bits tenemos $b_0^t(x, y) \Rightarrow b_7^t(x, y)$ (figura 3.4).

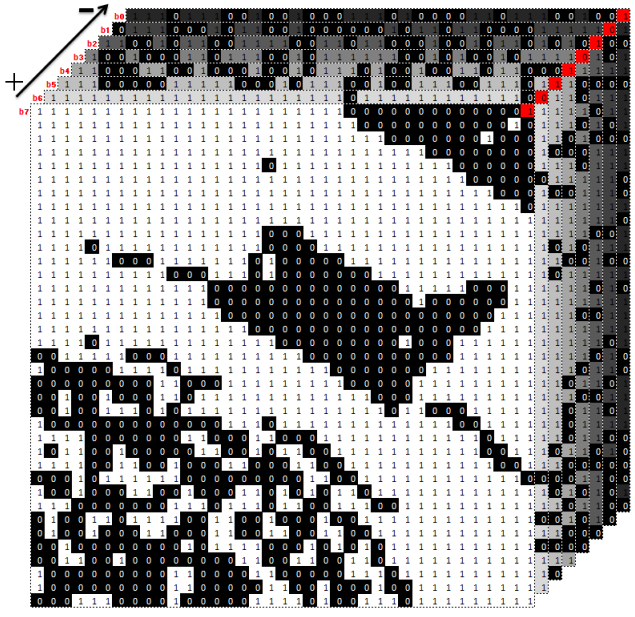


Figura 3.4: Planos de bits(escala de gris).

Cabe mencionar que el plano de bits $b_0(x, y)$ contiene los bits menos significativos de la imagen y el plano de bits $b_7(x, y)$ contiene todos los bits más significativos de la imagen, es por ello que en los planos de bits superiores se encuentran todas las características globales de la imagen y en los planos de bits menos significativos están contenidas todos los detalles de la imagen [15].

PLANO	a_{K-1}	$a_{K-1}2^{K-1}$	Total Acu.
a_7	1	$1 * 2^7 = 128$	128
a_6	0	$0 * 2^6 = 0$	128
a_5	1	$1 * 2^5 = 32$	160
a_4	1	$1 * 2^4 = 16$	176
a_3	0	$0 * 2^3 = 0$	176
a_2	1	$1 * 2^2 = 4$	180
a_1	0	$0 * 2^1 = 0$	180
a_0	1	$1 * 2^0 = 1$	181

Tabla 3.1: Plano de bits - escala de gris.

El principal problema que se presenta al hacer uso de los planos de bits que solo contienen cógigo-binario (figura 3.5),es que se afecta a más de un plano de bits con el cambio de un nivel de gris a otro, logrando con ello que la estimación de nuestro movimiento sea imprecisa.

DECIMAL	BINARIO	GRAY
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100

Figura 3.5: Código-Gray vs Binario.

Por esta razón una vez que obtenemos nuestros planos de bit hacemos uso del Código-Gray, el cual se obtiene solo con el uso de la operación OR exclusiva de la siguiente manera:

$$\begin{aligned} g_{K-1} &= a_{K-1} \\ g_k &= a_k \oplus a_{k+1} \quad 0 \leq k \leq K-2 \end{aligned} \tag{3.3}$$

donde, $\oplus$ es la operación Booleana OR exclusiva y a_k es el k-ésimo bit de base 2 representado por la ecuación 3.2.

Como se observa en la (figura 3.6) el séptimo plano de bits queda igual, los subsecuentes son los únicos que cambian, pero ya con esto aseguramos que ante cualquier cambio en la escala de gris solo va a diferir un plano de bits, como es el caso de 127 (011111111) y 128 (10000000) que en binario afecta a todos los bits, por lo que se presentaría un cambio drástico en nuestros planos de bits sin embargo con Gray 127 (11000000) y 128(01000000), donde solo se afecta un bit que en nuestro caso solo se afecta un plano de bits, obteniendo una mayor precisión de la estimación del movimiento [8].

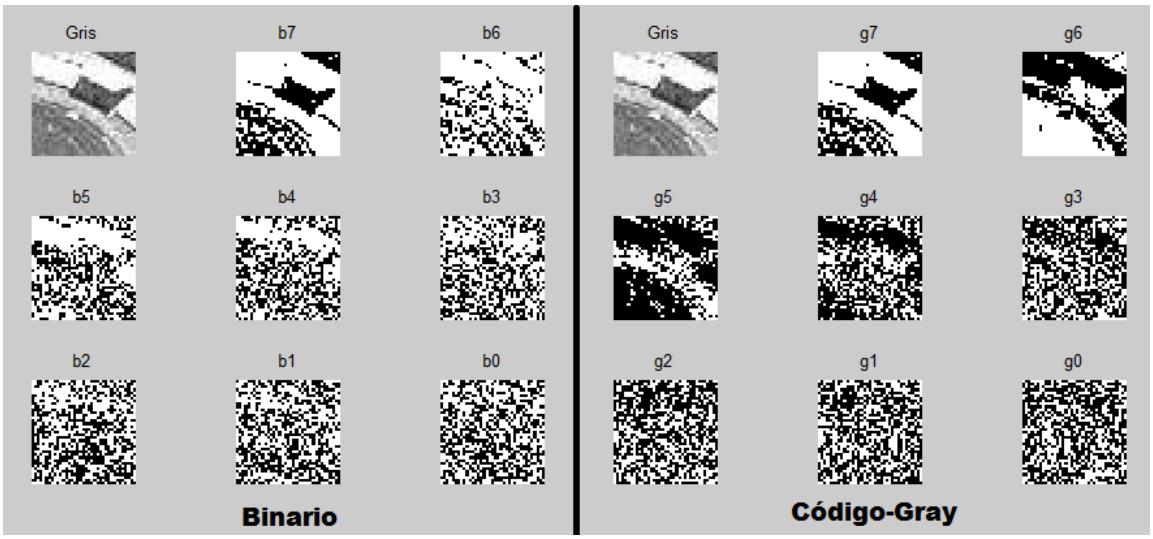


Figura 3.6: Planos de bits sin/con código Gray.

Finalmente tenemos nuestra ventana de búsqueda lista para el siguiente bloque de procesamiento. Como se va a observar en bloques siguientes los planos de bits que vamos a utilizar son el cuarto, quinto y sexto, ya que estos son de los que obtenemos las características globales y algunos de los detalles más significativos de cada una de las imágenes y con ello obtener el estimado de movimiento.

3.2 Estimación del movimiento (Local y Global).

Como ya se mencionó el cálculo de la estimación del movimiento se divide en dos partes, la primera es la estimación de los vectores locales, esta parte es muy importante ya que a partir del uso de estos vectores podemos determinar la segunda parte, la cual es el estimado del vector de movimiento global.

3.2.1 Estimación local.

Para la estimación de los vectores de movimiento local tenemos cuatro sub-imágenes (S_1, S_2, S_3 y S_4), las cuales obtuvimos de dividir la imagen principal en cuatro (figura

3.2) y de ahí obtener la ventana de búsqueda de cada sección de la imagen (Figura 3.3), obviamente obtendremos cuatro vectores de movimiento y cada vector de movimiento de cada sub-imagen en el actual Plano de Bits en Código-Gray ($PB-CG$) es determinado por la evaluación del Movimiento en el Plano de Bits en Código-Gray ($MPB-CG$) sobre cada sub-imagen en el anterior PB-CG y seleccionando cuál de los vectores produce la coincidencia más cercana. (Figura 3.7).

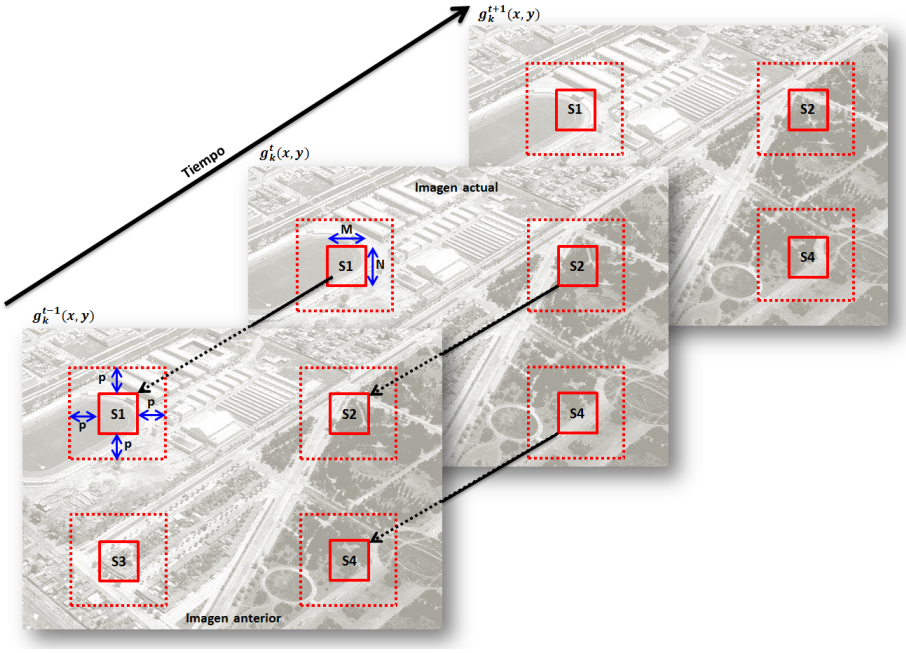


Figura 3.7: Vectores de movimiento.

Como ya se había comentado anteriormente aquí asumimos que todos los pixeles dentro de cada sub-imagen tienen movimiento uniforme y el rango del vector de movimiento se encuentra limitado por el tamaño de la ventana de búsqueda que seleccionamos. El cálculo de MPB-CG, lo obtenemos mediante la correlación de la siguiente manera [16]:

$$C_j(m,n) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g_k^t(x,y) \oplus g_k^{t-1}(x+m, y+n) \qquad -p \leq m,n \leq p \tag{3.4}$$

donde $g_k^t(x,y)$ y $g_k^{t-1}(x,y)$, son el actual y el anterior k-ésimo orden respectivamente en el PB-CG y p el máximo desplazamiento en la ventana de búsqueda. Para cada (m,n) dentro del intervalo de búsqueda se calcula la correlación $C_j(m,n)$, la cual es el número de bits no coincidentes entre la sub-imágenes en el PB-CG actual y las sub-imágenes en el PB-CG anterior.

El valor más pequeño de $C_j(m,n)$ es la mejor coincidencia para cada sub-imagen, y por lo tanto el vector de movimiento V en la j-ésima sub-imagen se obtiene de la siguiente manera:

$$V_j = argmin C_j(m,n), -p \leq m,n \leq p \tag{3.5}$$

En la figura 3.8 Observamos un diagrama a bloques de la parte del algoritmo presentado hasta este momento.

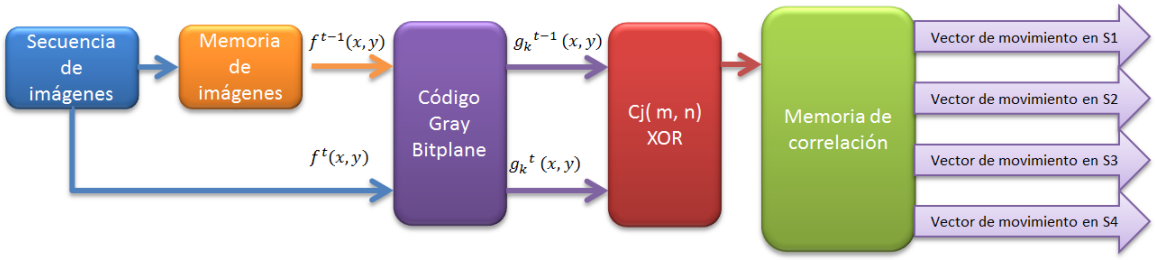


Figura 3.8: Diagrama a bloques.

Con el afán de reducir aún más el tiempo de procesamiento, la correlación antes mencionada, no se aplicará a toda la ventana de búsqueda, lo que se va a usar en este caso es una variante de la búsqueda de los tres pasos (figura 3.9), y aquí en donde hacemos uso de los planos de bits 4, 5 y 6 ya que en estos planos de bits se encuentran las características globales y detalles principales de las imágenes.

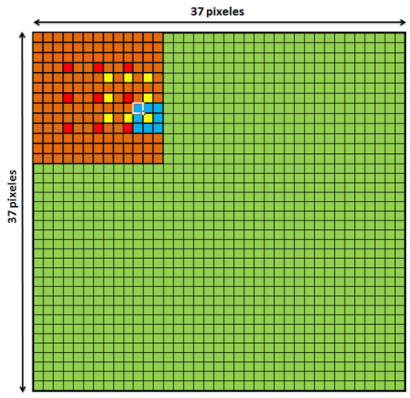


Figura 3.9: 3 Pasos tradicional.

Tomando en cuenta que seleccionamos un máximo desplazamiento p de 6 pixeles, la variante en la búsqueda de los 3 pasos consta en lugar de hacer uso de una ventana de $9X9$, utilizamos una ventana de $13X13$ y en el primer paso solo vamos a correlacionar 5 puntos, en el segundo y tercer paso correlacionamos los tradicionales 9 puntos (figura 3.10).

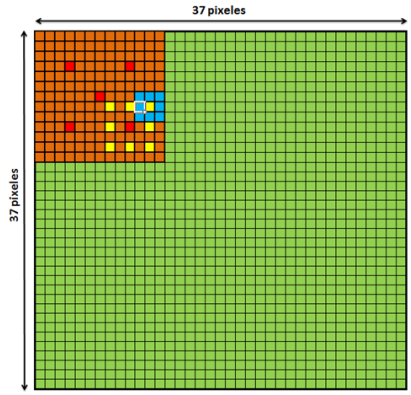


Figura 3.10: 3 Pasos modificado.

El primer paso lo aplicamos en el sexto plano de bits con una ventana de $7X7$, de este primer paso obtenemos 5 valores, seleccionamos el menor de esos cinco valores y desplazamos nuestro centro a la ubicación de dicho valor. Así pasamos al segundo paso

donde nuestra ventana se reduce a $5X5$ y ahora utilizamos el 5 plano de bits, pero en esta ocasión correlacionamos 9 puntos. De igual manera seleccionamos el menor de ellos y desplazamos de nuevo nuestro centro a la ubicación de ese nuevo valor. Ahora vamos al tercer y último paso, donde la ventana se reduce a $3X3$ y aquí utilizamos el cuarto plano de bits, y de igual manera que el paso dos obtenemos 9 valores de los cuales seleccionamos el menor y éste va a ser nuestro vector de movimiento para esa sub-imagen, por lo que estos pasos se aplican a cada una de las sub-imágenes.

Para poder comparar las búsquedas hacemos el siguiente comparativo donde podemos observar el ahorro computacional al utilizar la variación de los 3 pasos aquí propuesta.

Correlación	Puntos sub-imagen	Puntos imagen	Comparación	
Full	4,225	16,900	100%	—
3 Pasos	625	2,500	$\approx$ 14.8%	100%
Modificada	525	2,100	$\approx$ 12.5%	84%

Tabla 3.2: Comparación de correlación.

En la tabla 3.2 podemos observar que al hacer la correlación total por cada sub-imagen requerimos de 4,225 puntos y por las 4 sub-imágenes requerimos de hacer la correlación de 16,900 puntos, por lo que al hacer uso del tradicional método de los tres pasos por cada sub-imagen requerimos de solo correlacionar 625 puntos con un total de 2,500 puntos por cada imagen lo que nos indica que solo utilizaremos aproximadamente el 14.8% del total, además de observar que el total de los puntos por correlacionar es aproximadamente el 59% de los puntos que se correlacionan en una sub-imagen con el uso de la correlación total. Sin embargo al utilizar la variación de la búsqueda de los tres pasos por cada sub-imagen hacemos 100 punto menos que en el método tradicional

y al comparar la variación con la correlación total solo se requiere de correlacionar aproximadamente el 12.5% de los puntos, pero si lo comparamos con el método tradicional seria correlacionar el 84% de los puntos que se utilizarían con la búsqueda de los 3 pasos. En la figura 3.11 Observamos la diferencia en las ventanas de correlación de estos tres métodos.

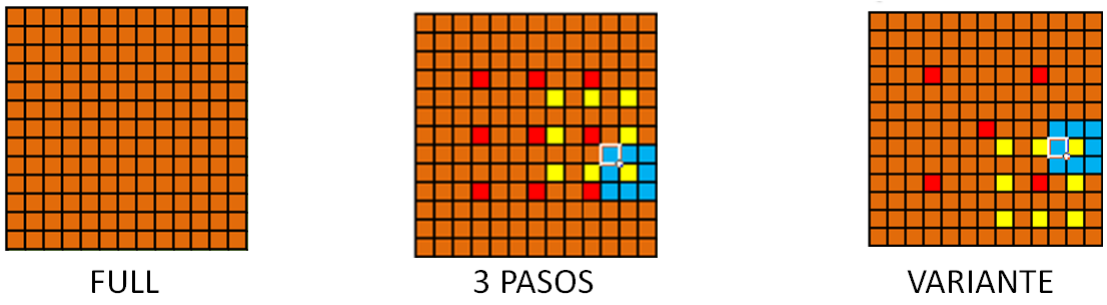


Figura 3.11: Comparación ventana de correlación.

Una vez que tenemos nuestros vectores de movimiento local en cada una de nuestras regiones (S_1, S_2, S_3 y S_4) pasamos a la obtención del vector de movimiento global.

3.2.2 Estimación global.

Los vectores de movimiento local obtenidos de las sub-imágenes resultan no ser confiables, por lo que debemos de obtener el vector de movimiento global. Considerando que las perturbaciones son relativamente lentas en comparación con la velocidad del video, los vectores de movimiento obtenidos deben de ser similares entre ellos [17].

Tomando en cuenta lo anterior utilizamos un esquema simple y robusto para la corrección del movimiento, por lo que el vector de movimiento global está determinado por la selección separada de las medianas para cada elemento del vector de movimiento en

cada sub-imagen. Por lo que el vector de movimiento global se obtiene de la siguiente manera:

$$V_g^t = mediana V_1^t, V_2^t, V_3^t, V_4^t, V_g^{t-1} \tag{3.6}$$

donde V_1^t, V_2^t, V_3^t y V_4^t, son los vectores de movimiento locales actuales y V_g^{t-1}, es el vector de movimiento global anterior.

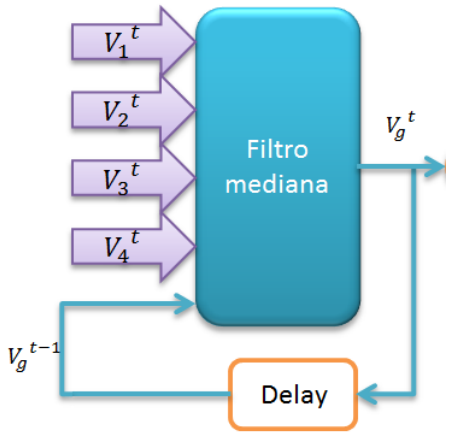


Figura 3.12: Vector de movimiento global.

