SISTEMA PARA EL SEGUIMIENTO DE UN OBJETO IMPLEMENTADO EN LA CABEZA DEL ROBOT INMOOV

NICOLAS ORTIZ VALENCIA
LUIS FELIPE VARGAS LONDOÑO

UNIVERSIDAD PILOTO DE COLOMBIA
FACULTAD DE INGENIERÍA MECATRÓNICA
BOGOTA D.C.
2016

SISTEMA PARA EL SEGUIMIENTO DE UN OBJETO IMPLEMENTADO EN LA CABEZA DEL ROBOT INMOOV

NICOLAS ORTIZ VALENCIA
LUIS FELIPE VARGAS LONDOÑO

Proyecto de grado para optar al título de Ingeniero Mecatrónico

Directores
Msc. Marco Antonio Jinete
Msc. Robinson Jimenez

UNIVERSIDAD PILOTO DE COLOMBIA
FACULTAD DE INGENIERÍA MECATRÓNICA
BOGOTA D.C.
2016

Nota de aceptación:

_____-

Una vez realizada la revisión metodológica y técnica del documento final de proyecto de grado, doy constancia de que el documento de los estudiantes Nicolás Ortiz Valencia y Luis Felipe Vargas Londoño ha cumplido a cabalidad con los objetivos propuestos, cumple a cabalidad con los Lineamientos de Opción de Grado vigentes del programa de Ingeniería Mecatrónica y con las leyes de derechos de autor de la República de Colombia, por tanto, se encuentran preparados para la defensa del mismo ante un jurado evaluador que considere idóneo el Comité de Investigaciones del Programa de Ingeniería Mecatrónica de la Universidad Piloto de Colombia.

_____

Marco Antonio Jinete Gómez
Director del Proyecto

Bogotá D.C. 6 de Febrero de 2017.

Resumen

La visión de máquina es una de las tareas más importantes en la interacción entre los robots y el medio en el cual están inmersos, pues esta le provee mayor información sobre los elementos que en éste existen.

El sistema implementado se realizó en el robot humanoide InMoov, el primer robot a escala real que puede ser producido por impresión 3D y cuyo diseño esta disponible al público.

Mediante el seguimiento del vector de movimiento, se implementó el control de la cabeza del robot con el propósito de seguir la trayectoria de un objeto. Fueron tomadas en cuenta las restricciones propias del diseño del robot junto con las restricciones cinemáticas del mismo para el desarrollo de los algoritmos a utilizar en el movimiento de la cabeza del robot bajo condiciones controladas.

En primer lugar, se diseñan y se definen los algoritmos de procesamiento de imágenes a utilizar y se realiza una comparación entre los mismos teniendo en cuenta su precisión y costo computacional.

En segundo lugar, se obtienen las restricciones de propias del diseño del robot junto con las restricciones cinemáticas del robot mediante la cinemática inversa y el diseño de este, seguido a esto se selecciona y se diseña la instrumentación adecuada conforme al modelo matemático obtenido y a los algoritmos implementados.

Finalmente, se diseña el algoritmo para el control de la cabeza del robot, se muestran los resultados obtenidos para el algoritmo seleccionado y se obtienen las conclusiones pertinentes.

Palabras Clave: Vector de movimiento, procesamiento de imágenes, costo computacional, precisión, rendimiento, cinemática inversa, modelo matemático.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Desde hace décadas, el estudio de los robots humanoides ha cobrado gran interés en la comunidad científica junto con cada uno de los retos que estos presentan, desde cómo resolver algunas de las necesidades de los seres humanos, como lo es el caso del robot ARMAR, del cual se hablará más adelante, hasta su uso en niveles industriales para el desarrollo de procesos, donde un ser humano podría cometer un error y así perder la vida.

Según Jonathan Ruiz de Garibay, Doctor en sistemas de información de la universidad de Deusto, la robótica es un campo tan amplio que su estudio es muy complejo, por lo que se hace necesario dividirla en secciones [1] con el fin de atender de manera individual a cada una de las necesidades del robot, con el objetivo de ampliar sus similitudes con el ser humano. Necesidades como lo son: la visión, el habla, las sensaciones, las respuestas auditivas y cognitivas, e inclusive la interacción que puede llegar a tener este con el entorno.

En este proyecto de grado, se pretende atacar una de estas necesidades imprescindibles para el desarrollo de un robot humanoide, la visión. Además, se propone afrontar las dificultades de implementar dicho sistema para la interacción entre el robot y el medio que lo rodea.

Dado el constante avance en este campo, se hace evidente la necesidad de implementar algoritmos con menores costos computacionales, así como mayor precisión y exactitud. Para estos es necesario seleccionar de manera adecuada cada uno de los algoritmos a implementar teniendo en cuenta factores como la cámara, la tarjeta electrónica, el lenguaje de programación a utilizar e incluso las técnicas de iluminación [7]. Esto con el fin de obtener un sistema que consuma la menor cantidad de recursos y tiempos de procesamiento.

El robot inmoov es un proyecto de software y hardware abierto, lo que significa que el diseño del mismo se puede obtener de forma gratuita junto con el software que utiliza el robot. En el presente documento se realiza un estudio comparativo de diferentes algoritmos para la detección y el seguimiento de un objeto, por medio de la obtención del vector de movimiento. Adicionalmente, se obtuviene el modelo cinemático del robot con el propósito de llevar estos algoritmos a un entorno físico, mediante la interacción entre el robot y el medio. Posteriormente, se realiza un algoritmo para la interacción entre los algoritmos para la detección del vector de movimiento y los actuadores del robot. Finalmente se realizan las pruebas sobre el robot y se evalúan los resultados para el algoritmo implementado.

1.1. Planteamiento y Formulación del Problema

Como se afirma en [32], una de las cuestiones principales en el campo de la robótica es el problema de la generación y modulación de la trayectoria, siendo esta alterada constantemente por diferentes factores, como por ejemplo obstáculos o perturbaciones en el sistema. Pese al avance que han tenido los robots humanoides en las últimas décadas, como lo es el caso del robot ASIMO de Honda [19], este tipo de dificultades siguen apareciendo y siguen siendo un reto para los investigadores en esta área.

1.1.1. Descripción del Problema

Como se mencionaba anteriormente, es necesario atacar una de estas dificultades de manera individual, en este caso la visión de máquina o visión artificial. De este modo, es necesario abordar las diferentes técnicas que existen en este campo para la detección de un objeto en movimiento, del tal forma que cumplan con el objetivo del robot, lo que a su vez conlleva a enfrentarse con la generación y modulación de los movimientos del mismo.

1.1.2. Formulación del Problema

Teniendo en cuanto lo anterior, se formúla la siguiente pregunta de investigación:

¿Cómo implementar un sistema de detección de movimiento por medio de visón artificial en un robot InMoov?

1.2. Justificación

Actualmente, como lo afirma Báez Ramos [29], los esfuerzos de muchos grupos de investigación nacionales en el campo de la robótica han generado grandes resultados de desarrollo, no obstante el nivel de investigación en este campo sigue siendo bastante bajo en comparación con otros países, debido a la falta de apoyo, tanto de la administración pública como del sector privado, así como la carencia de programas de doctarado en áreas relacionadas con la tecnología.

Según cifras del Observatorio Colombiano de Ciencia y Tecnología [31], el número de grupos de investigación en Ingeniería y Tecnología disminuyó en aproximadamente un 9 % entre el 2013 y el 2014, y representan tan solo el 17 % del total de grupos de investigación activos en el 2014.

Cabe rescatar que, ha pesar de la evidente falta de investigación en el país, la productividad y calidad científica en estos últimos años, como lo afirmó Colciencias [30] el pasado 26 de mayo del 2016, ha aumentado significativamente, registrando un incremento del 17 % en grupos de investigación y un 21 % en el número de investigadores con los que actualmente cuenta el país.

Por otra parte, como lo afirman Smola y Vishwanat [2], el campo del aprendizaje automático a tomado una gran importancia en las últimas décadas en el área de las tecnologías de la información, y cada vez se ha hecho mas evidente el afán por tratar de imitar y/o replicar la conducta humana en mayor o menor medida. En este sentido, es necesario decir que una gran parte de la informacion que es procesada por el ser humano es principalmente visual, y toda interacción física en el medio esta enmarcada en este ámbito. Estos estímulos sirven de ayuda para la comprensión y el entendimiento del mundo físico.

Debido a lo descrito anteriormente, se decide implementar un robot humanoide, el robot InMoov, de Hardware y Software libre, lo que permitirá una reducción en costos y tiempos de diseño, y se desarrolla un sistema para el seguimiento de un objeto implementado en la cabeza del robot, con el propósito de sentar una base sólida para el desarrollo de plataformas robóticas que impulsen la investigación en las áreas de visión de máquina y aprendizaje automático, concernientes al área de Ingeniería y Tecnología.

1.3. Objetivos

1.3.1. Objetivo General

Diseñar e implementar un sistema basado en la cabeza del robot inmoov para la detección y seguimiento de un objeto.

1.3.2. Objetivos Específicos

• Realizar el modelo cinemático de la cabeza del robot inmoov.

• Seleccionar e implementar la instrumentación requerida para el movimiento de la cabeza.

• Diseñar e implementar las tarjetas electrónicas requeridas para el control de los actuadores y la adquisición de datos provenientes de la instrumentación.

• Seleccionar e Implementar el algoritmo apropiado, basado en visión de máquina, para la detección del vector de movimiento bidimensional de un objeto en el campo de visión del robot.

• Diseñar e implementar un algoritmo de interacción entre el algoritmo de detección del vector de movimiento y los movimientos de la cabeza del robot

1.4. Estado del Arte

La visión de máquina es hoy uno de los ejes fundamentales para el desarrollo de robots con la capacidad de interactuar con el medio que los rodea. En este sentido se han desarrollado multiples técnicas que permiten substraer información del medio en el cual se encuentra el robot, de tal forma que le permita cumplir con el objetivo para el cual este fue diseñado.

La detección de objetos es fundamentalmente el tema más ampliamente abordado por la comunidad científica que estudia este campo, y es comúnmente utilizada en el desarrollo de robots humanoides para realizar taréas como el reconocimiento facial, el seguimiento de objetos móviles [8], la captura y manipulación de objetos [9], entre otras aplicaciones.

A continuación se describe el estado de diferentes técnicas que actualmente se han implementado en el desarrollo de robots humanoides, basadas en procesamiento digital de imágenes como en otro tipo de metodologías igualmente válidas.

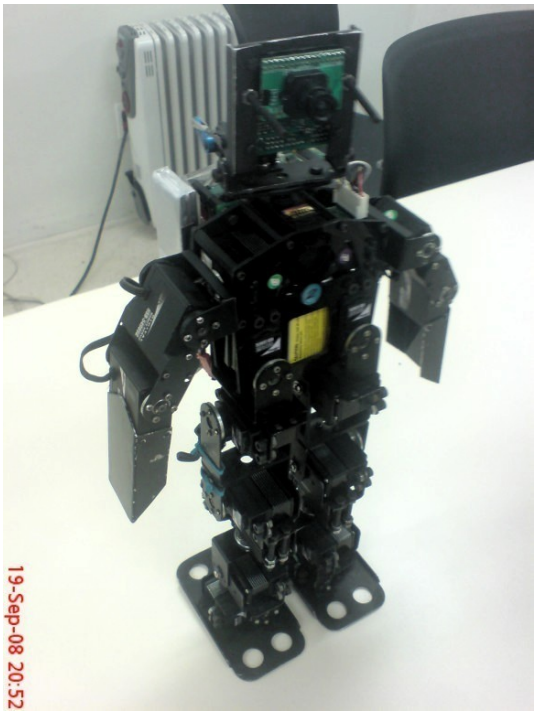
Figura 1: Robot Humanoide ARMAR

Fuente: Science Daily, Robot obeys to commands and gestures. [en linea]. <https://www.sciencedaily.com/releases/2012/02/120224110605.htm> [citado el 10 de septiembre de 2016].

La captura de objetos por medio de visión de máquina sigue siendo hoy en día un problema difícil de abordar en sistemas basados en robots humanoides como se afirma en [9], donde se implementa un sistema de visión estéreo 6D para la localización de objetos y su captura implementado en un robot

ARMAR (véase Figura 1), por medio de dos camáras ubicadas en posiciones similares a las de los ojos de un ser humano, este sistema percibe de manera tridimensional las características del medio sobre el cual esta sumergido el robot. Azad, Asfour y Dillmann presentan un sistema de alta precisión capas de lidiar con objetos con textura, así como con objetos que pueden ser segmentados de forma global y que cuya forma este ya predefinida, implementado en un sistema de coordenadas 3D sujeto a un entorno rigido o invariable y en objetos prismáticos rectángulares (cuboides).

Figura 2: Robot Humanoide Bogobot

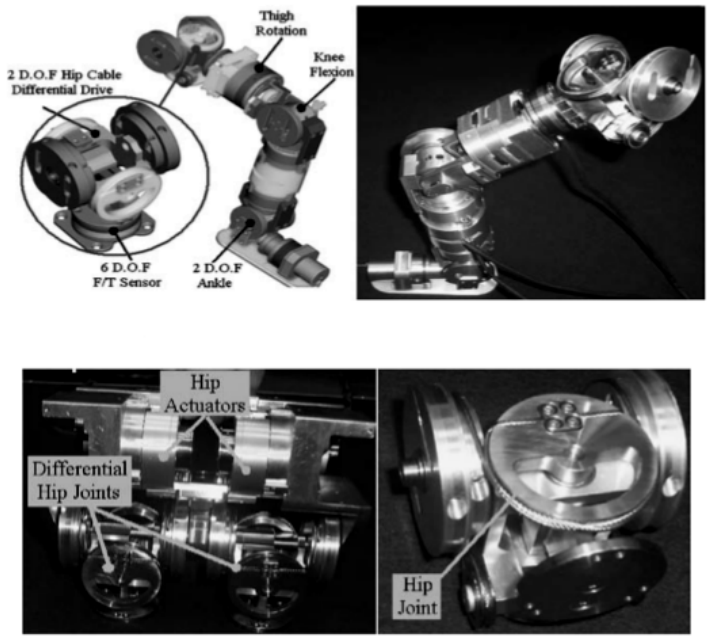


Fuente: Barrera [10]

De la misma forma Barrera [10] presenta un sistema de percepción y localización implementado a los robots Bogobots[33] (véase Figura 2)con el objetivo de mejorar su desempeño. Para esto se utilizaron dos clasificadores de color; uno para el uso de superficies implícitas en el espacio RGB y otro a través de seis umbrales para evaluar cada una de las componentes en el espacio HSI, con el propósito de segmentar la imágen e identificar los objetos inherentes en esta por medio de la separación de estas características con el uso de árboles de decisión. Esto permite estimar la distancia real entre los objetos y el robot,

permitiendole trazar un trayectoria para verificar la efectividad de los algoritmos implementados.

Figura 3: Realización mecánica de la pierna y enrutamiento del cable de transmisión del par de articulaciones diferenciales de cadera del iCub.



Fuente: Tsagarakis. et al. [6]

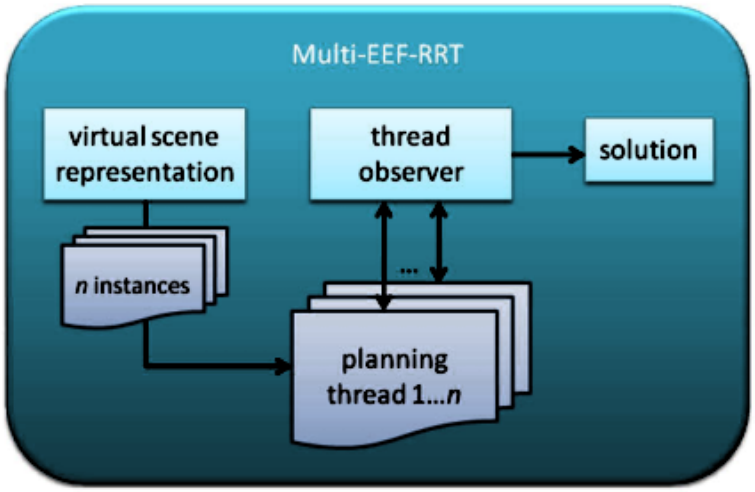
Los autores citados anteriormente, presentan una visión de su trabajo enmarcada exclusivamente en el procesamiento digital de imágenes, dejando a un lado lo concerniente a la instrumentación requerida, los modelos matemáticos del sistema de movimiento del robot y los aportes realizados a otros campos de la ciencia. Sin embargo, podemos encontrar trabajos como [4], donde se describe la implementación de un sistema de visión de maquina con un Kinect, con el objetivo de buscar la mayor naturalidad en los movimientos de un robot Bioloid. O como [5] en donde se detallan la cinemática, la dinámica y los sistemas de movimiento de un robot humanoide AILA. También lo es el caso del ICub [6], donde se muestra el diseño y desarrollo robot y además los aportes del mismo al campo de la neurociencia (véase Figura 3).

Además de las técnicas para la detección de objetos implementadas en robots humanoides como las abordadas en los trabajos descritos anteriormente y las descritas en [11] [12] [13], también es posible encontrar otro tipo de

metodologías abordadas como la que se menciona en [14], donde se implementa un *DForC* por sus siglas en inglés (Controlador dinámico de campo de fuerzas), sistema que intenta integrar la visión de máquina con las restricciones y características cinemáticas propias de un robot. También se pueden dislumbrar metodologías mucho más robustas como en [15], donde se presenta un algoritmo de visión de máquina junto con un algoritmo de aprendizaje de máquina que le permite al robot aprender a estimar de manera precisa posiciones tridimensionales relativas a este utilizando redes neuronales artificiales junto con algoritmos genéticos (ANN y GP respectivamente, por sus siglas en inglés).

En las tareas que requieren una interacción física entre el medio y el robot humanoide, es necesario también dislumbrar la forma en que se aborda el control de los actuadores del robot y las técnicas utilizadas para alcanzar el objetivo final, por ejemplo capturar un objeto en movimiento, seguir la trayectoria de este, asistir a un ser humano en una labor de rescate etc. Para esto son ampliamente utilizadas las técnicas de planeación de trayectoria.

Figura 4: Planificador MultiEEF-RRT



Fuente: Tsagarakis. et al. [16]

En [16] se propone un algoritmo MultiEEF-RRT para la planeación de trayectorias libres de colisiones para la captura de objetos utilizando búsquedas paralelas en cada uno de los brazos del robot ARMAR-III, lo cual le permitió

al robot por medio de visión de máquina calcular la trayectoria más adecuada para la captura del objeto de estudio, con la capacidad de escoger con cual brazo era posible o más eficiente realizar la tarea programada. En el esquema de la Figura 4 se puede observar la forma en funciona este algoritmo, paralelizando cada uno de los hilos posibles («*threads*») y observando cada una de las trajectorias correspondientes a estos para luego observar la mejor solución. Sin embargo existen también métodos, por llamarlos de cierta forma, poco ortodoxos como lo es el caso de [17] y [18], donde se implementa programación por demonstración para enseñarle a un robot humanoide a realizar tareas basado en las demonstraciones de un ser humano y las técnicas que permiten limitar los movimientos del robot respecto a las expresiones corporales realizadas por un ser humano.

1.5. Alcances y Limitaciones

1.5.1. Alcances

El proyecto a desarrollar tiene como alcance implementar la cabeza de un robot InMoov para la detección y seguimiento del vector de movimiento de un objeto. Mediante el análisis cinemático, se realizará el algoritmo correspondiente para el control de los actuadores de la cabeza del robot. Para la detección del vector de movimiento se implementará un algoritmo basado en el procesamiento digital de imágenes, permitiendo que la cabeza del robot tenga la capacidad de seguir el vector de movimiento de cualquier objeto.

Los aspectos puntales que comprende la investigación están referidos a la cinemática inversa y directa del robot para determinar la forma en que deberán moverse los actuadores. El diseño del sistema electrónico estará enfocado a la adquisición de las señales provenientes de los sensores y la etapa de potencia que requieran los actuadores para el control en lazo cerrado de los movimientos de la cabeza.

1.5.2. Limitaciones

El sistema está basado en el diseño de la cabeza del robot InMoov, por esta razón el análisis correspondiente a la resistencia de los materiales y al diseño mecánico del mismo no estarán contemplados en el desarrollo del proyecto.

El algoritmo de detección de movimiento basado en el procesamiento digital de imágenes se hará bajo condiciones de luz controladas, teniendo en cuenta las capacidades de la cámara y las necesidades del sistema.

La velocidad de respuesta del sistema estará sujeta a las capacidades y condiciones propias de la instrumentación de la cabeza del inMoov; como la

velocidad de adquisición de los fotogramas de la cámara, la capacidad mecánica de los actuadores entre otras.

1.6. Linea de Investigación del Programa

La línea de investigación a la cual se acoge este proyecto es:

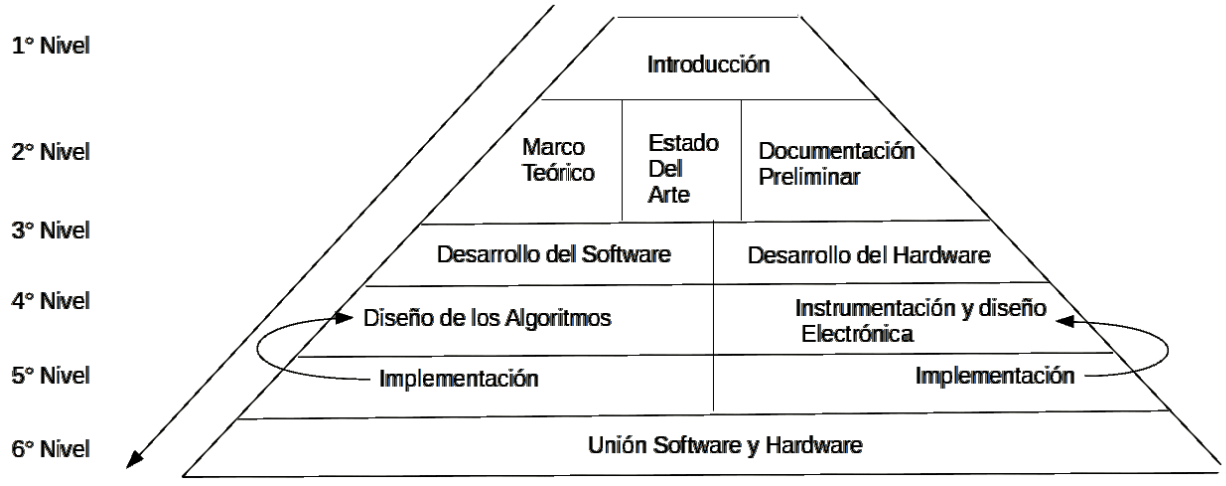
- Robótica y Biomecatrónica.

Esto debido a que el objetivo principal del proyecto es la aplicación de un algoritmo de visión de máquina en la cabeza de un robot humanoide (Robot InMoov).

1.7. Diseño Metodológico

El diseño metodológico se baso en el esquema piramidal mostrado en la Figura 5, que busca cumplir cada una de las tareas contenidas en cada nivel de manera descendente desde el primer hasta el último nivel. Se puede observar que las tareas del quinto y cuarto nivel se pueden reiterar dependiendo de los resultados obtenidos en el quinto nivel.

Figura 5: Esquema Metodológico



Fuente: Elaboración propia

1.8. Contribuciones

Durante el desarrollo del proyecto, se realizó un artículo sobre dos de los algoritmos utilizados y fue presentado como ponencia en el Simposio Internacional STSIVA, el cual se expone en el Anexo A:

• ORTIZ, Nicolás. VARGAS, Luis Felipe. JINETE, Marco Antonio y JIMÉNEZ, Robinson. Movement detection for object tracking applied to the InMoov robot head. En: 2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA). 2016. ISBN: 978-1-5090-3797-1.

Además se contó con la participación y el reconocimiento en los siguientes eventos:

• Conferencistas Mini Maker Fair, Diciembre de 2016. Biblioteca Virgilio Barco. Visión artificial aplicada al robot Inmoov.

• Ganadores: Robotic People Awards 2016, por los aportes realizados a la robótica y la tecnología. Diciembre de 2016.

2. Marco Teórico

2.1. Visión de Máquina y Procesamiento de imágenes

Para el desarrollo del proyecto es necesario aclarar la diferencia entre visión de máquina y procesamiento digital de imágenes. Aunque similares, técnicamente se distinguen por el propósito final que persiguen. Como se expone en [21], la visión por computador o el procesamiento digital de imágenes, es el estudio a través del cual se busca describir el mundo observable en una o más imágenes mediante la reconstrucción y extracción de sus propiedades tales como: la forma, la iluminación o la distribución de colores. La visión de máquina, en cambio, como se afirma en [35], busca crear un modelo del mundo físico, y es aquí donde rádica la diferencia, a partir de imágenes. El objetivo no es solo extraer características de este, si no que también crear un modelo que permita la interacción entre una máquina y el medio que la rodea. No obstante, la visión de máquina depende a su vez del procesamiento digital de imágenes, por lo que en primera instancia se describirán los pilares fundamentales de estas dos áreas.

2.1.1. Preliminares

Los investigadores en visión por computadora, desde sus inicios, han desarrollado modelos matemáticos y técnicas para el estudio de las imágenes. Por esta razón, se introduce a continuación la notación utilizada para describir geométricamente los elementos que en estas se ecuentran presentes.

2.1.1.1. Notación

2.1.1.1.1. Puntos Los puntos 2D, pueden ser denotados como un par de valores $p = (x, y) \in \mathbb{R}^2$, o alternativamente como:

$$p = \begin{bmatrix} x \\ y \end{bmatrix}$$

De igual forma los puntos 3D, pueden ser denotados por tres valores como $p = (x, y, z) \in \mathbb{R}^3$, o alternativamente como:

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

También se suelen representar en coordenadas homogéneas como se muestra a continuación:

Para puntos en 2D	**Para puntos en 3D**
$\widehat{p} = (\widehat{x}, \widehat{y}, \widehat{w}) \ \in \ \mathbb{P}^2$	$\widehat{p} = (\widehat{x}, \widehat{y}, \widehat{z}, \widehat{w}) \ \in \ \mathbb{P}^3$

Donde el espacio $\mathbb{P}^2$ es equivalente al espacio $\mathbb{R}^3 - (0, 0, 0)$ y de igual forma $\mathbb{P}^3$ es equivalente al espacio $\mathbb{R}^4 - (0, 0, 0, 0)$.

2.1.1.1.2. Líneas en 2D y Planos en 3D De igual forma que los puntos, las líneas pueden ser representadas en coordenadas homogéneas como $\widehat{l} = (a, b, c)$ para 2D, y $\widehat{l} = (a, b, c, d)$ para planos en 3D, correspondientes a:

Ecuación de la línea en 2D	**Ecuación del Plano en 3D**
$\overline{p} \cdot \widehat{l} = ax \ + \ by \ + \ c = 0$	$\overline{p} \cdot \widehat{l} = ax \ + \ by \ + \ cz \ + \ d = 0$
Donde $\overline{p} = (x, y, 1)$ es el vector aumentado en 2D.	Donde $\overline{p} = (x, y, z, 1)$ es el vector aumentado en 3D.

2.1.1.1.3. Líneas en 3D A diferencia de las representaciones anteriormente descritas, una línea en 3D no se puede expresar tan facilmente. Para esto usualmente se utilizan dos puntos sobre la línea. (m, n), de tal forma que cualquier otro punto sobre la línea puede ser expresado como una combinación líneal de la forma:

$$r = (1 - \lambda)m + \lambda n$$

o bien en coordenadas homogéneas como:

$$\widehat{r} = \mu\widehat{m} + \lambda\widehat{n}$$

2.1.1.2. Transformaciones 2D y 3D

2.1.1.2.1. Traslación Las traslaciones pueden ser denotadas como $p' = p + t$ o:

$$p' = \begin{bmatrix} I & t \end{bmatrix} \overline{p}$$

Donde I es una matriz identidad de (2×2) si se trabaja en $\mathbb{R}^2$ y (3×3) si se trabaja en $\mathbb{R}^3$, de igual forma podemos denotarla como:

$$\overline{p}' = \begin{bmatrix} I & t \\ \mathbf{0^T} & 1 \end{bmatrix} \overline{p}$$

donde $\mathbf{0}$ es el vector cero y t es el vector de traslación, denotado como:

$$t = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

para traslaciones en 2D, y como:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_x \end{bmatrix}$$

para traslaciones en 3D, donde t_x, t_y y t_z representan los desplazamientos en cada eje coordenado.

2.1.1.2.2. Rotación y Traslación También conocida como Transformación Euclidiana 3D en el espacio $\mathbb{R}^3$, puede ser denotada de la forma $p' = Rp + t$ o:

$$p' = \begin{bmatrix} R & t \end{bmatrix} \overline{p} \tag{1}$$

Donde R es una matriz de rotación ortonormal de (2×2) para $\mathbb{R}^2$ y de (3×3) para $\mathbb{R}^3$ con $RR^T = I$ y determinante $|R| = 1$. La diferencia de realizar esta transformación en 2D o en 3D, radica en la forma de expresar la matriz R. Mientras en un espacio bidimensional esta se expresa fácilmente como:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

En un espacio tridemensional, dependiendo de la rotación realizada, R se puede expresar de múltiples formas a través del producto de tres rotaciones alrededor de los tres ejes cardinales. Estas matrices pueden ser obtenidas por medio de los ángulos de Euler como se explica en [36].

En la sección 2.2.1.2 se describen de manera más formal estas matrices.

2.1.1.2.3. Rotación Escalada o Transformación de Similitud Esta transformación puede expresarse de forma similar a la rotación como $p' = sRp + t$, donde s es un factor de escala arbitrario. Esta puede ser expresa también como:

$$p' = \begin{bmatrix} sR & t \end{bmatrix} \overline{x}$$

En este caso la matriz R se comporta de igual forma que en la rotación para los diferentes espacios de trabajo, sin embargo en este caso ya no es estricatamente necesario que $|R| = 1$.

2.1.1.2.4. Transformación Afín Esta transformación permite que las líneas conserven la paralelidad que exista entre ellas. Se denota como $p' = A\overline{p}$, donde A es una matriz (2×3) para $\mathbb{R}^2$ y (3×4) para $\mathbb{R}^3$.

2.1.1.2.5. Transformación Proyectiva Esta transformación opera en coordenadas homogéneas, y se denota como:

$$\widehat{x}' = \left[\widehat{H}\right] \widehat{x}$$

donde $\widehat{H}$ es una matriz arbitraria (3×3) para $\mathbb{R}^2$ o (4×4) para $\mathbb{R}^3$.
En esta transformación se preservan las líneas rectas.

En la siguiente tabla se encuentran contenidas las transformaciones mencionadas anteriormente, junto con las matrices que las definen y el grado de libertad que poseen.

Cuadro 1: Transformaciones en 2D y 3D.

Transformación	**2D**	**3D**	**# DoF 2D**	**# DoF 3D**	**Gráfico**
Traslación	$\left[\ I \mid\ t\ \right]_{2\times3}$	$\left[\ I \mid\ t\ \right]_{3\times4}$	2	3	
Euclidiana	$\left[\ R \mid\ t\ \right]_{2\times3}$	$\left[\ R \mid\ t\ \right]_{3\times4}$	3	6	
Similitud	$\left[\ sR \mid\ t\ \right]_{2\times3}$	$\left[\ sR \mid\ t\ \right]_{3\times4}$	4	7	
Afín	$[A]_{2\times3}$	$[A]_{3\times4}$	6	12	
Proyectiva	$\left[\widehat{H}\right]_{3\times3}$	$\left[\widehat{H}\right]_{4\times4}$	8	15	

Fuente: Szeliski [21]. Nótese las siglas # DoF, estas corresponden a los grados de libertad de cada transformación por sus siglas en inglés «Degrees of Freedom».

2.1.1.3. Proyecciones 3D a 2D Las proyecciones 3D, consisten esencialmente en proyectar un punto en un espacio tridimensional a un ezpacio bidimensional.
En esta sección denotaremos como p a un punto en el espacio $\mathbb{R}^3$ y o a un punto en el espacio $\mathbb{R}^2$.

2.1.1.3.1. Proyección Ortigráfica Consiste en despecriar el valor real de la componente z de la coordenada tridimensional p para obtener la coordenada bidimensional 0. De esta forma:

$$o = \left[\; I_{2\times 2} \mid \quad \mathbf{0} \; \right] p$$

Expresado en coordenadas homogéneas como:

$$\widehat{o} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \widehat{p}$$

Podemos observar, que si se tiene el punto $p = \left[\; x \quad y \quad z \; \right]^T$, la proyección ortográfica o de p será $\left[\; x \quad y \; \right]^T$.

En términos prácticos, como lo afirman Sawhney y Hanson [37], este tipo de proyección es un modelo aproximado de los lentes de larga distancia focal y de los objetos cuya profundidad relativa, con respecto a la distancia que hay de la cámara a este, es despreciable.

2.1.1.3.2. Proyección Ortográfica Escalada Como se afirma en [21], en la práctica, las coordenádas físicas (medidas generalmente en metros) deben ser escaladas apropiadamente, de manera tal que correspondan con las medidas del sensor de la cámara (pixeles).

Denotada como:

$$o = \left[\; sI_{2\times 2} \mid \quad \mathbf{0} \; \right] p$$

Este modelo equivale a proyectar los puntos del mundo físico en un plano local paralelo al sensor y luego escalar nuevamente la imágen utilizando una proyección de perspectiva de la cual se hablará más adelante.
Este tipo de proyección, es ampliamente utilizada para la reconstrucción 3D a partir de imágenes en 2D. Bajo este modelo, el movimiento y la estructura de un objeto pueden ser facílmente estimados usando el método de factorización de Tomasi y Kanade en [38].

2.1.1.3.3. Proyección Perspectiva-paralela La proyección perspectiva-paralela (véase [40] [39]) es un modelo que busca proyectar los puntos de un objeto, dada un referencia local paralela al plano de la imágen como su nombre lo indica. No obstante, en lugar de ser proyectados de forma ortogonal al plano, son proyectados paralelos a la línea de visión hacia el centro del objeto. Esto seguido usualmente de una proyección final sobre el plano de la imágen, equivalente a una escala.
Este modelo puede ser denotado como:

$$\widehat{o} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \widehat{p}$$

2.1.1.3.4. Proyección Perspectiva Siendo una de las más utilizadas en visión de máquina, esta representa una perspectiva 3D real sobre la imágen. En coordenadas no homogéneas, consiste en dividir las compenentes x y y por la componente z del punto p. Se denota como sigue:

$$\overline{o} = P_z(p) \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$$

O en coordenadas homogéneas como:

$$\widehat{o} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \widehat{p}$$

2.1.1.4. Pixeles Vecinos Se puede deducir intuitivamente, que el uso de modelos como los descritos anteriormente, no es suficiente para describir de manera eficiente las características inmersas en una imágen.
Uno de los problemas que presenta una imagen como una estructura de datos, es la cantidad de información que debe ser procesada y analizada. A fin de realizar un procesamiento efectivo, es necesario adaptar la teoría existente (Teoría de Grafos o Topología, en general) para el manejo de estructuras matemáticas como los son las imágenes digitales. Bajo esta premisa, en esta sección se describe la estructura más elemental en Teoría de Grafos para el manejo de imágenes, los conjuntos de pixeles vecinos, expuesta de manera formal por Voss [46].