En la figura 3.12 podemos observar el proceso para la selección del vector de movimiento global, el cual consiste en utilizar el vector de movimiento global anterior y aplicarle la mediana junto a los vectores de movimiento local actual. Ahora que ya obtuvimos nuestro vector de movimiento global pasamos al tercer bloque el cual es el de decisión del movimiento.

3.3 Decisión del movimiento

Sabemos que las direcciones de movimiento son en principio todas paralelas. Una vez que determinamos el vector de movimiento global, el sistema de corrección de movimiento (figura 3.14)en una imagen decide si el movimiento presenciado es causado por el un movimiento intencional o por una perturbación.

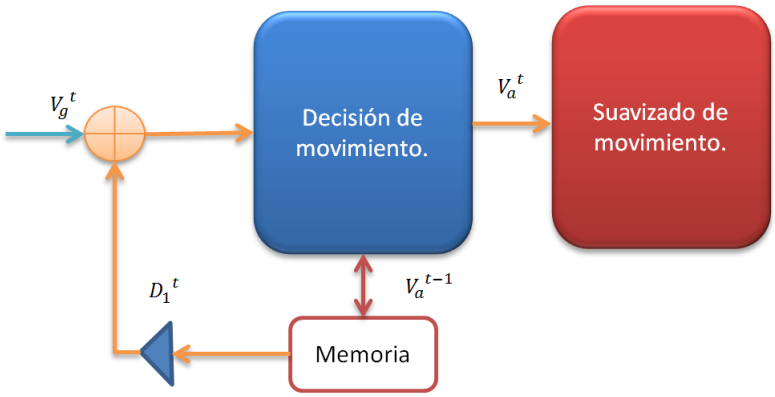


Figura 3.13: Vector de movimiento integrado.

Para tomar dicha decisión es necesario que, el vector de movimiento global se integre con un coeficiente de amortiguamiento, y dicho vector de movimiento integrado designa el vector de movimiento definitivo en una imagen para la corrección de movimiento. El vector de movimiento integrado V multiplicado por el coeficiente de amortiguamiento está dado por:

$$V_a^t = D_1 V_a^{t-1} + V_g^t \tag{3.7}$$

donde V_g^t es el vector de movimiento global obtenido en el bloque anterior y D_1 $(0 < D_1 < 1)$ es el coeficiente de amortiguamiento el cual desempeña un papel importante, ya que el vector de movimiento integrado converge a cero cuando no se presenta ningún movimiento de la cámara.

Para el cálculo del ángulo de rotación entre la imagen actual y la imagen anterior, hacemos uso de los cuatro vectores de movimiento (S_1, S_2, S_3 y S_4) y el vector de movimiento global anterior, de estos cinco vectores obtenemos el vector resultante, y éste nos indica la magnitud y dirección, a este valor al igual que en la traslación le aplicamos un coeficiente de amortiguamiento. Y con ello finalmente tenemos estimados tanto para la traslación como para la rotación.

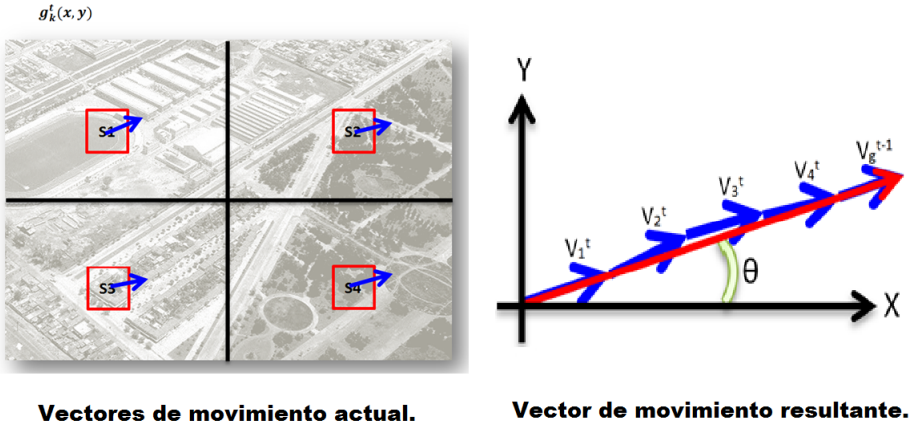


Figura 3.14: Ángulo de movimiento.

3.4 Suavizado del movimiento

Para el lograr el suavizado del movimiento hacemos uso del filtro de Kalman, el cual como ya sabemos nos proporciona una estimación para el estado de un proceso en tiempo discreto. Es un algoritmo unas series de medidas obtenidas por un tiempo, éstas contienen ruido (variaciones aleatorias) y otras inexactitudes, y produce un estimado de las variables desconocidas que tienden a ser más precisas que las basadas en una sola medida. Siendo un poco más formales el filtro de Kalman opera recursivamente sobre un flujo de datos de entrada ruidosos para producir una estimación estadísticamente óptima del estado del sistema subyacente [18].

Un vez que ya comprendemos que el filtro de Kalman lineal es un filtro recursivo y eficiente que estima el estado actual de un sistema dinámico lineal a partir de una serie de medidas ruidosas [19]. El filtro de Kalman asume que el verdadero estado del sistema en el instante k se relaciona con el estado en el instante $k-1$ de acuerdo con el siguiente modelo:

$$x_k = Ax_{k-1} + Bu_k + n_k \tag{3.8}$$

donde x_k es el estado en el instante k, A es la matriz de transición de estado aplicada al estado previo x_{k-1}, B es la matriz de control de entrada aplicada al vector de control u_k y n_k es el ruido blanco presente en el proceso con una distribución normal estándar descrita por la matriz de covarianza Q. En el tiempo k una observación o medida, z_k del estado verdadero de x_k se encuentra dado de la siguiente manera:

$$z_k = Hx_k + v_k \tag{3.9}$$

donde H es la matriz que indica la relación entre las mediciones obtenidas y el vector de estado de observación y v_k es el ruido blanco de valor promedio igual a cero descrito mediante la matriz de covarianza C. Para hacer uso del filtro de Kalman lineal se debe modelar el proceso acorde a las ecuaciones descritas anteriormente.

Dado a que el filtro de Kalman es un estimador recursivo, solo el estado estimado a partir del paso en el instante anterior ($\hat{x}_{k-1}$) y la medida actual (z_k) son necesarias para calcular el estimado del estado actual ($\hat{x}_k$). El filtro de Kalman actúa en dos diferentes fases: la predicción y la corrección. La fase de predicción produce un estimado *a priori* del estado ($\hat{x}_{\bar{k}}$) y el error del filtro, la distribución la cual es representada por la matriz de covarianza ($P_{\bar{k}}$).

$$\hat{x}_{\bar{k}} = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.10}$$

$$P_{\bar{k}} = AP_{k-1}A^T + Q \tag{3.11}$$

En la fase de corrección, el estimado a priori es considerado una combinación lineal del estimado estado a priori y del error (multiplicado por un factor apropiado) entre la observación y la predicción de la observación:

$$\hat{x}_k = \hat{x}_{\bar{k}} + K(z_k - H\hat{x}_{\bar{k}}) \tag{3.12}$$

Este factor es representado por la ganancia de Kalman K:

$$K = P_{\bar{k}}H^T(HP_{\bar{k}}H^T + C)^{-1} \tag{3.13}$$

En el caso de que el proceso tenga mucho ruido, la ganancia de Kalman le da prioridad a la correcćon, mientras que en el caso de que se presente mucho ruido en la observación el término de la estimación del estado a priori domina. La estimación mejorada se denomina como el estimado a posteriori del estado actual ($\hat{x}_k$) y el error del filtro es corregido conforme a la siguiente ecuación:

$$P_k = (I - KH)P_{\bar{k}} \tag{3.14}$$

En la correlación de bloques, el bloque de referencia se actualiza para tener en consideración los cambios que se presentan entre la apariencia del bloque de referencia con respecto al anterior. El filtro de Kalman puede utilizarse para la estimación del bloque de referencia el cual es dado por la imagen k mediante el proceso de modelado como:

$$x_k = R_k = R_{k-1} + n_k \tag{3.15}$$

$$z_k = M_{k-1} = R_{k-1} + v_k \tag{3.16}$$

donde k es el bloque de referencia y M es el bloque mejor correlacionado. Tomando como referencias a 3.8 and 3.9 $A = H = I$ y $B = 0$. Y por consecuencia las ecuaciones 3.11-3.14 son válidas para $x \equiv R$ y $z_k \equiv M_{k-1}$. Las matrices Q y C son consideradas proporcionales a la identidad ($Q = qI, C = cI$) y el método es optimizado en términos de los factores de multiplicación q, c. Al experimentar con estos parámetros se puede

lograr una maximización en el rendimiento siempre que $c >> q$ [20]. Por lo tanto ya que probamos con diferentes valores encontramos de manera heurística que $q = 0.0004$ y $c = 0.99$. Mediante la observación se encuentra que el error es minimizado cuando cada imagen tiende a mantener el estimado a priori, lo cual significa que el bloque previo es el mismo que el bloque de referencia, con una ligera mejora derivada de la diferencia entre el bloque mejor correlacionado y el bloque de referencia de la imagen previa.

3.5 Compensación del movimiento

Anteriormente estimamos el movimiento y suavizamos éste con un filtro de Kalman, y para compensar el movimiento utilizamos el modelo de movimiento afín, el cual es un modelo de movimiento de dos dimensiones, el cual consta de la descripción de la rotación y de la traslación mediante una transformación lineal. En el plano de coordenadas bidimensional, los puntos (x_i, y_i) rotan un ángulo θ alrededor del centro, y obtenemos el punto (x_j, y_j) por medio de la siguiente formula de transformación afín:

$$\begin{bmatrix} x_j \\ y_j \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{3.17}$$

Si hay un movimiento traslacional del centro, la formula 3.17 cambia a la siguiente forma para describir ambos movimientos:

$$\begin{bmatrix} x_j \\ y_j \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} m \\ n \end{bmatrix} \tag{3.18}$$

donde m y n son desplazamientos de la imagen en dirección horizontal y en dirección vertical respectivamente. Y θ toma valor positivo cuando la rotación es en sentido contrario a las manecillas del reloj y negativo cuando la rotación es en dirección a las manecillas del reloj.

Basándonos en el modelo de transformación afín, el movimiento global entre dos imágenes puede aproximarse al movimiento traslacional y rotacional alrededor de un eje óptico, en el marco de referencia con coordenadas (x_i, y_i) transformándose a (x_j, y_j) después de aplicar la rotación y en (x_k, y_k) después de aplicar la traslación. Con ello el algoritmo es capaz de compensar no solo la traslación si no también la rotación.

Con esto podemos aplicar la compensación del movimiento en cada imagen actual con respecto a la imagen anterior y una vez que se compensa el movimiento calculado, se sobrepone la imagen actual sobre la anterior y así se obtiene el video estabilizado con una velocidad de 30 imágenes por segundo. Los resultados se discutirán en la sección correspondiente, donde podremos analizar y comparar los dos algoritmos que se probaron.

CAPÍTULO 4

SEGUNDO ALGORITMO DE ESTABILIZACIÓN

El segundo algoritmo de estabilización en la presente tesis, consta del uso de cuatro de las funciones de OpenCV. Se escogió este algoritmo ya que buscamos hacer una comparación de nuestro primer algoritmo. Con ello se pretende encontrar las ventajas y desventajas de cada uno de los presentes algoritmos. Ya que el primer algoritmo se seleccionó por su bajo costo y complejidad computacional. Las cuatro funciones que utilizamos en la presente tesis son [21]: *GoodFeaturesToTrack, CalcOpticalFlowPyrLK, EstimateRigidTransform y WarpAffine.*

4.1 Detección de características principales.

Existen muchos tipos de características locales que se pueden rastrear. Por ejemplo si tomamos un punto cualquiera en una pared blanca, resulta imposible localizar tal punto en la siguiente imagen, lo mismo ocurre si todos los puntos son idénticos o muy similares. Es por ello que es de suma importancia la selección de los puntos que sean únicos, ya que con ello tenemos mayores posibilidades de localizar dichos puntos en la

siguiente imagen. Por lo tanto, el punto o característica que seleccionemos debe de ser único o por lo menos casi único, además éste debe de ser parametrizable, de tal manera que éste pueda ser comparado con otros puntos en la siguiente imagen. Otro problema que se puede presentar es el problema de la apertura, en donde podemos tomar una fuerte derivada sin embargo ésta puede pertenecer a algún tipo de borde y por ello se puede parecer a otros puntos a lo largo de dicho borde. Por lo que, se procede a observar las derivadas fuertes en ambas direcciones ortogonales, por lo que de esta manera aumentan las posibilidades de que dicho punto sea único. Intuitivamente, las esquinas y no los bordes son los puntos que contienen suficiente información. El método más utilizado para la detección de esquinas, es el propuesto por Harris [22], el cual se basa en la matriz de sus segundas derivadas $(\partial^2 x, \partial^2 y, \partial x \partial y)$ para determinar las intensidades en una imagen.

Podemos pensar en las derivadas de segundo orden de las imágenes, tomadas en todos los puntos de la imagen, como formando nuevas "imágenes de segunda derivada", o cuando se combinan éstas, una nueva imagen Hessiana. Esta terminología proviene de la matriz Hessiana en torno a un punto, la cual es definida en dos dimensiones por:

$$H(p) = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial y \partial x} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \tag{4.1}$$

Para las esquinas de Harris, consideramos la matriz de correlación de la segundad derivada de la imagen sobre una pequeña ventana alrededor de cada punto. Dicha matriz se define como:

$$M(x,y) = \begin{bmatrix} \sum w_{i,j} I_x^2(x+i, y+j) & \sum w_{i,j} I_x(x+i, y+j) I_y(x+i, y+j) \\ \sum w_{i,j} I_x(x+i, y+j) I_y(x+i, y+j) & \sum w_{i,j} I_y^2(x+i, y+j) \end{bmatrix} \tag{4.2}$$

(aquí $w_{i,j}$ es el término de ponderación, éste puede ser uniforme, pero a menudo es utilizado para crear una ventana circular o una ponderación gaussiana). Las esquinas de Harris por definición, *son lugares en la imagen donde la matriz de autocorrelación de las segundas derivadas tiene dos grandes valores propios.* En otras palabras, esto significa que hay textura (o bordes) que va en al menos dos direcciones separadas en torno a un determinado punto. Las segundas derivadas son útiles porque no responden a gradientes uniformes. Cabe mencionar que un gradiente se obtiene de la segunda derivada, por lo que si la primera derivada es uniforme (constante), entonces la segunda derivada es 0. Dicha definición tiene la ventaja adicional de que, si tenemos en cuenta sólo los valores propios de la matriz de autocorrelación, de esta manera estamos considerando valores que son invariantes también en rotación. Estos dos valores no solo determinan si un punto es una buena característica, sino que también proporciona una firma de identificación para dicho punto.

Harris toma el determinante de $H(p)$, restando la traza de $H(p)$ (con un coeficiente de ponderación), y posteriormente comparando esta diferencia con un umbral predeterminado. Posteriormente Shi y Tomasi [23] descubrieron que se tienen buenas esquinas siempre y cuando el menor de los valores propios sea mayor que el mínimo del umbral. La función *cvGoodFeaturesToTrack()* hace uso de la definición de Shi y Tomasi. Esta función calcula las segundas derivadas (utilizando los operadores de Sobel), las cuales son necesarias para el cálculo de los valores propios. A continuación, devuelve una lista de los puntos que satisfacen la definición que fue mencionado anteriormente.

En este caso, la imagen de entrada (*image*) debe de ser de 8-bits o de 32-bits (IPL_DEPTH_8U o IPL_DEPTH_32F) y de un solo canal. La imagen temporal ($tempImage$) y la ($eigImage$) deben de ser imágenes de 32-bits, de un solo canal y del mismo tamaño. Ambas imágenes se utilizan como cero por el algoritmo. En particular, cada entrada contiene el mínimo valor propio para el punto correspondiente en la imagen de entrada. La esquina ($corners$) es una matriz de puntos de 32-bits (CvPoint2D32f), que

```
void cvGoodFeaturesToTrack(
const CvArr     image,
CvArr           eigImage,
CvArr           tempImage,
CvPoint2D32f    corners,
int             corner_count,
double          quality_level,
double          min_distance,
const CvArr     mask        = NULL,
int             block_size  = 3,
int             use_harris  = 0,
double          k           = 0.4
);
```

Tabla 4.1: Función de OpenCV para el cálculo del las características.

contienen los puntos resultantes. Esta matriz debe de asignarse antes de llamar a la función *cvGoodFeaturesToTrack*(). Al asignarse dicha matriz, sólo se asignará una catidad finita de memoria. El término *corner_count* indica el máximo número de puntos a grabar, este término escribe sobre el número de puntos que encuentra realmente. El parámetro *quality_level* indica el mínimo valor propio aceptable para considerar un punto como esquina. El mínimo valor propio utilizado para el punto de corte es el producto del término *quality_level* y el más pequeño valor propio que se observe en la imagen. Por lo tanto el valor de *quality_level* no debe exceder de 1 (un valor típico podría ser 0.10 ó 0.01). Por ejemplo si el la minima medida es 1200 y *quality_level* es igual a 0.01 entonces los valores menores a 12 serán desechados. El parámetro *min_distance* garantiza que no existan dos puntos entre ese número de píxeles.

El término *mask* es opcional ya que usualmente es la imagen, ésta es interpretada como valores booleanos, indica que puntos deben o no ser considerados como posibles esquinas. Si *mask* se establece como $NULL$, esto indica que esta mascara no se utilizará. El *block_size* es la región en torno a un determinado píxel, el cual es considerado cuando se calcula la matriz de correlación de las derivadas. Resulta ser mejor resumir las derivadas a través de una pequeña ventana para calcular su valor en un solo punto (es decir, un *block_size* de 1). Si *use_harris* es distinto de cero, entonces la definición de esquina de Harris se utiliza en lugar de la definición de Shi-Tomasi. Si se establece *use_harris* con un valor diferente de cero, entonces el valor de k es el coeficiente de ponderación utilizado para establecer el peso relativo dado a la traza de la matriz Hessiana de correlación en comparación con el determinante de la misma matriz. El resultado de utilizar la función $cvGoodFeaturesToTrack()$, es un conjunto de ubicaciones de píxeles, los cuales esperamos encontrar en otra imagen similar. En nuestro caso, buscamos dichas características en las imágenes posteriores contenidas en el vídeo.

Figura 4.1: Características principales.

4.2 Flujo óptico

El flujo óptico es el patrón de movimiento aparente de los objetos, superficies y bordes en una escena causado por el movimiento relativo entre un observador (cámara) y la escena, refleja los cambios en la imagen debido al movimiento de los objetos presentes en la escena. En otras palabras, el flujo óptico es el movimiento aparente de los pixeles de una imagen a otra imagen en una secuencia de vídeo. En tres dimensiones el flujo óptico está definido como una medida de cómo se movieron los puntos en el espacio. El cálculo del flujo óptico no es fácil, por ello algunas de las hipótesis utilizadas son: intensidad constante, rigidez de los objetos, coherencia espacial, entre otras, sin embargo generalmente estas hipótesis no cumplen con escenarios reales.