Se denotará como N a la relación binaria de vecindad («neighborhood relation»). Para su descripción existen dos propiedades básicas en la noción de vecindad. Primero, un punto p, no puede ser vecino de si mismo, en consecuencia el par (p, p) no es un miembro de la relación binaria deseada. Por otro parte, un punto p es vecino de un punto q si a su vez, q es vecino de p. De esta forma $(p, q) \in N$ y $(q, p) \in N$.

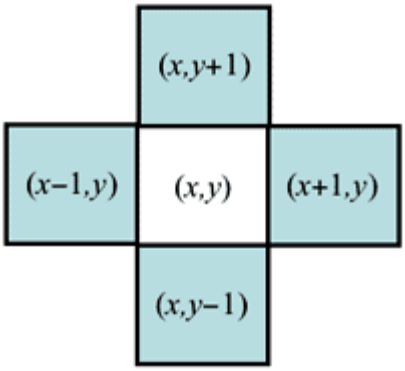
Definición 1 *Una tupla $[P, N]$, con $p \neq \varnothing$ y $N \subset P \times P$ como una relación irreflexiva y simétrica, es una estructura de vecindad.*

Definición 2 *Si $p \in P$ de una estructura de vecindad $[P, N]$, el conjunto de puntos adjuntos $N(p) = \{q \mid (p, q) \in N\}$, es llamado el conjunto de vecinos de p.*

Es a partir de estas dos definiciones, que el autor ya mencionado expone la teoría de grafos aplicada al procesamiento de imágenes. De esta teoría surgen varios modelos que permiten describir propiamente al conjunto de pixeles vecinos. Si se observa detenidamente, la relación binaria N indica cuando un

punto q es vecino de p. Uno de estos modelos es el de 4-pixeles vecinos, donde un punto $q = (x_q, y_q)$ es vecino de un punto $p = (x_p, y_p)$ si se cumple que $(x_q = x_p+1) \wedge (y_q = y_p)$, o $(x_q = x_p) \wedge (y_q = y_p+1)$, o $(x_q = x_p-1) \wedge (y_q = y_p)$, o $(x_q = x_p) \wedge (y_q = y_p - 1)$. Este modelo se observa más facilmente de forma gŕafica como se muestra a continuación.

Figura 6: 4-vecindad de p



Fuente: Image Segmentation [47].

2.1.2. Procesamiento de Imágenes

A continuación se realiza una revisión de algunos conceptos y técnicas para el procesamiento de imágenes, enmarcados en el objetivo principal de este proyecto.

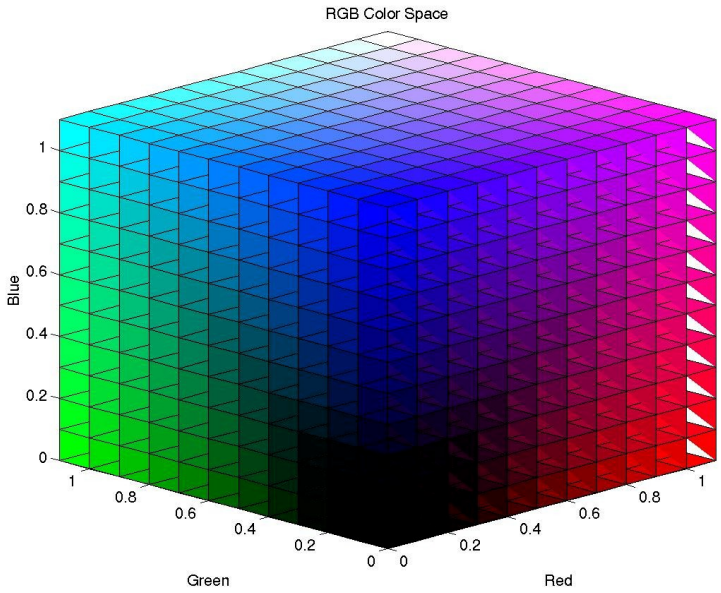
2.1.2.1. Espacios de Color Las imágenes que podemos adquirir a través de una cámara, usualmente estan estrictamente relacionadas a la forma en que el ser humano percibe el color. Dichas imágenes pueden ser entendidas como funciones arbitrarias de valor vectorial. Gonzalez y Woods en [41], afirman que el uso color en el procesamiento de imágenes esta motivado fundamentalmente por el potencial que tiene este como un descriptor de la imágen, lo que en ocasiones facilita considerablemente la detección de un objeto y su extracción de una escena.

El propósito de usar un espacio de color es la de facilitar la especificación de este en una forma estandarizada, generalizada y aceptada.

Un espacio de color es una representación de la gama de colores en un sistema coordenado, donde cada color esta determinado por un punto en el espacio.

2.1.2.1.1. RGB El espacio RGB es el espacio más ampliamente utilizado, pues es el modelo de color que utilizan los monitores y una amplia gama de video cámaras. Viene representado por un sistema de coordenadas cartesianas, donde cada eje representa la intensidad del color en cada uno de los canales R (rojo), G (verde) y B (azul) como se muestra a continuación:

Figura 7: Espacio RGB



Fuente: ArcSoft, What is Color Space?. [en linea]. <http://www.arcsoft.com/topics/photostudio-darkroom/what-is-color-space.html> [citado el 12 de septiembre de 2016].

Nótese que las coordenádas de la Figura 7 se encuentran en el rango $[\ 0 \quad 1\ ]$, pues estas estan normalizadas. Generalmente, encontramos que cada uno de los canales de la imagen posee 8 bits, es decir 256 valores enteros en un rango de 0 a 255. Estos valores a su vez, representan la intensidad de cada color en un punto o pixel.

2.1.2.1.2. CMY y CMYK El modelo CMY, al igual que el modelo RGB, esta compuesto de tres canales; C (Cian), M (Magenta) y Y (Amarillo). Estos tres colores corresponden a los colores secundarios de la luz, o alternativamente a los colores primarios de los pigmentos. Es mayormente utilizado por los dispositivos que depositan pigmentos sobre una superficie, tales como

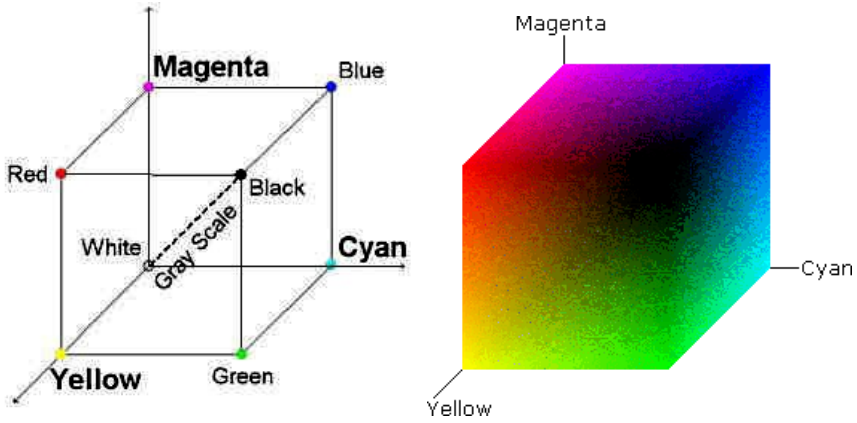
las impresoras a color.
La conversión del modelo RGB al modelo CMY se realiza de la siguiente forma:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Esto suponiendo que los valores de cada canal se encuentran normalizados.

Su representación gráfica se puede observar en la Figura que se muestra a continuación:

Figura 8: Espacio CMY



Fuente: (a) [42], (b) [43].

Como se afirma en [41], la combinación de los valores máximos de cada canal (C,M y Y) en teoría produce el color negro. Sin embargo, en la práctica, esto no ocurre del mismo modo. Por este motivo, es necesario añadir un cuarto canal K, a fin de conseguir el color negro, de esto surge el modelo CMYK.

2.1.2.1.3. Espacio XYZ Los espacios anteriormente descritos se limitan única y exclusivamente a describir el color. Aunque perfectamente válidos, en el mundo físico la forma en que se ven y se observan las cosas depende también de factores como la luz. Estos factores modifican la percepción del color, lo cual por ejemplo, impedira que observemos el color de un auto en la noche de la misma forma en que lo vemos en el día.

En este sentido, el modelo XYZ intenta compensar estos factores, suprimiendo dos parámetros que se suponen afectan el color del pixel; el matiz y la saturación. La transformación del espacio RGB al espacio XYZ se describe como:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \frac{R}{R+G+B} \\ \frac{G}{R+G+B} \\ \frac{B}{R+G+B} \end{bmatrix} \tag{2}$$

Poynton [44] señala que los valores de X, Y y Z, también llamados «tristimulus values» en inglés, representan las cantidades necesarias para formar cualquier color en particular, utilizados en imágenes digitales. Estos a su vez, son utilizados para la caracterización y calibración de dispositivos de salida como lo menciona Schanda [45]. En la sección 3.1.1 se describe de manera más detallada la forma en que este modelo funciona.

2.1.3. Flujo Óptico

Fleet y Weiss [49] mencionan que el movimiento es una propiedad intrínseca del mundo físico y una parte integral de la experiencia visual del ser humano. Es una gran fuente de información que permite realizar un sin número de taréas tales como; reconstrucción 3D, organización perceptual, reconocimiento de objetos entre otras.

El Flujo Óptico se define como el movimiento aparente de una imagen en una secuencia de imágenes [50]. El propósito de estimar el flujo óptico, es el de computar una aproximación al campo de movimiento, a partir del análisis de las variaciones en la intensidad de una imágen en el tiempo. Existen múltiples enfoques para estimar el movimiento, siendo la estimación basada en gradientes, el método más ampliamente utilizado. Por este motivo, en esta sección se ahondará en la descripción de este método. No obstante, el lector puede ver en [51] una descripción detallada y un estudio comparativo de otras técnicas utilizadas.

2.1.3.1. Estimación Basada en Gradientes Sean I y J imágenes en escala de grises 2D. Donde $I(n) = (i, j)$ y $J(n) = (i, j)$ son el primer y segundo valor de la intensidad de los pixeles correspondientes a las imágenes en la posición $n = [i\ j]^T$, se denotan para este caso como i y j a las posiciones rectangulares del punto n en la imagen. Considérese el punto en la imagen $u = [u_i\ u_j]^T$, el propósito en este caso es el de calcular la posición del punto $v = u + d = [u_i + d_i\ u_j + d_j]$ en la segunda imagen J. Notese, que esto implica estimar la traslación del punto de la primera a la segunda imagen, como se explicaba anteriormente en la sección 2.1.1.2.1. Si el desplazamiento del pixel se supone pequeño, se obtiene que la diferencia de las intensidades del pixel en las dos imágenes se expresa como:

$$J(i + d_i, j + d_j) - I(i, j) \approx 0$$

Expandiendo por series de Taylor a $J(i+d_i, j+d_j)$:

$$J(i+d_i, j+d_j) = J(i,j) + \frac{\partial J}{\partial i}d_i + \frac{\partial J}{\partial j}d_j$$

$$J(i,j) + \frac{\partial J}{\partial i}d_i + \frac{\partial J}{\partial j}d_j - I(i,j) \approx 0$$

Sea $J_i = \frac{\partial J}{\partial i}$ y $J_j = \frac{\partial J}{\partial j}$ entonces:

$$(J(i,j) - I(i,j)) + J_i d_i + J_j d_j \approx 0$$

$$J_t + \nabla J \cdot [d_i \ d_j] \approx 0 \tag{3}$$

El propósito final de esta técnica, es el de estimar el vector de desplazamiento $[d_i \ d_j]^T$ para cada pixel en la imágen. Sin embargo, la ecuación 3 corresponde a un sistema de una ecuación y dos incógnitas, a esto se le conoce como el problema de apertura. Las diferencias entre las diferentes técnicas existentes para la estimación del flujo óptico, radican en las condiciones adicionales que son utilizadas para completar el sistema de ecuaciones, y en este sentido calcular las variables deseadas.

2.1.3.2. Algoritmos para la Estimación del Flujo Óptico En [51], se presenta un estudio comparativo de diferentes técnicas concernientes a este enfoque en el procesamiento de imágenes. El cuadro que se muestra a continuación corresponde a los resultados obtenidos, en el documento ya mencionado, para diferentes algoritmos ante un desplazamiento producido mediante el desfase definido de una señal sinusoidal en la imagen. Se considera pertinente incluirlo, para observar los algoritmos más eficientes en este campo.

Cuadro 2: Resultados Obtenidos para un desfase sinusoidal definido

Technique	Average Error	Standard Deviation	Density
Horn and Schunck (original)	4.19°	0.50°	100%
Horn and Schunck (modified)	2.55°	0.59°	100%
Lucas and Kanade (no thresholding)	2.47°	0.16°	100%
Uras et al. (no thresholding)	2.59°	0.71°	100%
Nagel	2.55°	0.93°	100%
Anandan	30.80°	5.45°	100%
Singh ($n = 2$, $w = 2$, $N = 2$)	2.24°	0.02°	100%
Singh ($n = 2$, $w = 2$, $N = 4$)	91.71°	0.04°	100%
Waxman et al. $\sigma_J = 1.5$	64.26°	26.14°	12.8%
Fleet and Jepson $\tau = 1.25$	0.03°	0.01°	100%

Fuente: Barron y Beauchemin [51]

2.2. Cinemática

La cinemática de un robot debe ser entendida como una herramienta que permite describir la relación entre el movimiento de las articulaciones y el movimiento resultante de los cuerpos rigidos que componen al robot [34].

2.2.1. Preliminares

El estudio de la cinemática no es nada reciente, sin embargo, no fue sino hasta el año 1900 que Ball [48] introdujo una teoría formal sobre torsores, teoria que, hoy en día, es la base fundamental para el análisis y modelamiento moderno de robots.
Contrario a lo que se pueda pensar, los conceptos fundamentales, para la descripción del movimiento de un cuerpo rígido, que se presentan a continuación, no difieren en mayor medida de las representaciones y transformaciones geométricas utilizadas para el procesamiento de imágenes. Pues, el uso de coordenadas homogéneas y matrices de rotación es también de gran importancia en este estudio.

2.2.1.1. Transformaciones de un Cuerpo Rígido En este apartado, se entiende por un cuerpo rígido, a un cuerpo que es indeformable. Esta suposición, facilita el empleo de la geometría de torsores en la descripción del modelo de un robot, pues asume que las deformaciones que se producen sobre un elemento del sistema durante el movimiento son despreciables o nulas.

La locación de un partícula en un conjunto de ejes ortogonales, viene dada entonces por el conjunto de valores ordenados $(x, y, z) \in \mathbb{R}^3$, donde cada coordenada representa la proyección de la locación en cada uno de los ejes correspondientes. De esta forma, la trayectoria de una partícula puede ser representada por medio de la curva paramétrica $p(t) = (x(t), y(t), z(t)) \in \mathbb{R}^3$. Dada la consideración descrita anteriormente, un cuerpo rígido puede ser enendido más formalmente, como una colección de partículas de forma tal que, la distancia entre cada una de ellas permanece constante independientemente de los movimientos o fuerzas que se ejercen sobre el cuerpo. De esta forma si p y q son dos puntos cuales quiera de un cuerpo rígido, entonces:

$$\|p(t) - q(t)\| = \|p(0) - q(0)\| = K$$

Donde K es un valor real constante.

Dado un Objeto descrito como un subconjunto $O \in \mathbb{R}^3$, un movimiento rígido se puede representar como una familia de asignaciones de la forma $g(t) : O \longrightarrow \mathbb{R}^3$ que describe el movimiento relativo en función del tiempo de cada punto sobre el cuerpo alrededor de un sistema de ejes fijo. Por otra parte, un

desplazamiento rígido viene dado por una simple asignación $g : O \longrightarrow \mathbb{R}^3$. Las asignaciones o transformaciones descritas, pueden también ser extendidas para todo vector $v \in \mathbb{R}^3$ definido como un segmento de línea que une dos puntos p y q en el espacio, de forma tal que el vector de p a q viene dado por $v = q - p$. En este sentido el desplazamiento de un cuerpo rígido, esta definido como un vector que depende de los desplazamientos de p y q como:

$$g_*(v) = g(q) - g(p)$$

Bajo estas condiciones podemos definir una transformación de un cuerpo rígido como una asignación de la forma $\mathbb{R}^3 \longrightarrow \mathbb{R}^3$, y se escribe como sigue a continuación.

Definición 3 *Transformación de un Cuerpo Rígido.*

Una asignación $g : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ es una transformación de un cuerpo rígido si se cumple que:

1. $\|g(q) - g(p)\| = \|q - p\|$*, para todos los puntos* $q, p \in \mathbb{R}^3$
2. $g_*(v \times w) = g_*(v) \times g_*(w)$*, para todos los vectores* $v, w \in \mathbb{R}^3$

Teniendo en cuenta la *identidad de polarización*, donde para dos vectores v_1 y v_2:

$$v_1^T v_2 = \frac{1}{4}(\|v_1 + v_2\|^2 - \|v_1 - v_2\|^2)$$

y teniendo en cuenta que, por definición:

$$\|v_1 + v_2\| = \|g_*(v_1) + g_*(v_2)\|$$

y

$$\|v_1 - v_2\| = \|g_*(v_1) - g_*(v_2)\|$$

Se puede deducir que:

$$v_1^T v_2 = g_*(v_1)^T g_*(v_2)$$

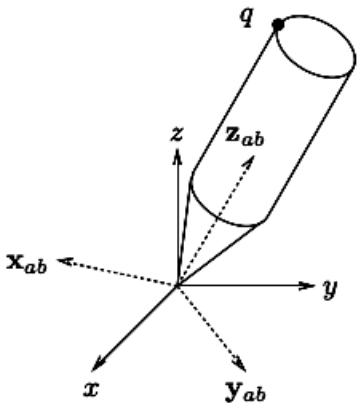
Esto implica que el producto punto y el producto cruz se preserva para toda transformación de un cuerpo rígido. Cabe aclarar, que el hecho de que estas propiedades se conserven, no implican que una partícula no pueda moverse de forma relativa respecto a otra, sino que puedan rotar más no transladarse relativamente entre si. Una consecuencia de esto, es la necesidad de unir un sistema de ejes coordenados a un punto y hacer un seguimiento del movimiento de este con respecto a un eje de coordenadas fijo. El movimiento individual de cada partícula puede ser abstraido nuevamente del movimiento del marco del cuerpo y del punto de unión entre el marco y este. De esta forma, si

la configuración de un cuerpo rígido esta dada por los vectores v_1, v_2 y v_3 atados o unidos al punto p, la configuración del cuerpo rígido luego de una transformación $g : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$, esta dada por los vectores $g_*(v_1)$, $g_*(v_2)$ y $g_*(v_3)$.

2.2.1.2. Movimiento Rotacional en $\mathbb{R}^3$ Como se expuso anteriormente, la descripción del movimiento relativo de las partículas en un mismo cuerpo, está bien definida para la rotación, pues el objetivo de la cinemática es el de describir el desplazamiento de un cuerpo rígido y la rotación del mismo alrededor de un punto. Por esta razón, el estudio del moviemto rotacional es esencial para un primer análisis.

Considérese la Figura mostrada a continuación:

Figura 9: Rotación de un cuerpo rígido alrededor de un punto



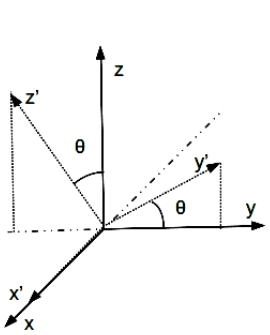
Fuente: Murray, Zexiang y Sastry [34]

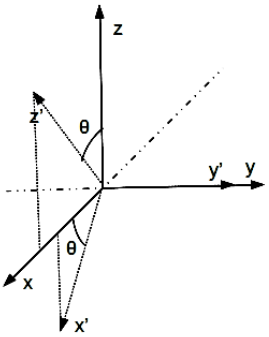
Sean A y B ejes coordenados ortogonales, donde A es el marco inercial del sistema y B el marco del cuerpo. Además, se tiene que $x_{ab}, y_{ab}, z_{ab} \in \mathbb{R}^3$ son los vectores de coordenadas principales del eje B relativos a A. La matriz de rotación $R_{3\times3} \in \mathbb{R}^{3\times3}$ de B con respecto a A viene dada por:

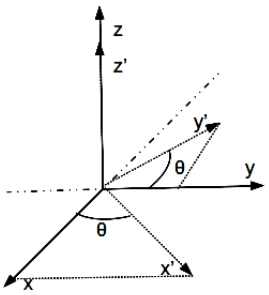
$$R = \begin{bmatrix} x_{ab} & y_{ab} & z_{ab} \end{bmatrix}$$

A continuación se describen las matrices de rotación más elementales, siendo θ el ángulo de rotación alrededor de una línea que atraviesa el eje x, y o z según el caso.

Cuadro 3: Matrices de Rotación Elementales







$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotación alrededor del eje X | Rotación alrededor del eje Y | Rotación alrededor del eje Z

Fuente: Elaboración Propia.

De estas matrices podemos observar dos propiedades que se pueden extender para cualquier matriz de rotación:

1. $RR^T = R^T R = I$
2. $\det(R) = 1$

Bajo estas condiciones se define entonces al conjunto $SO(n)$ como sigue:

$$SO(n) = \left\{ R \in \mathbb{R}^{n \times n} \mid RR^T = I,\ \det(R) = 1 \right\}$$

De forma particular, para $\mathbb{R}^3$, $SO(3)$ será el conjunto de matrices de (3×3) que satisfacen las propiedades ya mencionadas, en definitiva, este representa al grupo de matrices de rotación en $\mathbb{R}^3$.

Supongase ahora que, sea un punto q en el espacio, cuyas coordenadas respecto a un sistema de ejes coordenados B son $q_b = (x_b, y_b, z_b)$, se desea hallar las coordenadas del punto respecto a un sistema fijo de ejes coordenados A. De esta forma, dados los vectores x_{ab}, y_{ab}, z_{ab} de la matriz de rotación R_{ab} de B con respecto a A se tiene que:

$$q_a = \begin{bmatrix} x_{ab} & y_{ab} & z_{ab} \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = R_{ab} q_b$$

Este resultado se puede aplicar para definir la acción de una matriz de rotación sobre un vector, de tal forma que dado $v_b = q_b - p_b$, se tiene que:

$$R_{ab}(v_b) := R_{ab} q_b - R_{ab} p_b = q_a - p_a = v_a$$

Ahora bien, si se tiene un sistema de ejes coordenados C relativo a B, y a su vez B es relativo a A, se puede definir la matriz de rotación de C con respecto a A como:

$$R_{ac} = R_{ab}R_{bc}$$

En consecuencia de lo mostrado anteriormente, se puede deducir que la rotación corresponde a una transformación de un cuerpo rígido concorde con la Definición 3. La demostración formal de esta conclusión no se muestra en este documento, para mayor información el lector puede consultar en [34].

2.2.1.2.1. Coordenadas Exponenciales de Rotación Para la definición de este concepto en particular, se define como $\widehat{\omega} \in \mathbb{R}^{3\times 3}$ a una matriz de la forma:

$$\widehat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

Tal que, dado un vector de rotación $\omega = [\omega_1\ \omega_2\ \omega_3]^T$ y un vector arbitrario $\lambda \in \mathbb{R}^3$, $\omega \times \lambda = \widehat{\omega}\lambda$.

Las coordenadas exponenciales surgen de la necesidad de representar una matriz de rotación cualquiera $R \in SO(3)$, como una función de un vector de rotación unitario $\omega \in \mathbb{R}^3$ y un ángulo de desplazamiento $\theta \in \mathbb{R}$.
De manera general, se puede describir el movimiento a velocidad constante de una partícula en q alrededor de un eje de rotación ω en función del desplazamiento θ ($R(\omega, \theta)$) de la forma:

$$\dot{q}(\theta) = \widehat{\omega} \times q(\theta)$$

Resolviendo la ecuación diferencial, se obtiene:

$$q(\theta) = q(0)e^{\widehat{\omega}\theta}$$

Donde $e^{\widehat{\omega}\theta}$ es la matriz exponencial. Aplicando series de Taylor se tiene:

$$R(\omega, \theta) = e^{\widehat{\omega}\theta} = I + \widehat{\omega}\theta + \frac{(\widehat{\omega}\theta)^2}{2!} + \frac{(\widehat{\omega}\theta)^3}{3!} + \cdots \tag{4}$$

Dado que ω es un vector unitario que define el eje de rotación, la norma del vector esta dada por $\|\omega\| = \sqrt{\omega_1^2 + \omega_2^2 + \omega_3^2} = 1$, y se puede demostrar que $\widehat{\omega}^3 = -\|\omega\|\widehat{\omega}$. De forma iterativa se tiene que:

$$
\begin{aligned}
\widehat{\omega} &= \widehat{\omega} \\
\widehat{\omega}^2 &= \widehat{\omega}^2 \\
\widehat{\omega}^3 &= -\widehat{\omega} \\
\widehat{\omega}^4 &= -\widehat{\omega}^2 \\
\widehat{\omega}^5 &= \widehat{\omega} \\
\widehat{\omega}^6 &= \widehat{\omega}^2 \\
\widehat{\omega}^7 &= -\widehat{\omega} \\
&\vdots
\end{aligned}
$$

De tal forma que la ecuación 4 se puede reescribir como:

$$e^{\widehat{\omega}\theta} = I + \left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots\right)\widehat{\omega} + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} - \cdots\right)\widehat{\omega}^2$$

Se puede observar que el factor que multiplica a $\widehat{\omega}$, es la expansión en series de Taylor del $\sin\theta$, y el factor que multipla a ω^2 corresponde a la expansión de $1 - \cos\theta$. La ecuación queda de la forma:

$$e^{\widehat{\omega}\theta} = I + \widehat{\omega}\sin\theta + \widehat{\omega}^2(1 - \cos\theta) \tag{5}$$

Bajo esta definición, es mucho más fácil demostrar que la matriz de rotación de un punto relativo a un sistema de ejes coordenados fijo, equivale a la multiplicación de las matrices de rotación del punto respecto a otros sistemas. De esta forma si tenemos un punto q en un eje B rotado θ radianes, que a su vez esta rotado respecto a un sistema fijo A α radianes, la matriz exponencial de q se puede expresar como:

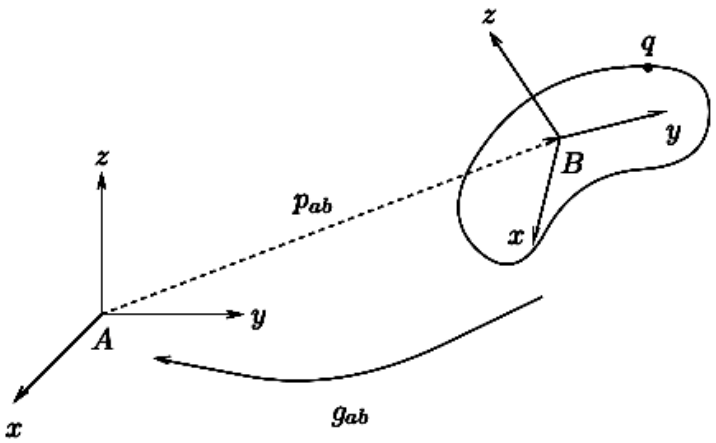
$$e^{\widehat{\omega_1}\theta + \widehat{\omega_2}\alpha} = e^{\widehat{\omega_1}\theta}e^{\widehat{\omega_2}\alpha} = R(\widehat{\omega_1}, \theta)R(\widehat{\omega_2}, \alpha)$$

2.2.1.3. Movimiento de un Cuerpo Rígido en $\mathbb{R}^3$ Si bien es cierto que la traslación de un punto no esta bien definida para partículas contenidas en un mismo cuerpo, esta transformación si se puede definir para describir la posición y orientación instantáneas de un sistema de un cuerpo atado a un sistema de coordenadas relativo a un sistema inercial.

En cinemática, el movimiento de un cuerpo rígido esta definido por la rotación del cuerpo alrededor de una línea seguida de la traslación del mismo sobre una línea paralela a esta, a esto se le conoce como movimiento torsor o movimiento rígido («screw motion»). En general, el movimiento de un cuerpo rígido esta definido por un movimiento de rotación y un movimiento de traslación.
Considérese la figura que se muestra a continuación:

Tomando en consideración la imágen, podemos definir el desplazamiento como una trayectoria $p(t) \in \mathbb{R}$, tal que $t \in [0, T]$. En la Figura 10, se desea

Figura 10: Marco de Coordenadas de un movimiento rígido



Fuente: Murray, Zexiang y Sastry [34]

hallar la orientación de un punto $q \in \mathbb{R}^3$ en el sistema de coordenas de B, teniendo en cuenta que B es relativo al sistema A, con $R_{ab} \in SO(3)$ la matriz de rotación de B a A y $p_{ab} \in \mathbb{R}^3$ el vector de posición del origen a A al origen de B. Se denota entonces como $SE(3)$ a un conjunto euclideano de la forma:

$$SE(3) = \left\{(p, R) \mid p \in \mathbb{R}^3\right\}$$

Este conjunto representa al conjunto de elementos (p, R) que sirven como descriptores de una transformación de un cuerpo rígido. Sean $q_a, q_b \in \mathbb{R}^3$ el punto q relativo a A y a B respectivamente. q_a se puede expresar como:

$$q_a = p_{ab} + R_{ab} q_b \tag{6}$$

Nótese, que la ecuación 6 corresponde a la ecuación 1 ya mostrada en la sección 2.1.1.2.2, y se denota de la misma forma en que se explica en esta misma sección.

2.2.2. Cinemática Directa

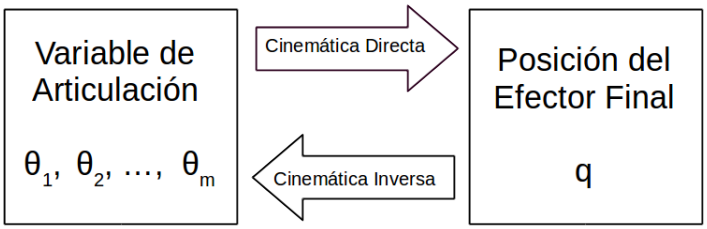
El espacio articular provee poca información sobre la posición y orientación del efector final de una cadena cinemática [64]. Con esto se hace referencia al hecho de que ubicar un partícula de un cuerpo rígido de interés en un espacio cartesiano local fijo de una cadena cinemática, requiere un análisis más profundo. La cinemática directa permite realizar un mapeo desde el espacio de la articulación del cuerpo a un espacio cartesiano local. Dada una cadena cinemática de m articulaciones, con ángulos de rotación $(\theta_1, \theta_2, \ldots, \theta_m)$, la cinemática directa permite encontrar la posición y orientación final de un efector.

2.2.3. Cinemática Inversa

De la misma forma, Kofinas [64] resalta el hecho de que un manipulador robótico necesita alcanzar puntos o seguir trayectorias en un espacio tridimensional. En este sentido, la cinemática inversa permite calcular o estimar los ángulos de rotación $(\theta_1, \theta_2, \ldots, \theta_m)$ a partir de la posición y orientación de un punto q en el espacio.

La Figura que se muestra a continuación representa de manera gráfica la forma en que la cinemática directa e inversa funcionan.

Figura 11: Cinemática Directa e Inversa



Fuente: Elaboración Propia.

3. Algoritmos para la Detección y el seguimiento de un Objeto

Los algoritmos utilizados y que se muestran en este apartado fueron implementados en C++ con el uso de *OpenCV*, librería ampliamente utilizada para el procesamiento de imágenes. *OpenCV* está enfocada principalmente al desarrollo de aplicaciones en tiempo real con un bajo costo computacional como se menciona en [65], razón por la cual su uso se ve completamente justificado al ser a su vez una plataforma de Software libre, característica tomada en cuenta en el marco del proyecto al estar este destinado a ser una plataforma de uso académico y de investigación.

3.1. Procesamiento de Imágenes

En la sección 2.1.3 se mencionaba que el Flujo Óptico es en esencia la cantidad de movimiento aparente de un pixel o un grupo de estos en una misma escena y, en este sentido, existen distintas técnicas para su estimación. Dichas técnicas permiten obtener el vector de movimiento, principal propiedad de interés para el desarrollo de este proyecto. Actualmente, la versión más reciente de *OpenCV* (3.2), cuenta con tres algoritmos para la estimación directa del Flujo Óptico; Lucas Kanade (**LK**), Farnebäck (**FB**) y Correlación de Fase (**PC**), sin embargo, se diseñó un cuarto algoritmo (**SB**) con la finalidad de obtener un mejor estudio comparativo en base a la aplicación directa sobre el robot InMoov. En este capítulo se describe el preproceso realizado para el tratamiento de las imágenes de entrada al sistema y se realiza una descripción del proceso de implementación de cada uno de estos algoritmos.

3.1.1. Preproceso

Como lo menciona Szeliski en [21], el primer paso en la mayoría de aplicaciones es el uso del procesamiento de las imágenes para preprocesar las imágenes, valga la redundancia, y de esta forma hacerlas idóneas para un análisis más profundo mediante el manejo de los factores que afectan la información contenida en la imagen tales como la luminosidad, la saturación, el ruido etc. Como se evidencia en el trabajo de Shi y Tomasi [53], uno de los factores esenciales para el uso del Flujo Óptico como técnica, es el manejo de la luz en la imagen, pues al no ser contralada adecuadamente, dificultará la tarea de estimar el movimiento. Esto se debe en gran parte al problema de apertura descrito en 2.1.3.1, que se aplica para los métodos basados en el gradiente espacial. En este sentido se realizó en primera instancia una transformación de la imágen al modelo XYZ, pues esta permite obtener información real sobre las cantidades necesarias para formar los colores de la imagen. Para un correcto análisis, como se verá en seguida, se hace la inclusión de un parámetro adicional s a la Ecuación 2, correspondiente a un factor de luminosidad

que se supone afecta de la misma forma a cada uno de los canales de la imagen.

Sea $I_{XYZ} = [a_{i,j}]$ la transformada XYZ de la imágen I_{RGB}, donde $[a_{i,j}]$ es la matriz cuyos miembros son los vectores de intensidades de cada pixel en la posición rectangular (i, j), entonces:

$$a_{i,j} = \begin{bmatrix} \frac{sR_{i,j}}{s(R_{i,j}+G_{i,j}+B_{i,j})} \\ \\ \frac{sG_{i,j}}{s(R_{i,j}+G_{i,j}+B_{i,j})} \\ \\ \frac{sB_{i,j}}{s(R_{i,j}+G_{i,j}+B_{i,j})} \end{bmatrix} \tag{7}$$

Donde s corresponde al factor de luminosidad y $R_{i,j}$, $G_{i,j}$ y $B_{i,j}$, corresponden a las intensidades del pixel en la posición (i, j) en cada uno de los canales de la imagen RGB (I_{RGB}). Como se puede apreciar, la transformada XYZ nos permite eliminar este factor s. Como consecuencia de esto, en cierto modo se obtendrá información directa sobre el color real de los objetos en la escena de la imagen.
Luego de esto, se aplica una normalización por blanco como se muestra en [66], donde se busca esencialmente normalizar la intensidad de color en la imagen multiplicando por el valor máximo admisible (255) y diviendo por la máxima intensidad de cada canal como se muestra:

$$b_{i,j} = \begin{bmatrix} \frac{255 \cdot X_{i,j}}{max(X)} \\ \\ \frac{255 \cdot Y_{i,j}}{max(Y)} \\ \\ \frac{255 \cdot Z_{i,j}}{max(Z)} \end{bmatrix} \tag{8}$$

donde $b_{i,j}$ es la intensidad de cada pixel luego de la normalización. Esto se realiza con el fin de obtener una mejor saturación del color en la imágen. Vale aclarar, que para realizar este proceso es necesario la inclusión de un elemento blanco en la escena, el cual servirá de referencia para obtener las intensidades reales del color de cada uno de los elementos inmersos en la imagen.

En la Figura 12 se muestra el resultado del preproceso para tres imágenes. En estas, si se observa detenidamente, se evidencia una mejoría en el contraste entre los pixeles cercanos a los contornos de la imagen, lo cual, como se menciona en [53], facilita el proceso para estimar el movimiento de un pixel. Además, como se puede observar en la imagen (b) de la misma Figura, cuando las intensidades de los pixeles son bajas el preproceso realzará la intensidad del color en la imagen y de esta forma permitirá trabajar con imágenes tomadas bajo condiciones de poca luminosidad.

Vale aclarar que cada uno de los algoritmos utilizados para calcular el Flujo Óptico, necesita de imágenes de un solo canal para poder realizar el debido proceso. De esta forma como parte del preproceso, es necesario transformar las imágenes obtenidas a Escala de Grises, es decir una imagen de un solo canal de 8 bits (256 Valores Posibles).

Figura 12: Resultados del Preproceso. Imágenes originales tomadas de la base de datos de Martin et al [67].

(a) Imagen Original 1 (b) Imagen Preprocesada 1

(c) Imagen Original 2 (d) Imagen Preprocesada 2

(e) Imagen Original 3 (f) Imagen Preprocesada 3

Fuente: Elaboración Propia.

3.1.2. Flujo Óptico

En este apartado se describe de manera más detallada los algoritmos de Lucas Kanade, Farnebäck, Correlación de Fase, y se realiza el diseño de un cuarto algoritmo **SB**. Además se muestran los resultados de su implementación mediante el uso de la base de datos provista por Baker et al [63] y mediante un experimento de validación se verifica su correcto funcionamiento y se selecciona el algoritmo a utilizar.

En los algoritmos siguientes las restas $I_2 - I_1$ e $I_3 - I_2$, es decir, las restas entre los frames 2 y 1 y 3 y 2 respectivamente, se utilizan como un paso más en el preprocesamiento de las imágenes para calcular el flujo óptico sobre un único objeto en movimiento. El sentido de incluir estas dos restas, se analiza de manerá más profunda en la sección que sigue a continuación.

3.1.2.1. SB Se diseño el algoritmo de substracción o **SB**, bajo el siguiente análisis. Sea $I_1(i,j)$ e $I_2(i,j)$ las intensidades de los pixeles en la pósición (i,j) del primer y segundo frame, correspondientes a la secuencia de imágenes de un objeto particular en movimiento en distintos instantes de tiempo respectivamente. Si se considera que solo existe un objeto en movimiento dentro de la escena y que la cámara permanece inmóvil durante la obtención de estas imágenes, entonces, siendo O el conjunto de las posiciones de los pixeles que componen al objeto en movimiento y $x = (i,j)$ la posición de un pixel en la imagen I_2:

$$x \in O \longleftrightarrow I_2(i,j) - I_1(i,j) \neq 0$$

De esta forma, $O = \{(i,j) \mid \mid I_2(i,j) - I_1(i,j) \mid > 0\}$. Nótese que O es un conjunto de pares ordenados. Esto indica básicamente que los únicos pixeles cuya intensidad se verá directamente afectada, serán aquellos que pertenezcan al objeto en movimiento. Sin embargo esta definición de O no es del todo acertada para aplicaciones reales, pues las condiciones de luz pueden variar de frame a frame ocasionando que existan pixeles que cumplan con esta condición sin pertenecer propiamente al objeto en movimiento. En este sentido, es necesario definir un parámetro δ correspondiente a un valor de umbral pequeño, suponiendo que las variaciones de luz son a su vez pequeñas. Así $O = \{(i,j) \mid \mid I_2(i,j) - I_1(i,j) \mid > \delta\}$.
Hasta este punto no es posible estimar el vector de movimiento del objeto, por esto es necesario la inclusión de una tercera imagen I_3 correspondiente a un tercer frame tomado unos instantes después de I_2.
Sea o_k el k-ésimo elemento de O, se define como O^c al centroide del objeto como sigue:

$$O^c = \frac{1}{n} \sum_{1}^{n} o_k$$

Donde n es el número de elementos del conjunto mencionado. Vale aclarar que o_k es un par ordenado de la forma (i, j) y O^c es igualmente un par ordenado correspondiente a una posición específica dentro de la imagen. Considérese ahora O_σ los pixeles del objeto en movimiento obtenidos a partir de I_1 e I_2 y O_η los pixeles del objeto en movimiento obtenidos a partir de I_2 e I_3. Por consiguiente el vector de movimiento $\vec{\nu}$ del objeto estará dado por:

$$\vec{\nu} = \overrightarrow{O^c_\sigma \; O^c_\eta} \tag{9}$$

Si $O^c_\sigma = (i_\sigma, j_\sigma)$ y $O^c_\eta = (i_\eta, j_\eta)$, entonces:

$$\mid \vec{\nu} \mid = \sqrt{(i_\sigma - i_\eta)^2 + (j_\sigma - j_\eta)^2} \tag{10}$$

y

$$\measuredangle \vec{\nu} = \arctan\left(\frac{j_\sigma - j_\eta}{i_\sigma - i_\eta}\right) \tag{11}$$

En este sentido, $\vec{\nu}$ es, de manera más general, el vector global de movimiento, pues corresponde al movimiento total del objeto y no al de cada uno de los pixeles del mismo.

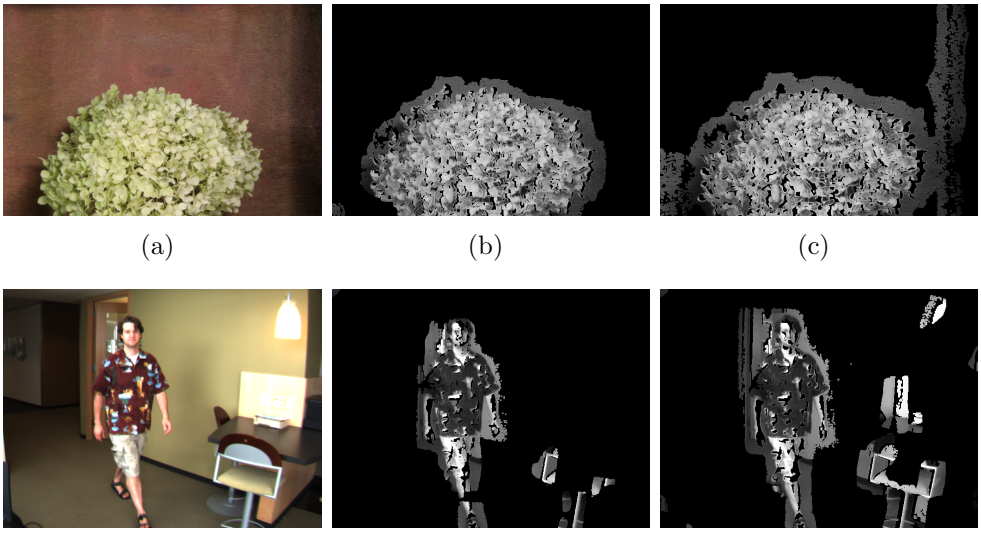
Cuadro 4: Pseudocódigo del Algoritmo de Substracción

Algoritmo SB
1. Declaración de Variables.
2. Adquisición de imágenes: I_1, I_2 e I_3.
3. Preprocesamiento de cada una de estas imágenes.
4. Obtención de $I_2 - I_1$ e $I_3 - I_2$.
5. Umbralización de $I_2 - I_1$ e $I_3 - I_2$ mediante el parámetro δ.
7. Obtención de O_σ y O_η.
6. Obtención de $\vec{\nu}$.

Fuente: Elaboración Propia.

Los resultados obtenidos se muestran a continuación:

Figura 13: Resultados de la substracción entre cada frame, implementados sobre la base de datos sunministrada por Baker et al [63]: Conjuntos Hydrangea y Walking. (a) y (d): Frames Iniciales, (b) y (e): Resultados de la Resta entre el segundo y primer frame y (c) y (f): Resultados de la Resta entre el tercer y segundo frame.

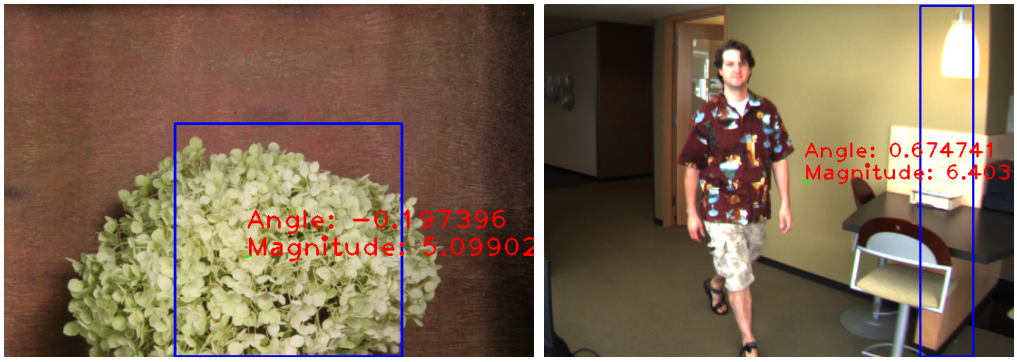


(a) (b) (c)

(d) (e) (f)

Fuente: Elaboración Propia.

Figura 14: Estimación del Movimiento por **SB** utlizando como referencia a las imágenes contenidas en la Figura 13.



(a) Hydrangea (b) Walking

Fuente: Elaboración Propia.

En la Figura 15 se muestra el modulo o la magnitud del vector de movimiento y el ángulo de este con respecto a la horizontal.

3.1.2.2. Lucas Kanade Esta técnica para el registro de imágenes fue descrita por primera vez en 1981 por Lucas y Kanade [52], el principal objetivo de esta es el de usar la información del gradiente espacial para realizar una busqueda directa de la posición de un mismo pixel en otra imagen. Con el trabajo de Shi y Tomasi [53], se lográ una mejor eficiencia y precisión del algoritmo, debido a la menor cantidad y mejor calidad de la información a procesar como lo mencionan Ortiz et al en [66].

Para la detección de movimiento, fue implementado el algoritmo de Lucas Kanade Piramidal descrito por Bouguet [56] que es una aplicación de la ecuación de gradiente espacial sobre un modelo piramidal de la imagen. A partir de la Ecuación 3 descrita en la sección 2.1.3, utlizando un modelo de pixeles vecinos se obtiene:

$$\begin{bmatrix} \sum J_i J_i & \sum J_i J_j \\ \sum J_i J_j & \sum J_j J_j \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} = - \begin{bmatrix} \sum J_i J_t \\ \sum J_j J_t \end{bmatrix} \tag{12}$$

Este sistema de ecuaciones permite encontrar los desplazamientos d_i y d_j de un pixel elimando así el problema de apertura. Junto con un modelo piramidal de la imagen, que basícamente consiste en proyectar los pixeles de esta sobre una nueva imagen proporcionalmente más pequeña, se consigue estimar movimientos de pixel más grandes, sin embargo el objetivo y la dinámica del algoritmo no se ven alterados.

En el Cuadro que se presentará a continuación, se encuentra contenido el pseudocódigo del algoritmo implementado, en este se denota como $LK(J, I)$ a la función de Lucas Kanade para estimar el movimiento entre dos imágenes J e I, imagen actual y previa respectivamente. Al igual que el algoritmo descrito previamente a este, **LK** permite hallar de manera directa los conjuntos de puntos O_σ, posiciones anteriores, y O_η, posiciones actuales, de los pixeles del objeto en movimiento y como consecuencia se puede hallar el vector global de movimiento del objeto $\vec{\nu}$.

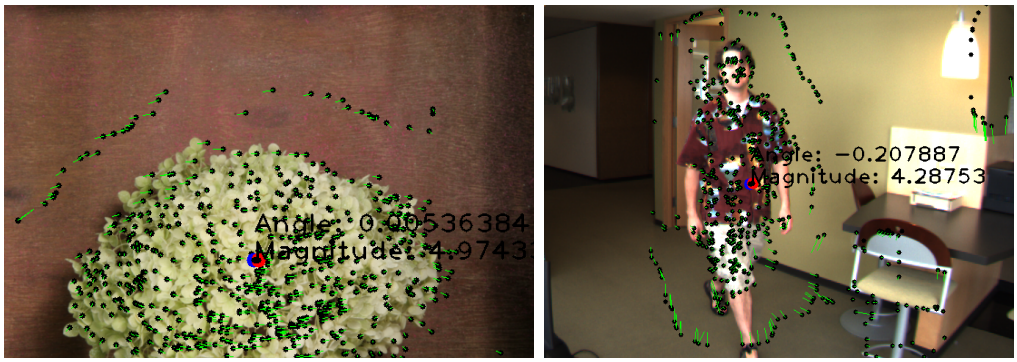
Cuadro 5: Pseudocódigo del Algoritmo de Lucas Kanade Piramidal

Algoritmo LK
1. Declaración de Variables.
2. Adquisición de Imágenes: I_1, I_2 e I_3.
3. Preprocesamiento de cada una de estas imágenes.
4. Obtención de $I = I_2 - I_1$ y $J = I_3 - I_2$.
5. $LK(J, I) \rightarrow O_\sigma$, O_η.
6. Obtención de $\vec{\nu}$.

Fuente: Elaboración Propia.

Los resultados obtenidos fueron los siguientes:

Figura 15: Estimación del Movimiento por **LK** utlizando la base de datos de Baker et al [63]: Conjuntos Hydrangea y Walking.



(a) Hydrangea (b) Walking

Fuente: Elaboración Propia.

3.1.2.3. Farnebäck Descrita por Farnebäck [61], esta técnica estima el movimiento utilizando una expansión polinomial, al igual que Lucas Kanade el propósito de esta es el mismo: encontrar la nueva locación de un pixel de interés.

Considere el modelo local de la señal de una imagen expresado en un sistema de coordenadas locales de la forma:

$$f(x) \sim x^T a x + b^T x + c$$

Donde a es una matriz, b un vector, y c un escalar. De acuerdo con esta definición la función de desplazamiento entre dos frames, $f_1(x)$ y $f_{2(x)}$ será:

$$f_2(x) = f_1(x - d) = x^T a_1 x + (b_1 - 2a_1 d)^T x + d^T a_1 d - b_1^T d + c_1$$

$$f_2(x) = x^T a_2 x + b_2^T x + c_2$$

Esto implica que:

$$a_2 = a_1 \tag{13}$$

$$b_2 = b_1 - 2a_1 d \tag{14}$$

$$c_2 = d^T a_1 d - b_1^T d + c_1 \tag{15}$$

Luego, si a_1 es una matriz no singular se puede resolver facilmente para d la Ecuación 14. A su vez, Farnebäck introduce la inclusión de un campo de desplazamiento *a priori* que en consecuencia, para una matriz a_1 singular,

completa la información necesaria para resolver la ecuación y encontrar los vectores de movimiento de cada punto. A diferencia de **LK** y **SB**, el algoritmo **FB** estima el movimiento de una matriz de puntos previamente definida sobre la imagen. A pesar de esto, contrario a lo que se podría pensar, la inclusión de estos nuevos puntos que no pertenecen al objeto de estudio, dada la substracción realizada previamente, el vector de movimiento de los mismos será nulo, por lo cual el resultado de las operaciones siguientes a realizar no se verá afectado.

En el siguiente cuadro se observa el pseudocódigo del algoritmo implementado, como se podrá evidenciar este es practicamente igual al del algoritmo **LK**, salvo que en este caso $FB(J, I)$ es la función de Farnebäck para estimar el movimiento entre dos imágenes y los elementos de los conjuntos O_σ y O_η no corresponderan en su totalidad a los pixeles del objeto en movimiento.

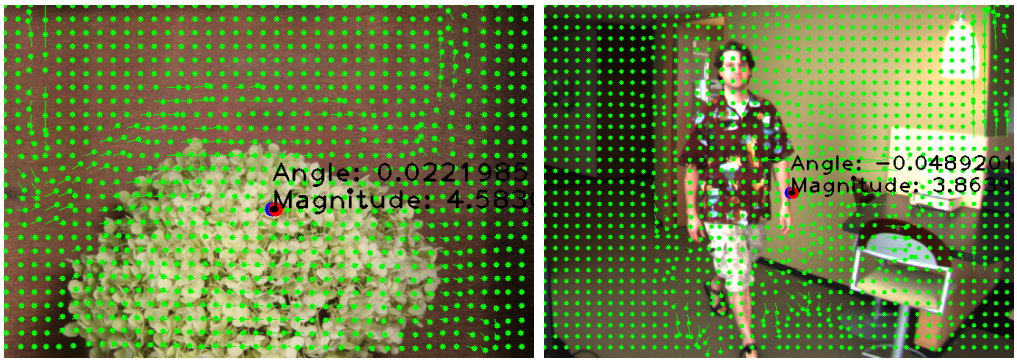
Cuadro 6: Pseudocódigo del Algoritmo de Farnebäck

Algoritmo FB
1. Declaración de Variables.
2. Adquisición de Imágenes: I_1, I_2 e I_3.
3. Preprocesamiento de cada una de estas imágenes.
4. Obtención de $I = I_2 - I_1$ y $J = I_3 - I_2$.
5. $FB(J, I) \rightarrow O_\sigma$, O_η.
6. Obtención de $\vec{\nu}$.

Fuente: Elaboración Propia.

Para este algoritmo se obtuvieron los siguientes resultados:

Figura 16: Estimación del Movimiento por **FB** utlizando la base de datos de Baker et al [63]: Conjuntos Hydrangea y Walking.



(a) Hydrangea (b) Walking

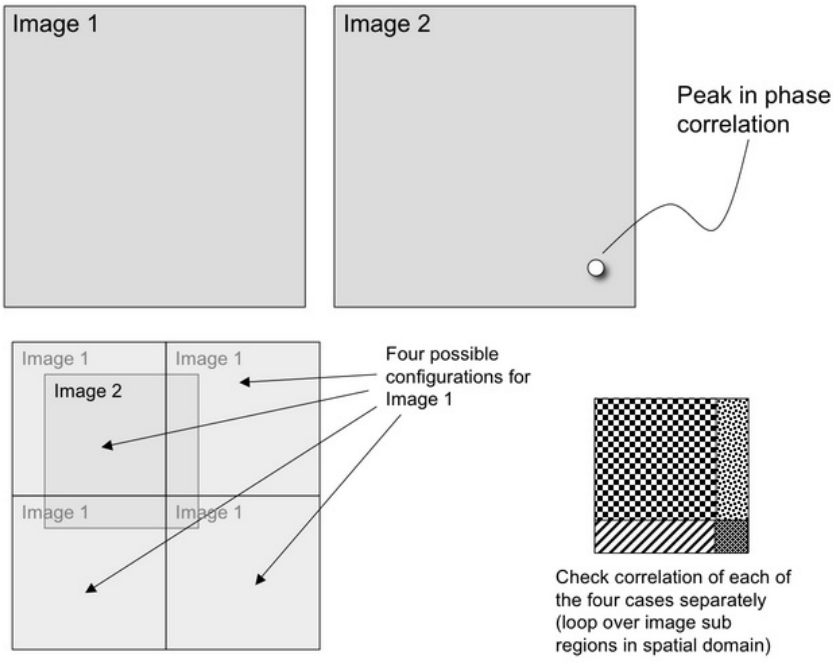
Fuente: Elaboración Propia.

3.1.2.4. Correlación de Fase Descrita por primera vez por De Castro y Morandi [69] para el registro de traslaciones y rotaciones de una imagen es hoy en un día un método ampliamente utilizado para estimar el movimiento, esto como consecuencia de su resistencia al ruido y las oclusiones al ser un método basado en el dominio de la frecuencia a diferencia de los métodos convencionales que trabajan en el dominio espacial. Lo que se busca con esta técnica es, por medio de una correlación en frecuencia entre dos imágenes, encontrar la locación de los picos de intensidades donde las imágenes coincidan mejor. Como se menciona en [68], se puede intuir que la imagen resultante de la correlación tendrá sus valores máximos en las locaciones donde estas coincidan. La correlación de fase entre dos imágenes dadas F y G se define como:

$$p(x, y) = \mathbb{F}^{-1} \left[\frac{F^*(u, v)G(u, v)}{|F^*(u, v)G(u, v)|} \right] \tag{16}$$

Donde $\mathbb{F}^{-1}$ es el operador correspondiente a la transformada inversa de Fourier, y * es el operador correspondiente al conjugado complejo de la función. En la siguiente Figura se explica de manera gráfica el funcionamiento del algoritmo.

Figura 17: Esquema gráfico del Algoritmo **PC**



Fuente: [68]

En la Figura ya mostrada, se indica que primero se halla el pico máximo en la imagen resultante de la correlación entre las imágenes 1 y 2. En este punto

existen cuatro orientaciones posibles de la imagen 1 con respecto a la 2, por consiguiente, de forma iterativa, se evalúa la correlación de las cuatro fases que dividen a la segunda imagen con respecto a la primera imagen hasta encontrar la orientación adecuada.

A continuación se presenta el pseudocódigo del algoritmo implementado, en este caso la función $PC(J, I)$ arroja un único resultado correspondiente al vector global de movimiento $\vec{\nu}$ por lo que su calculo es prescindible para esta técnica en particular.

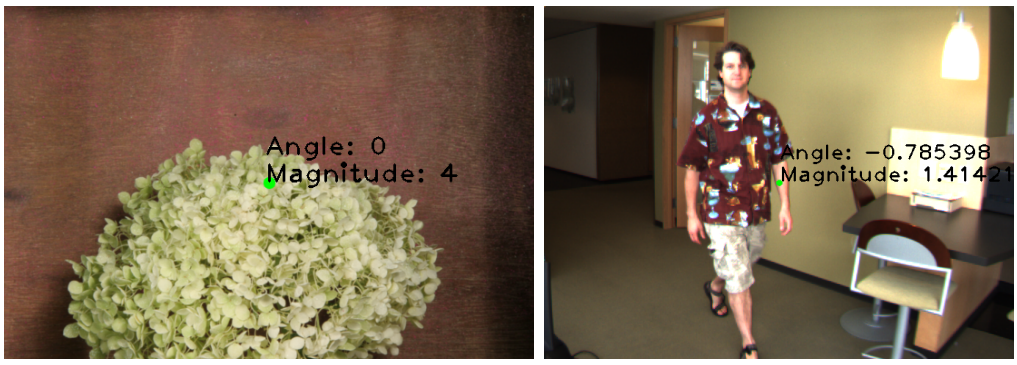
Cuadro 7: Pseudocódigo del Algoritmo de Correlación de Fase

Algoritmo PC
1. Declaración de Variables.
2. Adquisición de Imágenes: I_1, I_2 e I_3.
3. Preprocesamiento de cada una de estas imágenes.
4. Obtención de $I = I_2 - I_1$ y $J = I_3 - I_2$.
5. $\vec{\nu} = FB(J, I)$.

Fuente: Elaboración Propia.

Los resultados obtenidos fueron los siguientes:

Figura 18: Estimación del Movimiento por **PC** utlizando la base de datos de Baker et al [63]: Conjuntos Hydrangea y Walking.



(a) Hydrangea (b) Walking

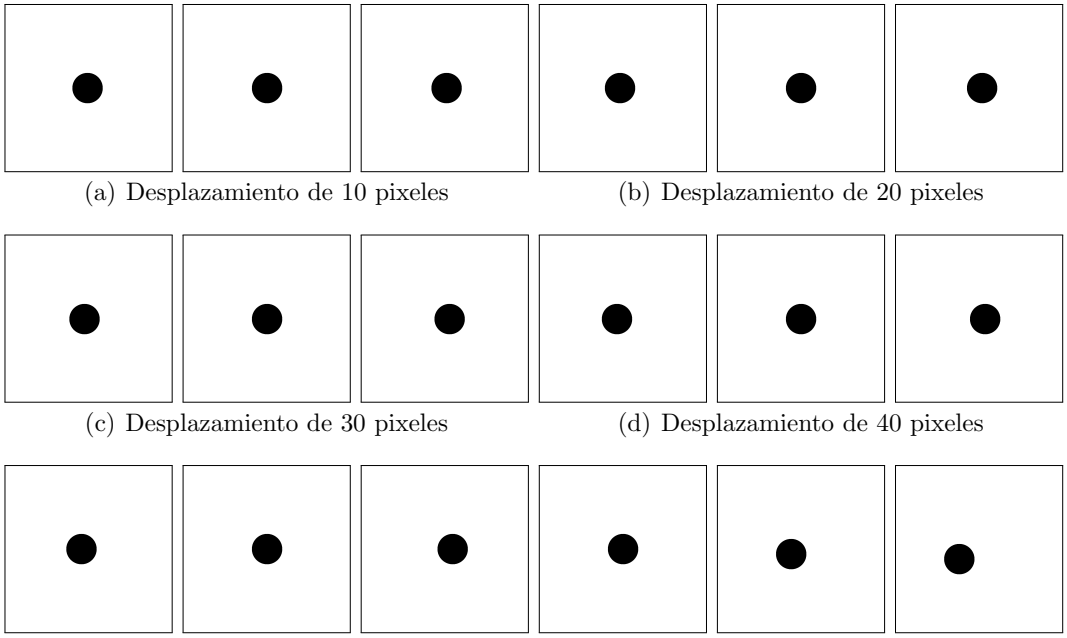
Fuente: Elaboración Propia.

3.2. Validación de Resultados

Para verificar el correcto funcionamiento de cada algoritmo se realizó un experimento de validación. Este consiste en crear una secuencia de tres frames con un desplazamiento constante preestablecido para luego calcular el error

entre el vector global de movimiento calculado por el algoritmo y el ya previamente definido.

Figura 19: Frames de Validación



(a) Desplazamiento de 10 pixeles (b) Desplazamiento de 20 pixeles

(c) Desplazamiento de 30 pixeles (d) Desplazamiento de 40 pixeles

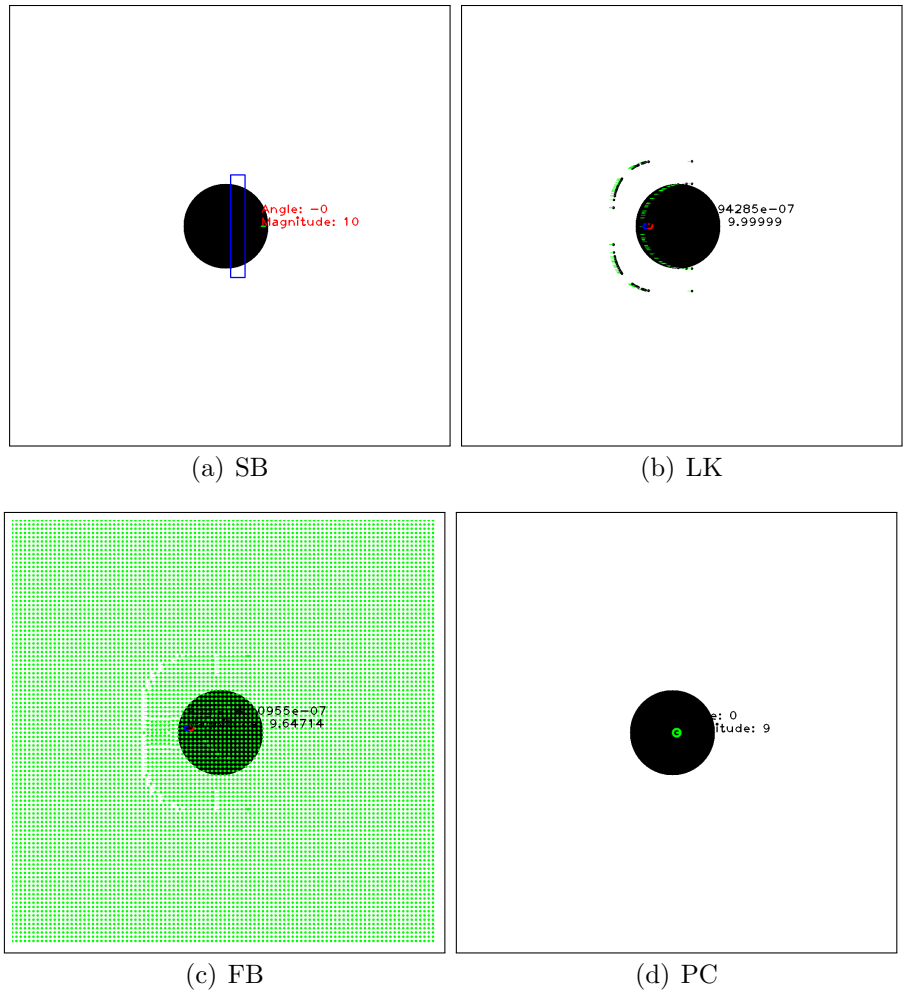
(e) Desplazamiento de 50 pixeles (f) Desplazamiento de 65 pixeles

Fuente: Elaboración Propia.

En la Figura 19 se muestran los frames creados para desplazamientos definidos de 10, 20, 30, 40, 50 y 65 pixeles de un círculo negro. Los valores mencionados corresponden a los módulos del vector de movimiento y los errores fueron calculados tomando como valor de referencia este parámetro.

Los resultados para cada uno de los algoritmos se muestran en las páginas siguientes.

Figura 20: Validación para un desplazamiento horizontal de 10 pixeles



(a) SB (b) LK

(c) FB (d) PC

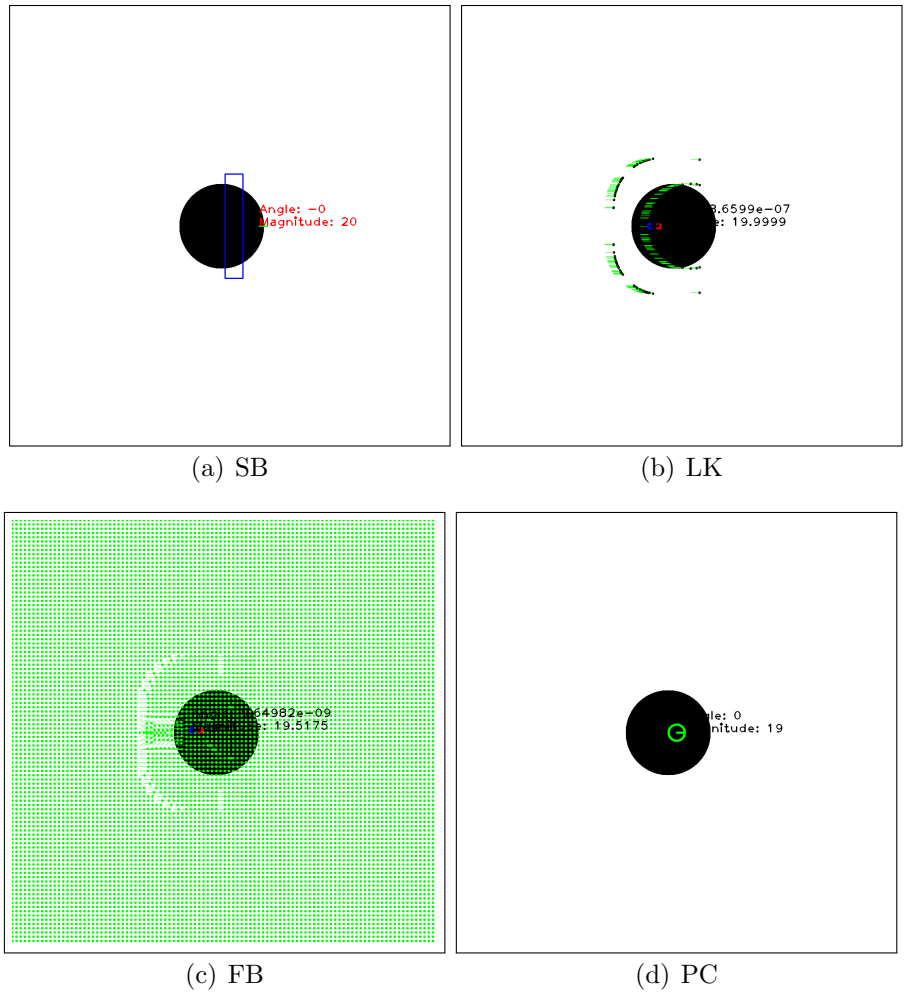
Fuente: Elaboración Propia

Cuadro 8: Errores calculados para un desplazamiento horizontal de 10 pixeles

Algoritmo	**Módulo del Vector de Movimiento Calculado (pixeles)**	**Error %**
SB	10.00	0.000
LK	9.999	0.010
FB	9.647	3.530
PC	9.000	10.00

Fuente: Elaboración Propia.