Como ya se mencionó, es muy frecuente que se busque evaluar el movimiento entre dos imágenes (o secuencia de imágenes) sin tener previo conocimiento de dichas imágenes. Un desplazamiento entre dos imágenes es representado por la distancia que ha recorrido un píxel entre la imagen anterior y la actual. Un método aparentemente sencillo es intentar igualar las ventanas alrededor de cada píxel de una imagen a la próxima imagen, a esto se le conoce como la correspondencia a bloques.

En la práctica no es fácil calcular el flujo óptico, por ejemplo si queremos calcular el flujo óptico de una hoja blanca de papel, esta tarea resulta técnicamente imposible. Por lo que, los métodos densos deben de tener un método de interpolación entre los puntos que sean más fáciles de seguir, esto con el fin de resolver los puntos que son más ambiguos. Una de las principales desventajas de utilizar el cálculo del flujo óptico denso es el aumento del coste computacional.

Existen algunos algoritmos para los casos donde se tiene un escaso flujo óptico. Estos algoritmos se basan en la especificación de los puntos que van a ser rastreados.

Si dichos puntos cuentan con ciertas propiedades deseables, tales como las esquinas, donde el seguimiento será relativamente robusto y fiable. Es por ello que utilizamos el método de Lucas Kanade, éste también se usa con una construcción de pirámides de cada imagen, lo que permite un seguimiento de los movimientos de una manera más rápida.

4.2.1 Método de Lucas-Kanade

Este método se aplica fácilmente a un subconjunto de puntos en la imagen de entrada. El algoritmo de Lucas Kanade (LK) se puede aplicar en un contexto donde el flujo óptico es escaso, ya que se basa únicamente en la información local que se deriva de una pequeña ventana que rodea cada uno de los puntos de interés. Una de las desventajas de utilizar pequeñas ventanas locales, tal como se hace en LK es que los grandes movimientos pueden mover algunos puntos fuera de la ventana local y por lo tanto resulta imposible de hallar estos puntos para el algoritmo. El problema anterior se soluciona con el algoritmo de LK piramidal, el cual sigue a partir del nivel más alto de una pirámide de una imagen (detalle más bajo) y trabajando a niveles más bajos (detalles más finos). El seguimiento sobre las pirámides en una imagen permite a las ventanas locales capturar grandes movimientos.

¿Cómo funciona el algoritmo de Lucas Kanade?

La idea básica del algoritmo de LK se basa en tres supuestos, los cuales son:

1. *Constancia en brillo.* Un píixel de la imagen de un objeto en la escena no cambia en apariencia, ya que (posiblemente) éste se mueve de cuadro a cuadro. Para las imágenes en escala de grises (LK también puede aplicarse en color) suponemos que el brillo de un píxel no cambia, ya que se hace un seguimiento de cuadro a cuadro.

2. *Persistencia temporal o "movimientos pequeños"*. El movimiento en la superficie de una imagen cambia lentamente en el tiempo. En otras palabras, esto significa que los incrementos temporales son lo suficientemente rápidos en relación con la escala de movimiento en la imagen, por lo que el objeto no se mueve mucho de una imagen a otra.

3. *Coherencia espacial*. Los puntos vecinos en una escena que pertenece a la misma superficie, tienen movimiento similar, así como la proyección de puntos cercanos en el plano de la imagen.

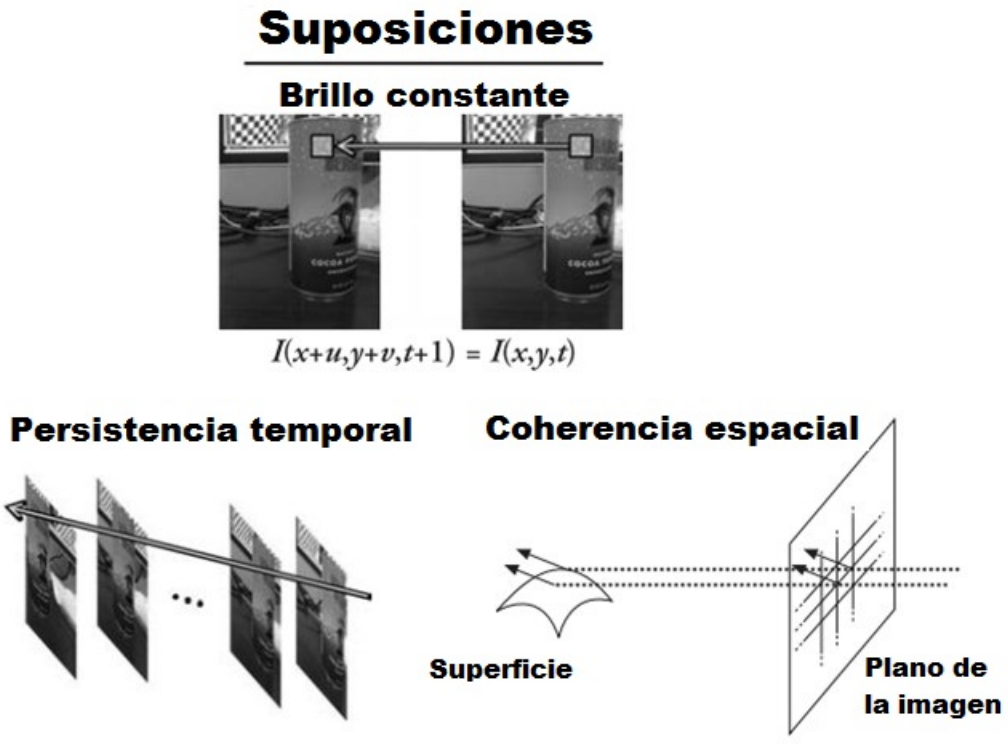


Figura 4.2: Lucas-Kanade supuestos.

En la figura 4.2 se muestran los supuestos que anteriormente se mencionaron. El primer requisito, la constancia de brillo, solo indica que los píxeles de una superficie

tienen el mismo aspecto con el paso del tiempo, por lo que:

$$f(x,t) \equiv I(x(t),t) = I(x(t+dt),t+dt) \tag{4.3}$$

Esto es bastante simple ya que significa que nuestra intensidad de los píxeles rastreados no presenta ningún cambio a través del tiempo:

$$\frac{\partial f(x)}{\partial t} = 0 \tag{4.4}$$

La persistencia temporal que es el segundo supuesto, significa en esencia que los movimientos presentes de cuadro a cuadro son pequeños. En otras palabras, podemos ver el cambio como la aproximación de la derivada de la intensidad con respecto al tiempo (es decir se afirma que el cambio entre una imagen y la siguiente, es una secuencia diferencialmente pequeña). Para comprender las implicaciones de este supuesto, consideramos el caso de una sola dimensión espacial. En este caso comenzamos con la ecuación de consistencia de brillo, sustituimos la definición de brillo $f(x,t)$ mientras que tenemos en cuenta la dependencia implícita de x en t, $I(x(t),t)$ y luego aplicamos la regla de la cadena para la diferenciación parcial. Por lo que:

$$\underbrace{\left.\frac{\partial I}{\partial x}\right|_t}_{I_x} \underbrace{\left(\frac{\partial x}{\partial t}\right)}_{v} + \underbrace{\left.\frac{\partial I}{\partial t}\right|_{x(t)}}_{I_t} = 0 \tag{4.5}$$

donde Ix es la derivada espacial a través de la primera imagen. Esto es la derivada entre las imágenes con el tiempo y v es la velocidad que se está buscando. De esta manera llegamos a la sencilla ecuación para le velocidad de flujo óptico en el caso de una sola dimensión:

$$v = -\frac{I_t}{I_x} \tag{4.6}$$

Una vez que se analizó la parte unidimensional, vamos a pasar a las imágenes en dos dimensiones, por lo que hacemos uso de y que es componente de la velocidad v y a x que es componente de la velocidad u, por lo que tenemos:

$$I_x u + I_y v + I_t = 0 \tag{4.7}$$

Desafortunadamente en esta simple ecuación tenemos dos incógnitas por cada píxel. Esto significa que las mediciones se encuentran restringidas y no se pueden utilizar para obtener una única solución para el movimiento de dos dimensiones en un determinado punto. En lugar de ello, solo podemos resolver para el componente de movimiento perpendicular o "normal" a la línea descrita por la ecuación de flujo. La figura 4.4 presenta los detalles matemáticos y geométricos.

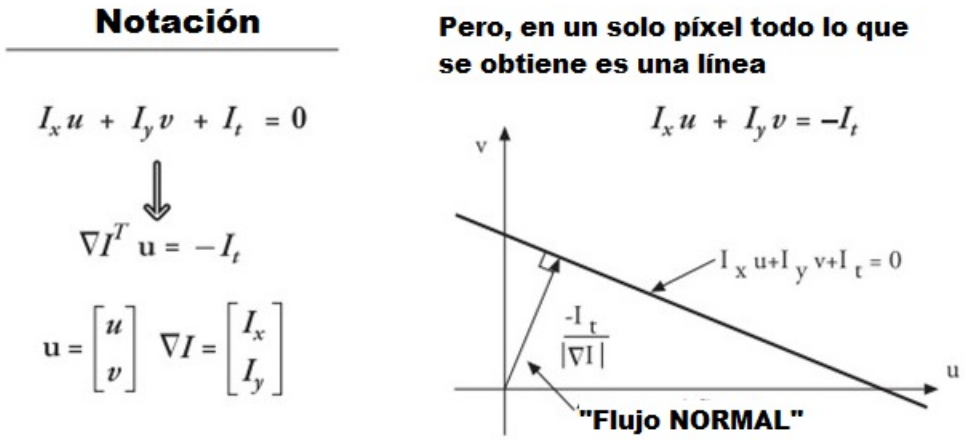


Figura 4.3: Flujo óptico en dos dimensiones para un sólo píxel.

Normalmente los resultados del flujo óptico ante el problema de la apertura, el cual surge cuando se tiene una pequeña apertura o ventana en la cual se mide el movimiento. Siempre que se tienen una abertura pequeña y detectamos un movimiento, observamos un borde y no una esquina. Sin embargo es insuficiente con un solo borde determinar cómo (en qué dirección) se mueve todo el objeto (figura 4.4). Es por ello que recurrimos al último supuesto del flujo óptico.

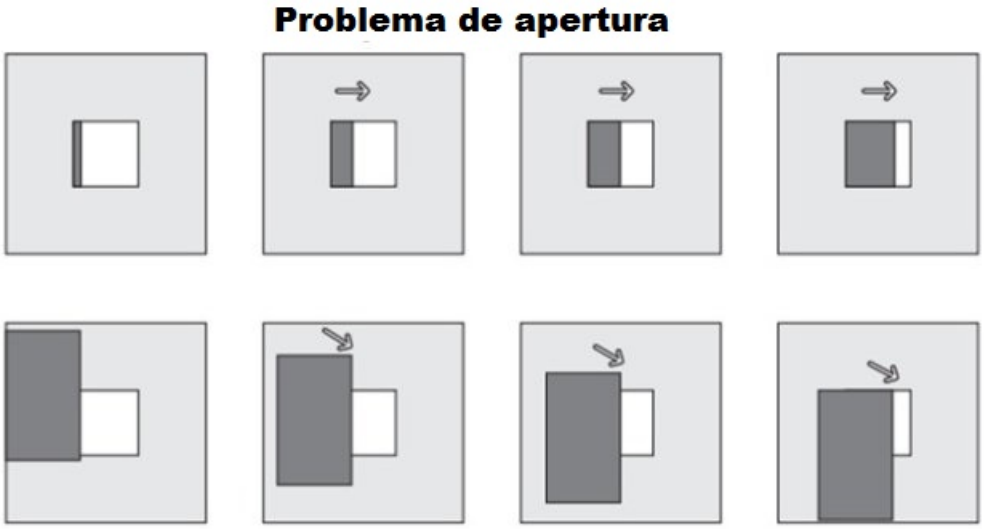


Figura 4.4: Problema de apertura.

Si un conjunto local de píxeles se mueve coherentemente (congruencia en el movimiento), entonces podemos encontrar fácilmente el movimiento que sufrió el píxel central, mediante el uso de los píxeles circundantes para establecer un sistema de ecuaciones. Por ejemplo, si utilizamos una ventana de valores de brillo de $5x5$ tomando en cuenta que la ventana puede ser de $3x3$, $7x7$ o cualquiera que se seleccione. Si la ventana es muy grande, entonces podemos contradecir el supuesto de que el movimiento es coherente y con ello el algoritmo sería incapaz de determinar el movimiento. De otro modo si la ventana es demasiado pequeña, de nuevo tendremos el problema de apertura, (en escala de grises, para color se debe triplicar) para calcular el movimiento alrededor de un píxel se pueden configurar 25 ecuaciones como se muestra a continuación:

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{A_{25X2}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d_{2X1}} = - \underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_{b_{2X1}} \tag{4.8}$$

Ahora tenemos un sistema más restringido, del cual podemos obtener una solución que contenga más que solo un borde en la ventana de $5x5$. Para resolver este sistema, se realiza una minimización de la ecuación de mínimos cuadrados, por lo cual $min||Ad-b||^2$ se resuelve de manera estándar como:

$$\underbrace{\left(A^T A\right)}_{2X2}\underbrace{d}_{2X1} = \underbrace{A^T b}_{2X2} \tag{4.9}$$

A partir de esta relación obtenemos los componentes de movimiento u y v. Con más detalle tenemos que:

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = -\underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b} \tag{4.10}$$

La relación de dichas ecuaciones es:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \left(A^T A\right)^{-1} A^T b \tag{4.11}$$

$\left(A^T A\right)$ es invertible cuando es de rango completo (2), lo cual ocurre cuando tiene dos grandes vectores propios. Esto ocurre en las regiones donde la imagen contiene textura en al menos dos direcciones. En este caso $\left(A^T A\right)$ tendrá las mejores propiedades, cuando la ventana de seguimiento se centra sobre la región de una esquina.

El flujo óptico de Lucas Kanade por sí solo no funciona muy bien, ya que necesitamos de una ventana más grande para capturar los grandes movimientos, pero una gran ventana puede romper la coherencia en el movimiento. Con el fin de evitar dicho problema, primero se rastrea en escalas espaciales más grandes usando una pirámide de imagen y después se refina los supuestos iniciales en la velocidad del movimiento, y recorriendo los niveles de la pirámide hasta que se llega a los píxeles en bruto. La técnica más usual es primero resolver el flujo óptico en la capa superior y después se utilizan las

estimaciones de movimiento resultantes como punto de partida para la siguiente capa de la pirámide. Se sigue bajando la pirámide hasta llegar al nivel más bajo. De esta manera se minimizan las suposiciones de movimiento erróneas y así se puede rastrear más rápido los movimientos más largos. Esta función se ilustra en la figura 4.5.

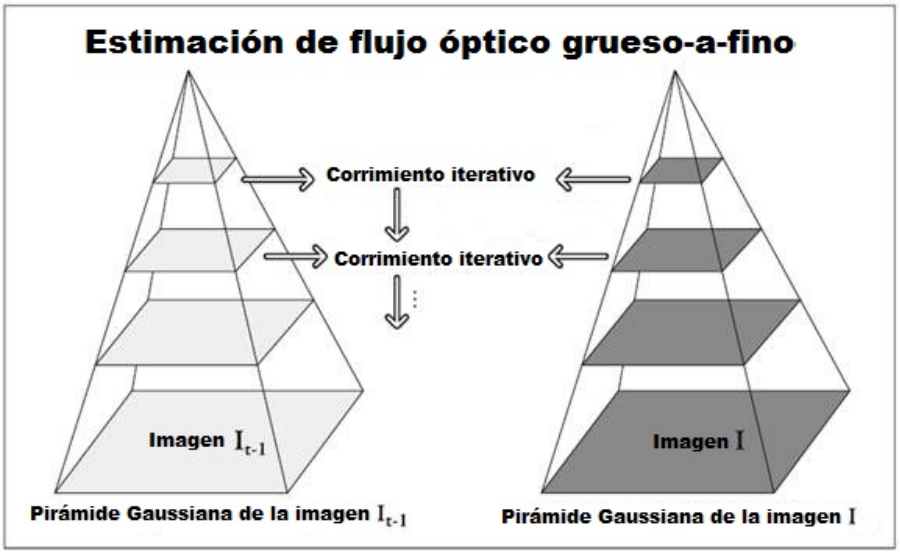


Figura 4.5: Lucas-Kanade piramidal.

La función en OpenCV que implementa el flujo óptico denso piramidal por Lucas-Kanade es $cvCalcOpticalFlowPyrLK()$. Esta función se apoya de la función anteriormente vista $goodFeaturesToTrack$ y también indica si se está llevando de manera adecuada el seguimiento en cada punto.

Como se puede observar esta función tiene muchas entradas. Los primeros dos argumentos son las imágenes inicial y final respectivamente, ambas deben de ser de un solo canal y de 8 bits. Los siguientes dos argumentos son buffers asignados para almacenar las imágenes de la pirámide. El tamaño de estos buffers debe de ser de al menos *el ancho de la imagen + 8 X la altura de la imagen /3 bytes* esto debido a que se necesita no solo acomodar la imagen sino acomodar toda la pirámide para cada una de las imágenes de entrada ($pyrA$ y $pyrB$). (Si estos dos punteros son establecidos en $NULL$ entonces automáticamente se liberará la memoria adecuada, sin embargo esto

```
void cvCalcOpticalFlowPyrLK(
const CvArr        imgA,
const CvArr        imgB,
CvArr              pyrA,
CvArr              pyrB,
CvPoint2D32f       featuresA,
CvPoint2D32f       featuresB,
int                count,
CvSize             winSize,
int                level,
char               status,
float              track_error,
CvTermCriteria     criteria,
int                flags
);
```

Tabla 4.2: Función de OpenCV para el cálculo del flujo óptico

no es bueno para el rendimiento). El arreglo $featuresA$ contienen los puntos para los cuales se detecta el movimiento y $featuresB$ es un arreglo similar donde son colocadas las nuevas ubicaciones de $featuresA$, *count* es el número de puntos en la lista de $featuresA$. La ventana que se utiliza para calcular el movimiento local coherente está dada por *winSize*. Debido a que se construye una imagen piramidal, el argumento *level* se utiliza para ajustar la profundidad de la pila de imágenes. Si *level* es 0, entonces las pirámides no son utilizadas. El parámetro *status* es de la longitud de *count*. Al finalizar la función, cada entrada en *status* será 1 (si se ha encontrado el punto correspondiente en la segunda imagen) o 0 (si no se ha encontrado). El parámetro *track_error* es opcional y se pude desactivar estableciéndola en $NULL$. Si *track_error* está activo,

entonces éste es una serie de números, uno para cada punto de seguimiento, igual a la diferencia entre el parche alrededor de un punto rastreado en la primer imagen y el parche alrededor de la ubicación en la que este punto fue rastreado en la segunda imagen. Se puede utilizar *track_error* para quitar puntos cuyos cambios son tan grandes como los puntos en movimiento.

```
cvTermCriteria(
int      type,
int      max_iter,
bouble   epsilon
);
```

Tabla 4.3: Función de OpenCV para generar la estructura de datos.