Figura 21: Validación para un desplazamiento horizontal de 20 pixeles



(a) SB (b) LK

(c) FB (d) PC

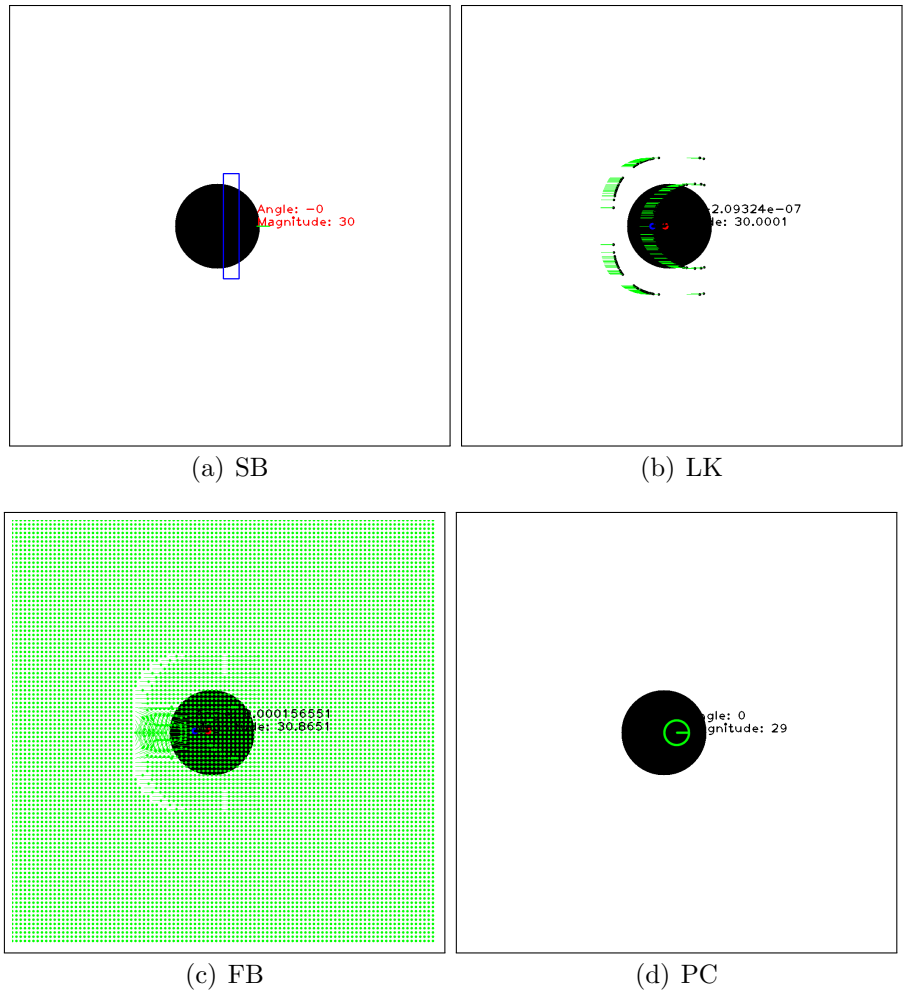
Fuente: Elaboración Propia

Cuadro 9: Errores calculados para un desplazamiento horizontal de 20 pixeles

Algoritmo	**Módulo del Vector de Movimiento Calculado (pixeles)**	**Error %**
SB	20.000	0.000
LK	19.999	0.005
FB	19.517	2.415
PC	19.000	5.000

Fuente: Elaboración Propia.

Figura 22: Validación para un desplazamiento horizontal de 30 pixeles



(a) SB (b) LK

(c) FB (d) PC

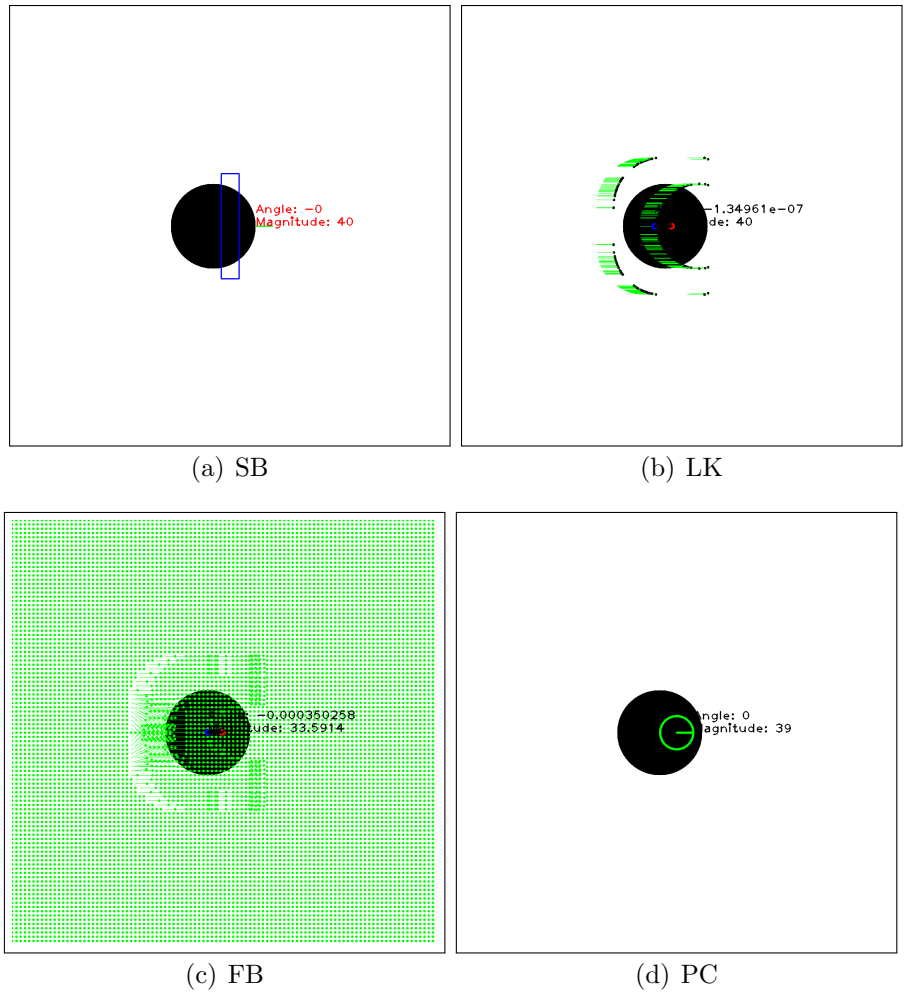
Fuente: Elaboración Propia

Cuadro 10: Errores calculados para un desplazamiento horizontal de 30 pixeles

Algoritmo	**Módulo del Vector de Movimiento Calculado (pixeles)**	**Error %**
SB	30.000	0.000
LK	30.000	0.000
FB	30.865	2.883
PC	29.000	3.333

Fuente: Elaboración Propia.

Figura 23: Validación para un desplazamiento horizontal de 40 pixeles



(a) SB (b) LK

(c) FB (d) PC

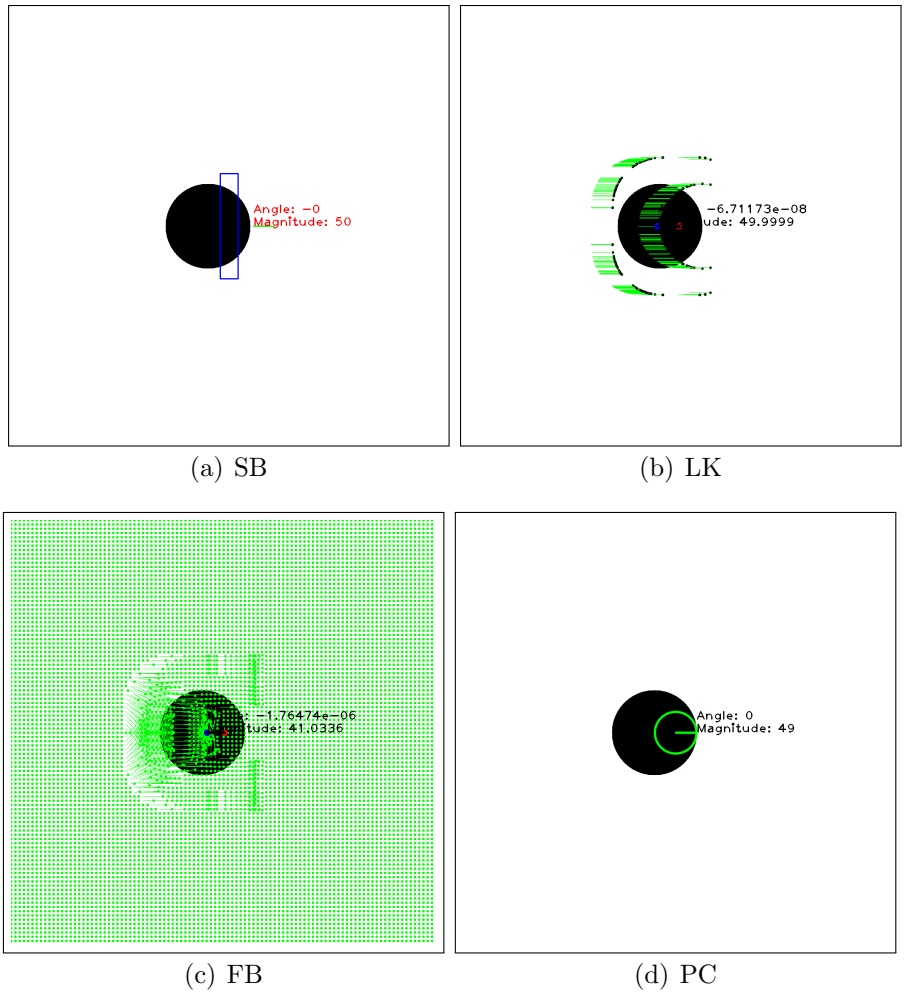
Fuente: Elaboración Propia

Cuadro 11: Errores calculados para un desplazamiento horizontal de 40 pixeles

Algoritmo	**Módulo del Vector de Movimiento Calculado (pixeles)**	**Error %**
SB	40.000	0.000
LK	40.000	0.000
FB	33.591	16.02
PC	39.000	2.500

Fuente: Elaboración Propia.

Figura 24: Validación para un desplazamiento horizontal de 50 pixeles



(a) SB (b) LK

(c) FB (d) PC

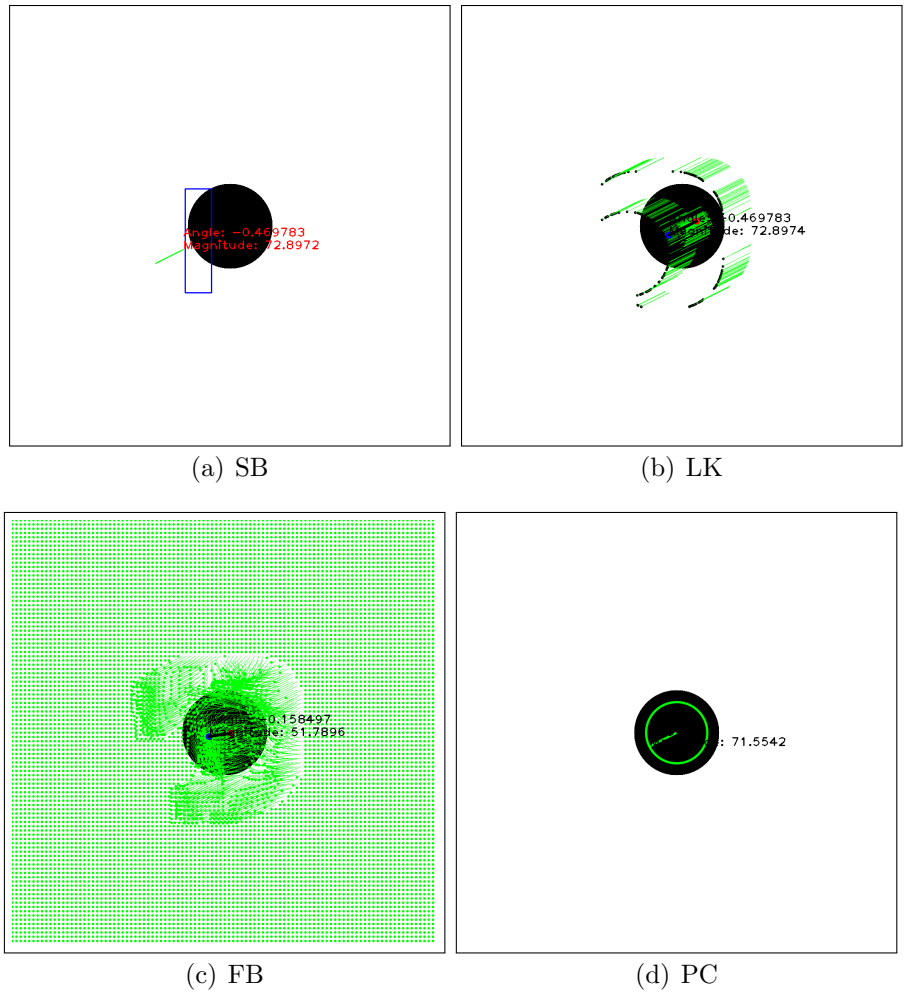
Fuente: Elaboración Propia

Cuadro 12: Errores calculados para un desplazamiento horizontal de 50 pixeles

Algoritmo	**Módulo del Vector de Movimiento Calculado (pixeles)**	**Error %**
SB	50.000	0.000
LK	49.999	0.002
FB	41.033	17.93
PC	49.000	2.000

Fuente: Elaboración Propia.

Figura 25: Validación para un desplazamiento horizontal de 65 pixeles



(a) SB (b) LK

(c) FB (d) PC

Fuente: Elaboración Propia

Cuadro 13: Errores calculados para un desplazamiento horizontal de 65 pixeles

Algoritmo	**Módulo del Vector de Movimiento Calculado (pixeles)**	**Error %**
SB	72.897	12.149
LK	72.897	12.149
FB	51.789	20.324
PC	71.554	10.083

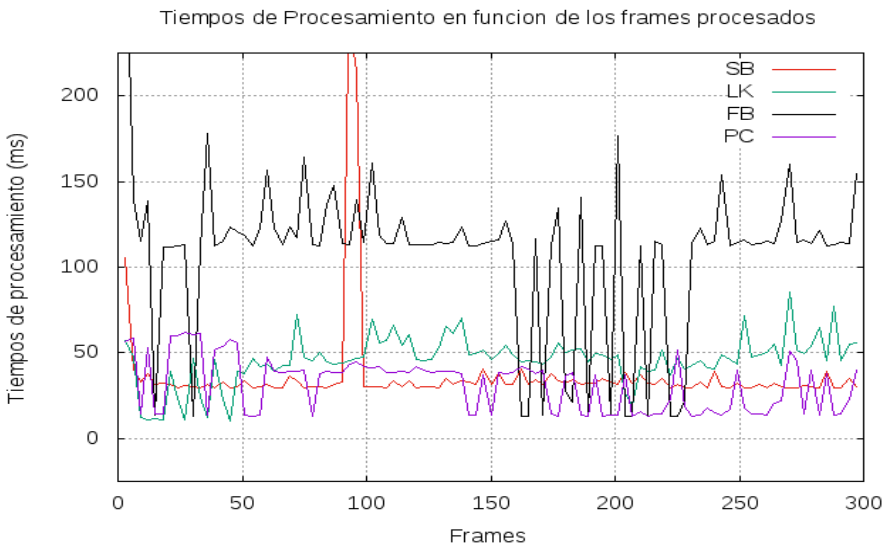
Fuente: Elaboración Propia.

En primera instancia, se puede observar que para desplazamientos mayores a 40 pixeles aproximadamente, el algoritmo **FB** pierde precisión con respecto a los demás algoritmos, y ante desplazamientos de no más de 65 pixeles, el error de los algoritmos **SB**, **LK** y **PC** no sobrepasa lo que podríamos considerar un error admisible de menos del 15 %. También se puede observar que el algoritmo **SB** fue el más preciso en casí la totalidad de frames de validación. Por último, a manera de análisis, podemos observar que los errores de los algoritmos **LK** y **FB**, algoritmos basados en el dominio espacial, parecen decrecer a medida que se incrementa el desplazamiento hasta un valor límite, para **FB** de 40 pixeles, donde empiezan a incrementar nuevamente. Esto podría indicar que existe un valor óptimo de desplazamiento de frame a frame para cada algoritmo, que a su vez podría ser controlado mediante la variación de los tiempos entre la captura de una imagen y otra. No Obstante, en el documento presente no se estudiará a profundidad esta observación.

3.2.1. Costo Computacional

El expermiento de Validación que se describió, pretendía evaluar la precisión de cada uno de los algoritmos implementados. Sin emabargo, dado que su aplicación deberá ser en tiempo real, es necesario evaluar el rendimiento de cada uno de estos. Para esto se tomarón los tiempos de procesamiento de 3 frames, durante una prueba de 300 frames y se realizó una gráfica del tiempo de procesamiento en un función de los frames procesados. Las pruebas fueron realizadas en un procesador Intel® Core^{TM} i5-5200U. De esta forma se obtubo la siguiente gráfica:

Figura 26



Fuente: Elaboración Propia.

Conforme a los resultados obtenidos, se encontraron los tiempos de procesamiento promedio basdos en la Figura 26 y se anexaron en el Cuadro que se muestra a continuación.

Cuadro 14: Tiempos de Procesamiento Promedio para cada Algoritmo

Algoritmo	**Tiempo de Procesamiento promedio (ms)**
SB	36.920
LK	45.833
FB	107.932
PC	31.620

Fuente: Elaboración Propia.

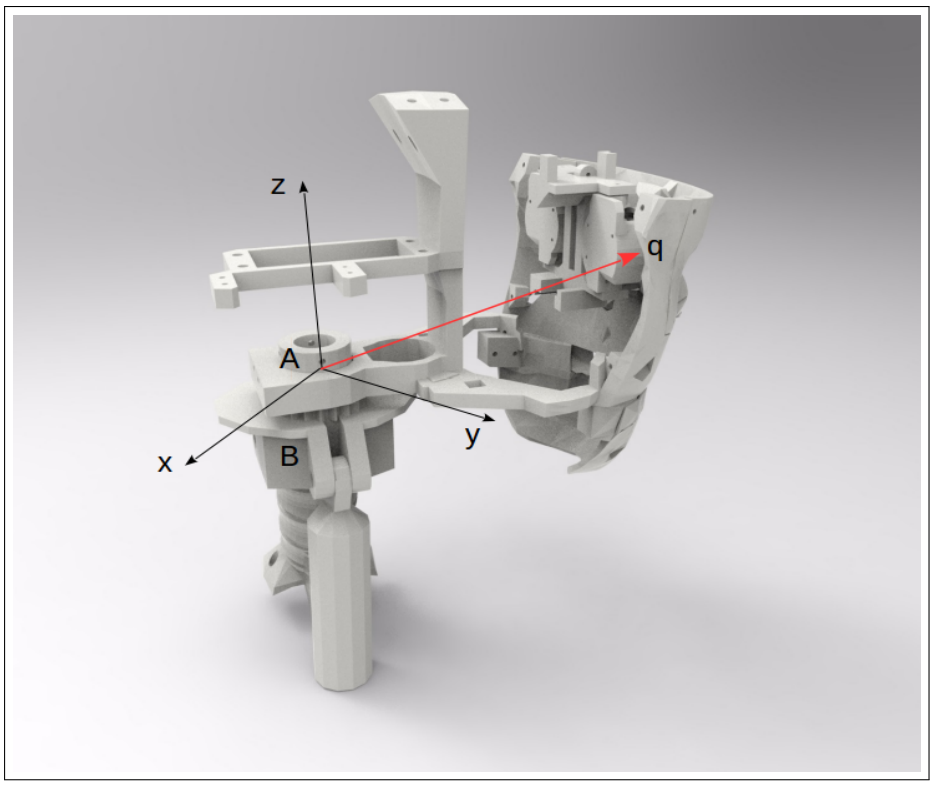
Se puede ver que los tiempos de procesamiento para los algoritmos **SB** y **PC** son los más bajos, y a su vez estos algoritmos, son relativamente más estables, es decir que para cualquier secuencia de tres frames el tiempo de procesamiento siempre será aproximadamente el mismo, a diferencia de **LK** por ejemplo, donde los tiempos de procesamiento oscilan considerablemente dependiendo de las características de los frames a procesar. Se puede concluir que el algoritmo que mejor desempeño obtuvo, teniendo en cuenta tanto la precisión como el costo computacional del mismo, fue **SB**.

Vale aclarar que el rendimiento de los algoritmos se evaluó en tiempos de procesamiento con la finalidad de obtener la velocidad máxima que detectará el sistema. En base a los resultados obtenidos, tomando como referencia el tiempo de procesamiento promedio del algoritmo seleccionado, **SB**, la velocidad límite de detección de movimiento será de aproximadamente 1760 pixeles por segundo. En las secciones siguientes se discutirá el significado físico de esta velocidad para su correcto análisis con respecto a los parámetros del sistema.

4. Modelo Cinemático

En este documento se toma en cuenta única y exclusivamente el movimiento de la cabeza del robot, que se obtiene a partir del mecanismo que se muestra a continuación:

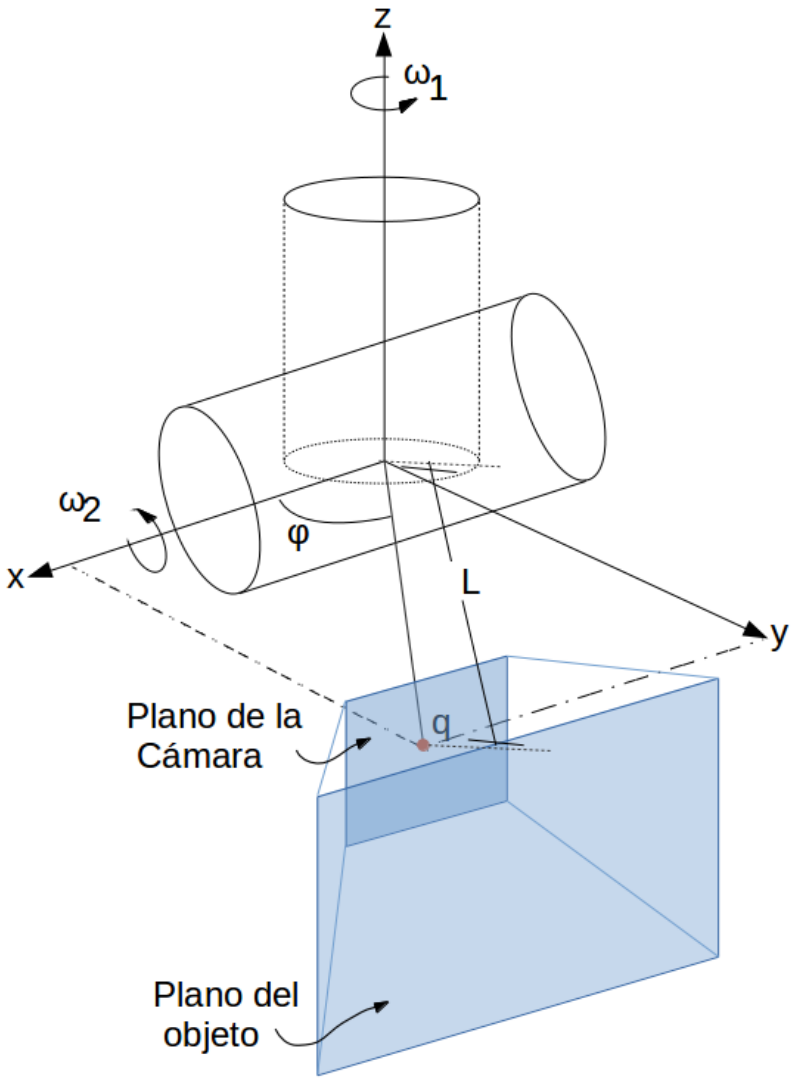
Figura 27: Mecanismo para el Movimiento de la cabeza del robot InMoov



Fuente: Elaboración Propia.

Este mecanismo es equivalente a la cadena cinemática que se muestra en la Figura 28. Respecto a esta se define el sistema coordenado B que rota a razón de ω_1 y se encuentra restringido al movimiento de un sistema A, que a su vez rota a razón de ω_2. Se observa que q es la posición de la cámara con respecto al mecanismo, cabe resaltar que la posción de la misma esta sujeta al movimiento del sistema B.

Figura 28: Cadena Cinemática de la Cabeza del Robot InMoov



Fuente: Elaboración Propia.

4.1. Cinemática Directa

En esencia el desarrollo de la cinemática directa, en este caso, busca encontrar la nueva posición del punto q (Locación del plano de la cámara) luego de una rotación en los sistemas de coordenadas A y B de θ_2 y θ_1 respectivamente. Por definición (Sección 2.2.1.2.1), la cinemática directa respecto al punto q de la cabeza del robot será equivalente a:

$$q_o = e^{\widehat{\omega_2}\theta_2} e^{\widehat{\omega_1}\theta_1} q_B$$

Donde q_o corresponde a la posición del punto q con respecto al origen y q_B a la posición del punto con respecto a B. De la Figura 28 se puede deducir que:

$$q_B = \begin{bmatrix} L\cos\varphi \\ L\sin\varphi \\ 0 \end{bmatrix}$$

Notese que q_B siempre será constante.

$$\omega_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad\qquad \omega_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

De esta forma las matrices exponenciales de rotación quedarán de la forma:

$$e^{\widehat{\omega_2}\theta_2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & -\sin\theta_2 \\ 0 & \sin\theta_2 & \cos\theta_2 \end{bmatrix}$$

$$e^{\widehat{\omega_1}\theta_1} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Por consiguiente, se resuelve para q_o:

$$q_o = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & -\sin\theta_2 \\ 0 & \sin\theta_2 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L\cos\varphi \\ L\sin\varphi \\ 0 \end{bmatrix}$$

$$q_o = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \cos\theta_2\sin\theta_1 & \cos\theta_2\cos\theta_1 & -\sin\theta_2 \\ \sin\theta_2\sin\theta_1 & \sin\theta_2\cos\theta_1 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} L\cos\varphi \\ L\sin\varphi \\ 0 \end{bmatrix}$$

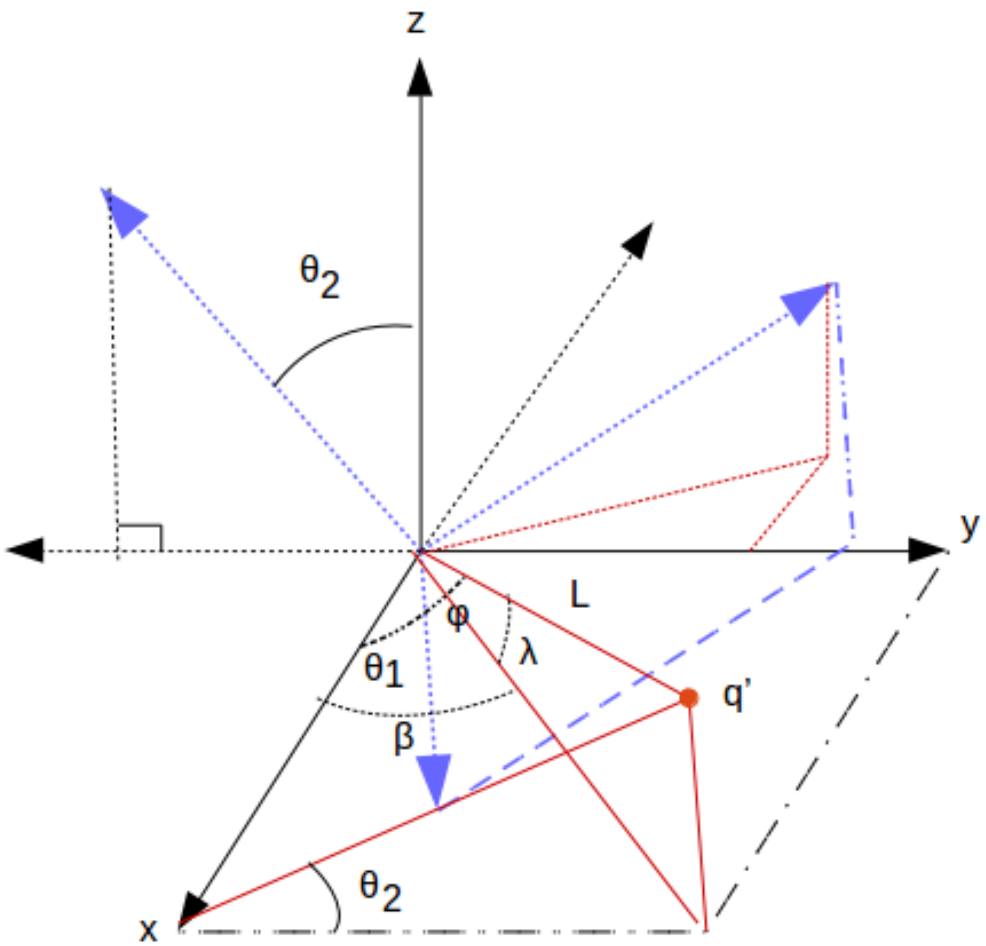
$$q_o = \begin{bmatrix} L\cos\theta_1\cos\varphi - L\sin\theta_1\sin\varphi \\ L\cos\varphi\cos\theta_2\sin\theta_1 + L\sin\varphi\cos\theta_2\cos\theta_1 \\ L\cos\varphi\sin\theta_2\sin\theta_1 + L\sin\varphi\sin\theta_2\cos\theta_1 \end{bmatrix}$$

$$q_o = \begin{bmatrix} L\cos(\varphi+\theta_1) \\ L\cos\theta_2\sin(\varphi+\theta_1) \\ L\sin\theta_2\sin(\varphi+\theta_1) \end{bmatrix} \tag{17}$$

4.2. Cinemática Inversa

Análogamente a la cinemática directa, la cinemática inversa busca ahora encontrar los ángulos de rotación θ_1 y θ_2, dada la posición de q con respecto al origen q_o, en este caso, la nueva posición de q será la posición que debe alcanzar el sistema para seguir continuar el seguimiento de un objeto. Esta se resuelve fácilmente de manera análitica, véase la siguiente Figura:

Figura 29: Posición del punto q luego de rotar A y B, θ_2 y θ_1, diferentes de cero, respectivamente



Fuente: Elaboración Propia.

En una primera instancia se sabe que la posición de q con respecto al origen esta dada por:

$$q_o = \begin{bmatrix} L\cos\lambda\cos\beta \\ L\cos\lambda\sin\beta \\ L\sin\lambda \end{bmatrix} \tag{18}$$

Donde los ángulos β y λ se suponen conocidos. Mediante la Figura 29 se deduce que:

$$\cos\lambda\cos\beta = \cos(\varphi + \theta_1)$$
$$\theta_1 = \arccos(\cos\lambda\cos\beta) - \varphi \tag{19}$$

$$\sin\lambda = \sin(\varphi + \theta_1)\sin\theta_2$$
$$\theta_2 = \arcsin\left(\frac{\sin\lambda}{\sin(\varphi + \theta_1)}\right) \tag{20}$$

De la Figura mencionada se puede deducir a su vez que:

$$\sin\beta\cos\lambda = \sin(\varphi + \theta_1)\cos\theta_2$$

Ecuación que junto con las Ecuaciones 19 y 20 demuestran de forma directa que la Ecuación 18 es equivalente a la Ecuación 17 para los ángulos θ_1 y θ_2 hallados en términos de λ y β, esto verifica y le da validez al modelo utlizado.

4.3. Restricciones

Las Restricciones Mecánicas del robot estarán sujetas al diseño del mismo y a las Ecuaciones 19 y 20. De estas podemos observar que para que la Ecuación 20 tenga validez se debe cumplir que:

$$\sin(\varphi + \theta_1) \neq 0$$

Entonces:

$$\theta_1 \neq -\varphi \tag{21}$$

Mediante la Figura 29, se puede intuir que $\theta_1 = -\varphi$ si:

$$q_o = \begin{bmatrix} L \\ 0 \\ 0 \end{bmatrix}$$

Esto quiere decir, por medio de la Ecuación 17, que para los valores de λ y β tales que $\theta_1 = -\varphi$, las ecuaciones 19 y 20 se cumplen para cualquier $\theta_2 \in \mathbb{R}$. De esta forma la Ecuación 21 deja de ser una restricción si se define a θ_2 como sigue:

$$\theta_2 = \begin{cases} \arccos\left(\frac{\cos\lambda \sin\beta}{\sin(\varphi+\theta_1)}\right) & \theta_1 \neq \varphi \\ \\ 0 & \theta_1 = \varphi \end{cases} \tag{22}$$

Las restricciones propias del diseño del robot (véase [77]), se listan a continuación:

$$\theta_1 \in [20^\circ, 160^\circ] \tag{23}$$

$$\theta_2 \in [30^\circ, 150^\circ] \tag{24}$$

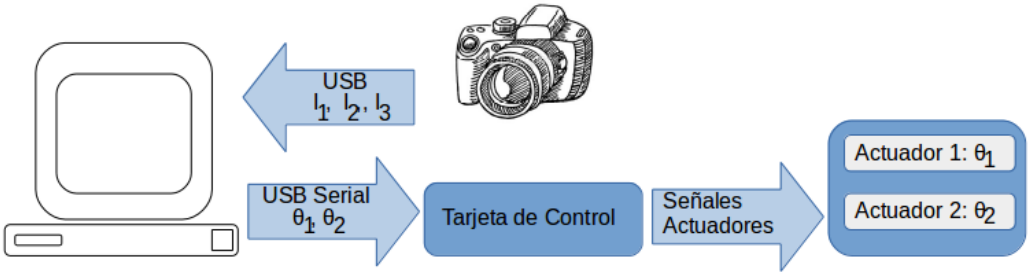
Estos valores corresponden de manera directa a las restricciones de los actuadores que permiten el movimiento de la cabeza [77], es decir que, para la implementación de los algoritmos, luego de haber hallado θ_1 y θ_2 los actuadores deberán a su vez girar un ángulo θ_1 y θ_2 según sea el caso.

Las restricciones correspondientes a la visión del robot no fueron tomadas en cuenta, pues estas surgen del mecanismo correspondiente al movimiento de los ojos del robot el cual no fue contemplado en el desarrollo de este proyecto. Para mayor información el lector puede consultar en [66] y en [77].

5. Instrumentación

Para la correcta selección de la instrumentación se tuvo en cuenta el siguiente esquema de control propuesto:

Figura 30: Esquema de Control del Sistema



Fuente: Elaboración Propia.

En la Figura 30, el computador recibe las imágenes provenientes de la cámara del robot, este mismo realiza el procesamiento correspondiente para obtener el flujo óptico y, del vector movimiento obtenido junto con la cinemática directa, obtiene los ángulos θ_1 y θ_2. Este proceso se describe mas detalladamente en la sección 6. Luego de esto, se envian los datos, vía USB Serial, a la tarjeta de control, la cual produce las señales correspondientes para cada uno de los actuadores.

Se utilizó una Camara Web convencional con una resolución SVGA 800x600. En este apartado no se describirá la selección de la misma, pues las referencias convencionales que existen en el mercado no difieren en mayor medida las unas de las otras. Asi mismo, no se contemplo la posibilidad de utilizar una cámara con una resolución mayor con el objetivo de no comprometer el rendimiento del Algoritmo a utilizar.

5.1. Actuadores Mecánicos

Dado que los datos que se envían a la tarjeta de control corresponden a los ángulos de movimiento del mecanismo de la cabeza (véase Figura 28), se decidió utilizar Servomotores para controlar directamente la posición de la cámara del robot con respecto al objeto en movimiento.

Respecto a los planos del robot contenidos en el Anexo B y dado que en los alcances del proyecto no se contempla la modificación del diseño mecánico del robot provisto en [70], se eligierón dos referencias cuyas medidas se ajustan

Cuadro 15: Características Técnicas de los Servomotores HS-805BB y Savox SV-0235MG

ITEM	HS-805BB	Savox SV-0235MG
		
SISTEMA DE CONTROL	Modulación por ancho de pulso 1500 Useg	Modulación por ancho de pulso 1500 Useg
VOLTAJE DE OPERACIÓN	4.8 - 6 V	6 - 7.4 V
RANGO DE TEMPERATURA	-20° - +60°	-20° - +60°
VELOCIDAD DE OPERACIÓN	0.16 Seg/60°	0.16 Seg/60°
TORQUE DE PARADA	$19{,}8Kg \cdot cm$	$28Kg \cdot cm$
ÁNGULO DE OPERACIÓN	45° A 400 Useg	45° A 400 Useg
CORRIENTE	8 mA parado, 700 mA mov. Sin carga	5 mA parado, 700 mA mov. Sin carga
DIMENSIONES	66x30x57.6mm	65.8x30x57.4mm
PESO	152g	200g
PRECIO	$150.000	$260.000

Fuente: [72] y [71].

a las medidas del robot, el HS-805BB y el Savox SV-0235MG. En el siguiente cuadro se encuentran contenidas las características técnicas de cada uno.

Del cuadro anterior, se puede evidenciar que el servomotor más idóneo para los movimientos horizontales y verticales, correspondientes al mecanismo de la cabeza del robot, es el Savox SV-0235MG debido al torque que este posee. Sin embargo, como también se puede observar, las demás características técnicas del servomotor HS-805BB no difieren en mayor medida a las de este, por lo que, dado que esta referencia es la recomendada por el diseñador del InMoov ([70]) y junto con que su costo es menor al del Savox SV-0235MG, se decidió utilizar este motor.

5.2. Electrónica

5.2.1. Tarjeta de Control

Para la selección de la tarjeta de control se tuvieron en cuenta los demás motores con los que puede llegar a contar el robot InMoov y que se especifican en el siguiente cuadro:

Cuadro 16: Descripción de Motores utilizados en el robot InMoov

Grupo	Mecanismo Asociado
Manos	Pulgar x 2 Índice x 2 Medio x 2 Anular x 2 Meñique x 2 Muñeca x 2
Brazo	Hombro x 2 Codo x 2 Rotador x 2 Bícep x 2 Omoplato x 2
Cabeza	Cuello x 2
Boca	Jaw
Ojos	Lateral

Fuente: [70].

De acuerdo al cuadro anterior, para el completo control de todos los mecanismos mencionados es necesario, suponiendo que estos motores sean a su vez servomotores, un total de 26 salidas PWM (Modulación de Ancho de Pulso por sus siglas en Inglés). Asumiendo que, para una primera etapa luego del desarrollo de este proyecto, se implementarán los mecanismos para el movimiento de los Brazos, la Boca y los Ojos junto con el de la Cabeza, se decidió utilizar un Arduino Mega 2560, el cual cuenta con 15 salidas PWM y a su vez permite una fácil comunicación USB Serial. Las características técnicas del mismo se muestran a continuación:

Cuadro 17: Características Técnicas Arduino Mega 2560

Microcontrolador	ATMega2560
Voltaje de Operación	5V
Voltaje de Entrada (Recomendado)	7-12V
Voltaje de Entrada (Límite)	6-20V
Pines de Entrada y Salida (I/O)	54 (15 PWM)
Corriente DC por Pin (I/O)	20mA
Costo (COP)	$40.460 ([74])

Fuente: [73] y [74]

5.3. Elementos de Potencia

Entiendase como los elementos que permiten el correcto funcionamiento de los dispositivos por medio de la regulación de las corrientes y voltajes suministrados por la fuente, en contraste con los requeridos por cada uno de estos.

5.3.1. Fuente de Voltaje

Al igual que para la tarjeta de control, la selección de la Fuente de Voltaje se realizó teniendo en cuenta los actuadores de los mecanismos que pueden llegar a ser utilizados, para un total de 26 motores. Suponiendo que cada uno de estos consuma un 30 % más de la corriente de consumo del motor HS-805BB sin carga (véase Cuadro 15), la corriente máxima del sistema será de aproximadamente $26A$. Para esto no se tuvo en cuenta la corriente de consumo de la tarjeta de control, pues esta se alimenta vía USB con el computador. En este sentido se seleccionó una Fuente de $12V$ a $120W$ (Figura 32) para un total de $30A$, con lo que se contará con un margen de seguridad del 15.38 % en caso de llegar a ser utilizados todos los motores.

Figura 31: Fuente de Voltaje S-360-12

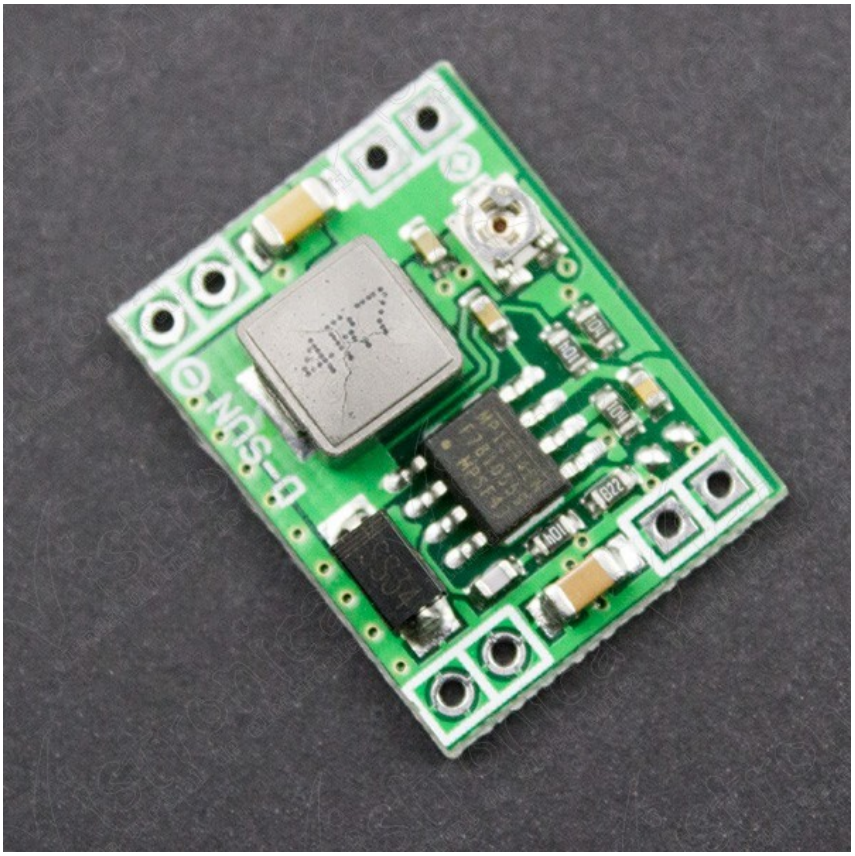
Fuente: [75].

5.3.2. Reguladores de Voltaje

En el Cuadro 15 se puede observar que el voltaje de operación de los motores seleccionados esta entre $4{,}8V - 6V$. Puesto que la fuente seleccionada

posee un voltaje de salida de $12V$, es necesario el uso de reguladores de voltaje para la alimentación a los motores. En este caso su diseño se considerá innecesario debido al alto costo de fabricación de un circuito similar (alrededor de $10.000 COP para una PCB de 5cm x 5cm sin incluir los materiales) en comparación con el bajo costo de las tarjetas prefabricadas que se venden en el mercado (alrededor de los $4.000 COP) [76]. En este sentido, y puesto que las características técnicas y los precios de estos dispositivos no varían significativamente entre si, se seleccionó un Conversor DC-DC Tipo Buck MP1584EN Ajustable a 3A, el cual se muestra en seguida:

Figura 32: Conversor DC-DC Tipo Buck MP1584EN Ajustable 3A



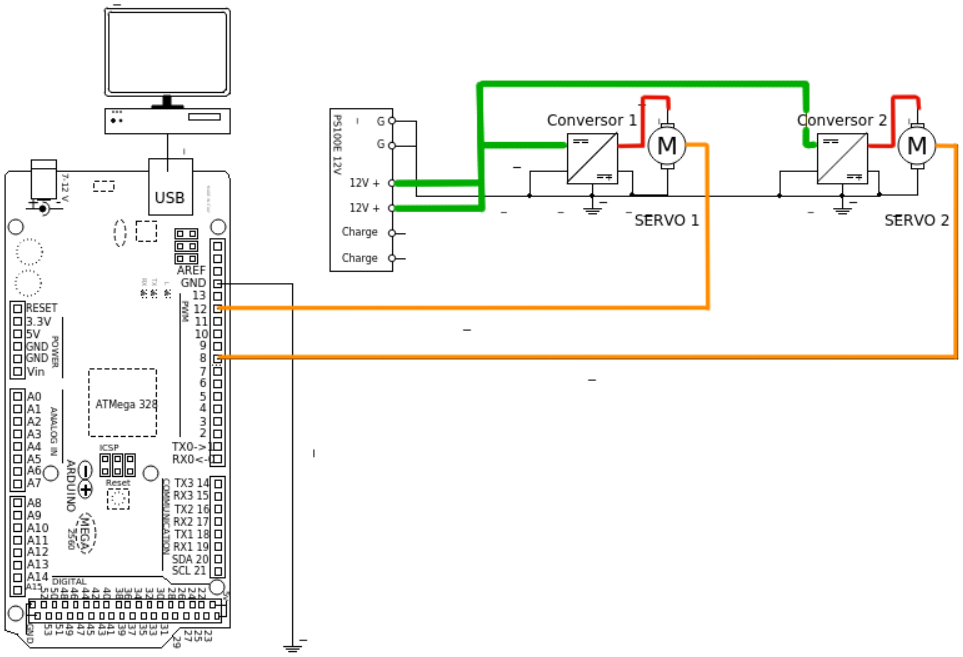
Fuente: [76].

Al ser este dispositivo ajustable, es posible modular el voltaje de salida de tal forma que coincida con el voltaje de operación de los actuadores.

5.4. Esquema Eléctrico

Teniendo en cuenta los motores seleccionados, la tarjeta de control y los elementos de potencia utilizados, se propone el siguiente diagrama eléctrico:

Figura 33: Diagrama Eléctrico del Sistema



Fuente: Elaboración Propia.

En la Figura 33, los Conversores 1 y 2 corresponden al convertidor DC-DC Tipo Buck MP1584EN, y los Servomotores 1 y 2 a los motores HS-805BB correspondientes al mecanismo de la cabeza del robot para el movimiento horizontal y vertical de la misma.

Vale resaltar que, dados los componentes seleccionados y el esquema eléctrico propuesto, no fue requerido el diseño de una tarjeta de control ni el de ninguna etapa o circuito de potencia.

6. Algoritmo de Control

El algoritmo de control corresponde a un algoritmo de visión de máquina que permitirá la interacción entre los algoritmos de procesamiento de imágenes y la respuesta directa de los actuadores basada en la cinnemática inversa del sistema. Para esto, dado que los algoritmos trabajan con imágenes bidimensionales, se abogó por el trabajo de Ortiz et al [66], de donde se puede afirmar que, dado que el módulo de los ángulos de desplazamiento del robot InMoov son proporcionales al desplazamiento en pixeles calculado, es posible encontrar una relación entre los ángulos β y λ con respecto a las compenentes vertical (ν_j) y horizontal (ν_j) del vector global de movimiento $\vec{\nu}$, de tal forma que:

$$\begin{bmatrix} \beta \\ \lambda \end{bmatrix} \sim \begin{bmatrix} \alpha_i \nu_i \\ \alpha_j \nu_j \end{bmatrix} \tag{25}$$

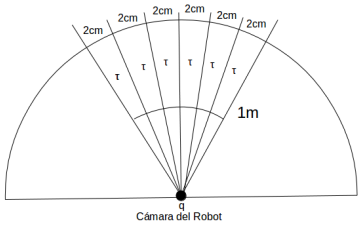
Donde α_i y α_j son las constantes de proporcionalidad de las componentes horizontal y vertical respectivamente.

Conforme a lo mencionado, se realizó el siguiente experimento para determinar los valores de α_i y α_j.

6.1. Obtención de α_i y α_j

Para la obtención de estos dos parámetros se dispuso de dos objetos dados a una distancia de $1 \pm 0{,}01\ m$ de la cámara del robot. Luego de esto, se desplazó el primer objeto $20cm$ en el eje X con respecto a la imagen, y se capturarón las posiciones del objeto en intervalos de $2cm$. Con el segundo objeto se realizó el mismo procedimiento pero para un desplazamiento de $20cm$ en el eje Y con respecto a la imagen.

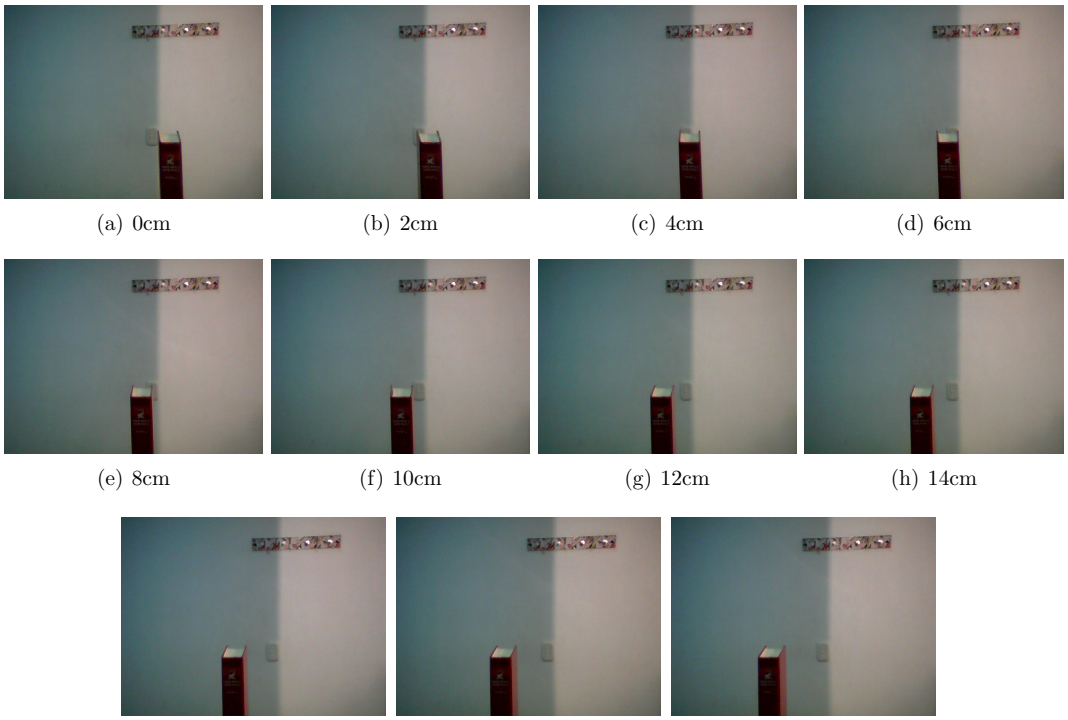
Figura 34: Esquema del Experimento realizado para la obtención de α_i y α_j



Fuente: Elaboración Propia

En las Figuras siguientes se muestran las capturas de las posiciones de cada uno de los objetos y los desplazamientos en pixeles calculados por el algoritmo **SB** para desplazamientos físicos de 2cm, 4cm, 6cm y 8cm a una distancia de 1m.

Figura 35: Posiciones del Objeto 1 para un desplazamiento total de 20cm en el eje X



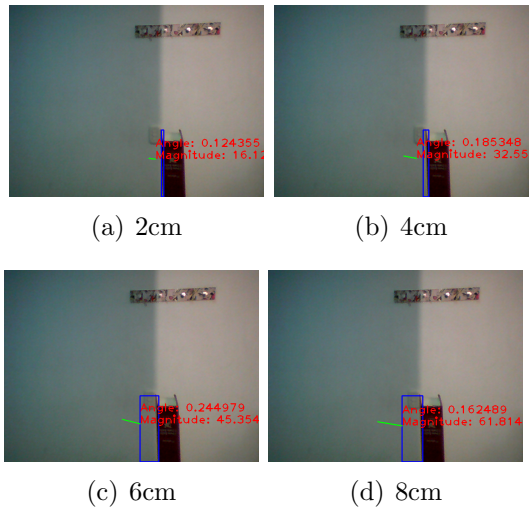
(a) 0cm (b) 2cm (c) 4cm (d) 6cm

(e) 8cm (f) 10cm (g) 12cm (h) 14cm

(i) 16cm (j) 18cm (k) 20cm

Fuente: Elaboración Propia.

Figura 36: Desplazamientos Calculados en Pixeles por **SB** para el Objeto 1

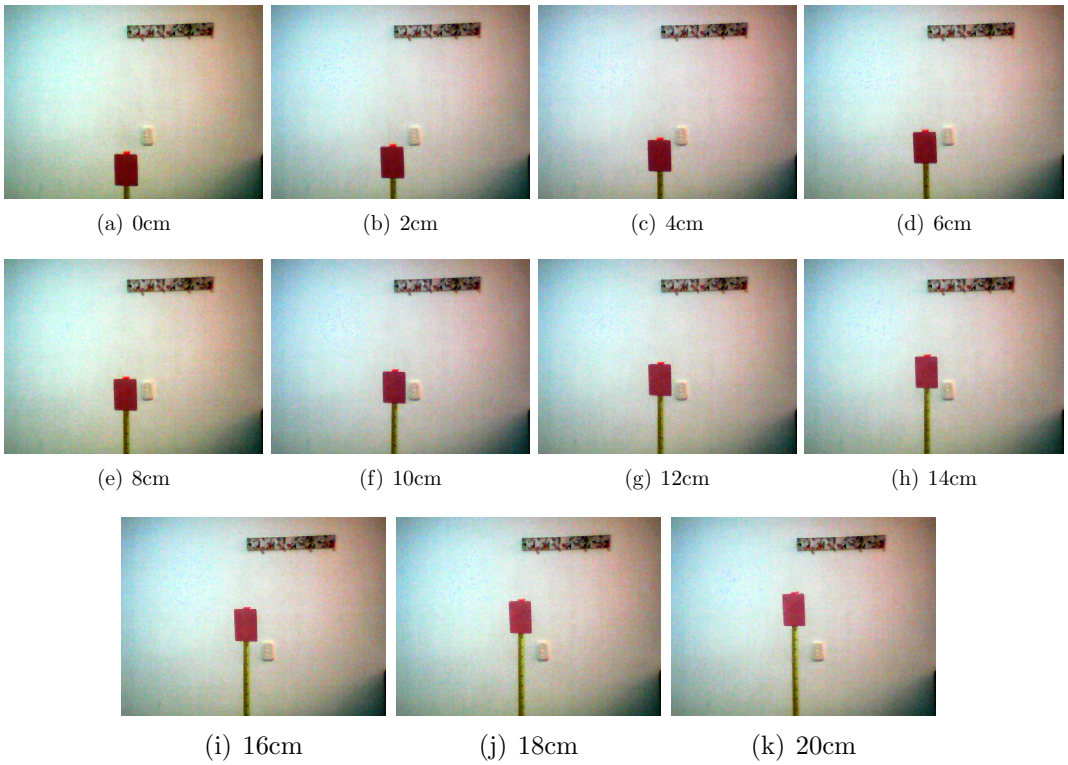


(a) 2cm (b) 4cm

(c) 6cm (d) 8cm

Fuente: Elaboración Propia.

Figura 37: Posiciones del Objeto 2 para un desplazamiento total de 20cm en el eje Y



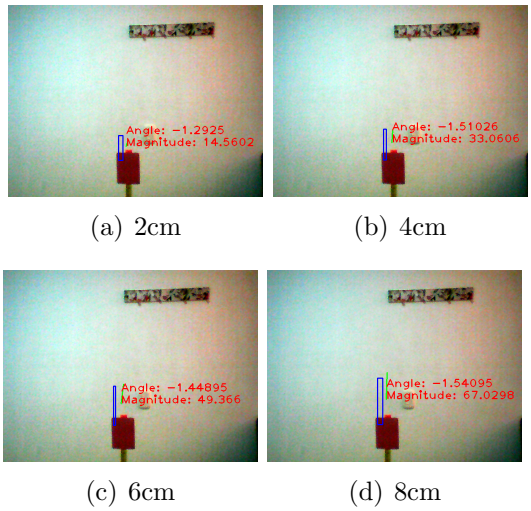
(a) 0cm (b) 2cm (c) 4cm (d) 6cm

(e) 8cm (f) 10cm (g) 12cm (h) 14cm

(i) 16cm (j) 18cm (k) 20cm

Fuente: Elaboración Propia.

Figura 38: Desplazamientos Calculados en Pixeles por **SB** para el Objeto 2



(a) 2cm (b) 4cm

(c) 6cm (d) 8cm

Fuente: Elaboración Propia.

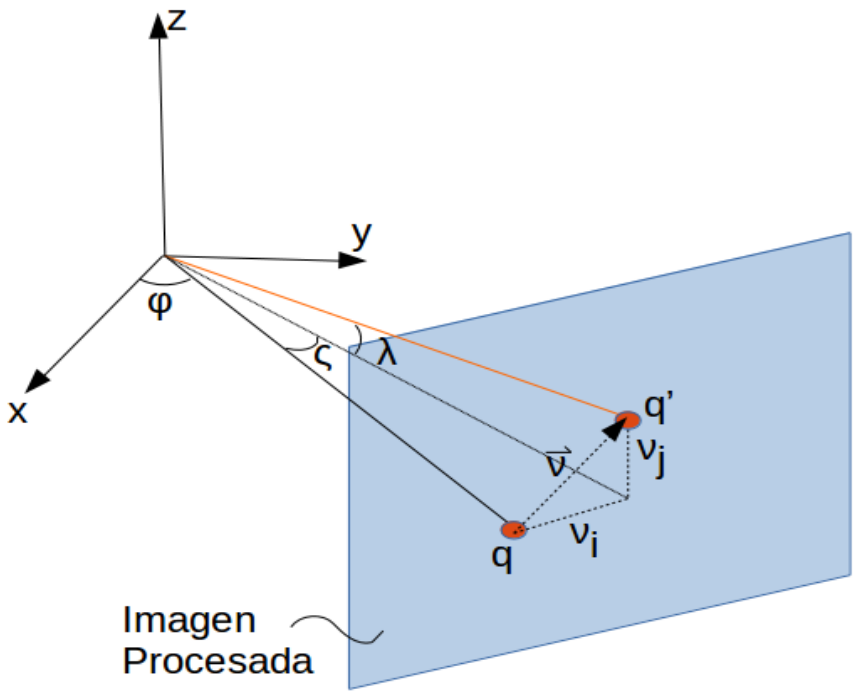
Sea $\tau = [\tau_x \ \tau_y]^T$, donde τ_x y τ_y son los ángulos de desplazamiento del robot con respecto a un desplazamiento físico de 2cm a 1m de distancia en X y Y respectivamente, de tal forma que, conforme al experimento realizado:

$$\tau = \begin{bmatrix} 1{,}136^\circ \\ 1{,}137^\circ \end{bmatrix}$$

Así, para un desplazamiento de 4cm en el eje X tendremos un ángulo de $2 \cdot \tau_x$, para uno de 6cm tendremos $3 \cdot \tau_x$ etc., y de manera análoga para los desplazamientos en el eje Y.

Para determinar la relación expuesta en la Ecuación 25 es necesario el siguiente análisis. Sea la Figura mostrada:

Figura 39



Fuente: Elaboración Propia.

Se puede evidenciar fácilmente que $\beta = \varphi + \zeta$, y de esta forma:

$$\begin{bmatrix} \varphi + \zeta \\ \lambda \end{bmatrix} = \begin{bmatrix} \beta \\ \lambda \end{bmatrix} = \begin{bmatrix} \alpha_i \nu_i + \varphi \\ \alpha_j \nu_j \end{bmatrix} \tag{26}$$

Esta ecuación aplica para cualquier posición de q, pues el plano de la imagen siempre será perpendicular al plano XY del sistema sobre el cual se encuentra sujeta la cámara (véase la Sección 4.1 junto con la Figura 28). Teniendo en

cuenta lo ya mencionado, se presenta el siguiente cuadro donde se encuentra la información contenida en las Figuras 36 y 38, y los resultados obtenidos para α_i y α_j.

Cuadro 18: Resultados Obtenidos en las Figuras 36 y 38

	Desplazamiento (cm)	**Ángulo de Desplazamiento Λ_i**	**Despalzamiento en Pixeles ν_i**	α_i (Λ_i/ν_i)
Eje X	2	τ_x	16.120	0.0704
	4	$2 \cdot \tau_x$	32.550	0.0698
	6	$3 \cdot \tau_x$	45.354	0.0751
	8	$4 \cdot \tau_x$	61.814	0.0735
	Desplazamiento (cm)	**Ángulo de Desplazamiento Λ_j**	**Desplazamiento en Pixeles ν_j**	α_j (Λ_j/ν_j)
Eje Y	2	τ_y	14.560	0.0780
	4	$2 \cdot \tau_y$	33.060	0.0687
	6	$3 \cdot \tau_y$	49.366	0.0690
	8	$4 \cdot \tau_y$	67.029	0.0678

Fuente: Elaboración Propia.

Concorde a los resultados expuestos, se define nuevamente:

$$\begin{bmatrix} \beta \\ \lambda \end{bmatrix} = \begin{bmatrix} \overline{\alpha_i}\nu_i + \varphi \\ \overline{\alpha_j}\nu_j \end{bmatrix} \tag{27}$$

Donde $\overline{\alpha_i}$ y $\overline{\alpha_j}$ son los promedios de las constantes de proporcionalidad descritas en el Cuadro 18. De esta forma:

$$\begin{bmatrix} \beta \\ \lambda \end{bmatrix} = \begin{bmatrix} 0{,}0722 \cdot \nu_i + \varphi \\ 0{,}0708 \cdot \nu_j \end{bmatrix} \tag{28}$$

En el Cuadro 18 se observa que para un desplazamiento de $8cm$ a $1m$ de distancia del robot, se alcanza el desplazamiento máximo en pixeles permitido discutido en la Sección 3.2.1. Esto implica que la máxima velocidad del objeto de estudio que podrá detectar el sistema, teniendo en cuenta el tiempo promedio de procesamiento del algoritmo seleccionado, será de $2{,}166\frac{m}{s}$ a una distancia del robot de aproximadamente $1m$. Esta afirmación, aunque correcta, no es del todo válida, pues no tiene en cuenta la velocidad de respuesta de los actuadores.

6.2. Algoritmo de Control

Teniendo en cuenta lo detallado anteriormente, se diseño el Algoritmo de Control cuyo pseudocódigo se muestra en seguida, en este no se tuvo en cuenta

el algoritmo de Flujo Óptico a utilizar, pues la dinámica del mismo se aplica para cualquiera de los algoritmos de procesamiento de imágenes descritos en este documento. En cambio, se denotó como $OPTFLOW(I_1, I_2, I_3)$ a la función que realiza el preprocesamiento de las imágenes, calcula el flujo óptico (sin importar el método) y arroja como resultado el vector global de movimiento $\vec{\nu}$. Además, utilizando el Diagrama Eléctrico mostrado en la Figura 33, se define como $ArduinoSerial(\theta_1, \theta_2)$ a la función que transmite los datos necesarios vía USB Serial al Arduino, para así mover los actuadores del sistema.

Cuadro 19: Pseudocódigo del Algoritmo de Control

Algoritmo de Control
1. Declaración de Variables.
while(1)
2. Adquisición de las imágenes: I_1, I_2 e I_3.
3. $\vec{\nu} = OPTFLOW(I_1, I_2, I_3)$
4. Obtención de ν_i y ν_j.
5. Obtención de β y λ.
6. Obtención de θ_1 y θ_2 (Cinemática Inversa).
7. $ArduinoSerial(\theta_1, \theta_2)$.
8. Tiempo de espera: t_s.
end while
end

Fuente: Elaboración Propia.

Se puede observar que la adquisición de las imágenes se realiza durante el inicio del ciclo **while** y además al final del ciclo se realiza un tiempo de espera, esto se debe a que es necesario tomar las imágenes mientras el robot se encuentre estático, puesto que los algoritmos no solo detectan el movimiento del objeto si no a su vez el movimiento de la escena en caso de que la cámara se encuentre en movimiento.

El Pseudocódigo mostrado en el cuadro anterior corresponde al algoritmo que se ejecuta desde el computador, en seguida se muestra el pseudocódigo correspondiente al código que se ejecutará en el Arduino.

Cuadro 20: Pseudocódigo del Algoritmo del Arduino

Algoritmo

1. Declaración de Variables ($\theta_{1_{Old}}$,$\theta_{2_{Old}}$, $\theta_{1_{New}}$, $\theta_{2_{New}}$).
2. Mover Servomotores 1 y 2 un ángulo de 70° y 60 respectivamente (Posición media del robot Ecuaciones 23 y 24).
3. $\theta_{1_{Old}} = 70$, $\theta_{2_{Old}} = 60$.
loop

4. Obtención de $\theta_{1_{New}}$ y $\theta_{2_{New}}$ vía USB Serial.
if($20° < \theta_{1_{Old}} + \theta_{1_{New}} < 160°$ **y** $30° < \theta_{2_{Old}} + \theta_{2_{New}} < 150°$ **)**
5. Mover Servomotor 1 un ángulo $\theta_{1_{Old}} + \theta_{1_{New}}$
6. Mover Servomotor 2 un ángulo $\theta_{2_{Old}} + \theta_{2_{New}}$.
end if
7. Tiempo de espera: t_s.

end loop
end

Fuente: Elaboración Propia.

En el pseudocódigo se definen como $\theta_{1_{Old}}$ y $\theta_{2_{Old}}$ a los ángulos de movimiento anteriores, y $\theta_{1_{New}}$ y $\theta_{2_{New}}$ a los ángulos de movimiento actuales. La declaración de estas cuatro variables es de suma importancia, pues es necesario almacenar la posición pasada de la cabeza del robot para moverla de manera correcta en el siguiente ciclo. Así, si en el ciclo anterior el robot se desplazó $\theta_{1_{Old}}$ y $\theta_{2_{Old}}$, los servomotores deberán desplazarse como se muestra en los pasos **4** y **5**.

En los pasos **8** y **6** de los Cuadros 19 y 20 hacen referencia al tiempo correspondiente a la velocidad de respuesta de los actuadores. Este se manejó como un tiempo variable que, teniendo en cuenta el Cuadro 15, se define como:

$$t_s = max\left(\frac{\theta_1}{375°/s}\ ,\ \frac{\theta_2}{375°/s}\right) + t_a \tag{29}$$

Esto indica que los algoritmos deberán esperar el tiempo de respuesta máximo, correspondiente al servomotor que deba girar un ángulo mayor, más un tiempo t_a correspondiente al tiempo mínimo para realizar la transmisión de datos. Nótese que, para el pseudocódigo del Arduino, el tiempo t_a no se debe tomar en cuenta para el cálculo de t_s, pues este es el tiempo que demora el dispositivo en realizar el proceso de recepción de datos. Por este motivo, la velocidad límite del sistema descrita en la Sección 6.1 no es del todo acertada, pues esta sujeta a la velocidad de respuesta de los actuadores y al tiempo t_a

7. Resultados

En el siguiente Cuadro se encuentran contenidos los valores de cada uno de los parámentros utilizados en el algoritmo **SB** para la correcta validación de los resultados obtenidos:

Cuadro 21: Parámetros del Sistema

Parámetro	**Descripción**	**Valor**
δ	Valor Umbral del Algoritmo	10
$\overline{\alpha_i}$	Factor de Conversión Eje X	$0{,}0722\frac{^\circ}{pixeles}$
$\overline{\alpha_j}$	Factor de Conversión Eje Y	$0{,}0708\frac{^\circ}{pixeles}$
φ	Ángulo de Posición de q_B	$74{,}63^\circ$
t_a	Tiempo de Espera para la Transmisión de Datos	$400ms$

Fuente: Elaboración Propia.