Normalmente el término *criteria* se utiliza para generar la estructura que necesitamos. El primer argumento debe ser *CV_TERMCRIT_ITER o CV_TERMCRIT_EPS*, el cual le indica a la función si queremos terminar después de un cierto número de iteraciones o cuando la métrica converga a algún valor pequeño (respectivamente). Los siguientes dos argumentos establecen los valores en los que uno, el otro o ambos criterios deben terminar la función. Finalmente, $flags$ permite algún control fino de la contabilidad interna de la rutina; puede ajustarse a cualquiera o todos (usando bit a bit OR) de los siguientes:

- $CV\_LKFLOW\_PYR\_A\_READY$ La pirámide para la primera imagen es calculada antes de llamar y almacenar en $pyrA$.
- $CV\_LKFLOW\_PYR\_B\_READY$ La pirámide para la segunda imagen es calculada antes de llamar y almacenar en $pyrB$.
- $CV\_LKFLOW\_INITIAL\_GUESSES$ La matriz B ya contiene una estimación inicial de las coordenadas de las características cuando la rutina es llamada.

Estas banderas son particularmente útiles para la manipulación de la secuencia del video. Las pirámides de la imagen tienen un cierto costo computacional, por lo que se debe evitar recalcular éstas siempre que sea posible. La imagen final para el par de imagenes que se calculó será la imagen inicial para la siguiente pareja que será calculada. Si estos buffers son asignados por uno mismo (en lugar de que la rutina lo haga), entonces las pirámides de cada imagen estarán contenidas en estos buffers cuando regrese la rutina. Si se le indica a la rutina que esta información ya se encuentra calculada, entonces ya no será recalculada. Del mismo modo, si calculamos el movimiento de los puntos de la imagen anterior entonces podremos hacer una buena conjetura inicial de hacia dónde estará la siguiente imagen.

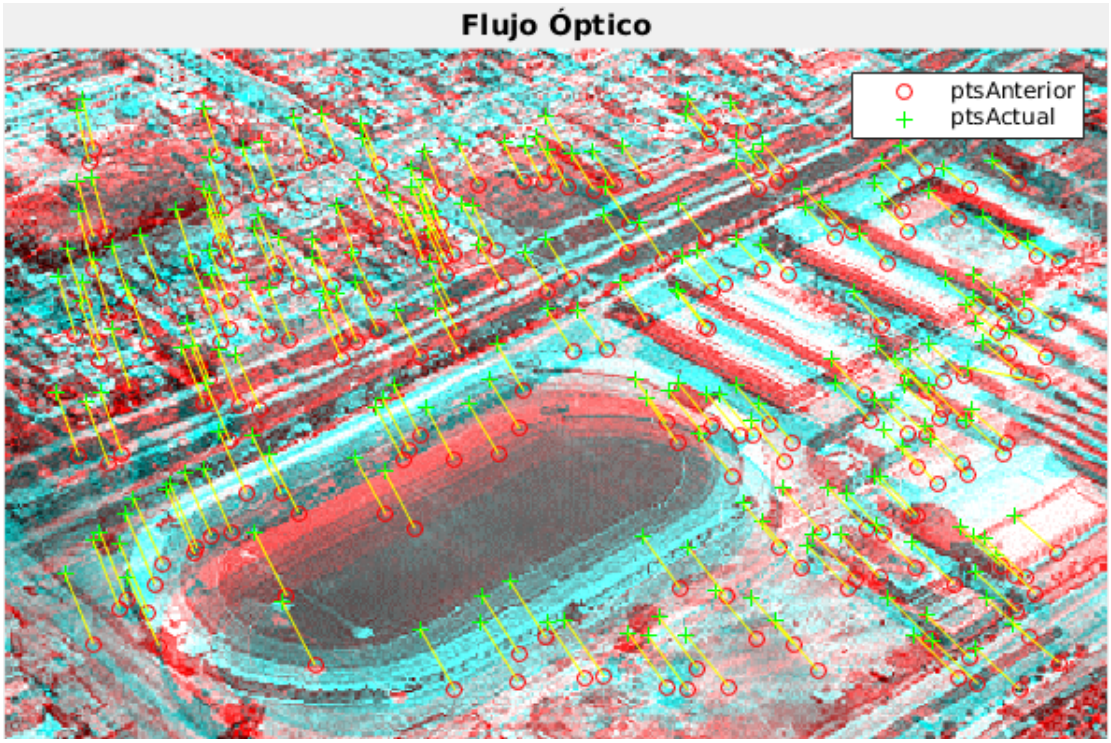


Figura 4.6: Resultado del flujo óptico.

Por lo que el procedimiento es simple: primero se suministran las imágenes, se hace una lista de los puntos a seguir $featuresA$ y se llama a la función. Cuando la función

regresa, devuelve la matriz de estado para ver qué puntos fueron rastreados con éxito y luego se verifica $featuresB$ para hallar la nueva ubicación de estos puntos. Un ejemplo se muestra en la figura 4.6.

4.3 Transformación afín

Existen dos tipos de transformaciones geométricas para áreas planas. Las trasformaciones que usan una matriz de $2X3$ son llamadas *transformaciones afín* y las transformaciones basadas en una matriz de $3X3$, las cuales son llamadas *homografía*. La homografía se puede ver como un método para calcular la forma en la que un plano en tres dimensiones es percibido por un particular observador, quien podría no estar en línea de vista directa con el plano.

Como solo corregimos el movimiento en traslación y rotación entonces haremos uso de la transformación afín, la cual es cualquier transformación que puede ser expresada en forma de una multiplicación de la matriz seguido por una adición vectorial. En OpenCV el estilo estándar para representar una transformación de este tipo es como una matriz de $2X3$. Donde definimos, lo siguiente:

$$A \equiv \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \quad B \equiv \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \quad T \equiv \begin{bmatrix} A & B \end{bmatrix} \quad X \equiv \begin{bmatrix} x \\ y \end{bmatrix} \quad X' \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.12}$$

Es fácil de observar que el efecto de la transformación afín $A \cdot X + B$ es exactamente equivalente a la extensión del vector X en el vector X' y simplemente multiplicar por la izquierda X' por T. Las transformaciones afín se pueden visualizar de la siguiente manera. Cualquier paralelogramo $ABCD$ en un plano se puede asignar a cualquier paralelogramo $A'B'C'D'$ por alguna transformación. Si las áreas de estos paralelogramos son diferentes de cero, entonces la transformación afín implícita se define únicamente

por (tres vértices de) los dos paralelogramos. Si se quiere, se puede pensar en una transformación afín como la elaboración de su imagen en una hoja, empujando o tirando de las esquinas para hacer diferentes tipos de paralelogramos

Cuando tenemos dos imágenes que tienen una ligera diferencia entre ellas, buscamos calcular las transformaciones reales que relacionen los diferentes puntos de vista. En este caso, las transformaciones afín se utilizan a menudo para modelar los diferentes planos de vista porque se tiene menos parámetros y es más fácil de resolver. La desventaja es que la verdadera distorsión de perspectiva solo puede ser modelada por una homografía, por lo que una transformación afín da una representación que no da cabida a todas las posibles relaciones entre diferentes planos de vista. Por otro lado, para pequeños cambios en los planos de vista de una distorsión resultante es afín, por lo que en algunas circunstancias es suficiente con el cálculo de la transformación afín.

Las transformaciones afín pueden convertir rectángulos en paralelogramos. Pueden aplastar la forma, pero deben mantener los lados paralelos; que pueden girar y/o reducir la escala. La homografía ofrece más flexibilidad; una perspectiva transformada puede convertir un rectángulo en un trapezoide. Por supuesto, las transformaciones afín son un subconjunto de homografías. En la figura 4.7 observamos ejemplos de transformaciones afín y de homografías.

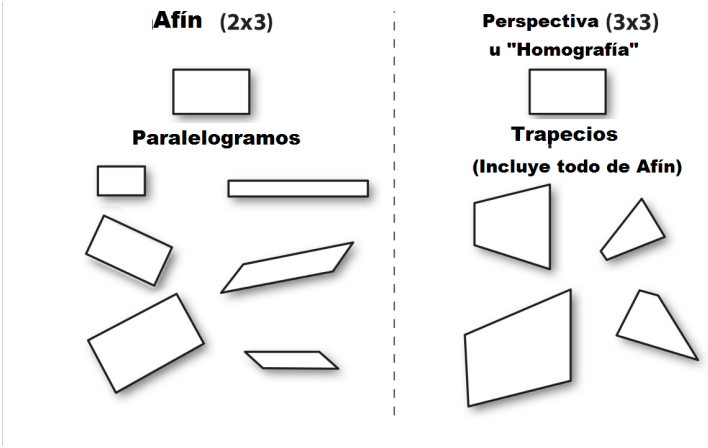


Figura 4.7: Transformación afín y homografía.

Hay dos transformaciones que se presentan al trabajar con transformaciones afín. En primer caso tenemos una imagen (o una región de interés) que queremos transformar; en el segundo caso, tenemos una lista de los puntos para los que nos gustaría calcular la transformación resultante.

En el primer caso, los formatos de entrada y salida son obviamente imágenes, y el requisito implícito es que la deformación asume que los píxeles son una representación densa de la imagen subyacente. Esto significa que la deformación de la imagen debe manejar necesariamente interpolaciones de manera que las imágenes de salida son suaves y de un aspecto natural. La función de transformación afín que proporciona OpenCV para el cálculo de la transformación es $cvWarpAffine()$ donde:

```
cvWarpAffine(
const CvArr    src,
CvArr          dst,
const CvMat    map_matrix,
int            flags    =CV_INTER_LINEAR — CV_WARP_FILL_OUTLIERS,
CvScalar       fillval  = cvScalarAll(0)
);
```

Aquí *src* y *dst* se refieren a una imagen, que puede ser de uno o de tres canales y de cualquier tipo (siempre que ambas sean del mismo tipo y tamaño). El término *map_matrix* es la matriz de $2X3$ que se introdujo antes, la cual cuantifica la transformación deseada. El penúltimo argumento $flags$ controla el método de interpolación, así como una o ambas de las siguientes opciones adicionales (como es de costumbre, se combinan con operadores OR booleanos).

- *CV_WARP_FILL_OUTLIERS* Generalmente la imagen transformada de *src* no encaja perfectamente en la imagen *dst*, ya que hay algunos píxeles "mapeados" desde el archivo de origen que en realidad no existen. Si se establece este indicador,

los valores perdidos serán llenos con $fillval$ (llena de negro los píxeles faltantes).

- *CV_WARP_INVERSE_MAP* Esta bandera permite por conveniencia, la deformación inversa de *dst* a *src* en lugar de desde *src* a *dst*.

Para poder utilizar la función anterior requerimos de utilizar primero la función de $estimateRigidTransform$, la cual calcula una óptima transformación afín entre dos conjuntos de puntos en 2D. Los detalles de esta función son:

Mat estimateRigidTransform(InputArray ***src****, InputArray* ***dst****, bool* ***fullAffine****)* donde **src** es la entrada del conjunto de puntos 2D y se almacenan en *std* :: *vector* o Mat, o la imagen almacenada en Mat. **dst** es la segunda entrada del conjunto de puntos 2D, debe de ser del mismo tipo y tamaño que el primer término. **fullAffine** si si valor es verdadero, la función encuentra una óptima transformación afín sin restricciones adicionales (6 grados de libertad). De lo contrario, la transformación será limitada a traslación, rotación y escalamiento uniforme (5 grados de libertad).

Por lo que podemos concluir que primero se deben de obtener las caracteristicas principales tanto de la imagen actual como de la imagen anterior. Posteriormente se obtiene el flujo óptico de dichos funtos. Después pasamos a estimar la transformada rigida afín y por último se aplica la corrección de la transformación previamente calculado.

CAPÍTULO 5

RESULTADOS Y CONCLUSIONES

Este capítulo esta dedicado a analizar y comparar los resultados que se obtuvieron durante la elaboración de esta tesis. Como primer instancia veremos la simulación del modelado y control para un Gimbal de dos grados de libertad y las gráficas que se obtuvieron de dicha simulación. Posteriormente observaremos las gráficas de los resultados experimentales que se obtuvieron del primer y segundo algoritmo de estabilización. A cada algoritmo se le aplicaron tres pruebas. Las dos primeras pruebas consisten en dos vídeos perturbados manualmente. El primer vídeo es a base de una imagen que se trasladó en los dos ejes y en las cuatro direcciones a una longitud de -15 píxeles a +15 píxeles. El segundo vídeo es para la rotación el cual consta en rotar una imagen de -5° a +5°. La última prueba consiste en un vídeo obtenido en un vuelo en tiempo real. Las gráficas obtenidas de dichas pruebas serán comparadas con el objetivo de obtener una conclusión respecto a ambos algoritmos de estabilización.

5.1 Simulación de un control PD en un Gimbal de 2GDL

El primer resultado a analizar es la simulación que se realizó en MATLAB- SIMULINK. En esta simulación se aplicó el modelado de sistema, el cual se obtuvo en el capítulo 2 de la presente tesis, de la misma manera se introdujó un control PD capaz de seguir una referencia establecida. En la figura 5.1 se muestra como quedó la simulación en SIMULINK.

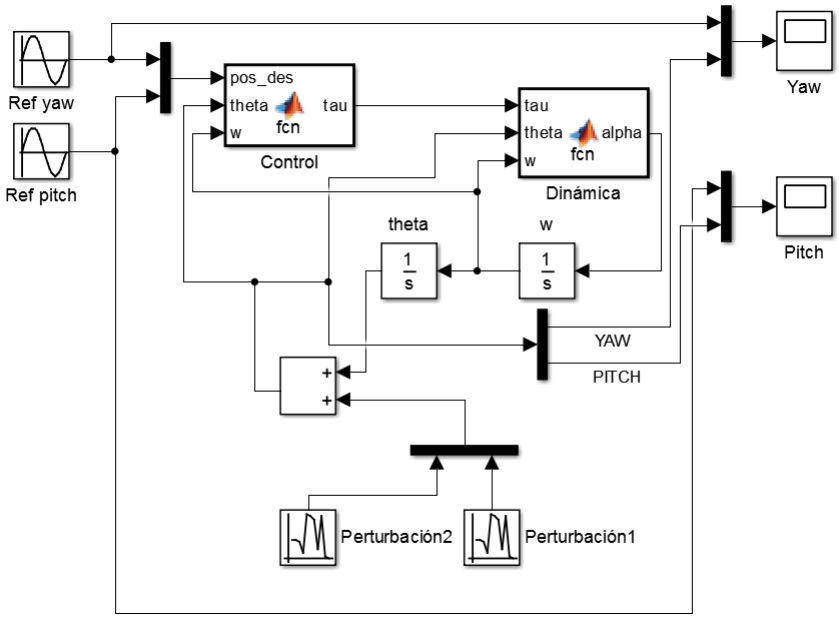


Figura 5.1: Simulación PD.

En la Figura 5.1 podemos observar que a nuestro control le aplicamos dos referencias ya que este análisis se basa en un Gimbal de dos grados de libertad. Por lo tanto, tenemos una referencia por cada grado de libertad (Yaw y $Pitch$). Como referencia de Yaw es un seno de amplitud $= 1$ y con frecuencia de 1 rad/seg. Para la referencia de $Pitch$ utilizamos también un seno pero de amplitud $= 2$ y frecuencia de 2 rad/seg, las cuales como ya se mencionó anteriormente, las utilizamos en nuestro control, además del ángulo θ, la velocidad angular ω y las ganancias K_p y K_v, para hacer uso de la parte

derecha de la ecuación 2.26, la cual es:

$$\tau = K_p\tilde{q} - K_v\dot{q} \tag{5.1}$$

donde $\tilde{q}$ es el error y es igual a $\tilde{q} = q_d - q$

Cabe mencionar que se por comodidad se utilizó un control PD y también porque el control del Gimbal no es el tema medular de esta tesis.

Una vez que tenemos nuestra ley de control, ésta la aplicamos en la dinámica de nuestro sistema (la parte izquierda de la ecuación 2.26). Una vez que ya observamos nuestra simulación en SIMULINK obtenemos las siguientes gráficas:

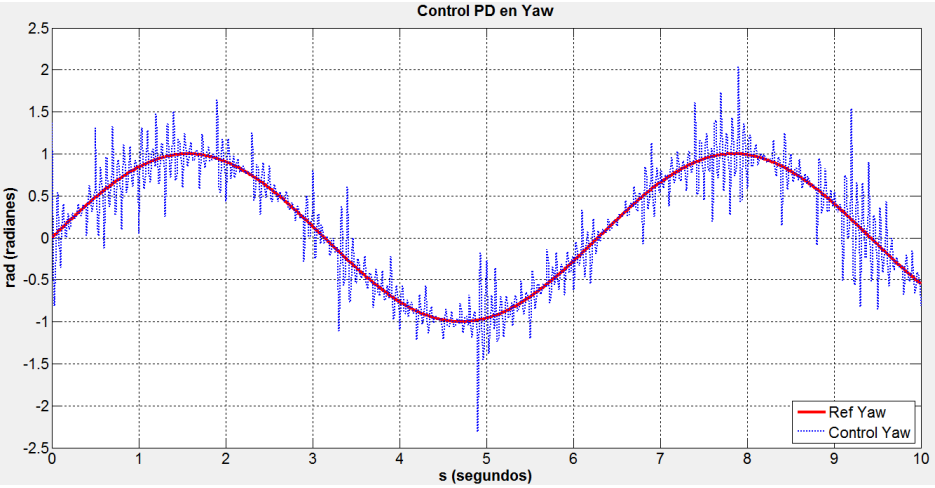


Figura 5.2: Control PD en Yaw.

En nuestra figura 5.2 observamos que a pesar de que al sistema le aplicamos una perturbación (simula el viento o las vibraciones mecánicas tanto en el gimbal como en la plataforma), con la ley de control PD el sistema trata de seguir la referencia, la cual se encuentra dada por una línea continua, y el control en Yaw, lo podemos observar en línea azul.

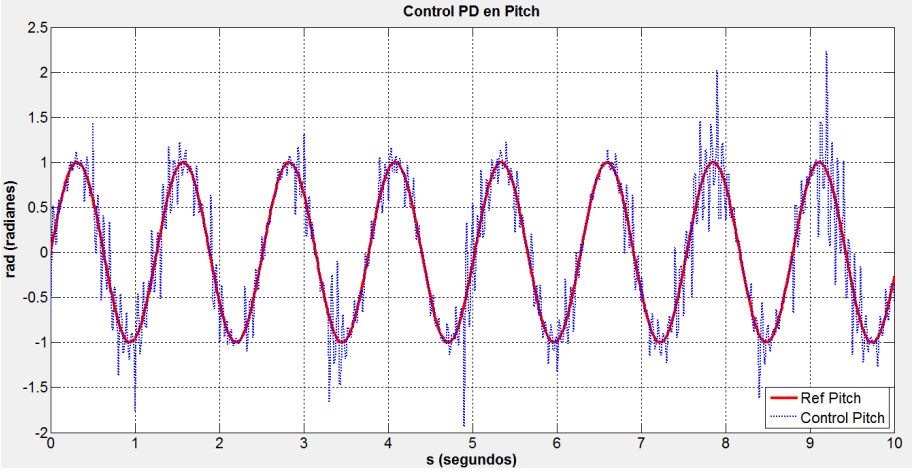


Figura 5.3: Control PD en Pitch.