Del Cuadro mostrado se puede verificar que el tiempo de espera mínimo encontrado para realizar la correcta transmisión de datos fue de $400ms$, esto implica que la velocidad límite del sistema tomando en cuenta lo descrito en la Sección 3.2.1 será:

$$\nu_{max} = \frac{8cm}{36{,}920ms + t_{s_{max}} + 400ms}$$

$$\nu_{max} = \frac{0{,}08m}{436{,}920ms + \frac{140^\circ}{375\frac{^\circ}{1000ms}}}$$

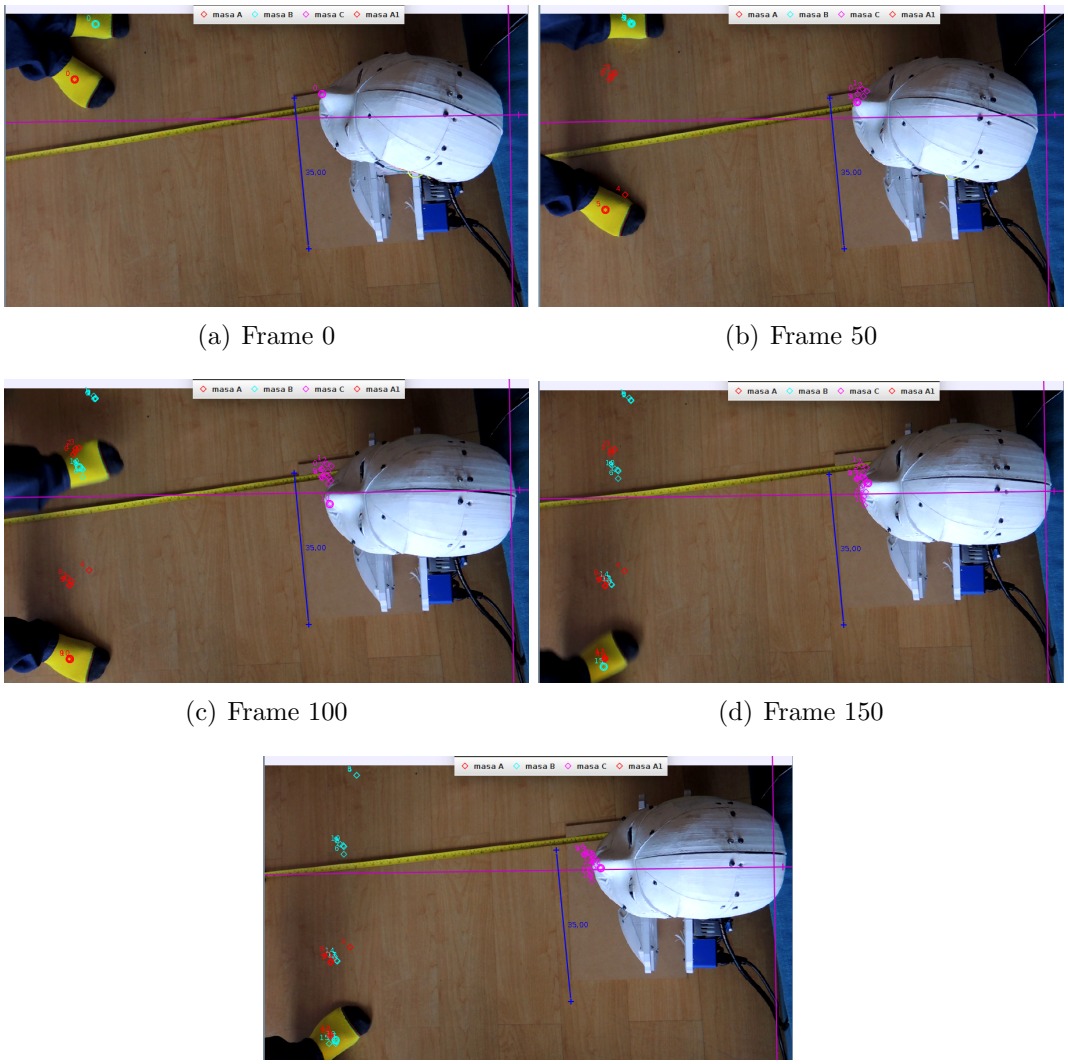
$$\nu_{max} = 0{,}9873\frac{m}{s}$$

Esta Velocidad corresponde a velocidad límite de detección a una distancia de 1m.

Para exponer los resultados obtenidos, se realizó un video del Robot siguiendo la Trayectoria de un objeto en movimiento y mediante el uso del Software Tracker [78] se obtuvierón los ángulos de desplazmiento del mismo y del plano de la cámara con respecto al origen del sistema. A su veź se muestra la trayectoría de estos sobre el plano XY.

En la siguiente Figura se muestran los frames 0, 50, 100, 150 y 200 del video realizado.

Figura 40: Frames del Video Realizado para las Trayectorias del Robot y el Objeto de Estudio



(a) Frame 0

(b) Frame 50

(c) Frame 100

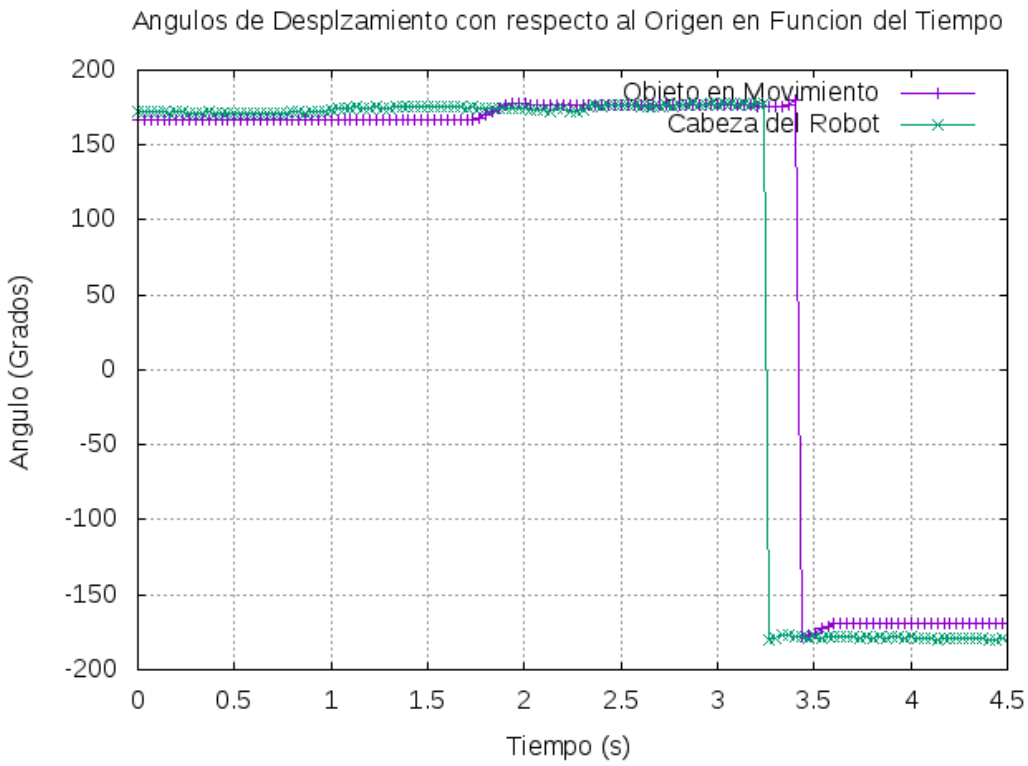
(d) Frame 150

(e) Frame 200

Fuente: Elaboración Propia.

Respecto al Video y al plano donde se encuentra contenido el robot y el objeto de estudio, como se observa en los Frames de la imagen superior, se obtuvierón las siguientes Gráficas:

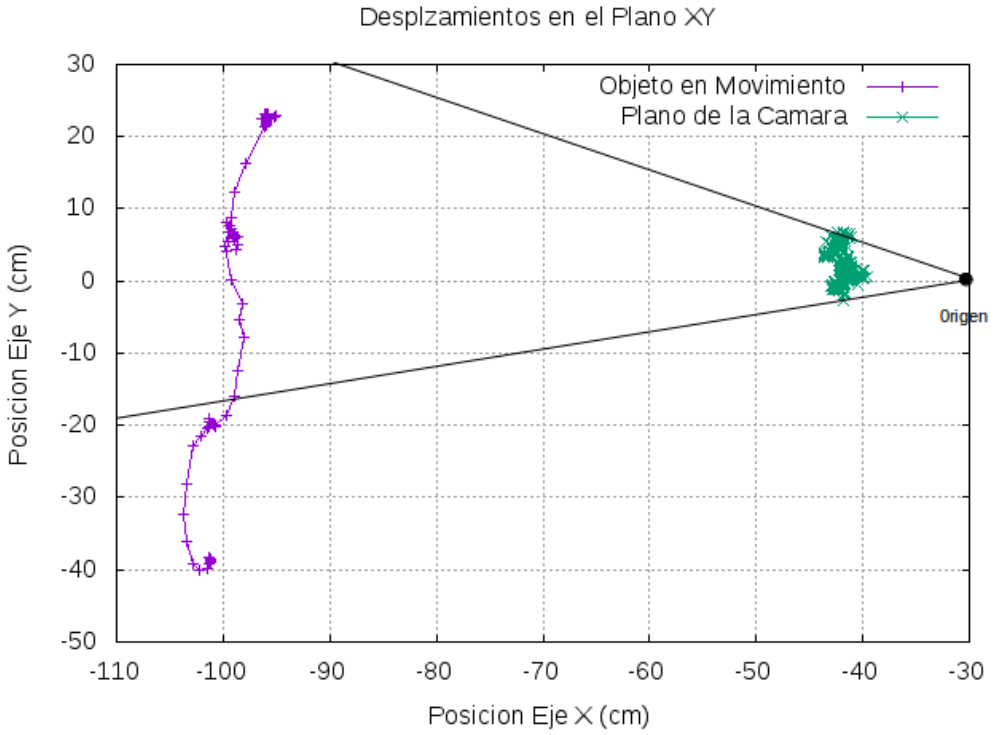
Figura 41



Fuente: Elaboración Propia.

Con respecto a la Gráfica Mostrada, podemos observar que los ángulos de la cabeza del robot corresponden de manera muy aproximada a los obtenidos para el objeto en movimiento, sin embargo, estos se encuentran desfasados, esto se debe al tiempo que demorá el robot en recibir la información y mover los actuadores hasta la posición deseada. En seguida se muestran las trayectorias obtenidas en el plano XY del Video:

Figura 42



Fuente: Elaboración Propia.

En esta Gráfica se puede evidenciar que efectivamente el robot realiza un seguimiento del objeto, sin embargo dado que utiliza información referente al movimiento del mismo y no a su a posición, el plano de la imagen nunca estará centrado con respecto al objeto sino con respecto a su movimiento aparente.

8. Conclusiones

En este trabajo se expusieron y se encontraron las parámetros y modelos necesarios para implementar un sistema basado en la cabeza del robot InMoov, para la detección y seguimiento de un objeto en movimiento. Dicho sistema, utiliza la información del movimiento obtenida de calcular el Flujo Óptico de una imagen a otra, lo que expone una nueva dinámica entorno a los trabajos e investigaciones expuestos en el estado del arte, donde, en su mayoría, se utiliza información espacial correspondiente a la posición del objeto.

Se realizó un estudio comparativo entre 4 diferentes técnicas para calcular el Flujo Óptico, y teniendo en cuenta la precisión y el costo computacional del Algoritmo se encontró que el más adecuado para el sistema fue el diseñado por los autores (**SB**).
Respecto a este, se encontró que la velocidad límite que puede detectar el algoritmo **SB** es de aproximadamente $2{,}166\frac{m}{s}$ a una distancia de $1m$ del plano de la cámara. Para que el lector se haga una idea más clara de lo que esto implica, el algoritmo es capaz de detectar un automóvil que se desplace a una velocidad de $7{,}797\frac{Km}{h}$ a un metro de distancia del plano de la cámara.
Sin embargo, la velocidad límite del sistema, como se mencionaba en la Sección 6.1, depende a su vez de otros factores. Para este caso en particular, se encontró que la velocidad límite que permite el correcto funcionamiento y desempeño del sistema es de $0,9873\frac{m}{s}$ a 1m de distancia del robot.

Se obtuvo la cinemática inversa del mecanismo que permite el movimiento de la cabeza del robot, y se obtuvieron las restricciones correspondientes al mismo que junto con las restricciones propias de diseño, permitierón el correcto desarrollo y diseño del algoritmo de control del sistema. El modelo encontrado se validó por medio de la cinemática directa del mecanismo.

Se seleccionarón los dispositivos y actuadores adecuados teniendo en cuenta el diseño del robot, las recomendaciones realizadas por el fabricante del mismo [70] y el esquema de control propuesto (Figura 30). Con respecto a estos, se pudo concluir que no fue requerido el diseño de las tarjetas electrónicas y se abogó por utilizar los dispositivos más adecuados teniendo en cuenta sus características técnicas y costo.

Se diseño el algoritmo de control en base a los parámetros del sistema, hallados por medio de los experimentos realizados y de la instrumentación seleccionada, más propiamente hablando α_i, α_j y la definición de t_s, y a las ecuaciones obtenidas para la cinemática inversa (Ecuaciones 20 y 19).
Por medio de los resultados obtenidos se demuestra la validez de la Ecuación 25 expuesta por Ortiz [66].

9. Contribuciones y Trabajos Futuros

9.1. Contribuciones

• ORTIZ, Nicolás. VARGAS, Luis Felipe. JINETE, Marco Antonio y JIMÉNEZ, Robinson. Movement detection for object tracking applied to the InMoov robot head. En: 2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA). 2016. ISBN: 978-1-5090-3797-1.

• Conferencistas Mini Maker Fair, Diciembre de 2016. Biblioteca Virgilio Barco. Visión artificial aplicada al robot Inmoov.

• Ganadores: Robotic People Awards 2016, por los aportes realizados a la robótica y la tecnología. Diciembre de 2016.

9.2. Trabajos Futuros

Como se mencionó en la Sección 1.2, el desarrollo del proyecto esta enfocado al área de investigación en Ingeniería y Tecnología, sobre todo en el área de aprendizaje de máquina. A futuro, se prevee trabajar en el desarrollo de una aplicación que permita una interacción directa humano-máquina, donde el robot tenga la capacidad de aprender del ser humano e interactuar con el a travez de estímulos no solo visuales sino también auditivos.

Referencias

[1] RUIZ DE GARIBAY PASCUAL, Jonathan. Robótica: Estado del Arte. Tesis doctoral en Sistemas de la Información. Bilbao: Universidad de Deusto, 2006.

[2] SMOLA, Alex and VISHWANATHAN, S.V.N. Introduction to Machine Learning. United Kingdowm: Cambridge University Press, 2008. p.3.

[3] RUSSEL, J. Stuart and NORVIG, Peter. Artificial Intelligence. A modern Approach. New Jersey, 1995. p.3.

[4] JIMÉNEZ, Robinson. ESPINOSA, Fabián y AMAYA, Darío. Control de movimiento de un robot humanoide por medio de visión de máquina y réplica de movimientos humanos. INGE CUC, 2013. Vol. 9. No. 2. p. 44–51.

[5] MRONGA, D. y AGGARVAL, A. Motion Control of the Humanoid Robot AILA. German Research Center for Artificial Intelligence (DFKI) GmbH, 2013. RR-13-03.

[6] TSAGARAKIS, N.G. et al. iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research. En: Advanced Robotics, 2007. Vol. 21. No. 10. p. 1151–1175.

[7] GONZALES MARCOS, Ana. et al. Técnicas y Algoritmos Básicos de Visión Artificial. Integrantes del Grupo de Investigación EDMANS. Logroño: Universidad de La Rioja, Servicio de Publicaciones, 2006. p. 11.

[8] PEÑA BAEZA, Alberto. Módulo de visión artificial del robot humanoide HOAP3. Aplicación al seguimiento de objetivos móviles. Tesis para obtar por el grado en Ingeniería de Sistemas y Automática. Leganés: Universidad Carlos III de Madrid, 2010.

[9] AZAD, Pedram. ASFOUR, Tamin y DILLMANN, Ruediger. Stereo-based 6D Object Localization for Grasping with Humanoid Robot Systems. En: IEEE International Conference on Intelligent Robots and Systems, 2007. p. 919-924

[10] BARRERA TOVAR, Lucia. Desarrollo e Implementación de Algoritmos para el Sistema de Percepción Y Localización de Los Robots Bogobots. Tesis para optar por el grado en Ciencias de la Ingeniería. Monterrey: Instituto Tecnológico y de Estudios Superiores de Monterry, 2010. p. 2-4.

[11] BROWATZKI, Björn. et al. Active object recognition on a humanoid robot. IEEE International Conference on Robotics and Automation, 2012. p. 2021–2028.

[12] GARCIA, G. et al. Visual servoing path tracking for safe human-robot interaction. IEEE International Conference on Mechatronics, 2009. p. 5479-5485.

[13] STÜCKLER, Jörg. y BEHNKE, Sven. Following human guidance to cooperatively carry a large object. EEE-RAS International Conference on Humanoid Robots, 2011. p. 218-223.

[14] GORI, I. et al. DForC: A real-time method for reaching, tracking and obstacle avoidance in humanoid robots. IEEE-RAS International Conference on Humanoid Robots, 2012. p. 544-551.

[15] LEITNER, Jürgen. et al. Learning spatial object localization from vision on a humanoid robot. International Journal of Advanced Robotic Systems, 2012. Vol. 9.

[16] VAHRENKAMP, Nikolaus. et al. Planning and execution of grasping motions on a humanoid robot. 9th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS09, 2009. p. 639-645.

[17] HOYOS GUTIÉRREZ, José. et al. Hacia el manejo de una herramienta por un robot NAO usando programación por demostración. Tecnologías, 2014. Vol.7. No. 33. p. 65-76.

[18] POLLARD, Nancy. et al. Adapting Human Motion for the Control of a Humanoid Robot. International Conference on Robotics and Automation, 2002.

[19] ASIMO. [en linea]. <http://asimo.honda.com/innovations/> [citado el 4 de agosto de 2016].

[20] LEE, Christine. DURAN, Ulises y XIA, Yuan. Hepatic feedback robotic hand. Illinois, 2015. ECE 445.

[21] SZELISKI, Richard. Computer Vision: Algorithms and Applications. Springer, 2010.

[22] VISIÓN ONLINE, Actualidad. [en linea]. <http://www.visiononline.es/es/actualidad-en-vision-artificial> [citado el 4 de agosto de 2016].

[23] HERNANDEZ, Nelson. Visión artificial. Bogotá D.C: Revista Tecnología, 2003. Vol 2. p. 48.

[24] RANJAN, R. et al. Classification and Identification of Surface Defects in Friction Stir Welding: An Image Processing Approach. Journal of Manufacturing Processes, 2016. Vol. 22. No.4. p. 237. ISSN 15266125.

[25] SORRENTINO, D.A. Improved Algorithms for Image Super-Resolution. Ann Arbor: University of Victoria (Canada), 2009. ISBN 9780494606582.

[26] GALLARDO, Alvarado. et al. Un Algoritmo para Resolver la Cinemática Directa de Plataformas GoughStewart Tipo 6-3. Mexico: Computación y Sistemas, 2004. Vol. 8. No. 2. p. 132-149

[27] RAMÍREZ, Cardona. et al. Cinemática directa utilizando Denavit-Hartenberg y generación de trayectorias para el robot FANUC LR-Mate200iB/5. Mexico: Revista Investigación Científica, 2008. Vol. 4. No. 2. ISSN 1870-8196.

[28] RAMIREZ, Kryscia. Cinemática Inversa del robot. Costa Rica. 2012 .UCR-ECCI. CI-2657

[29] UNIVERSIDAD DE BOGOTA JORGE TADEO LOZANO: Robótica y Automatización Industrial, Realidad de la Robótica en Colombia. [en linea]. <http://www.utadeo.edu.co/es/noticia/opinion/robotica-y-automatizacion-industrial/85/realidad-de-la-robotica-en-colombia> [citado el 24 de Agosto de 2016].

[30] COLCIENCIAS, Estado de la Ciencia en Colombia: grupos de investigación e investigadores van en aumento. [en linea]. <http://www.colciencias.gov.co/sala_prensa/estado-de-la-ciencia-en-colombia-grupos-de-investigacion-e-investigadores-van-en-aumento> [citado el 2 de septiembre de 2016].

[31] Observatorio Colombiano de Ciencia y Tecnología. Indicadores de Ciencia y Tecnología, Colombia 2015. 1 ed. Bogotá D.C.: Ediciones Ántropos ltda, 2015. p. 72.

[32] GAMS, Andrej. et al. Performing Periodic Tasks: On-Line Learning, Adaptation and Synchronization with External Signals. *En:* The Future of Humanoid Robots- Research and Applications. Rijeka: Teodora Smiljanic, 2012. p. 3-28.

[33] TECNOLÓGICO DE MONTERREY, Bogobots. [en linea]. <homepage.cem.itesm.mx/aaceves/Bogobots/losrobots.html> [citado el 15 de septiembre de 2016].

[34] MURRAY, Richard M. LI, Zexiang y SASTRY, S. Shankar. A mathematical Introduction to Robotic Manipulator. CRC press, 1994.

[35] RAMESH, Jain. KASTURI, Rangachar y SCHUNCK, Brian G. Machine Vision. McGRAW-HILL, 1995. ISBN 0-07-032018-7.

[36] DIEBEL, James. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. Stanford, 2006. 94301–9010.

[37] SAWHNEY, Harpreet S. y HANSON, Allen R. Identification and 3D description of 'shallow' environmental structure over a sequence of images. En IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91). 1991. p. 179–185.

[38] TOMASI, Carlo. y KANADE, Takeo. Shape and motion from image streams under orthogra-phy: A factorization method. En International Journal of Computer Vision, 1992. Vol. 9. No. 2. p. 137–154.

[39] POELMAN, Conrad J. y KANADE, Takeo. A paraperspective factorization method for shape and motion recovery. En IEEE Transactions on Pattern Analysis and Machine Intelligence, 1997. Vol. 19. No. 3. p. 206–218.

[40] ALOIMONOS, Yiannis J. Perspective approximations. En Image and Vision Computing, 1990. Vol. 8. p. 177-192.

[41] Gonzales, Rafael C. y WOODS, Richard E. Color Image Processing. En: Digital Image Processing. 3 ed. New York: Prentice Hall, 2006. p. 416- 482.

[42] Color Models for MapInfo. [en linea]. <http://georezo.net/jparis/MI_Enviro/Colors/color_models.htm> [citado el 14 de septiembre de 2016].

[43] Microsoft, CMY and CMYK Color Spaces. [en linea]. <https://msdn.microsoft.com/en-us/library/windows/desktop/dd371926(v=vs.85).aspx> [citado el 14 de septiembre de 2016].

[44] POYNTON, Charles A. A technical Introduction to Digital Video. New York: John Wiley & Sons, 1996. ISBN 0-471-12253-X.

[45] SCHANDA, János. EPPELDAUER, George. y SAUTER, Georg.Tristimulus Color Measurement of Self-Luminous Sources. En: Colorimetry: Understanding the CIE System. New Jersey: John Wiley & Sons, 2007. ISBN 978-0-470-04904-4.

[46] VOSS, Klaus. Discrete Images, Objects, and Functions in Z^n. 1 ed. Berlín: Springer-Verlag, 1993. p. 9-13.

[47] Image Segmentation. [en linea]. <https://www.cs.auckland.ac.nz/courses/compsci773s1c/l
html/topic3.htm> [citado el 15 de septiembre de 2016].

[48] BALL, Robert S. A Treatise on the Theory of Screws. Londres: Cambridge University Press, 1900.

[49] FLEET, David J. y WEISS, Yair. Optical Flow Estimation. En: Mathematical models for Computer Vision: The Handbook. 2005. p. 239-257. ISBN 0387263713.

[50] NEGAHDARIPOUR, Shahriar. Revised Definition of Optical Flow: Integration of Radiometric and Geometric Cues for Dynamic Scene Analysis. En: IEEE Transactions on Pattern Analysis and Machine Intelligence. 1998. Vol. 20. No. 9. p. 961-979.

[51] BARRON, J.L. FLEET, David J. y BEAUCHEMIN, S.S. Performance of optical flow techniques. En: International Journal of Computer Vision, 1994. Vol. 12. No. 1. p. 43-77.

[52] LUCAS, Bruce D. y KANADE, Takeo. An Iterative Image Registration Technique with an Application to Stereo Vision. En: Proceedings of Imaging Understanding Workshop, 1981. p. 121-130.

[53] SHI, Jiambo. y TOMASI, Carlo. Good Features to Track. En: IEE Conference on Computer Vision and Pattern Recognition (CVPR94), 1994.

[54] BAKER, Simon. y MATTHEWS, Iain. Lucas-Kanade 20 Years On: A Unifying Framework. En: International Journal of Computer Vision. Vol. 56. No. 3, 2004. p 221–255.

[55] ORON, Shaul. BAR-HILLEL, Aharon. AVIDAN, Shai. Extended Lucas-Kanade Tracking. En: European Conference on Computer Vision, 2014.

[56] BOUGUET, Jean-Yves. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm. En: Intel Corporation, Microprocessor Research Labs. 2000.

[57] DEGING, Sun. et al. Learning Optical Flow. En: European Conference on Computer Vision, 2008.

[58] HORN, Berthold K.P. y SCHUNCK, Brian G. Determining Optical Flow. En: Artificial Intelligence (MA 02139), 1980.

[59] FEIL-SEIFER, David. y MATARIC, Maja J. Human-Robot Interaction. Invited contribution to Encyclopedia of Complexity and Systems Science, Robert A. Meyers (eds.). New York: Springer, 2009. p. 4643-4659.

[60] GOODRICH, Michael A. y SCHULTZ, Alan C. Human–Robot Interaction: A Survey. En: Foundations and Trends in Human–Computer Interaction, 2007. Vol. 1, No. 3. p. 203–275.

[61] FARNEBÄCK, Gunnar. Two-Frame Motion Estimation Based on Polynomial Expansion. En: 13th Scandinavian Conference, SCIA, 2004. p. 363-370.

[62] MOHAMMED, Adulmalik D. y MORRIS, Tim. Optical Flow Estimation Using Local Features. En: Proceedings of the World Congress on Engineering, 2015. Vol 1. p. 562-565.

[63] BAKER, Simon. et al. A Database and Evaluation Methodology for Optical Flow. En: International Journal of Computer Vision, 2011. Vol 92. p. 1–31.

[64] KOFINAS, Nikolaus. Forward and Inverse Kinematics for the NAO Humanoid Robot. Tesis para optar al grado en Ingeniería Electrónica y Infromática. La Canea: Universidad Técnica de Creta, 2012. p. 10.

[65] OpenCV. [en linea]. <https://www.opencv.org> [citado el 21 de septiembre de 2016].

[66] ORTIZ, Nicolás. et al. Movement detection for object tracking applied to the InMoov robot head. En: 2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA). 2016. ISBN: 978-1-5090-3797-1.

[67] D. Martin. et al. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. En: 8th International Conference in Computer Vision, 2001. Vol. 2. p. 416-423.

[68] Image Processing CS 6640 Project-4 (Mosaic). [en linea]. <http://www.cs.utah.edu/ ssingla/IP/P4/Index.html> [citado el 10 de Noviembre de 2016].

[69] DE CASTRO, E. y MORANDI, C. Registration of Translated and Rotated Images Using Finite Fourier Transforms. En: IEEE Transactions on pattern analysis and machine intelligence. 1987.

[70] InMoov. [en linea]. <http://inmoov.fr> [citado el 3 de Julio de 2016].

[71] SAVÖX. [en linea]. <http://www.savoxusa.com/Savox_SV0235MG_Digital_Servo_p/savsv0
[citado el 4 de noviembre de 2016].

[72] SERVOCITY. [en linea]. <https://www.servocity.com/hs-805bb-servo>
[citado el 4 de noviembre de 2016].

[73] Arduino. [en linea]. <https://www.arduino.cc/en/Main/arduinoBoardMega2560>
[citado el 4 de noviembre de 2016].

[74] Vistronica. Tienda Virtual de Electrónica. [en linea]. <https://www.vistronica.com/arduino/board/arduino-mega-2560-con-ch340-detail.html> [citado el 10 de Enero de 2017].

[75] OfferAny. [en linea]. <http://www.offerany.com/p-13193252229.html> [citado el 4 de Enero de 2017].

[76] Vistronica. Tienda Virtual de Electrónica. [en linea]. <https://www.vistronica.com/conversores-dc-dc/?p=1> [citado el 10 de Enero de 2017].

[77] Myrobotlab. [en linea]. <http://myrobotlab.org/service/inmoov> [citado el 2 de Enero de 2017].

[78] Tracker. [en linea]. < http://physlets.org/tracker/> [citado el 2 de Enero de 2017].

Anexos

A. Movement Detection for Object Tracking Applied to the InMoov Robot Head

Movement Detection for Object Tracking Applied to the InMoov Robot Head

Nicolás Ortiz Valencia
Universidad Piloto
de Colombia
Bogotá D.C, Colombia
nicolas-ortiz@upc.edu.co

Luis Felipe Vargas Londoño
Universidad Piloto
de Colombia
Bogotá D.C, Colombia
luis-vargas@upc.edu.co

Marco Antonio Jinete
and Robinson Jiménez
Universidad Piloto
de Colombia
Bogotá D.C, Colombia

Abstract

Machine vision is one of the most important tasks for the interaction between the robot and the environment, because it provides more information about the elements that exist there. The InMoov robot is the first life-size robot that can be produced by 3D printing and its design is open to the public. Between the following of the movement vector, the head control of this robot was implemented with the purpose of trackig the movement of an object. In the project the movement restrictions were taken corresponding to the range vision of the robot to develop the tracking and movement algorithms that will be aplied to the head movement of the robot under controlled conditions.

1. Introduction

Robots are filling today a growing number of roles, from Industrial Applications to Health and medical care, and computer vision is one of the variety of fields that are involved in the study of the human-robot interaction (HRI) tasks [10]. With Humanoid robots, which main purpose is to imitate a certain human behavior or behaviors, object tracking is very important to achieve a good human-robot proximate interaction (HRPI) [11].

In this paper it is presented the process to achieve this goal. Movement dection algorithm was selected, taking into account the computational cost and accuracy. Then it is described the InMoov movement restrictions to obtain a physical interaction between the robot and the studied object. Finally, it is developed an algorithm to control the InMoov robot head momevement based on the computer vision process realized.

2. Movement Detection

There are many methods for movement vector detection; global and local differential methods, phase based methods, correlation and feature based methods and so on [14]. However, in this paper, we review two commonly used algorithms: Lucas Kanade (**LK**) and Farnebäck (**FB**), and we also used a subtraction algorithm as a filter before movement vector estimation process.

2.1. Image Preprocessing

As an initial step, we consider light conditions an important fact for a better optical flow calculation, because of that, as an initial step, we transform the input images to the XYZ model as follows.
Consider $I_{XYZ} = [a_{i,j}]$ the XYZ image transform of the I_{RGB} image, were $[a_{i,j}]$ is a matrix that have as members the intensity vector for each prixel on the rectangular position (i, j), then:

$$a_{i,j} = \begin{bmatrix} \frac{sR_{i,j}}{s(R_{i,j}+G_{i,j}+B_{i,j})} \\ \frac{sG_{i,j}}{s(R_{i,j}+G_{i,j}+B_{i,j})} \\ \frac{sB_{i,j}}{s(R_{i,j}+G_{i,j}+B_{i,j})} \end{bmatrix} \quad (1)$$

Were s is a brightness factor and $R_{i,j}$, $G_{i,j}$, $B_{i,j}$ are the channel intensities of each pixel of the RGB image. As it can be appreciate, the XYZ transform allows us to avoid this factor.
Then we did a white threshold to obtain a better filtering of the image brightness conditions.
Suppose that because of light condition it was obtained an image where the colors look darker, or simply look different. As is known white has the higher intensity value on each channel, then if there is something white in the image:

$$b_{i,j} = \begin{bmatrix} \frac{255 \cdot R_{i,j}}{max(R)} \\ \frac{255 \cdot G_{i,j}}{max(G)} \\ \frac{255 \cdot B_{i,j}}{max(B)} \end{bmatrix} \quad (2)$$

978-1-5090-3797-1/16/$31.00

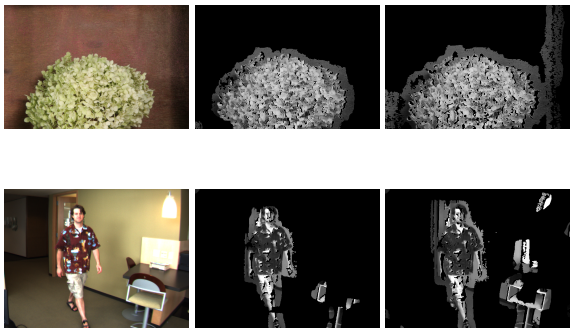


Figure 1: Subtraction Algorithms Results implemented in the Baker *et al*. [15] database: Hydrangea and Walking sets. From left to right; first frame, subtraction between the second and the first frame and subtraction between the third and the first frame.

Where $b_{i,j}$ is the intensity for each pixel after white thresholding. With this transformation we are normalizing the color intensities to obtain a better intensity information for each pixel.

2.2. Optical Flow: Lucas Kanade and Farnebäck Algorithms

For Optical Flow it was implemented a subtraction algorithm to obtain an image just with the studied object as it can be seen on Figure 1, and then we estimate the movement vector for each pixel using the algorithms mentioned. We also estimate a global movement vector (ν), as we are going to explain below, to control de InMoov head movement.

2.2.1 Lucas Kanade

This image registration technique was first described by Lucas and Kanade [1] in 1981, its main goal was to use spatial gradient information to direct the search for the position that yields the best match. With the Shi and Tomasi [2] work, this technique becomes more efficient and accurate, because of the better quality and less quantity of information to be processed.

For movement detection, it was implemented the Pyramidal Lucas Kanade described by Bouguet [5] that is an application of the lucas Kanade spatial gradient equation applied to an image pyramidal representation. Let I and J 2D grayscale images. Where $I(n) = (i, j)$ and $J(n) = (i, j)$ are first an second quantities of the grayscale image at the $n = [i\ j]^T$ location, where i and j are the rectangular positions of a n image point. Consider an image point $u = [u_i\ u_j]^T$, and the goal is to find the location $v = u + d = [u_i + d_i\ u_j + d_j]$ on the second image J. This means that:

$$J(i + d_i, j + d_j) - I(i, j) \approx 0$$

Taking the Taylor series expansion of $J(i + d_i, j + d_j)$:

$$J(i + d_i, j + d_j) = J(i, j) + \frac{\partial J}{\partial i} d_i + \frac{\partial J}{\partial j} d_j$$

$$J(i, j) + \frac{\partial J}{\partial i} d_i + \frac{\partial J}{\partial j} d_j - I(i, j) \approx 0$$

Let $J_i = \frac{\partial J}{\partial i}$ and $J_j = \frac{\partial J}{\partial j}$ then:

$$(J(i, j) - I(i, j)) + J_i d_i + J_j d_j \approx 0$$

$$J_t + \nabla J \cdot [d_i\ d_j] \approx 0$$

Therefore we are going to obtain the next system of equations:

$$\begin{bmatrix} \sum J_i J_i & \sum J_i J_j \\ \sum J_i J_j & \sum J_j J_j \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} = - \begin{bmatrix} \sum J_i J_t \\ \sum J_j J_t \end{bmatrix} \quad (3)$$

The goal here is to find the movement vector $[d_i\ d_j]^T$ for each pixel. The idea of using a pyramidal model is to be able to handle large pixel motions solving (3) for this image respresentation, but the main goal of this algorithm still being the same.

2.2.2 Farnebäck

This technique described by Farnebäck [12], consist on motion estimation by a polinomial expansion, the goal is the same as in Lucas Kanade algorithm: to find the new location of any pixel of interest.