De igual manera que en la figura 5.2, en la cual analizamos los resultados del control en Yaw. En la figura 5.3, en línea continua observamos la referencia que debe de seguir el control en $Pitch$, en línea azul está el comportamiento del sistema Gimbal. Él cual como ya mencionamos anteriormente se encuentra controlado por un control PD. En ambos casos podemos concluir que el control cumple con su objetivo, el cual es lograr que el sistema siga a la referencia que se le plantea. En la figura 5.4 podemos observar el comportamiento del sistema al ser controlado por un control PD en ambos grados de libertad (Yaw y $Pitch$).

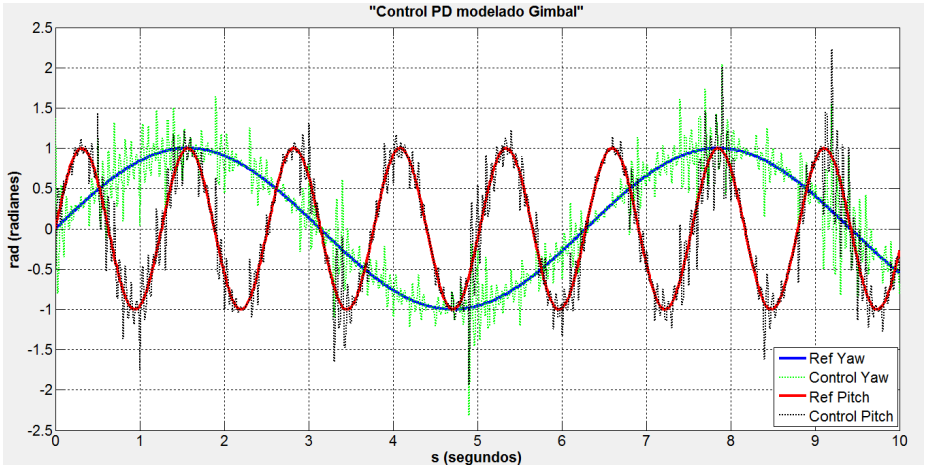
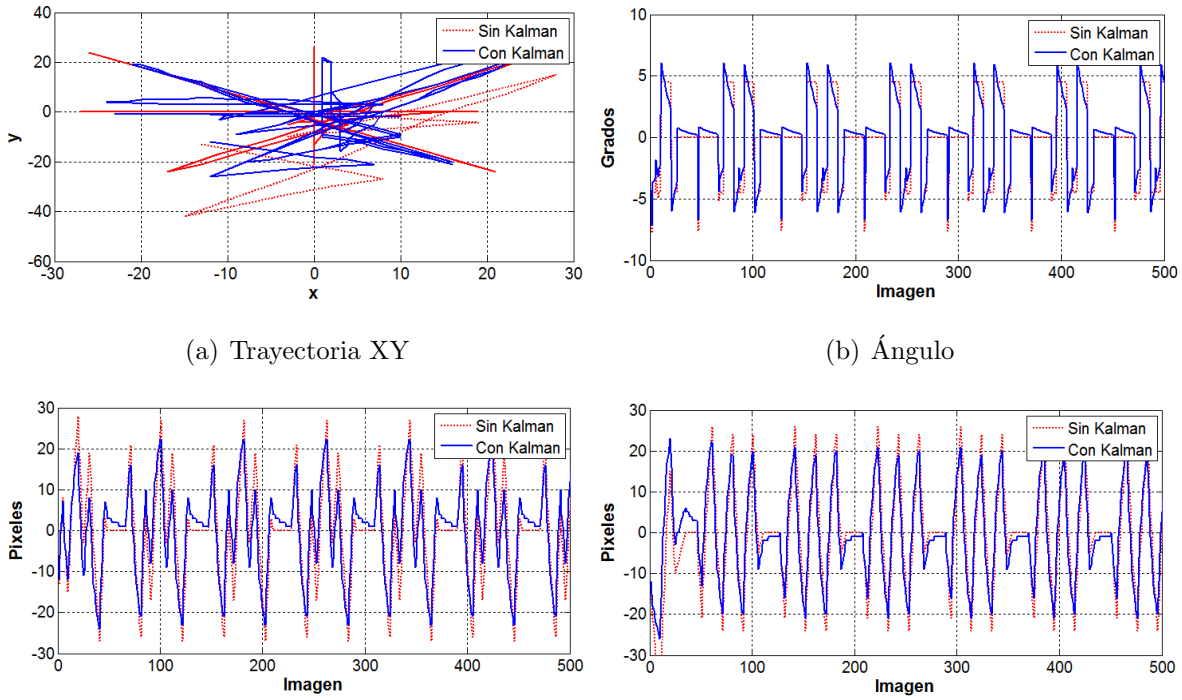


Figura 5.4: Control PD en el Gimbal.

5.2 Resultados del primer algoritmo de estabilización.

Como ya se mencionó anteriormente, para probar el funcionamiento del primer algoritmo, el cual se explica a detalle en el capítulo 3. Utilizamos tres vídeos, el primero fue elaborado artificialmente con movimientos únicamente traslacionales en los dos ejes en las cuatro direcciones con magnitud de -15 a +15 píxeles. Al mencionar un vídeo elaborado artificialmente nos referimos a que tomamos una imagen y la rotamos o trasladamos sobre la imagen anterior. En la gráfica 5.5 que se muestra a continuación, podemos observar el comportamiento del algoritmo basado en la correlación de los planos de bits en código Gray ante los movimientos de traslación.



(a) Trayectoria XY

(b) Ángulo

(c) Trayectoria X

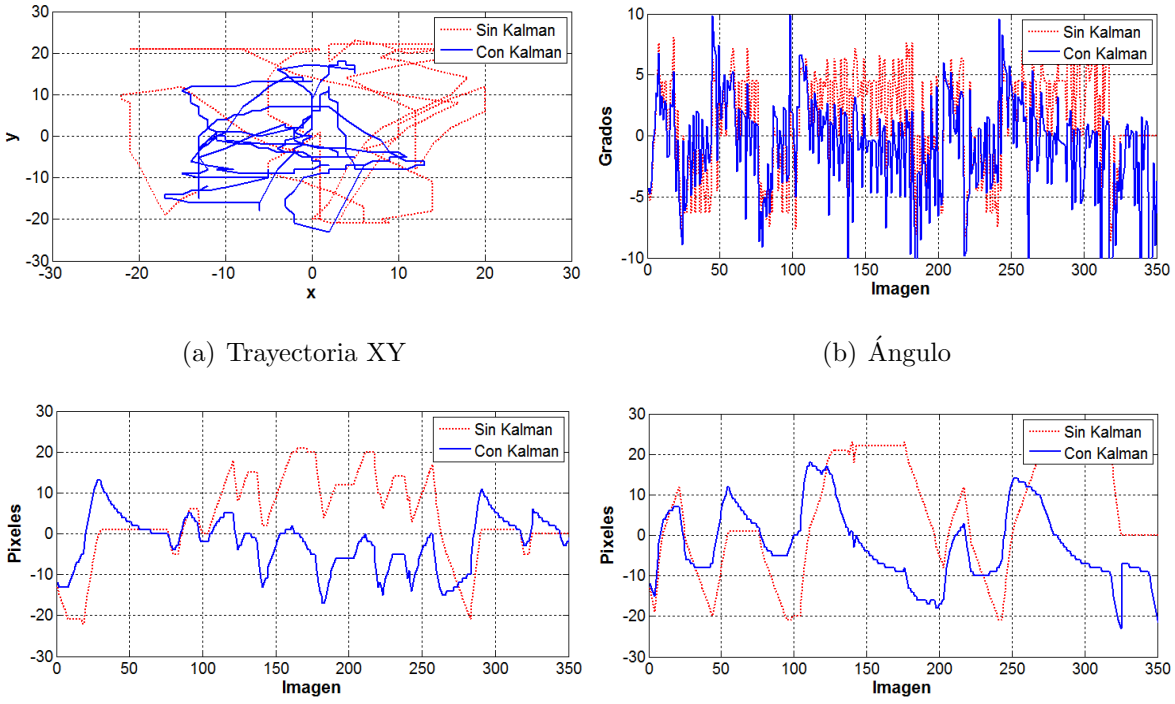
(d) Trayectoria Y

Figura 5.5: Traslación con algortimo BP-GCM.

En la gráfica 5.5(a) se observa en línea roja la trayectoria de la imagen ante el primer algoritmo sin aplicar el filtro de Kalman y en línea azul la trayectoria ante el

primer algoritmo una vez que ya se aplicó el filtro de Kalman. De las figuras 5.5(b), 5.5(c) y 5.5(d) podemos deducir que al aplicar el filtro de Kalman observamos no solo un suavizado en el movimiento, sino también una mejora en la predicción y con ello la corrección del movimiento.

A continuación se muestran los resultados del algoritmo ante un vídeo elaborado artificialmente en las dos direcciones de rotación con una magnitud de -5 a +5 grados. Donde la rotación positiva es en sentido contrario a las manecillas del reloj y la rotación negativa va en sentido de las manecillas del reloj.



(a) Trayectoria XY

(b) Ángulo

(c) Trayectoria X

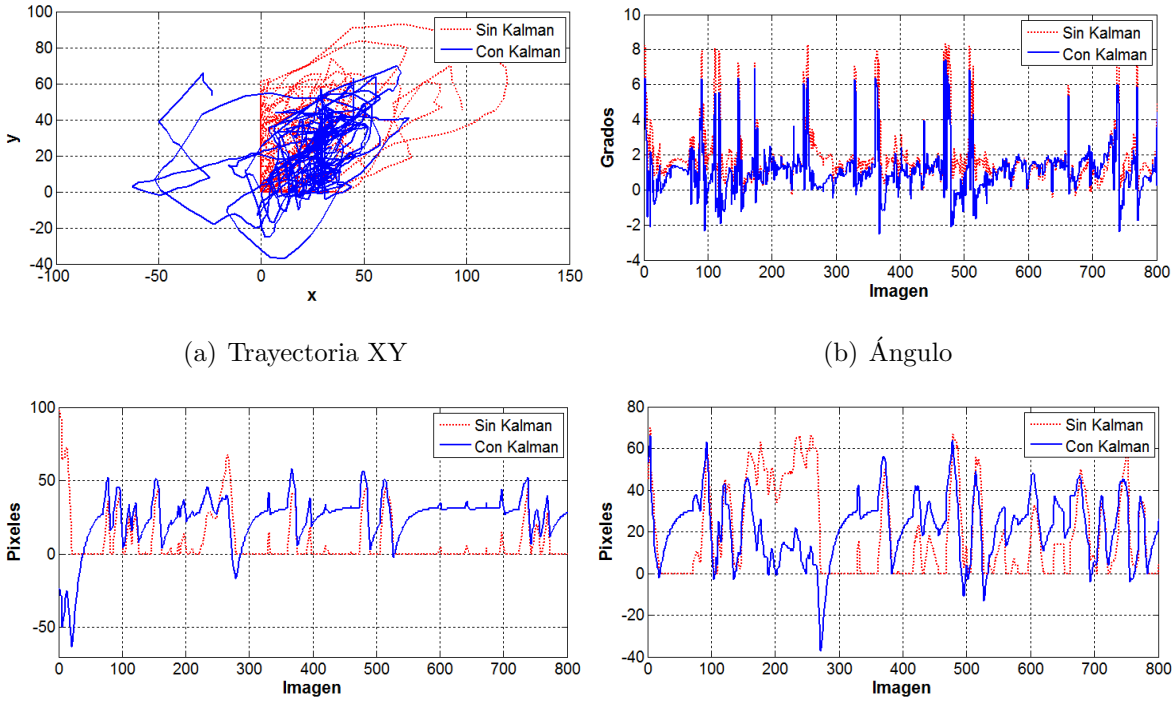
(d) Trayectoria Y

Figura 5.6: Rotación con algortimo BP-GCM.

Ahora, en la figura 5.6(a) observamos la trayectoria calculada por el primer algoritmo con y sin filtro de Kalman. En la figura 5.6(b) se observa la comparación del

movimiento calculado por el algoritmo. En línea roja está el primer algoritmo de planos de bits sin filtro de Kalman y en línea azul el mismo algoritmo pero con filtro de Kalman. Nuevamente en las figuras 5.6(c) y 5.6(d) observamos que la diferencia es mínima entre las medidas con Kalman y sin Kalman. La última prueba es con un vídeo de un vuelo en tiempo real en un UAV.

La figura 5.7 muestra los resultados en las mediciones que se obtuvieron con el primer algoritmo basado en la correlación de planos de bits ante un vuelo en tiempo real. En línea roja observamos el algoritmo sin aplicar Kalman y en línea azul algoritmo una vez que se le aplicó el filtro de Kalman.



(a) Trayectoria XY

(b) Ángulo

(c) Trayectoria X

(d) Trayectoria Y

Figura 5.7: Vuelo en tiempo real con algortimo BP-GCM.

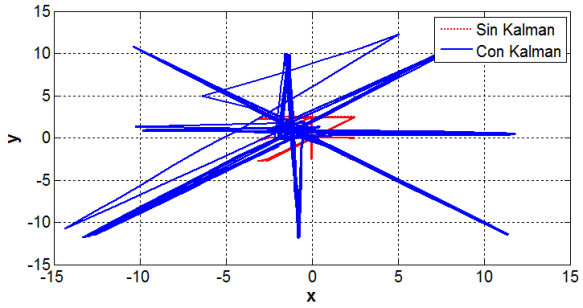
Finalmente en las gráficas 5.7(b), 5.7(c) y 5.7(d) observamos que las líneas en rojo se encuentran delimitadas. Lo anterior es debido a la ventana de búsqueda que se definió.

Ya que dependiendo del tamaño de la ventana, ésta se encuentra delimitada por el número de píxeles que esperamos que se mueva la imagen. Sin embargo la limitación observada en las líneas rojas, la resolvemos mediante el uso del filtro lineal de Kalman. El resultado de aplicarle el filtro lineal de Kalman se observa en las líneas azules. Por lo que podemos concluir que en las tres pruebas realizadas, se observó que el primer algoritmo obtiene medidas muy aproximadas a las reales y en tiempo real.

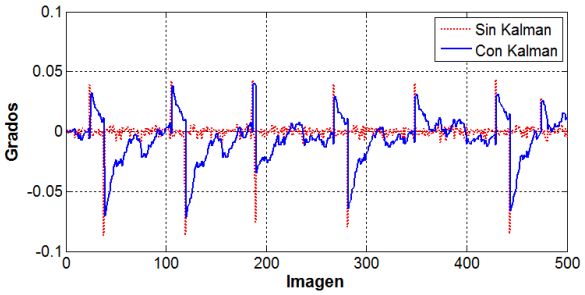
Por medidas reales nos referimos a que consideramos que en la prueba de traslación dichas medidas deben de oscilar entre ± 15 en ambos ejes x y y. Y en la segunda prueba los registros en la gráfica del ángulo deben de oscilar entre $\pm 5°$. Es muy importante el hecho de que los resultados son inmediatos, ya que uno de los principales objetivos de esta tesis es lograr la estabilización de los vídeos obtenidos en vuelos de UAVs en tiempo real. Se reitera que para lograr que dichos resultados se obtengan en tiempo real deben tener una baja complejidad computacional.

5.3 Resultados segundo algoritmo de estabilización.

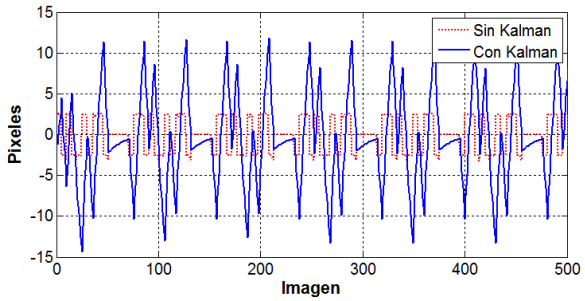
Para poner a prueba el segundo algoritmo de estabilización de nuevo hacemos uso de los dos vídeos modificados elaborados artificialmente tanto en traslación como en rotación. Y por último utilizaremos el segundo algoritmo con un vídeo en tiempo real de un vuelo en un UAV. Es importante mencionar que al referirnos a las medidas reales en la primera y segunda prueba, nos referimos a que las medidas en la prueba uno deben de oscilar entre ± 15 píxeles en ambos ejes x y y. Y en la segunda prueba las oscilaciones deben de ser entre $\pm 5°$ en la gráfica que muestra los ángulos. Estas medidas las conocemos, ya que son los movimientos que simulamos en los vídeos que elaboramos artificialmente. Las gráficas que se obtuvieron como resultados son las siguientes:



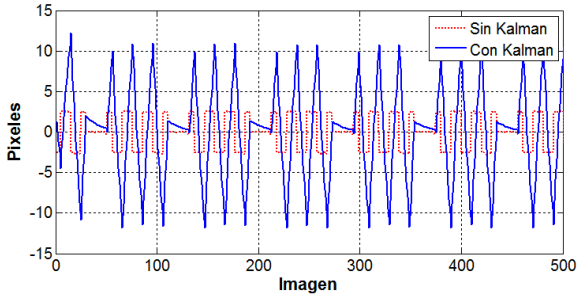
(a) Trayectoria XY



(b) Ángulo



(c) Trayectoria X



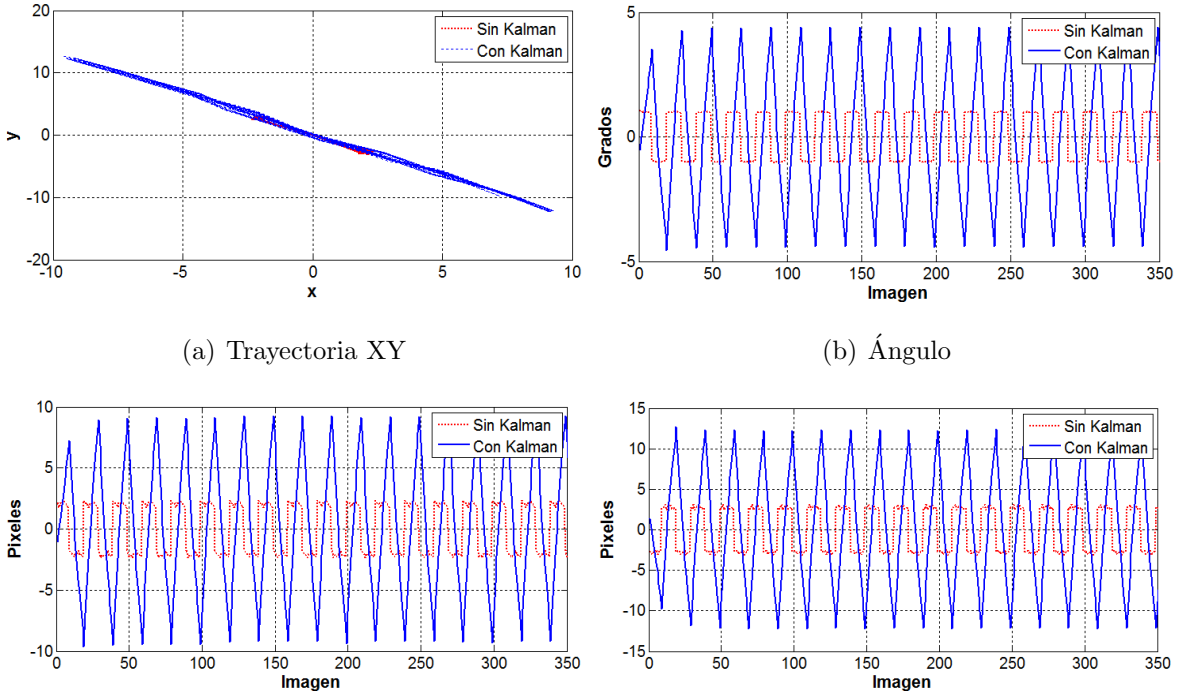
(d) Trayectoria Y

Figura 5.8: Traslación con algoritmo OpenCV.

En la figuras 5.8(c) y 5.8(d) observamos que las medias obtenidas por el segundo algoritmo de estabilización, el cual está basado en las funciones de OpenCV, son aproximadas a los datos reales. En línea roja observamos el algoritmo basado en las funciones de OpenCV sin aplicarle el filtro de Kalman y en línea azul tenemos el comportamiento de dicho algoritmo una vez que se le implemento el filtro de Kalman.

En las figuras 5.8(a) y 5.8(b) observamos la trayectoria total y los ángulos calculados respectivamente. En dichas gráficas también podemos observar que las medidas obtenidas sin el filtro de Kalman son más alejadas a las medidas reales, por lo que los resultados de la línea azul son más aproximados a los datos reales.

Las siguientes pruebas son la del vídeo de rotación y la del vídeo en vuelo real, donde los resultados son los siguientes:



(a) Trayectoria XY

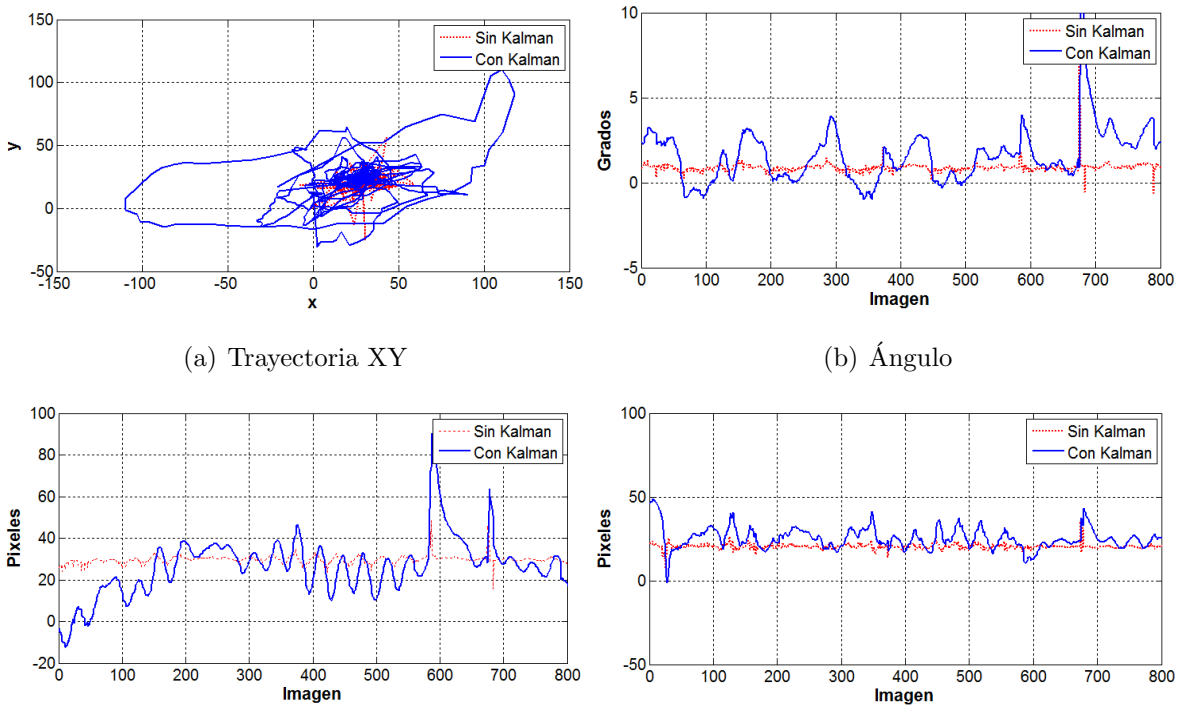
(b) Ángulo

(c) Trayectoria X

(d) Trayectoria Y

Figura 5.9: Rotación con algoritmo OpenCV.

Aunque en la figura 5.9(b) se observa que el algoritmo de estabilización del vídeo basado en las funciones de OpenCV tiene mediciones más aproximadas a los datos reales que las obtenidas con el algoritmo de estabilización de vídeo basado en la correlación de bloques de planos de bits en código Gray. Uno de los problemas que nos encontramos con el segundo algoritmo de estabilización es la complejidad computacional que presenta éste algoritmo, es mayor que la que presenta el primer algoritmo. Cabe mencionar que se llegó a la deducción anterior, ya que al correr el segundo programa se puede observar que se tiene un retraso de aproximadamente de medio segundo. Además en las figuras 5.9(a), 5.9(c) y 5.9(d) observamos los resultados obtenidos en la trayectoria XY,



(a) Trayectoria XY

(b) Ángulo

(c) Trayectoria X

(d) Trayectoria Y

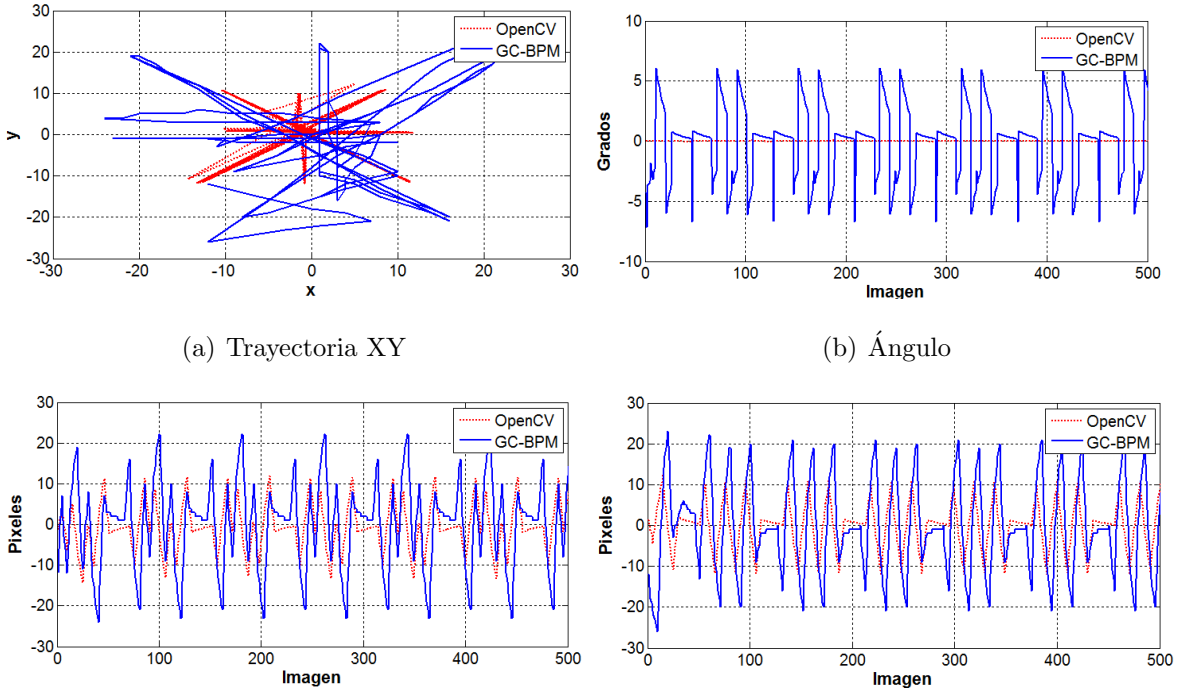
Figura 5.10: Vuelo en tiempo real con algoritmo OpenCV.

trayectoria X y trayectoria Y respectivamente. Finalmente la tercer prueba es la del vuelo en tiempo real. En las figuras 5.10(a), 5.10(b), 5.10(c) y 5.10(d) observamos los resultados de la trayectoria XY, los ángulos calculados, la trayectoria X y la trayectoria Y respectivamente. En dichas gráficas podemos constatar una vez más que los algoritmos mejoran significativamente tanto la predicción como la corrección del movimiento no deseado en los vídeos cuando a los algoritmos de estabilización de vídeo se les aplica un filtro de Kalman lineal.

Por ejemplo las medidas observadas en las gráficas 5.10(a), 5.10(b), 5.10(c) y 5.10(d) en línea roja tenemos los resultados del segundo algoritmo de estabilización sin aplicarle el filtro de Kalman y en línea azul los resultados de dicho algoritmo pero en este caso se le implementó el filtro de Kalman.

5.4 Primer algoritmo vs segundo algoritmo.

Ya hemos corroborado que ambos algoritmos obtienen una mayor aproximación en la estimación y corrección de los movimiento no deseados en los vídeos. En las siguientes gráficas podemos observar la comparación entre el primer algoritmo basado en la correlación de bloques contra el segundo algoritmo de estabilización, el cual está basado en las funciones de OpenCV.



(a) Trayectoria XY

(b) Ángulo

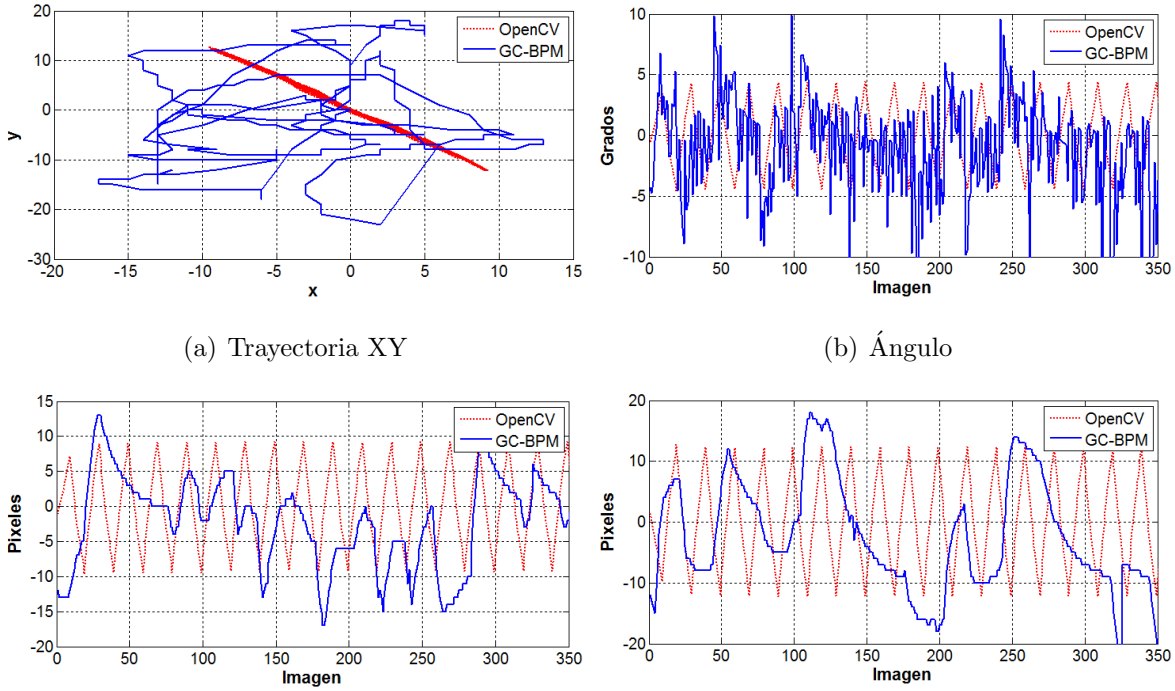
(c) Trayectoria X

(d) Trayectoria Y

Figura 5.11: GC-BPM vs OpenCV traslación.

En las gráficas 5.11(a), 5.11(b), 5.11(c) y 5.11(d) observamos la comparación de las trayectorias XY, de los cálculos de los ángulos, de las trayectorias X y de las trayectorias Y respectivamente. En dichas gráficas podemos observar que la diferencia en las medidas obtenidas no es grande. Sin embargo, si hablamos de la complejidad computacional, el algoritmo a base de las funciones de OpenCV presenta un retraso de

aproximadamente de medio a un segundo entre imagen a diferencia del algoritmo a base de la correlación de planos de bits.



(a) Trayectoria XY

(b) Ángulo

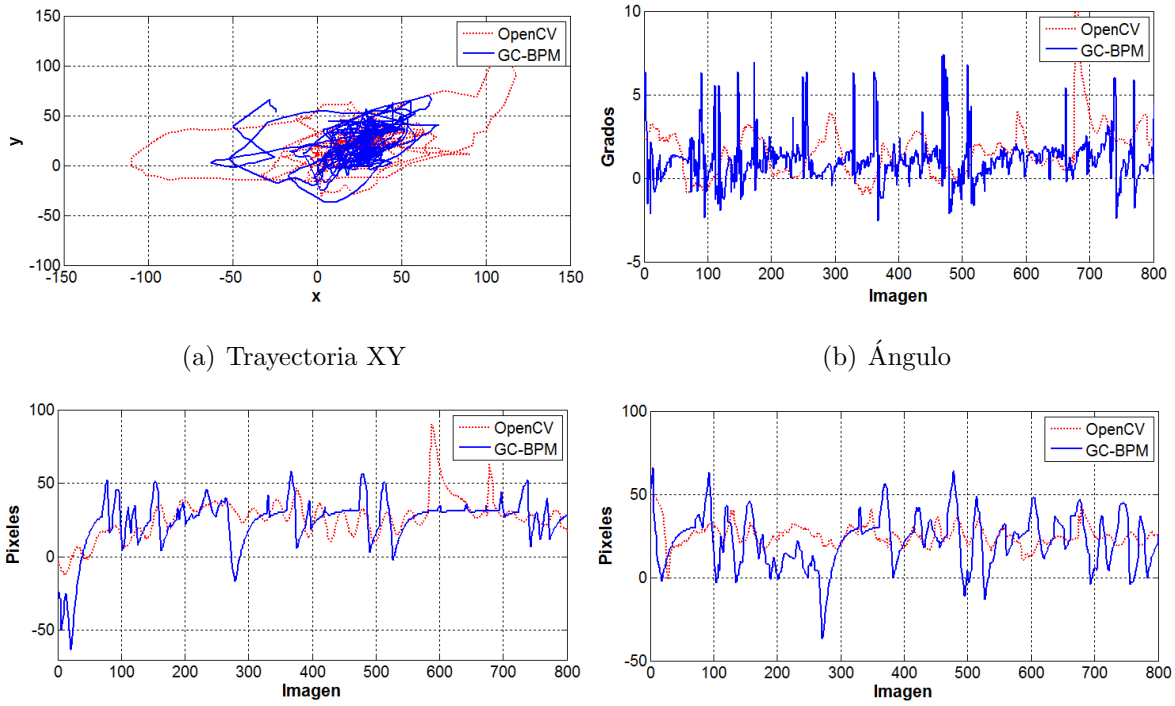
(c) Trayectoria X

(d) Trayectoria Y

Figura 5.12: GC-BPM vs OpenCV rotación.

Las figuras 5.12(a), 5.12(b), 5.12(c) y 5.12(d) nos muestran la comparación de las trayectorias XY, de los ángulos, de las trayectorias X y de las trayectorias Y respectivamente. Estas figuras son muy importantes ya que el segundo algoritmo está basado en la propuesta descrita en [8], donde claramente nos menciona que el algoritmo de estabilización por correlación de los planos de bits no está diseñado para contrarrestar las perturbaciones rotacionales presentes en los vídeos, sin embargo en esta tesis implementamos una forma de calcular los movimientos rotacionales. En la gráficas 5.12(b) observamos que las mediciones que se obtuvieron del algoritmo de planos de bits son cercanas a las que se obtuvieron con el algoritmo de las funciones de OpenCV. Éste

último algoritmo si esta diseñado para contrarrestar las perturbaciones rotacionales.



(a) Trayectoria XY

(b) Ángulo

(c) Trayectoria X

(d) Trayectoria Y

Figura 5.13: GC-BPM vs OpenCV vuelo en tiempo real.

Finalmente en y 5.13(a), 5.13(b), 5.13(c) y 5.13(d) observamos la comparación entre los calculos obtenidos entre el primero y el segundo algoritmo de estabilización ante un vídeo de un vuelo en tiempo real. En dichas figuras también, podemos observar que si tomamos como referencia al segundo algoritmo, el cual esta basado en las funciones de OpenCV. El primer algoritmo se acerca bastante al segundo algoritmo. Por lo que como conclusión final, podemos decir que los cálculos obtenidos del primer algoritmo no son tan precisos como los del segundo algoritmo. Sin embargo con ambos algoritmos se obtiene una estabilización aceptable. Sin dejar de mencionar que el segundo algoritmo presentó un retraso de aproximadamente medio segundo.

5.5 Conclusiones.

De este trabajo de tesis podemos concluir que el primer algoritmo de estabilización basado en la correlación de bloques en planos de bits en código Gray logra una estabilización considerable y lo más importante es que dicha estabilización se logra en tiempo real. Cabe mencionar que dicho algoritmo tiene base en el trabajo del profesor S. Ko descrito en [8], donde claramente nos indica que este algoritmo no está diseñado para corregir las perturbaciones rotacionales. En esta tesis se propuso una manera de calcular dichos movimientos rotacionales, además de proponer una manera más eficiente para calcular el movimiento total en cada imagen del vídeo que se busca estabilizar. Por lo tanto con esta tesis ahora se cuenta con un algoritmo eficaz que estabiliza tanto los movimientos rotacionales como los movimientos traslacionales.