Consider de local signal model, expressed in a local cordinate system:

$$f(x) \sim x^T a x + b^T x + c$$

Where a is a matrix, b is a vector, and c is a scalar. Accordingly to this we define the displacement function between two frames, $f_1(x)$ and $f_2(x)$, as:

$$f_2(x) = f_1(x - d) = x^T a_1 x + (b_1 - 2a_1 d)^T x + d^T a_1 d - b_1^T d + c_1$$

$$f_2(x) = x^T a_2 x + b_2^T x + c_2$$

This means that:

$$a_2 = a_1 \quad (4)$$

Optical Flow Implemented Algorythm

1. Obtain the first three frames; *Initial, Previous, Actual* and the time t between *Previous and Actual*

2. XYZ an white thresholding transformations
3. *Previous=Previous-Initial* & *Actual=Actual-Initial*.
4. Optical Flow calculation $\rightarrow \sigma_n$, η_n.
(Lucas Kanade or Farnebäck)
5. Estimate $\vec{\nu}$

6. end

Table 1

$$b_2 = b_1 - 2a_1 d \tag{5}$$

$$c_2 = d^T a_1 d - b_1^T d + c_1 \tag{6}$$

Then if a_1 is not singular we can solve for d equation (5). Farnebäck introduce then the inclusion of an a priori displacement field that in consequence complete the necessary information to solve this problem in the algorithm and find the displacement vectors for each point.

2.2.3 Global Movement Vector

Lucas Kanade and Farnebäck algorithms give us a several number of old points ($\sigma_n = (i_{\sigma(n)}, j_{\sigma(n)})$) and new points ($\eta_n = (i_{\eta(n)}, j_{\eta(v)})$), where $(i_{\eta(n)}, j_{\eta(n)})$ is the new rectangular position of the old n-point. Then if t is the time between the first and the second frame:

$$\mid \vec{\nu} \mid = \frac{1}{nt} \sum_{m=1}^{n} \sqrt{(i_{\sigma(m)} - i_{\eta(m)})^2 + (j_{\sigma(m)} - j_{\eta(m)})^2} \tag{7}$$

$$\measuredangle \vec{\nu} = \frac{1}{n} \sum_{m=1}^{n} \arctan\left(\frac{j_{\sigma(m)} - j_{\eta(m)}}{i_{\sigma(m)} - i_{\eta(m)}}\right) \tag{8}$$

Note that $\mid \vec{\nu} \mid$ is given in *pixels per time unit*, on section 3 we described the method used to make this quantity a real meassure that we can use to move the robot.

2.3. Computational Cost Comparison

Comparison between both algorithms was made on an Intel® Core^{TM} i5-5200U processor, herewith is the processing times (Figure 2) for the algorithm showed on Table 1, it was implemented with Lucas Kanade and Farnebäck during 300 frames:

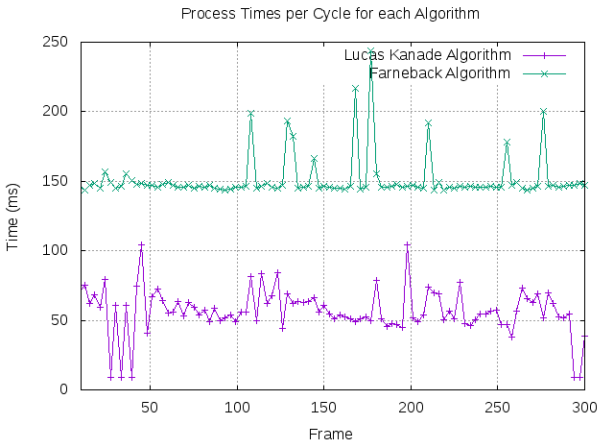


Figure 2: Computational Cost for both algorithms.

	PTA (*ms*)	**RDC** (*pixels*)	**RDM** (*pixels*)	**Error** (%)
FB	151,283	9,647	10	3.53
LK	57,152	9,999	10	0.01

Table 2: Comparison data for computational cost evaluation. (**PTA:** Process Time Average, **RDC:** Real Displacement Calculated, **RDM:** Real Displacement Meassured)

Comparassion was made taking into account the process time average taked from data showed on Figure 2 the error between the real displacement calculated by each algorithm and the real displacement meassured.

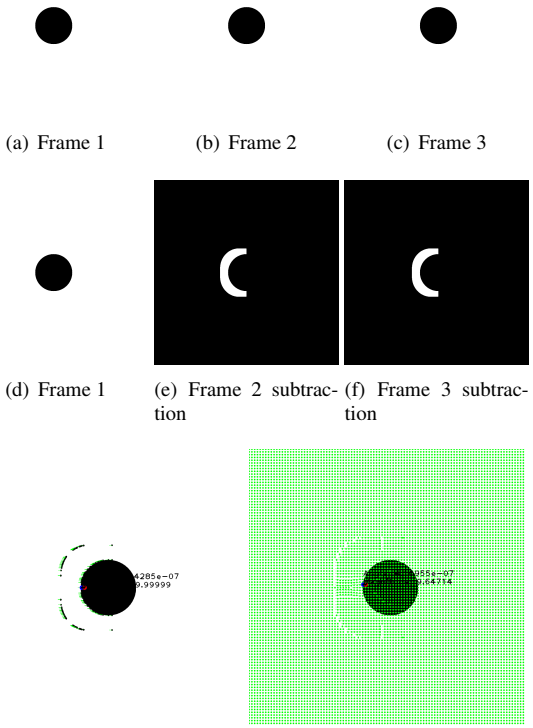
2.4. Validation Experiment

A validation experiment was made for real displacement meassured and movement direction. We made 3 photos of a circle with a well defined displacement pattern of meassure of 10 pixels. Then it was obtained the displacemente calculation for both algorithms, and finally it was meassured the corresponding errors as it is shown on Table 2.

Below are listed Validation Parameters:

- **Conversion Factor:** $0.2\frac{mm}{pixelsunits}$.
- **real movement displacement meassured:** 4mm.

We can see that **FB** algorithm is more stable than Lucas Kanade, which process time change depending on the image conditions. In spite of this it was observed that **LK** algorithm is more accurated and faster. Considering this it was decided to use **LK** for momevent tracking



(a) Frame 1 (b) Frame 2 (c) Frame 3

(d) Frame 1 (e) Frame 2 subtraction (f) Frame 3 subtraction

(g) Lucas Kanade (h) Farnebäck

Figure 3: Validation Experiment and Results for a movement displacement of 10 pixels

implementation.

Results for both algorythms are showed on Figure 4, where also are included the magnitude and angle of the global movement vector meassured without taking into account the time between frames.

3. InMoov Movement Restrictions

Movement restrictions were made based on the range of vision of the robot. Then it is described the method for movement vector estimation used for a real movement meassure.

3.1. Range of Vision Parameters

First its necessary to find the restriction angle θ and the critical occlusion point F, showed on Figure 6 based on the meassures of Figure 5:

$$\theta = 180^\circ - (70^\circ + \alpha)$$
$$\theta = 73.8^\circ$$

$$F = \frac{62.27mm}{2} \tan(\theta)$$
$$F = 107.167mm$$

Therefore, we defined Oclussion O as the set of points which are into the area covered by the triangle $\triangle\ AOB$ in the horizontal plane and the points . As it can be infer, the objects that are into this area cannot be tracking.

3.2. Movement vector estimation

Lets $P = \{P_\sigma, P_\eta\}$ and $P' = \left\{P'_\sigma, P'_\eta\right\}$ the sets of old and new points of the studied object on the farthest (Z_2) and closest (Z_1) spatial object image planes respectively. As farther Z_1 is from Z_2 the distance d between P_σ and P_η is going to be larger than the distance d' between P'_σ and P'_η, obviously, if the real distance $D = \epsilon_1 d$ is equal to $D' = \epsilon_2 d'$, where d and d' are defined as the image distance vectors of the closest and farthest images as follows:

$$d = (i_\eta - i_\sigma, j_\eta - j_\sigma)$$
$$d' = (i'_\eta - i'_\sigma, j'_\eta - j'_\sigma)$$

The goal is to find a vector of functions $K = [K_1(\Delta i)\ K_2(\Delta j)]^T$, to obtain the vector of vertical and horizontal angular displacement $\Omega = [\Omega_1\ \Omega_2]^T$. Where:

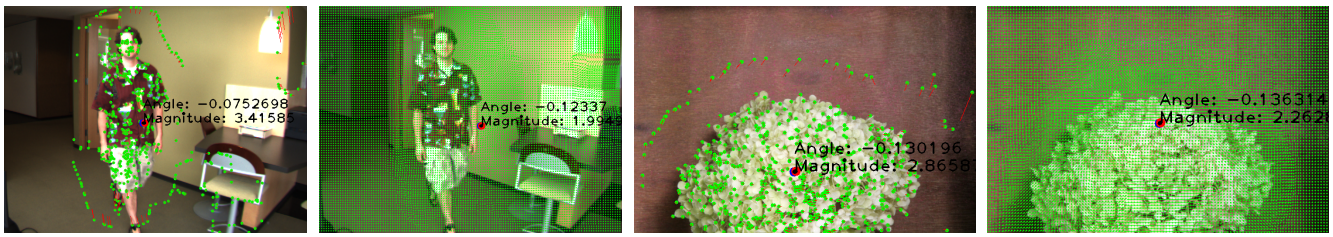
$$\begin{bmatrix} \Omega_1 \\ \Omega_2 \end{bmatrix} = \begin{bmatrix} K_1(i_\eta - i_\sigma) \\ K_2(j_\eta - j_\sigma) \end{bmatrix}$$

The idea was to find K_1 and K_2 making a linear or polinomial regression based on the distribution of points Figure 7 of Ω_1 and Ω_2 as functions of Δi and Δj.

In Table 3 is presented the final Algorithm for the InMoov movement tracking.

4. Acknowledgements

We want to thanks MSC. Marco Antonio Jinete and MSC. Robinson Jiménez for his support during the realization of this project. We also want to thanks Gael Langevin, InMoov designer and creator, for being aware of our work with the InMoov Robot. We give special thanks to the Image Processing *OpenCreator* research group.



(a) Lucas Kanade Walking set (b) Farnebäck Walking set (c) Lucas Kanade Hydrangea set (d) Farnebäk Hydrangea set

Figure 4: Lucas Kanade and Farnebäck algorithms results for the Walking and Hydrangea sets. (a) $\mid \vec{\nu} \mid = 3.4158/t$ and $\measuredangle\vec{\nu} = -0.0752$. (b) $\mid \vec{\nu} \mid = 1.994/t$ and $\measuredangle\vec{\nu} = -0.1233$. (c) $\mid \vec{\nu} \mid = 2.8658/t$ and $\measuredangle\vec{\nu} = -0.1301$. (d) $\mid \vec{\nu} \mid = 2.2625/t$ and $\measuredangle\vec{\nu} = -0.1363$

.

(a) Vertical View (b) Front View (c) Horizontal View (d) Isometric View

Figure 5: InMoov Horizontal, Frontal, Vertical and Isometric views, with the range vision parameters

.

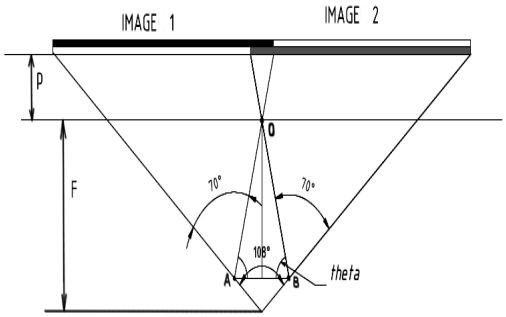


Figure 6: Horizontal vision plane

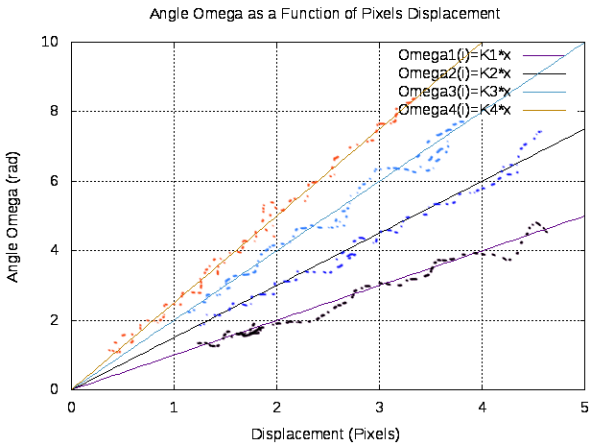


Figure 7

5. References

[1] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. Proceedings of Imaging Understanding Workshop, pp. 121-130. 1981.

[2] Jiambo Shi and Carlo Tomasi. Good Features to Track. IEE Conference on Computer Vision and Pattern Recognition (CVPR94). Seattle. 1994.

Optical Flow Implemented Algorythm
1. Obtain the first three frames; *Initial, Previous, Actual* and the time t between *Previous and Actual* **2.** XYZ an white thresholding transformations **3.** *Previous=Previous-Initial* & *Actual=Actual-Initial*. **4.** LK optical Flow between *Previous* & *Actual* $\rightarrow \sigma_n$, η_n. **5.** Estimate $\vec{\nu}$ **6.** $\vec{\nu}$ Conversion for angle displacement. **7.** *Forward Kinematics* $\rightarrow$ Actuators Signals. **8.** end

Table 3

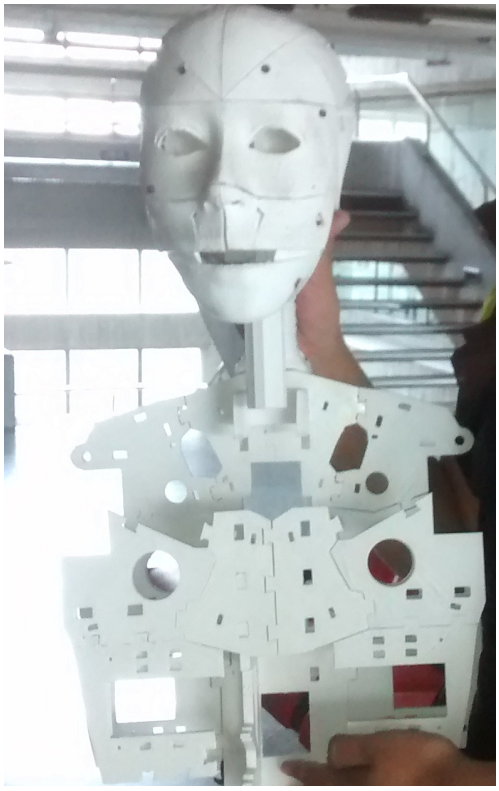


Figure 8: InMoov Robot

[3] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. International Journal of Computer Vision 56(3), pp 221–255. 2004.

[4] Shaul Oron, Aharon Bar-Hillel and Shai Avidan. Extended Lucas-Kanade Tracking. European Conference on Computer Vision. 2014.

[5] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm. Intel Corporation, Microprocessor Research Labs. 2000.

[6] Deging Sun *et al*. Learning Optical Flow. European Conference on Computer Vision. 2008.

[7] Berthold K.P. Horn and Brian G. Schunck Determining Optical Flow. Artificial Intelligence (MA 02139). 1980.

[8] Klaus Voss. Discrete Images, Objects, and Functions in Zn. ISBN-13: 978-3-642-46781-3. Springer Verlag. Jena, Germany. 1991.

[9] Richard M. Murray, Zexiang Li and S. Shankar Sastry. A Mathematical Introduction to Robotic Manipulation. CRC Press. 1994.

[10] David Feil-Seifer and Maja J. Mataric. Human-Robot Interaction. Invited contribution to Encyclopedia of Complexity and Systems Science, Robert A. Meyers (eds.), Springer New York, 4643-4659, 2009.

[11] Michael A. Goodrich and Alan C. Schultz. Human–Robot Interaction: A Survey. Foundations and Trends in Human–Computer Interaction Vol. 1, No. 3. pp 203–275. 2007.

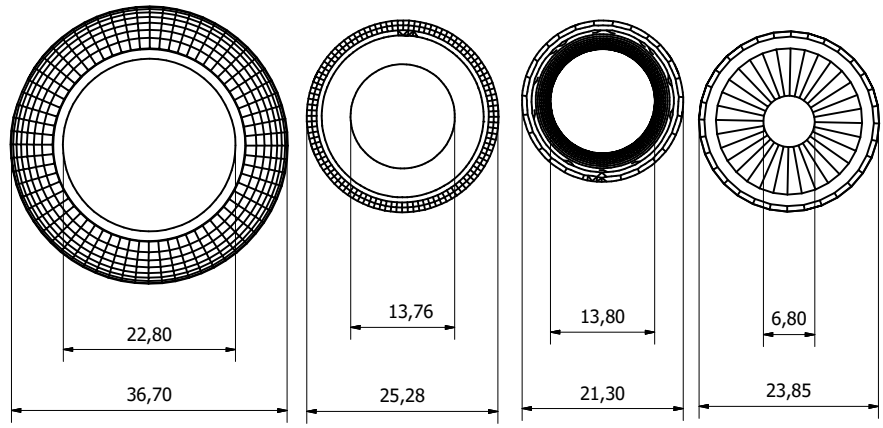
[12] Gunnar Farnebäck. Two-Frame Motion Estimation Based on Polynomial Expansion. 13th Scandinavian Conference, SCIA. pp 363-370. 2003.

[13] David J. Fleet and Yair Weiss. Optical Flow Estimation. Handbook of Mathematical Models in Computer Vision. pp 237-257. 2006.

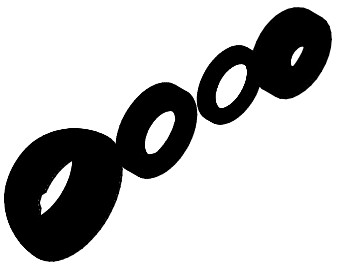
[14] Adulmalik D. Mohammed and Tim Morris. Optical Flow Estimation Using Local Features. Proceedings of the World Congress on Engineering. Vol 1. pp 562-565. 2015.

[15] Simon Baker *et al*. A Database and Evaluation Methodology for Optical Flow. International Journal of Computer Vision. Vol 92. pp 1–31. 2011.

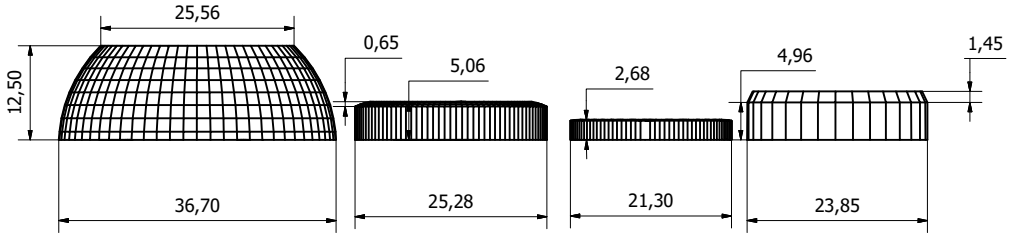
B. Planos



VISTA SUPERIOR

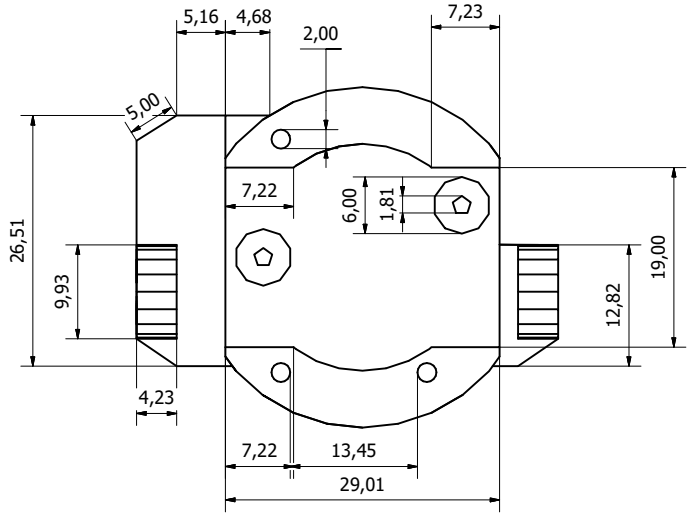


VISTA GENERAL

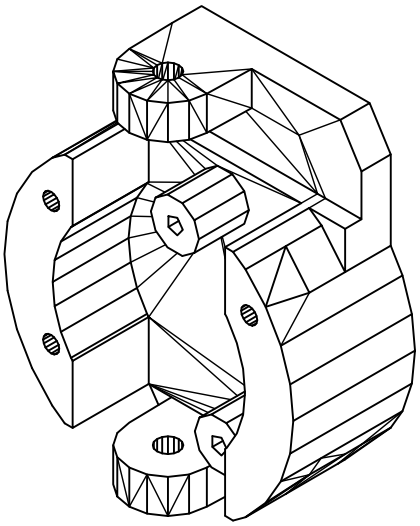


VISTA LATERAL

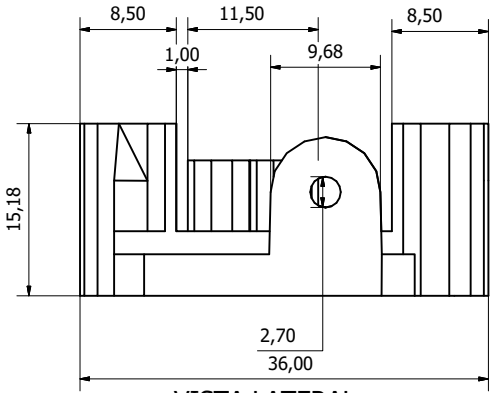
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	1.ipt	Escala 2: 1	1



VISTA SUPERIOR

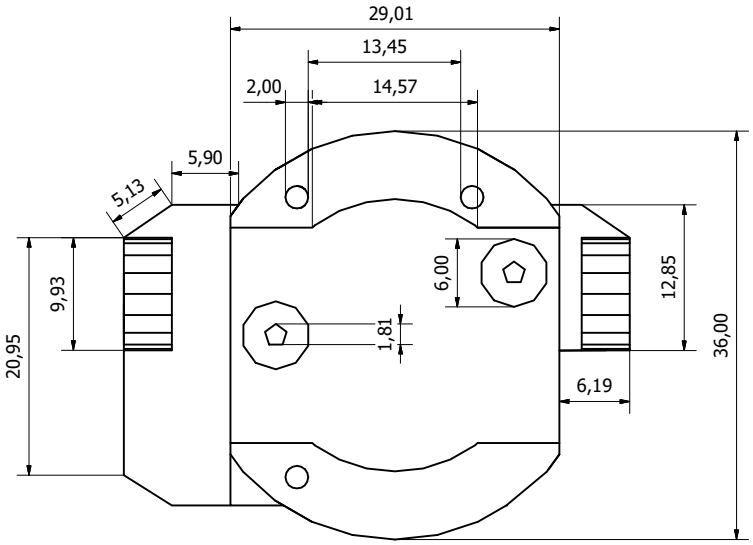


VISTA GENERAL

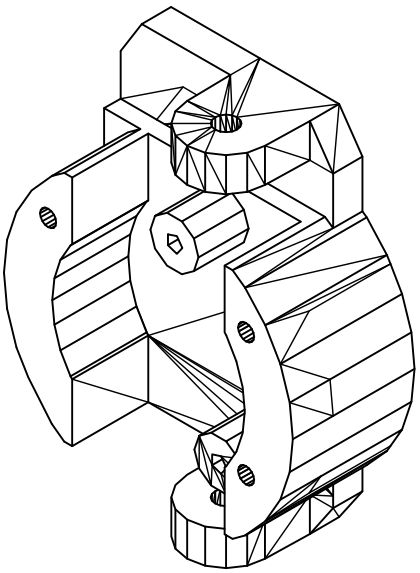


VISTA LATERAL

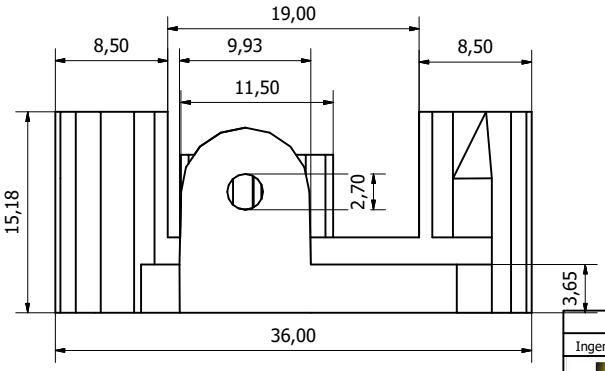
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	2.ipt	Escala 2.5 : 1	2



VISTA SUPERIOR



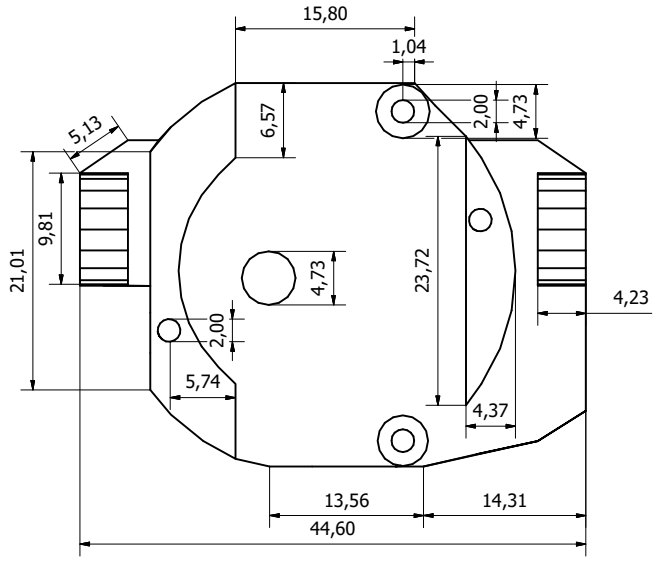
VISTA GENERAL



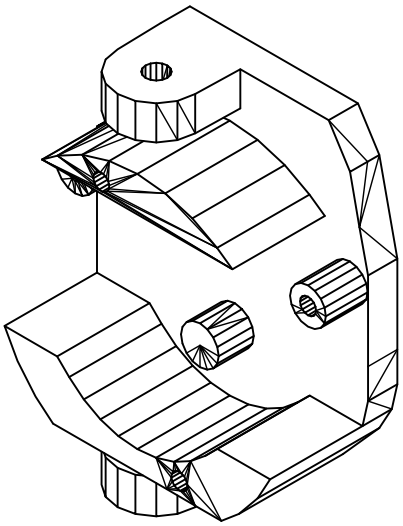
VISTA LATERAL

Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	

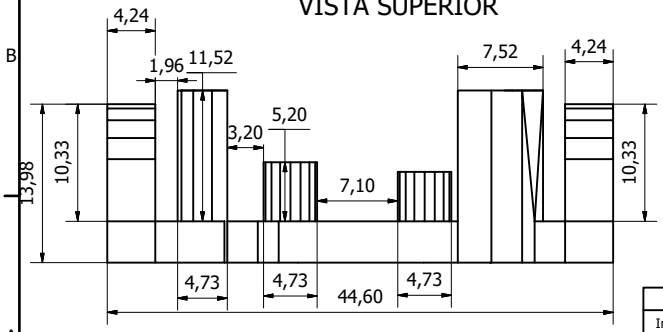
Universidad Piloto de Colombia	Universidad Piloto de Colombia	3.ipt	Escala 3 : 1	3



VISTA SUPERIOR



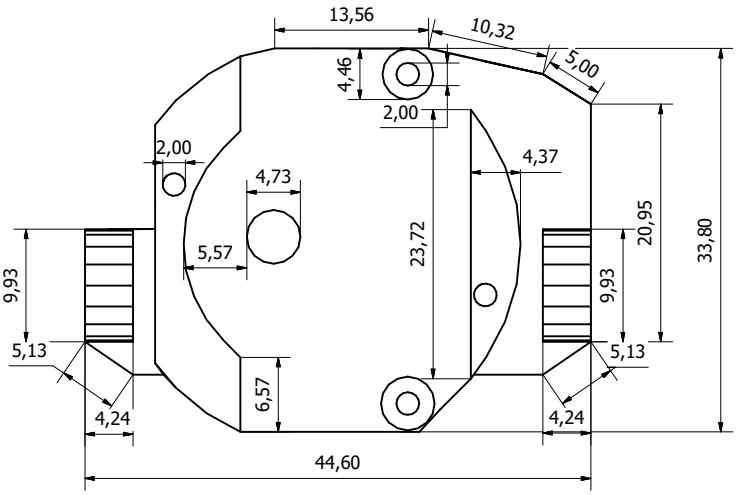
VISTA GENERAL



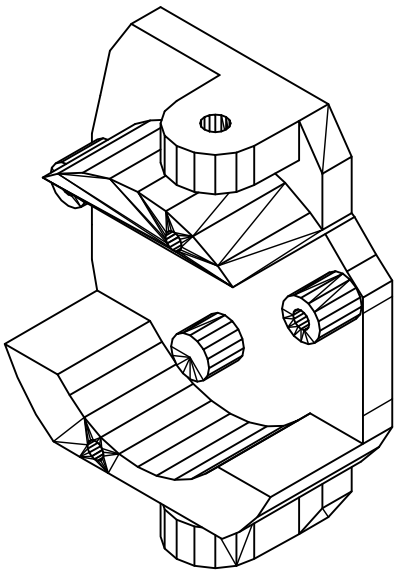
VISTA LATERAL

Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	

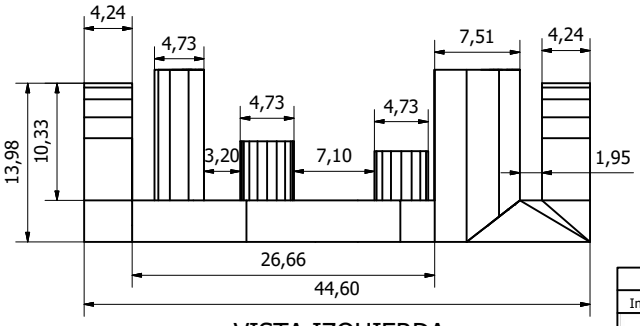
Universidad Piloto de Colombia	Universidad Piloto de Colombia	4.ipt	Escala 3 : 1	4



VISTA FRONTAL

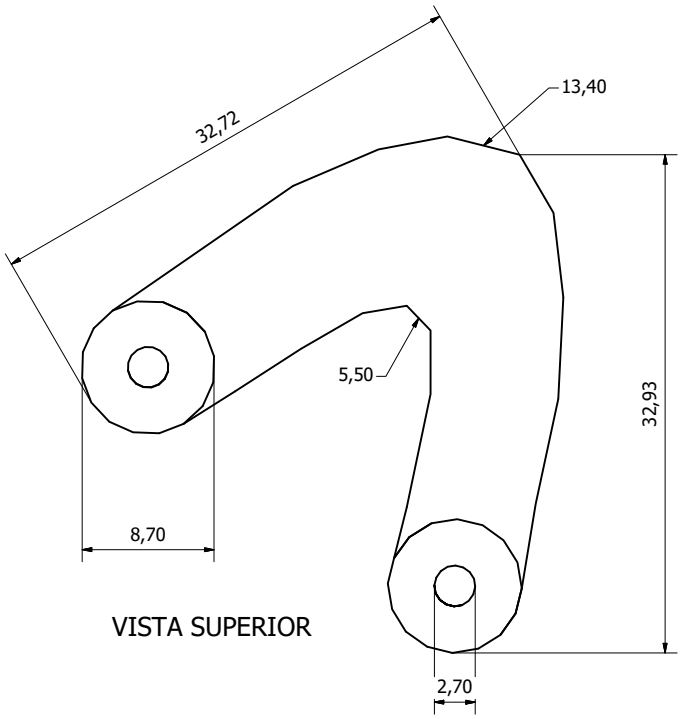


VISTA GENERAL

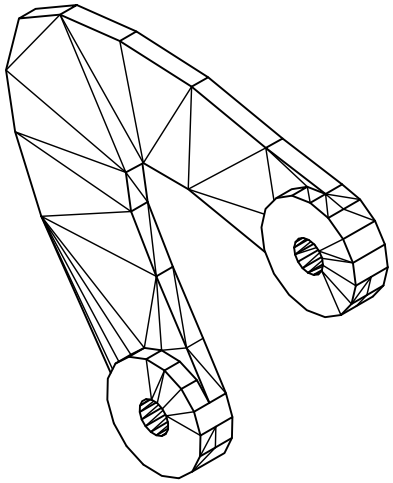


VISTA IZQUIERDA

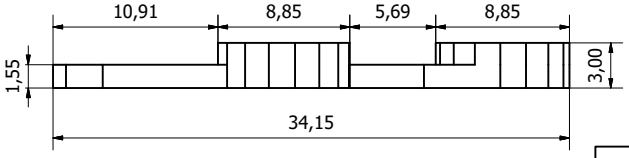
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	5.ipt	Escala 3 : 1	5



VISTA SUPERIOR

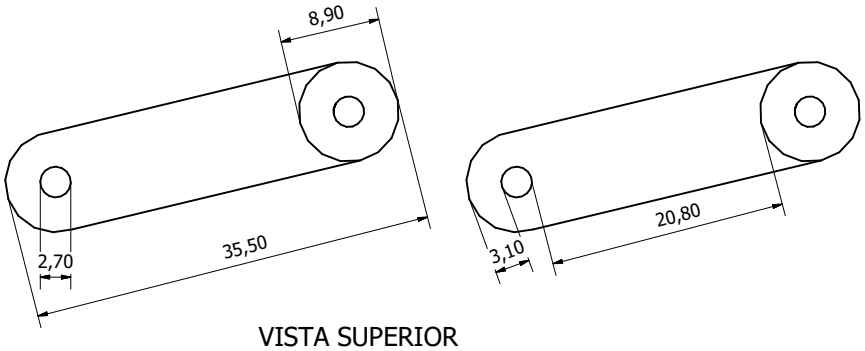


VISTA GENERAL

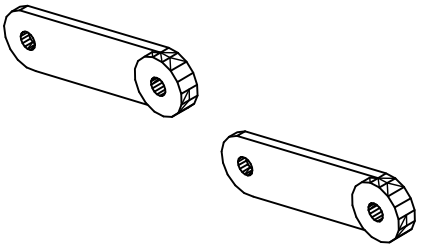


VISTA FRONTAL

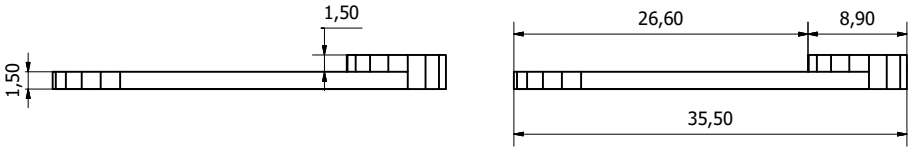
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	6.ipt	Escala 4 : 1	6