El segundo algoritmo de estabilización de vídeo el cual está basado en las funciones de OpenCV presenta una mayor estabilización, sin embargo este algoritmo tiene una mayor complejidad computacional, por lo que si búscamos embeber este algoritmo vamos a tener algunos retrasos al visualizar el vídeo estabilizado, por lo que ya no se estaría cubriendo uno de los principales objetivos de esta tesis. Además debemos mencionar que no todos los ordenadores de placa reducida cuentan con desarrollo en OpenCV. Por lo que nos enfrentamos a otro problema, encontrar un ordenador de placa reducida que cuente con desarrollo en OpenCV.

Finalmente, podemos concluir que aunque ambos algoritmos cumplieron los objetivos de la presente tesis, si pensamos en los objetivos a futuro, el primer algoritmo el cual está basado en la correlación de bloques de planos de bits en código Gray es el que mas nos conviene. Primero este algoritmo no requiere de que el ordenador de placa reducida donde se pretende embarcar tenga desarrollo basado en OpenCV y su costo computacional es muy bajo en comparación al algoritmo que usa las funciones de OpenCV.

5.6 Trabajo futuro.

Como trabajo futuro se proponen dos principales ideas, las cuales cada una presenta una gran complejidad. Estas propuestas son las siguientes:

- ***Embeber el algoritmo:*** Esta tarea presenta una complejidad ya que se debe de lograr embeber el primer algoritmo en un ordenador de placa reducida que sea capaz de procesar imágenes a una velocidad de 30 imágenes por segundo.

- ***Desarrollo de otros algoritmos:*** El algoritmo de estabilización de vídeo a base de la correlación de planos de bits, puede ser la base para la elaboración de algún algoritmo de búsqueda y seguimiento de objetivos. Debemos de tomar en cuenta que cualquier algoritmo que se busque programar no debe de tener gran complejidad computacional, ya que como se mencionó en el primer punto, estos algoritmos deben de ser embebidos en un ordenador de placa reducida.

A pesar de que solo se proponen dos puntos como trabajo a futuro, también debemos de tener en mente que no solo estas propuestas son las que se pueden realizar. Además de ello, podemos pensar en que los algoritmos sean capases de actuar con cámaras infrarrojas, con el fin de utilizarlas en alguna tarea nocturna. Posteriormente podemos desarrollar algún algoritmo de reconocimiento facial con el cual le daremos mas autonomía al UAV. Recordemos que uno de los objetivos de los UAV es realizar tareas que pueden ser peligrosas para los seres humanos y con esto podremos lograr que el UAV pueda tomar decisiones por si solo y que no requiera necesariamente da la intervención humana.

ANEXOS A

TEORÍA DE MODELADO

Convención de Denavit-Hartenberg (D-H)

Este método [24] [25] permite establecer un sistema de coordenadas ligado a cada eslabón i de una cadena articulada, y con ello determinar a continuación las ecuaciones cinemáticas de la cadena completa. Según la representación de D-H, escogiendo adecuadamente los sistemas de coordenadas asociados para cada eslabón, será posible pasar de uno al siguiente mediante 4 transformaciones básicas que dependen exclusivamente de las características geométricas del eslabón.

Estas transformaciones básicas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento i con el sistema del elemento *i-1*. Las transformaciones en cuestión son las siguientes:

- Rotación alrededor del eje Z_{i-1} un ángulo θ_i.
- Translación a lo largo de Z_{i-1} una distancia d_i.
- Translación a lo largo de X_i, una distancia a_i .
- Rotación alrededor del eje X_i, aun ángulo α_i.

Como ya sabemos el producto de las matrices no es conmutativo, por lo que las transformaciones se han de realizar en el orden indicado. Dónde A_i es el producto de las 4 transformaciones básicas:

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i}$$

$$A_i = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i} & 0 & 0 \\ S_{\theta_i} & C_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C_{\alpha_i} & -S_{\alpha_i} & 0 \\ 0 & S_{\alpha_i} & C_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_i = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i} C_{\alpha_i} & S_{\theta_i} S_{\alpha_i} & a_i C_{\theta_i} \\ S_{\theta_i} & C_{\theta_i} C_{\alpha_i} & -C_{\theta_i} S_{\alpha_i} & a_i S_{\theta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

Los parámetros de Denavit Hartenberg son como se muestran en la (figura A.1). Estos cuatro parámetros $(\theta_i, d_i, a_i, \alpha_i)$ dependen únicamente de las características geométricas de cada eslabón y de las articulaciones que le unen con el anterior y con el siguiente.

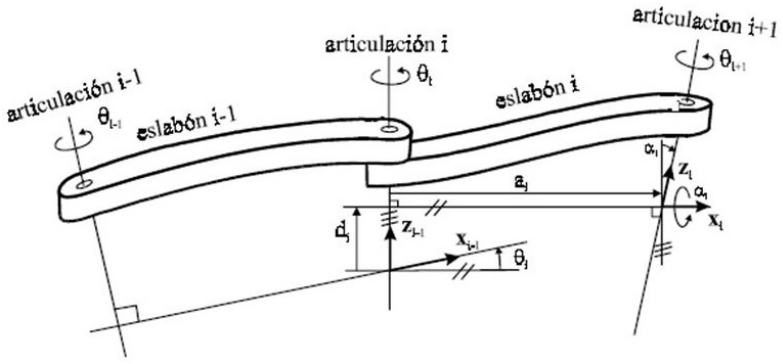


Figura A.1: Parámetros D-H.

De la figura A.1, podemos observar que,

- θ_i Es el ángulo que forman los ejes X_{i-1} y X_i medido en un plano perpendicular al eje Z_{i-1}, utilizando la regla de la mano derecha. Se trata de un parámetro variable en articulaciones giratorias.

- d_i Es la distancia a lo largo del eje Z_{i-1} desde el origen del sistema de coordenadas $(i-1)-esimo$ hasta la intersección del eje Z_{i-1} con el eje X_i. Se trata de un parámetro variable en articulaciones prismáticas.

- a_i Es a la distancia a lo largo del eje X_i que va desde la intersección del eje Z_{i-1} con el eje X_i hasta el origen del sistema $i-esimo$, en el caso de articulaciones giratorias. En el caso de articulaciones prismáticas, se calcula como la distancia más corta entre los ejes Z_{i-1} y Z_i.

- α_i Es el ángulo de separación del eje Z_{i-1} y el eje Z_i, medido en un plano perpendicular al eje X_i, utilizando la regla de la mano derecha.

El algoritmo de Denavit-Hartenberg se resume en los siguientes 8 pasos:

1. Localiza y etiqueta los ejes de articulación de $Z_0, ..., Z_{n-1}$.

2. Establezca la base. Selecciona el origen en cualquier lugar en el eje z_0. Los ejes X_0 y Y_0 se seleccionan convenientemente para formar un marco con la regla de la mano derecha.

3. Localiza el origen O_i donde normalmente Z_i y Z_{i-1} se intersectan. Si Z_i intersecta Z_{i-1} localiza O_i en esta intersección. Si Z_i y Z_{i-1} son paralelos localiza O_i en la articulación i.

4. Establece X_i a lo largo de Z_{i-1} y Z_i a través de O_i, o en la dirección normal al plano $Z_{i-1} - Z_i$, si Z_{i-1} y Z_i se intersectan.

5. Establece Y_i para completar el marco de la regla de la mano derecha.

6. Establece el marco del efector final.

7. Elabora una tabla con los parámetros $\alpha_i, a_i, \theta_i, d_i$ donde:

   - α_i = es al ángulo entre Z_{i-1} y Z_i medido sobre X_i, positivo en dirección contraria a las manecillas del reloj.
   - a_i = distancia a lo largo de X_i hasta O_i hasta la intersección de los ejes X_i y Z_{i-1}.
   - θ_i = ángulo entre X_{i-1} y X_i medido sobre Z_{i-1}, positivo en dirección contraria a las manecillas del reloj. θ_i es variable si la articulación es de revolución.
   - d_i = distancia a lo largo de Z_{i-1} hasta O_{i-1} hacia la intersección de los ejes X_i y Z_{i-1}. d_i es variable si la articulación es prismática.

8. Finalmente se forman las matices de transformación homogénea A_i y T_0^n

Cinemática Diferencial

El objetivo de la cinemática diferencial como su nombre lo indica, consiste en expresar el modelo diferencial de un robot manipulador a través de la matriz Jacobiana. Como ya sabemos, la cinemática directa e inversa establece la relación entre las variables de las articulaciones y el efector final, por lo que la cinemática diferencial otorga la relación entre las velocidades, lineal y angular (rotacional) del efector final.

Jacobiano Geométrico

Expresaremos la velocidad lineal $\dot{p}$ y la velocidad angular ω del efector final como una función de las velocidades $\dot{q}$ de cada uno de las articulaciones. Por lo que:

$$\dot{p} = J_v(q)\dot{q} \tag{A.2}$$

$$\omega = J_\omega(q)\dot{q} \tag{A.3}$$

Y sabemos que por definición $v = \begin{bmatrix} \dot{p} & \omega \end{bmatrix}^T$ por lo que las ecuaciones A.2 y A.3 pasan a la forma compacta siguiente:

$$v = J(q)\dot{q} \tag{A.4}$$

dónde la matriz J es el Jacobiano geométrico (o Jacobiano reducido), por lo que:

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \tag{A.5}$$

Velocidad Angular

Consideramos la velocidad angular ω_{i-1}^i del eslabón i en el marco del eslabón $i-1$. Si la articulación i es prismática entonces:

$$\omega_{i-1}^i = 0 \tag{A.6}$$

pero si la articulación i es de revolución entonces:

$$\omega_{i-1}^i = \dot{q}_i r_z \tag{A.7}$$

donde $r_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ es el eje rotacional. Ahora podemos expresar A.7 en el marco base de la siguiente manera:

$$\omega_{i-1,0}^i = \dot{q}_i R_0^{i-1} r_z \tag{A.8}$$

Considerando que las velocidades angulares pueden agregarse si son expresadas en el mismo marco de coordenadas. Por lo tanto, podemos expresar la velocidad angular en términos generales como:

$$\omega \equiv \omega_0^n = \sum_{i=1}^{n} \rho_i z_{i-1} \dot{q}_i \tag{A.9}$$

donde

$$z_i = R_0^i r_z \tag{A.10}$$

y $\rho_i = 0$ si la articulación i es prismática y $\rho_i = 1$ si la articulación i es de revolución. Por lo tanto J_ω está dada de la siguiente manera:

$$J_\omega = \begin{bmatrix} \rho_0 z_0 & \dots & \rho_n z_{n-1} \end{bmatrix} \tag{A.11}$$

Velocidad Linear

Siendo que $\dot{d}_0^n$ denota la velocidad lineal del efector final. Entonces por la regla de la cadena tenemos que:

$$\dot{d}_0^n = \sum_{i=1}^{n} \frac{\partial d_0^n}{\partial q_i} \dot{q}_i \tag{A.12}$$

y podemos ver que la $i - ésima$ columna de J_v es:

$$J_{vi} = \frac{\partial d_0^n}{\partial q_i} \tag{A.13}$$

Por lo que podemos obtener J_{vi} a partir de A.12, si $\dot{q}_i$ es igual a la unidad y todas las velocidades de cada articulación son iguales a cero. Si la articulación i es prismática tenemos que:

$$d_{i-1}^i = d_i r_z + R_{i-1}^i \alpha_i r_x \tag{A.14}$$

dónde $r_x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$. Fijamos todas las articulaciones excepto la i-ésima y diferenciar d_0^n.

$$\dot{d}_0^n = R_0^{i-1}\dot{d}_i^{i-1} = \dot{d}_i z_{i-1} \tag{A.15}$$

Por lo que:

$$J_{vi} = z_{i-1} \tag{A.16}$$

para una articulación prismática. Pero si nuestra articulación es de revolución tenemos que:

$$d_0^n = d_0^{i-1} + R_{i-1}^0 d_{i-1}^n \tag{A.17}$$

y

$$\dot{d}_0^n = R_{i-1}^0 \dot{d}_{i-1}^n \tag{A.18}$$

si todas las articulaciones son fijas excepto i. El movimiento del eslabón i puede expresarse de la siguiente manera:

$$\dot{d}_{i-1}^n = \dot{q} z_0 \times d_{i-1}^n \tag{A.19}$$

y por lo tanto:

$$\dot{d}_0^n = \dot{q} z_{i-1} \times (d_0^n - d_0^{i-1}) \tag{A.20}$$

Ahora venos que la i-ésima columna de J_ω puede expresarse como:

$$J_{\omega i} = z_{i-1} \times (d_0^n - d_0^{i-1}) \tag{A.21}$$

Jacobiano geometrico para un manipulador de n-eslabones

Partiendo de los resultados obtenidos anteriormente, ahora podemos expresar el Jacobiano geométrico para un manipulador de n-eslabones como:

$$J = \begin{bmatrix} J_1 & \dots & J_n \end{bmatrix} \tag{A.22}$$

donde:

$$J_i = \begin{bmatrix} z_{i-1} \times (d_0^n - d_0^{i-1}) \\ z_{i-1} \end{bmatrix} \tag{A.23}$$

si la articulación i es de revolución, y

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} \tag{A.24}$$

si la articulación i es prismática.

Notemos que todas las variables en las ecuaciones A.23 Y A.24 se pueden encontrar en la matriz de transformación homogénea.

- z_{i-1} está dada por los tres primeros elementos de la tercera columna de T_0^{i-1}.
- d_0^n está dada por los tres primeros elementos de la cuarta columna de T_0^n.
- d_0^{i-1} está dada por los tres primeros elementos de la cuarta columna de T_0^{i-1}.

Dinámica.

El modelo dinámico es la descripción de la relación del torque de la articulación y el movimiento de la estructura. Este es muy útil para simular el movimiento, analizar la estructura y diseñar estrategias de control.

En esta tesis se usa el método basado en la formulación de Lagrange para derivar la ecuación de movimiento para un manipulador general.

Ecuaciones de Lagrange

Un conjunto de coordenadas generalizadas son elegidas, las cuales describen eficazmente la posición del eslabón en un sistema mecánico. Entonces el Lagrangiano puede ser descrito como una función de las coordenadas generalizadas como:

$$L = K - V \tag{A.25}$$

donde K y V son las energías cinética y potencial respectivamente. Mediante el uso de las ecuaciones de movimiento de Lagrange podemos ahora derivarlas de manera sistemática, por lo que las ecuaciones de Lagrange son expresadas de la siguiente manera:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\lambda}_i} - \frac{\partial L}{\partial \lambda_i} = F_i \qquad\qquad i = 1, 2, ..., n \tag{A.26}$$

donde F_i es la fuerza generalizada asociada a las coordenadas generalizadas λ_i. Para un robot manipulador una selección obvia de las coordenadas generalizadas son las variables de las articulaciones, p. ej. $\lambda_i = d_i$ para una articulación prismática y $\lambda_i = \theta_i$ para las articulaciones de revolución. Las fuerzas generalizadas están dadas por las fuerzas no conservativas, p. ej. los torques y fricciones en las articulaciones.

Ecuaciones de movimiento de un robot manipulador

El Lagrangiano es la diferencia entre la energía cinética total y la energía potencial, por lo que derivaremos las expresiones para las energías cinética y potencial de un robot manipulador.

Cálculo de la energía cinética

La definición de la energía cinética es:

$$K = \frac{1}{2}\int_B v^T(x,y,z)v(x,y,z)dm \tag{A.27}$$

La velocidad de un punto en el objeto puede expresarse de la siguiente manera:

$$v = v_c + \omega \times r \tag{A.28}$$

Haciendo uso de los Anexos C, C y C podemos expresar a A.28 como:

$$v = v_c + S(\omega)r \tag{A.29}$$

donde $S(\omega)$ es la matriz antisimétrica. Entonces la energía cinética puede ser expandida en cuatro términos K_i, $i = 1, 2, 3, 4$ como se muestra a continuación:

$$\begin{aligned} k &= \frac{1}{2}\int_B [v_c + S(\omega)r]^T[v_c + S(\omega)r]dm \\ &= \frac{1}{2}\int_B v_c^T v_c dm + \frac{1}{2}\int_B v_c^T S(\omega)r dm \\ &\quad + \frac{1}{2}\int_B [S(\omega)r]^T v_c dm + \frac{1}{2}\int_B [S(\omega)r]^T S(\omega)r dm \end{aligned} \tag{A.30}$$

Ya que v_c es independiente de la variable de integración, el primer término es :

$$K_1 = \frac{1}{2}\int_B v_v^T v_c dm = \frac{1}{2}m v_c^T v_c \tag{A.31}$$

Ésta es solo la energía cinética de una partícula de masa m con velocidad v_c y es la parte traslacional de la energía cinética. El segundo y tercer término ambos son cero porque el centro de masa se encuentra en el origen. Por lo tanto:

$$K_2 = \frac{1}{2}\int_B v_c^T S(\omega) r dm = \frac{1}{2} v_c^T S(\omega) \int_B r dm = 0 \tag{A.32}$$

$$K_3 = \frac{1}{2}\int_B [S(\omega)r]^T v_c dm = \frac{1}{2}\int_B r^T dm S^T(\omega) v_c = 0 \tag{A.33}$$

Haciendo uso del Anexo C el último término puede expresarse como:

$$K_4 = \frac{1}{2}\int_B [S(\omega)r]^T S(\omega) r dm = \frac{1}{2}\int_B tr[S(\omega) r r^T S^T(\omega)] dm \tag{A.34}$$

$$= \frac{1}{2} tr[S(\omega)\int_B r r^T dm S^T(\omega)] = \frac{1}{2}\omega^T I \omega \tag{A.35}$$

donde I es el tensor de inercia y está definido por:

$$I = \begin{bmatrix} \int y^2 + z^2 dm & -\int xy dm & -\int xz dm \\ -\int xy dm & \int x^2 + z^2 dm & -\int yz dm \\ -\int xz dm & \int yz dm & \int x^2 + y^2 dm \end{bmatrix} \tag{A.36}$$

El cuarto término es conocido como la parte rotacional de la energía cinética. Por lo que la energía cinética para un objeto rígido se encuentra dado por:

$$K = \frac{1}{2} m v_c^T v_c + \frac{1}{2}\omega^T I \omega \tag{A.37}$$

Podemos observar que la ecuación A.37 depende del sistema de coordenadas calculadas de v_c, ω e I. $v_c^T v_c$ es el cuadrado de la longitud y la longitud es la misma independientemente del marco de referencia. El primer término es el mismo independientemente del marco inercial donde se exprese v_c. De la misma manera es el mismo término en todos los marcos de referencia. Se busca expresar este término en el marco inercial asociado al objeto, ya que la matriz de inercia será constante. Entonces si ω_0 es la velocidad angular del objeto medida en el marco inercial, entonces $R^T\omega_0$ es la velocidad angular del objeto medido en el marco del objeto.

Ahora consideramos a un robot manipulador de n-eslabones, tenemos las velocidades linear y angular de cualquier punto en cualquier eslabón como:

$$v_i = J_{vi}(q)\dot{q} \tag{A.38}$$

$$\omega_i = R_i^T(q)J_{\omega i}(q)\dot{q} \tag{A.39}$$

Expresando la energía cinética de un robot manipulador de n-eslabones en términos de la matriz Jacobiana y de las coordenadas generalizadas como:

$$K = \frac{1}{2}\dot{q}^T \sum_{i=1}^{n} [m_i J_{vi}(q)^T J_{vi}(q) + J_{\omega i}(q)^T R_i(q) I_i R_i(q)^T J_{\omega i}(q)]\dot{q} \tag{A.40}$$

La energía cinética tiene la siguiente forma:

$$K = \frac{1}{2}\dot{q}^T D(q)\dot{q} \tag{A.41}$$

donde $D(q)$ es la matriz de inercia y es una matriz simétrica definida positiva.

Cálculo de la energía potencial

La energía potencial está dada por la suma:

$$V = \sum_{i=1}^{n} V_i \tag{A.42}$$

donde V_i es la energía potencial del eslabón i. Si todos los eslabones son rígidos, entonces la energía potencial es causada solamente por la gravedad como:

$$V_i = \int_{Bi} g^T r_i dm = g^T \int_{Bi} r_i dm = g^T r_{ci} m_i \tag{A.43}$$

donde g es el vector de aceleración de la gravedad.

Cálculo de la ecuación de Lagrange

El Lagrangiano de un robot manipulador de n-eslabones puede escribirse de la siguiente manera:

$$L = K - V = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}(q)\dot{q}_i\dot{q}_j - V(q) \tag{A.44}$$

Dividimos la ecuación A.26 en pequeñas partes, y la primer parte que tenemos es:

$$\frac{\partial L}{\partial \dot{q}_i} = \sum_{j} d_{ij}(q)\dot{q}_j \tag{A.45}$$

y después:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} = \sum_{j} d_{ij}(q)\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n}\frac{\partial d_{ij}}{\partial q_k}\dot{q}_k\dot{q}_j \tag{A.46}$$

Y también:

$$\frac{\partial L}{\partial q_i} = \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n}\frac{\partial d_{jk}}{\partial q_i}\dot{q}_k\dot{q}_j - \frac{\partial V}{\partial q_i} \tag{A.47}$$

Las ecuaciones de Lagrange después de algunos cálculos, se pueden expresar de la siguiente manera:

$$\sum_{j=1}^{n} d_{ij}(q)\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}(q)\dot{q}_k\dot{q}_j + g_i(q) = \tau_i, \qquad i = 1, ..., n \tag{A.48}$$

donde c_{ijk} son los coeficientes de Christoffel y son definidos como:

$$c_{ijk} = \frac{1}{2}\left[\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{\partial d_{ij}}{\partial q_k}\right] \tag{A.49}$$

y g_i se encuentra definido como:

$$g_i(q) = \frac{\partial V}{\partial q_i} \tag{A.50}$$

Ahora escribiendo A.48 en forma de matriz tenemos:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{A.51}$$

donde los elementos de la matriz C se encuentran definidos como:

$$c_{ij} = \sum_{k=1}^{n} c_{ijk}(q)\dot{q}_k \tag{A.52}$$

Ecuaciones de movimiento

Finalmente las ecuaciones de movimiento de un robot manipulador de n-eslabones puede expresarse de la siguiente manera:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + g(q) = \tau \tag{A.53}$$

donde:

- $D(q)$ es la matriz de los momentos de inercia.
- $C(q,\dot{q})$ representa la matriz de coriolis.
- F_v representa la matriz de fricción.
- $g(q)$ es la matriz de gravedad.

Una interpretación física de las ecuaciones de movimiento muestran que:

* Los coeficientes d_{ii} representan el momento de inercia en la articulación i.
* El término $c_{ijj}\dot{q}_j^2$ representa el efecto centrífugo en la articulación i por la velocidad de la articulación j.
* El término $c_{ijk}\dot{q}_j\dot{q}_k$ representa el efecto de coriolis en la articulación i por la velocidad de las articulaciones j y k.

* El término g_i representa el momento generado en la articulación i causado por la gravedad.

* El término $\tau_{\alpha i}$ es el torque del motor i en el eslabón i.

ANEXOS B

MATRIZ DE TRANSFORMACIÓN

Considerando dos imágenes $f^t(x, y)$ y $f^{t-1}(x, y)$ descritas por una transformación. Obteniendo un punto correspondiente tentativamente $p_k^t \rightarrow p_k^{t-1}$ para $k = 0...N$, buscamos estimar la transformación T de manera que:

$$f^t(x, y) = f^{t-1}(T(x, y)) \tag{B.1}$$

La función T puede llegar a ser muy compleja, pero en esta ocasión nos vamos a limitar al caso más simple donde T es una transformación de coordenadas lineales.

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{B.2}$$

donde la matriz H de $3X3$ representa una proyección. La forma de la matriz de transformación H determina el tipo de transformación geométrica que representa, por ejemplo si solo queremos representar la rotación la matriz de transformación tiene la

siguiente forma:

$$H = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} R & 0 \\ \hline 0^T & 1 \end{array}\right] \tag{B.3}$$

donde θ representa el ángulo de rotación. Ahora si solo buscamos representar la translación la matriz de transformación ahora tendrá la siguiente forma:

$$H = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} R & t \\ \hline 0^T & 1 \end{array}\right] \tag{B.4}$$

Finalmente para representar tanto la rotación como la translación, nuestra matriz de transformación queda de la siguiente manera:

$$H = \begin{bmatrix} cos(\theta) & -sin(\theta) & t_x \\ sin(\theta) & cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} R & t \\ \hline 0^T & 1 \end{array}\right] \tag{B.5}$$

La matriz B.5 es la que vamos a utilizar en esta tesis.

ANEXOS C

CINEMÁTICA DE VELOCIDAD

Velocidad angular: Para el caso del eje fijo.

Cuando un cuerpo rígido presenta solo movimiento de rotación alrededor de un eje fijo, cada punto del cuerpo se mueve en forma circular. Los centros de estos círculos se encuentran sobre el eje de rotación. A medida que el cuerpo gira, de forma perpendicular desde cualquier punto del cuerpo al eje de barrido en el ángulo θ, y este ángulo es el mismo para cada punto en el cuerpo. Si k es un vector unidad en la dirección del eje de rotación entonces la velocidad angular está dada por:

$$\omega = \dot{\theta}k \tag{C.1}$$

en dónde $\dot{\theta}$ es la derivada con respecto al tiempo de θ. Dada la velocidad lineal en cualquier punto en el cuerpo como:

$$v = \omega \times r \tag{C.2}$$

en dónde r es un vector de la forma original(el cual se supone que se encuentra en el eje de rotación) al punto.

Matriz antisimétrica.

Se dice que una matriz S de nxn es antisimétrica si y solo si:

$$S^T + S = 0 \tag{C.3}$$

Denotamos un conjunto de $3x3$ matrices antisimétricas como so(3). Si $S \in so(3)$ que tiene componentes s_{ij}, $i, j = 1, 2, 3$ entonces la ecuación C.3 es equivalente a nueve ecuaciones:

$$s_{ij} + s_{ji} = 0 \qquad\qquad i, j = 1, 2, 3 \tag{C.4}$$

De la ecuación C.4 podemos observar que $s_{ii} = 0$; esto significa que los términos de la diagonal principal en S son cero y los términos fuera de la diagonal principal $s_{i,j}$, $i \neq j$ satisface que $s_{ij} = -s_{ji}$. En consecuencia S contiene solo tres términos independientes y cada matriz antisimétrica es de la forma:

$$S = \begin{bmatrix} 0 & -s_3 & s_2 \\ s_3 & 0 & -s_1 \\ -s_2 & s_1 & 0 \end{bmatrix} \tag{C.5}$$

Si $a = [a_x, a_y, a_z]^T$ es un vector, definimos la matriz antisimétrica como:

$$S(a) = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \tag{C.6}$$

Propiedades de la matriz antisimétrica

Las matrices antisimétricas poseen varias propiedades que son de utilidad en las derivaciones. Entre estas propiedades tenemos que:

1. El operador S es lineal, entonces:

$$S(\alpha a + \beta b) = \alpha S(a) + \beta S(b) \tag{C.7}$$

para los vectores a y b pertenecientes a $\mathbb{R}^3$ y los escalares α y β.

2. Para cualquier vector a y p que pertenecen a $\mathbb{R}^3$

$$S(a)p = a \times p \tag{C.8}$$

donde a y p denotan el producto cruz de los vectores. La ecuación C.8 puede ser verificada por cálculo directo.

3. Para $R \in SO(3)$ y $a \in \mathbb{R}^3$

$$RS(a)R^T = S(Ra) \tag{C.9}$$

Para demostrar esto, usamos el hecho de que si $R \in SO(3)$ y a y b son vectores en $\mathbb{R}^3$

$$R(a \times b) = Ra \times Rb \tag{C.10}$$

Esto se puede demostrar por cálculo directo. La ecuación C.10 no es cierta en general, a menos que R sea ortogonal. Esto quiere decir que si primero rotamos los vectores a y b utilizando la transformación de rotación R y después formamos el producto cruzado de los vectores de rotación Ra y Rb, el resultado es el mismo que el obtenido al formar primero el producto vectorial $a \times b$ y la rotación para obtener $R(a \times b)$. La ecuación C.9 ahora se obtiene fácilmente de las ecuaciones C.8 y C.10. Hacemos $b \in \mathbb{R}^3$ siendo un vector arbitrario. Entonces:

$$\begin{aligned} RS(a)R^T b &= R(a \times R^T b) \\ &= (Ra) \times (RR^T b) \\ &= (Ra) \times b \\ &= S(Ra)b \end{aligned}$$

donde observamos que el lado izquierdo de la ecuación C.9 representa una similitud a la matriz de transformación $S(a)$. Por lo tanto la ecuación dice que la representación de la matriz $S(a)$ en coordenadas del marco de rotación por R es la misma que la matriz antisimétrica $S(Ra)$ corresponde al vector a rotado por R.

4. Para una matriz antisimétrica S de $n \times n$ y un vector $X \in \mathbb{R}^n$.

$$X^T S X = 0 \tag{C.11}$$

Derivada de la matriz de rotación.

Suponemos que la matriz de rotación R es una función de una variable θ. Entonces $R = R(\theta) \in SO(3)$ para todo θ. Entonces R es ortogonal para todo θ, esto resulta que:

$$R(\theta)R(\theta)^T = I \tag{C.12}$$

Diferenciando ambos lados de la ecuación C.12 con respecto a θ usando la regla del producto tenemos que:

$$\left[\frac{d}{d\theta}R\right] R(\theta)^T + R(\theta)\left[\frac{d}{d\theta}R^T\right] = 0 \tag{C.13}$$

Definimos la matriz S como:

$$S = \left[\frac{d}{d\theta}R\right] R(\theta)^T \tag{C.14}$$

Después se transpone a S:

$$S^T = \left(\left[\frac{d}{d\theta}R\right] R(\theta)^T\right)^T = R(\theta)\left[\frac{d}{d\theta}R^T\right] \tag{C.15}$$

Por lo tanto la ecuación C.13 queda como:

$$S + S^T = 0 \tag{C.16}$$

En otras palabras, la matriz S definida por la ecuación C.14 es la matriz antisimétrica. Al multiplicar ambos lados de la ecuación C.14 a la derecha por R y usando el hecho de que $R^T R = I$ tenemos que:

$$\frac{d}{d\theta}R = SR(\theta) \tag{C.17}$$

La ecuación C.17 es muy importante, ya que muestra que el cálculo de la derivada de la matriz de rotación R es equivalente a multiplicar dicha matriz por la matriz antisimétrica S.

Velocidad lineal de un punto fijo a un marco que se mueve.

Ahora consideramos la velocidad lineal de un punto fijo a un marco que se mueve, donde suponemos que el punto p es fijo al marco $o_1x_1y_1z_1$ y que $o_1x_1y_1z_1$ rota relativamente al marco $o_0x_0y_0z_0$. Entonces las coordenadas de p con respecto al marco $o_0x_0y_0z_0$ están dadas por:

$$p^0 = R_1^0(t)p^1 \tag{C.18}$$

La velocidad $\dot{p}^0$ está dada por la regla de diferenciación del producto como:

$$\begin{aligned} \dot{p}^0 &= \dot{R}_1^0(t)p^1 + R_1^0(t)\dot{p}^1 \\ &= S(\omega^0)R_1^0(t)p^1 \\ &= S(\omega^0)p^0 = \omega^0 \times p^0 \end{aligned} \tag{C.19}$$

Notamos que la ecuación C.20 está dada por el hecho de que p está fijo al marco $o_1x_1y_1z_1$ y por lo tanto las coordenadas relativas al marco $o_1x_1y_1z_1$ no cambian, dadas por $\dot{p}^1 = 0$. Ahora suponemos que el movimiento en el marco $o_1x_1y_1z_1$ relativo a $o_0x_0y_0z_0$ son más generales. Suponemos que la transformación homogénea que describe

a los dos marcos es dependiente del tiempo, por eso:

$$H_1^0(t) = \begin{bmatrix} R_1^0(t) & o_1^0(t) \\ 0 & 1 \end{bmatrix} \tag{C.20}$$

Con el fin de simplificar, omitimos el argumento t y los subíndices y superíndices en R_1^0 y o_1^1, y escribimos:

$$p^0 = Rp^1 + o \tag{C.21}$$

Diferenciamos la expresión anterior usando la regla del producto y tenemos que:

$$\begin{aligned} \dot{p}^0 &= \dot{R}p^1 + \dot{o} \\ &= S(\omega)Rp^1 + \dot{o} && \text{(C.22)} \\ &= \omega \times r + v && \text{(C.23)} \end{aligned}$$

dónde $r = Rp^1$ es el vector de o_1 a p expresado en la orientación del marco $o_0x_0y_0z_0$, y v es la velocidad a la que el origen o_1 se mueve. Si el punto p se mueve relativamente al marco $o_1x_1y_1z_1$, entonces debemos agregar al término v el término $R(t)\dot{p}^1$, el cual es la velocidad de cambio de las coordenadas p^1 expresadas en el marco $o_0x_0y_0z_0$.

Traza de una matriz

La traza de una matriz cuadrada A está definida como la suma de los elementos de la diagonal principal y es denotada como $tr[A]$. Por lo tanto:

$$tr[A] = \sum_{i=1}^{n} a_{ii} \tag{C.24}$$

dónde:

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix} \tag{C.25}$$

Es fácil observar que:

$$tr[AB] = tr[BA] \tag{C.26}$$

para dos matrices cualquiera A y B, y además:

$$a^T b = tr[ab^T] \tag{C.27}$$

para cualquier vector a y b.

BIBLIOGRAFÍA

[1] J. S. Lim, *Two-Dimensional Signal and Image Processing*, Prentice-Hall, Inc., Upper Saddle River, new Jersey. 1990.

[2] K. Sato, S. Ishizuka, A. Nikami and M. Sato, *Control techniques for optical image stabilizing system*, IEEE Trans. Consumer Electron., vol. 39, no. 3, pp. 461-466, Jun. 1993.

[3] S. Kimura et al, *Angular Velocity Measuring Instrumen*, USP 2544646, Mar.1985.

[4] Image Stabilizer, *Automatic image stabilizing system by full-digital signal processing*, USP 6064827, May. 2000

[5] T. Chen, *Video Stabilization Using a Block-Based Parametric Motion Model*, Technical report, Stanford University, Information Systems Laboratory, Dept. of Electrical Engineering, Winter 2000.

[6] J. Chang, W. Hu, M. Cheng and B. Chang, *Digital image translational and rotational motion stabilization using optical flow technique*, IEEE Transactions on Consumer Electronics, vol. 48, no. 1, pp. 108-115, Feb. 2002.

[7] C. Erdem and A. Erdem, *An illumination invariant algorithm for subpixel accuracy image stabilization and its effect on MPEG-2 video compression*, Elsevier Signal Processing: Image Communication, vol. 16, pp. 837-857, 2001.

[8] S. Ko, S. Lee, S. Jeon and E. Kang, *Fast digital image stabilizer based on gray-coded bit-plane matching.* , IEEE Transactions on Consumer Electronics, vol. 45, no. 3, pp. 598-603, Aug. 1999.

[9] M. Ben-Ezra, S. Peleg and M. Werman, *A Real-Time Video Stabilizer Based on Linear Programming.*

[10] A. Censi, A. Fusiello and V. Roberto, *Image Stabilization by Features Tracking*, Technical report, University of Udine, Machine Vision Laboratory, Dept. of Mathematics and Informatics, 1998

[11] J. Jin, Z. Zhu and G. Xu, *Digital Video Sequence Stabilization Based on 2.5D Motion Estimation and Inertial Motion Filtering*, Real-Time Imaging, vol. 7, pp. 357-365, 2001.

[12] J. Jin, Z. Zhu and G. Xu, *A Stable Vision System for Moving Vehicles*, IEEE Transactions on Intelligent Transportation Systems, vol. 1, no. 1, pp. 32-39, Mar. 2000

[13] K. Uomori, A. Morimura, H. Ishii, T. Sakaguchi and Y. Kitamure, *Automatic image stabilizing system by full-digital signal processing*, IEEE Transactions on Consumer Electronics, vol. 36, no. 3, ppl 510-519, Aug. 1990.

[14] Y-H. Yeh, H-C. Chiang and S-J. Wang, *Digital camcorder image stabilizer based on gray-coded bit-plane block matching*, Society of Photo-Optical Instrumentation Engineers, October, 2001.

[15] J. Zhu and B. Guo, *Fast layered bit-plane matching for electronic video stabilization*, International conference on Multimedia and Signal Processing, May, 2011.

[16] S-J. Ko, S-H. Lee and K-H. Lee, *Digital image stabilizing algorithms based on bit-plane matching*, IEEE Consumer Electronics Society, August, 2002.

[17] L. Fang and Q. Xiaozhen, *An Electronic Image Stabilization Algorithm Based on Efficient Block Matching on the Bitplane*, open Journal of Applied Sciences, March, 2013.

[18] H. Yu and W. Zhang, *Moving Camera Video Stabilization Based on Kalman Filter and Least Squares Fitting*, Proceeding if the 11th World Congress on Intelligent Control and Automation Shenyang, June, 2014.

[19] S.M. Bozic, *Digital and Kalman Filtering*, Edward Arnold Publishers, 1979.

[20] A. Gastounioti, S. Golemati, J. Stoitsis and K.S. Nikita, *Kalman Filter Based Block Matching for Arterial Wall Motion Estimation from B-Mode Ultrasound*, IEEE International Conference on Imaging Systems and Techniques (IST) , July, 2010.

[21] G. Bradski and A. Kaehler, *Learning OpenCV*, O'REILLY , United States of America, 2008.

[22] C. Harris and M. Stephens, *A combined corner and edge detector*, Proceedings of the 4th Alvey Vision Conference (pp. 147-151) , 1988.

[23] J. Shi and C. Tomasi, *Good Features to Track*, IEEE Conference on Computer Vision and Pattern Recognition, Seattle, June, 1994.

[24] M. Spong, S. Hutchinson and M. Vidyasagar, *Robot Modeling and Control*, John Wiley & Sons, 2005.

[25] R. Kelly and V. Santibañez, *Control de movimientos de robots manipuladores*, Pearson Education, 2003.