VISTA SUPERIOR

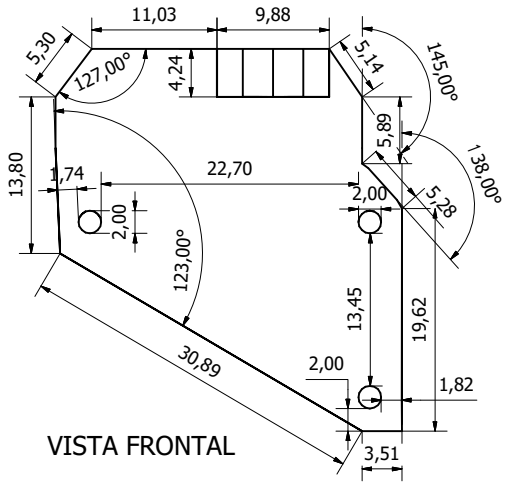


VISTA GENERAL

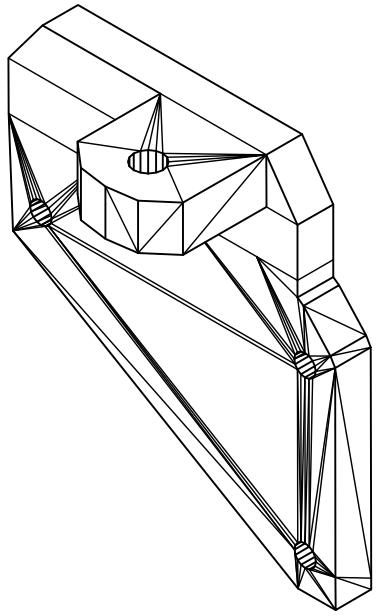


VISTA LATERAL

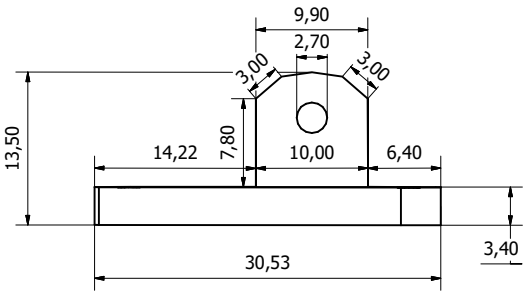
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	7.ipt	Escala 3 : 1	7



VISTA FRONTAL

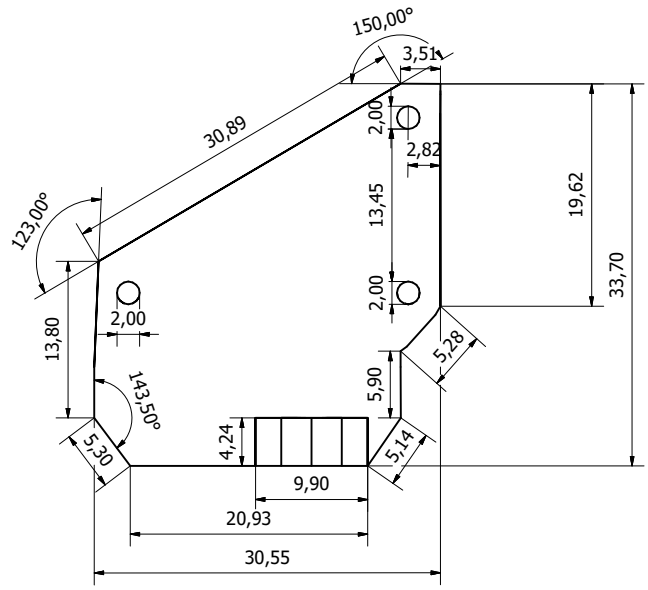


VISTA GENERAL

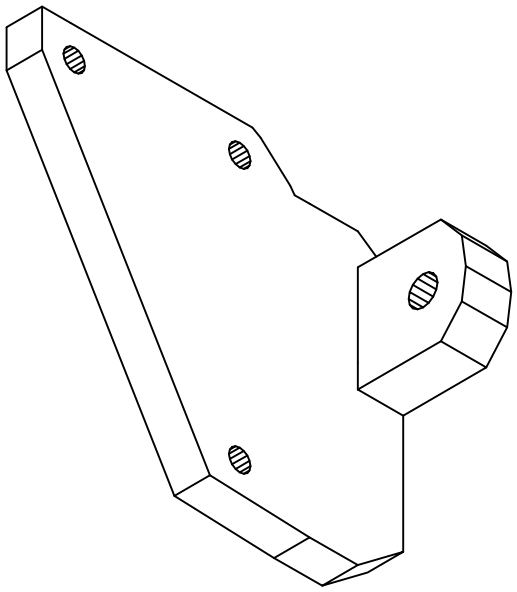


VISTA INFERIOR

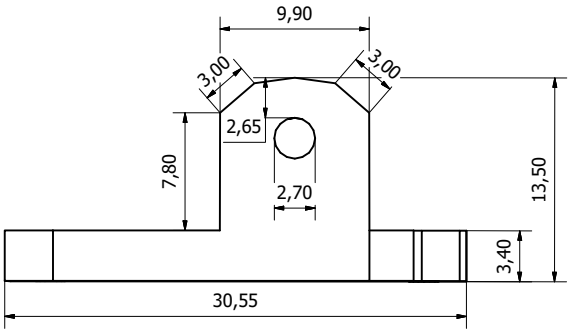
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	8.ipt	Escala 3 : 1	8



VISTA FRONTAL

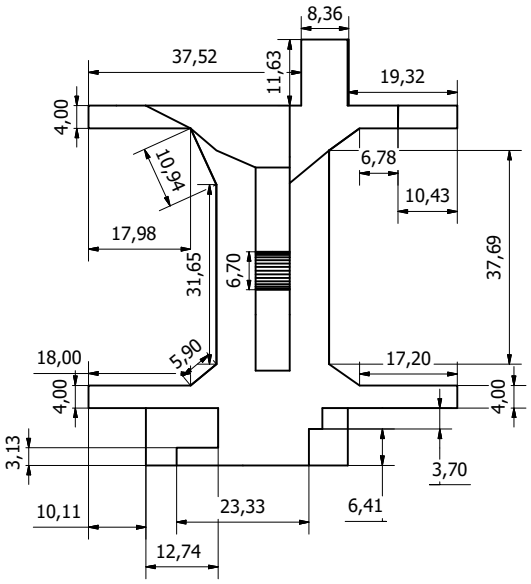


VISTA GENERAL

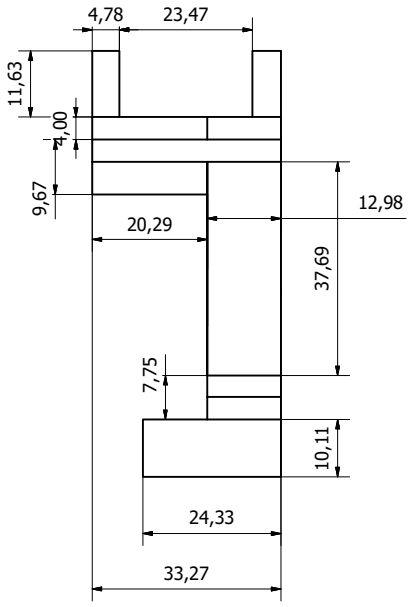


VISTA INFERIOR

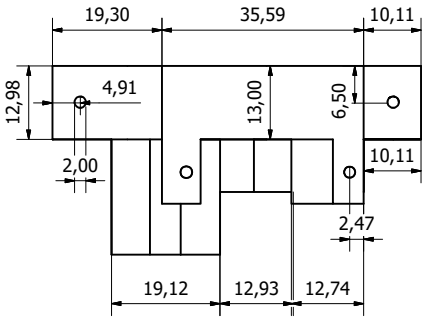
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	9.ipt	Escala 3 : 1	9



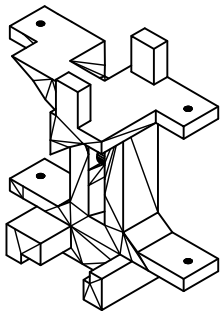
VISTA FRONTAL



VISTA DERECHA

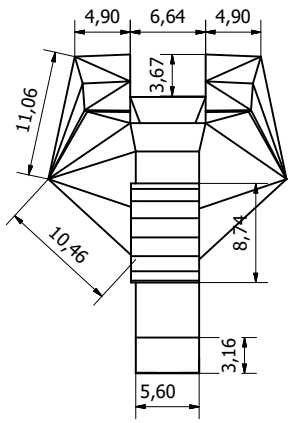


VISTA INFERIOR

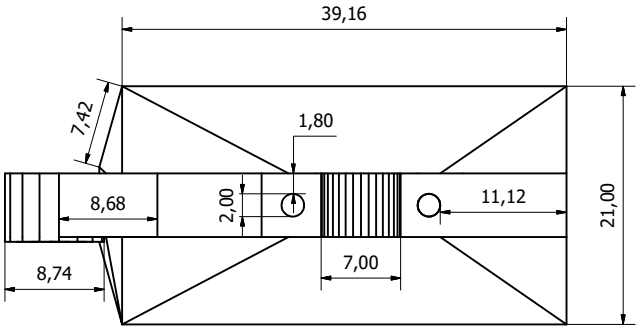


VISTA GENERAL

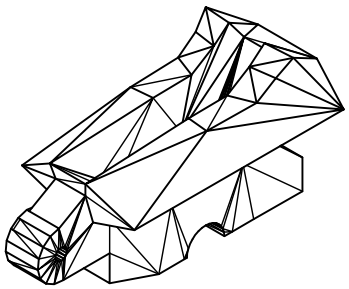
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	10.ipt	Escala 1.5 : 1	10



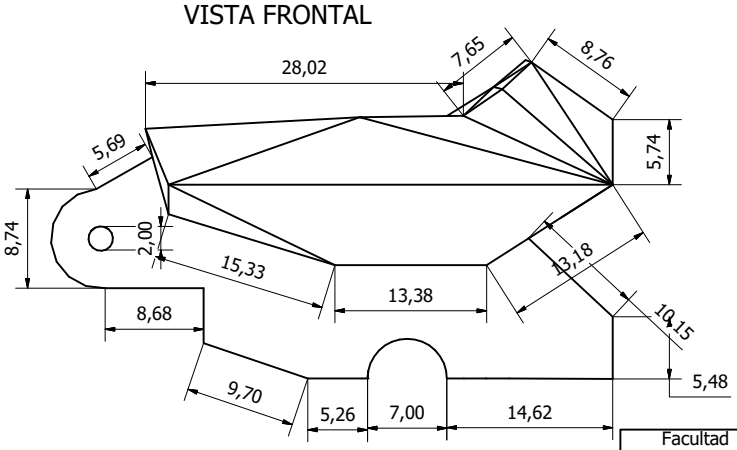
VISTA FRONTAL



VISTA INFERIOR

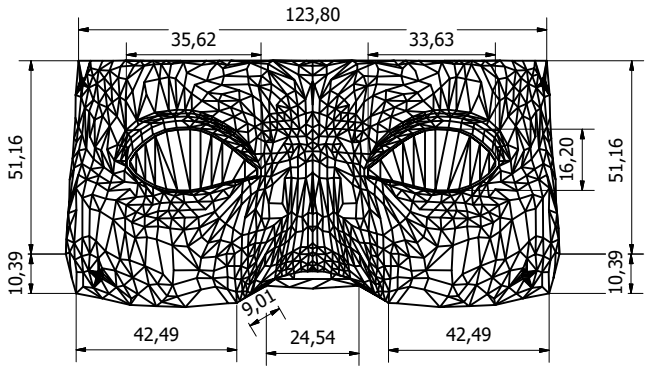


VISTA GENERAL

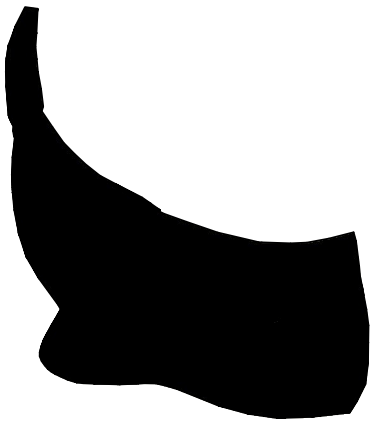


VISTA DERECHA

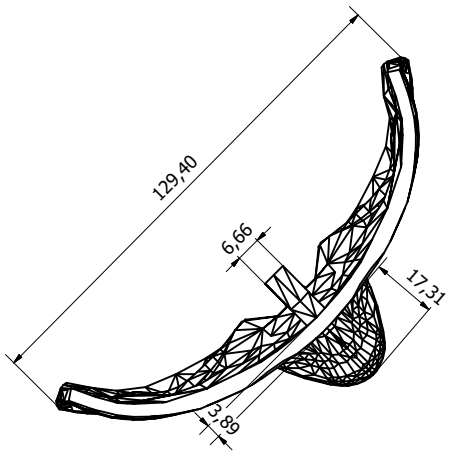
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	11.ipt	Escala 3 : 1	11



VISTA FRONTAL

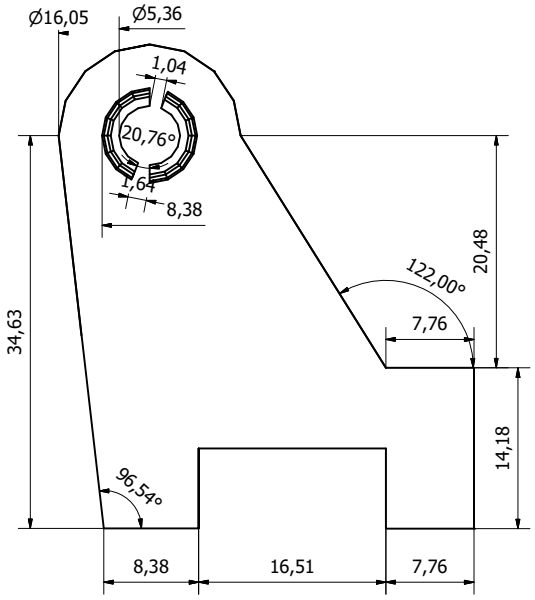


VISTA GENERAL

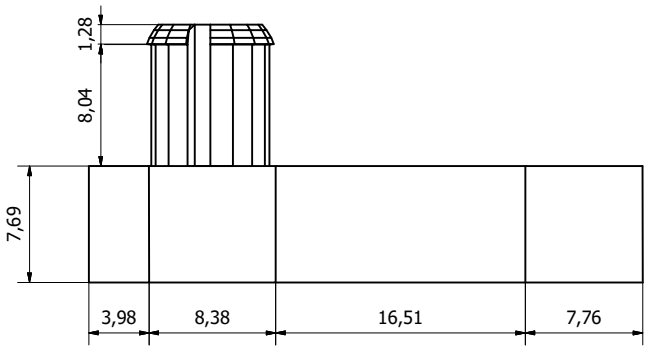


VISTA SUPERIOR

Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	12.ipt	Escala 1 : 1	12

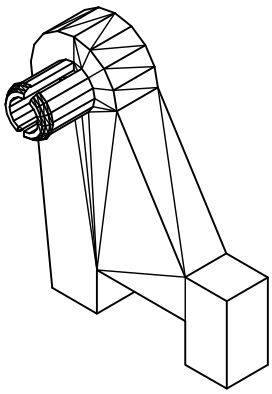


VISTA FRONTAL

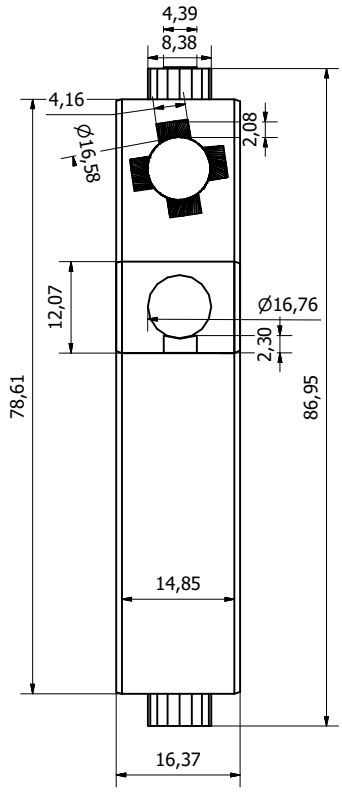


VISTA INFERIOR

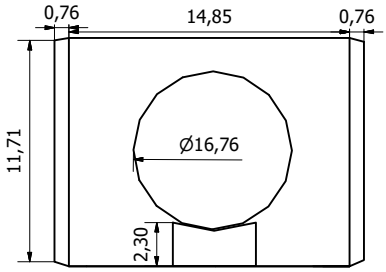
VISTA GENERAL



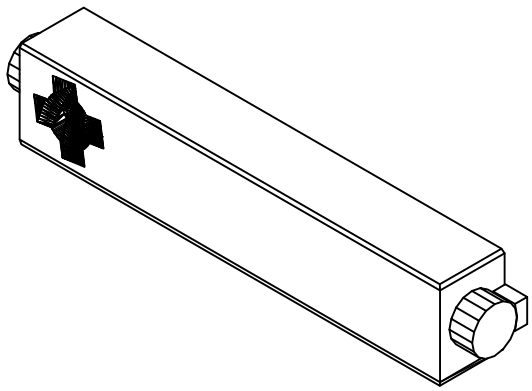
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	15.ipt	Escala 3: 1	13



VISTA DERECHA

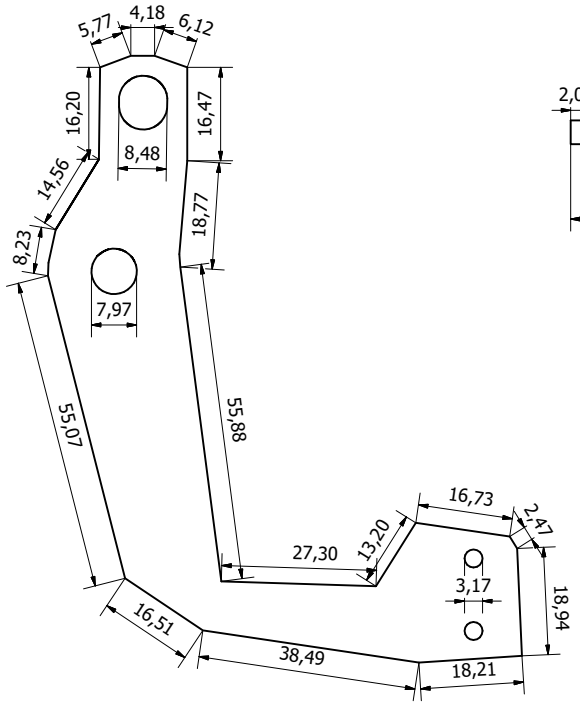


VISTA SUPERIOR

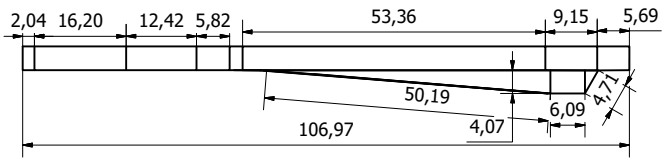


VISTA GENERAL

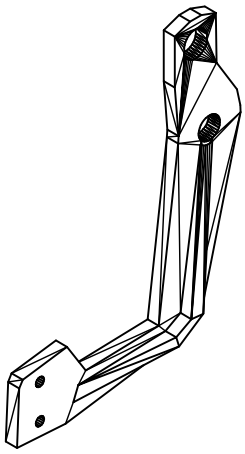
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia		Escala 2 : 1	14



VISTA POSTERIOR

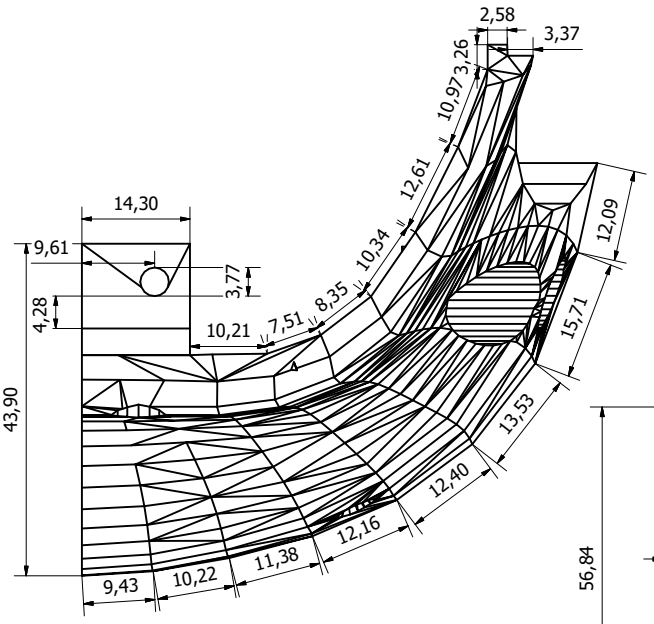


VISTA DERECHA

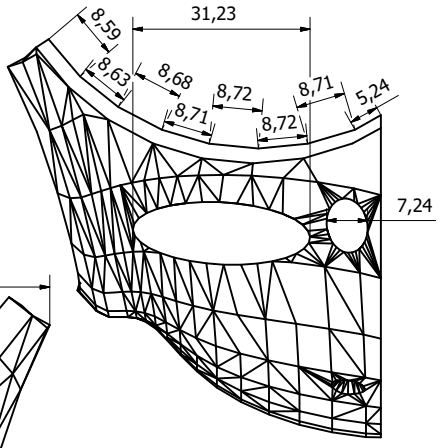


VISTA GENERAL

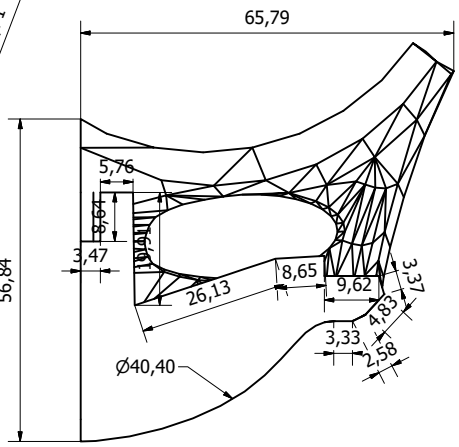
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	14.ipt	Escala 1,5 : 1	15



VISTA FRONTAL

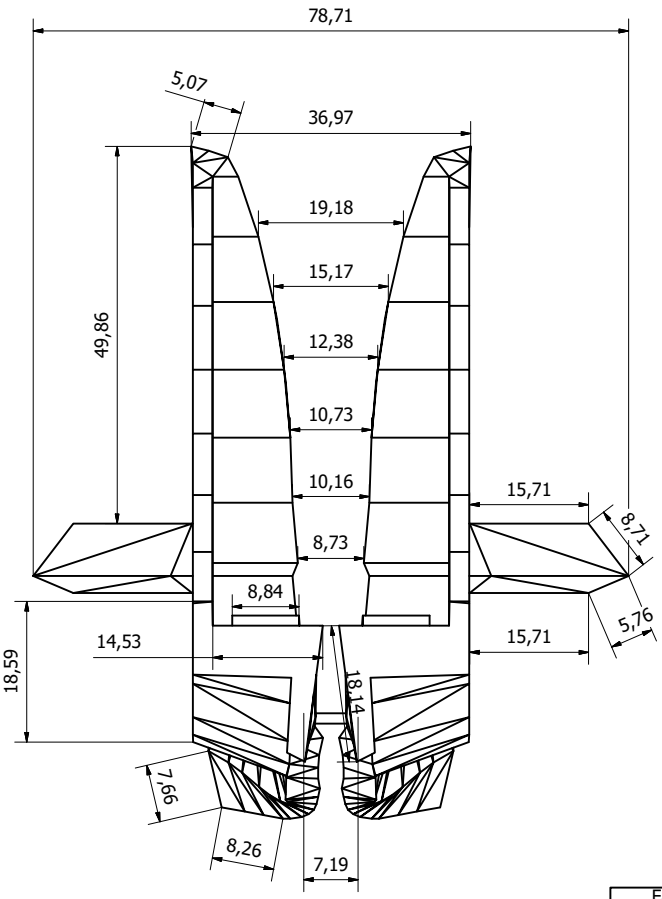


VISTA DERECHA

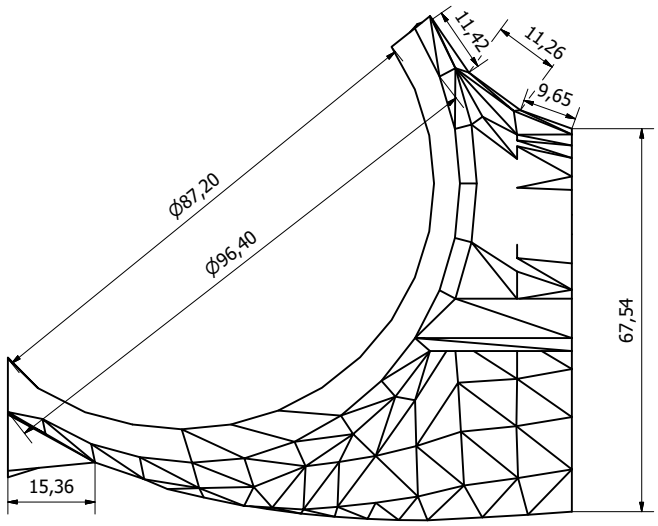


VISTA IZQUIERDA

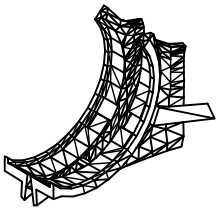
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	17.ipt	Escala 2 : 1	16



VISTA FRONTAL

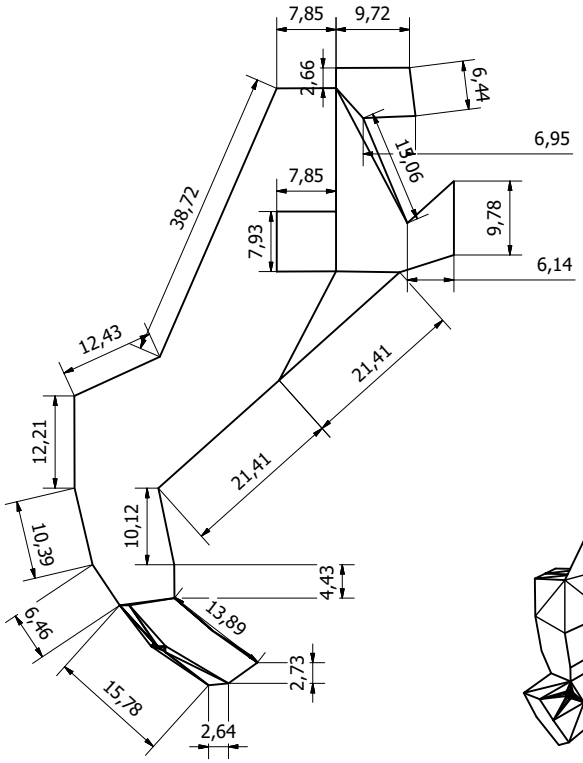


VISTA DERECHA

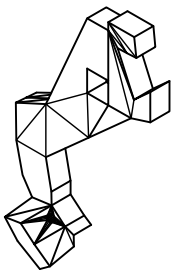


VISTA GENERAL

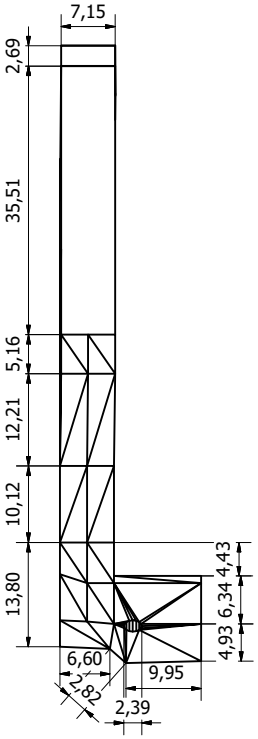
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	18.ipt	Escala 2: 1	17



VISTA FRONTAL

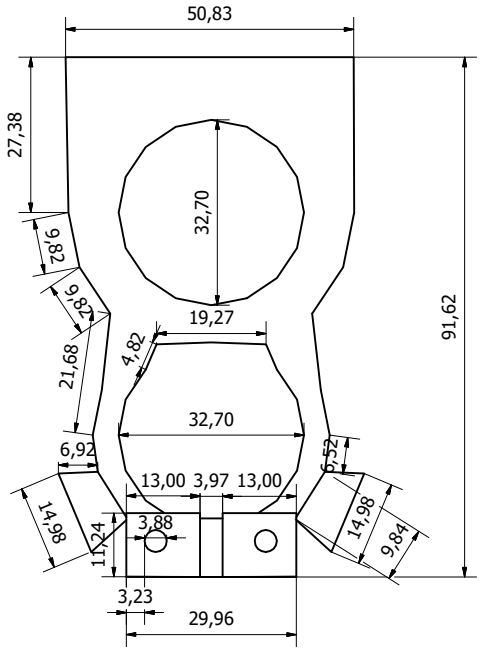


VISTA GENERAL

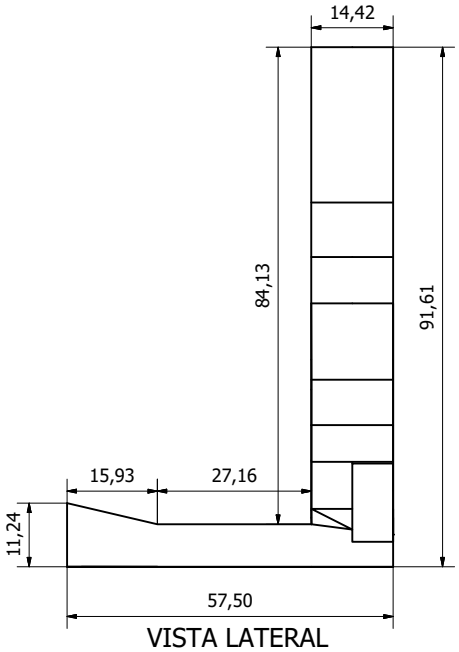


VISTA IZQUIERDA

Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	21.ipt	Escala 2: 1	18



VISTA FRONTAL

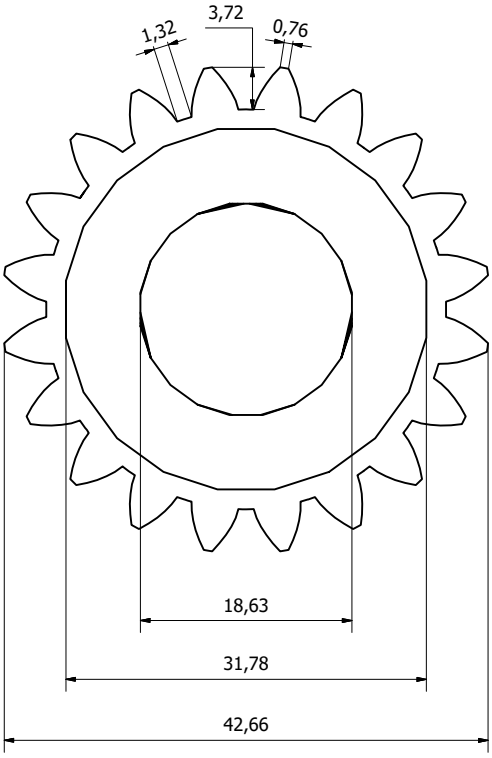


VISTA LATERAL

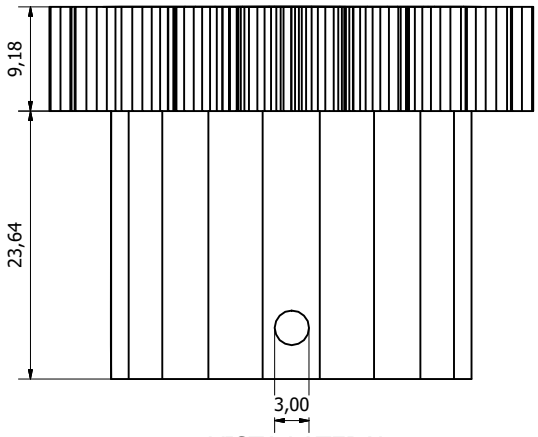
VISTA GENERAL

Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	

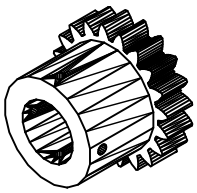
Universidad Piloto de Colombia	Universidad Piloto de Colombia	22.ipt	Escala 1.5: 1	19



VISTA SUPERIOR

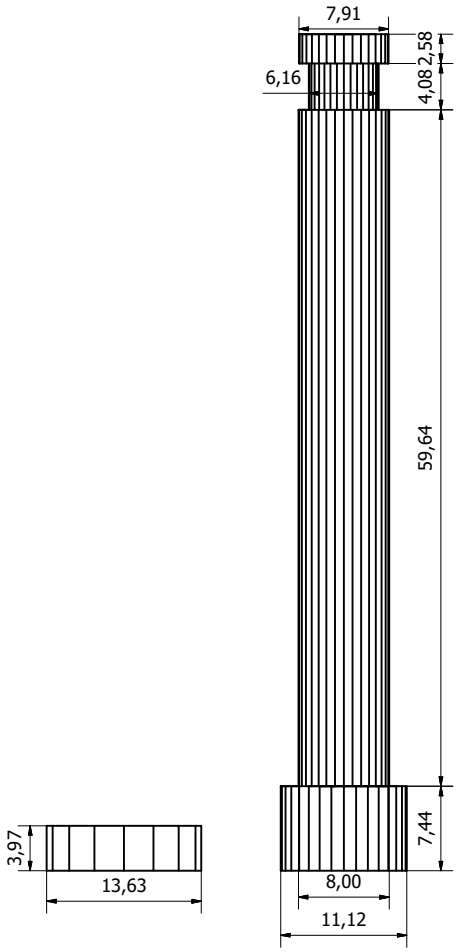


VISTA LATERAL

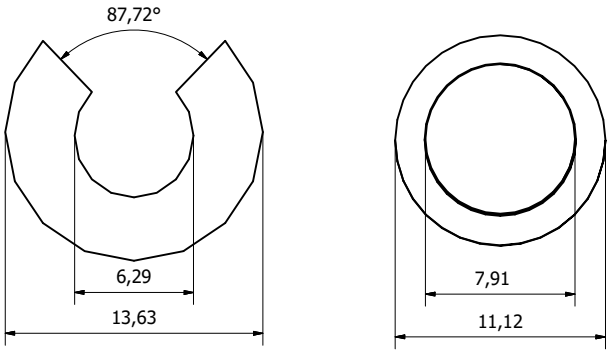


VISTA GENERAL

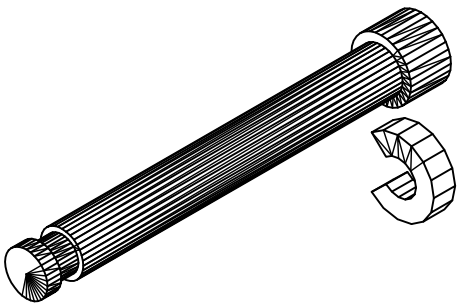
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	23.ipt	Escala 3: 1	20



VISTA LATERAL

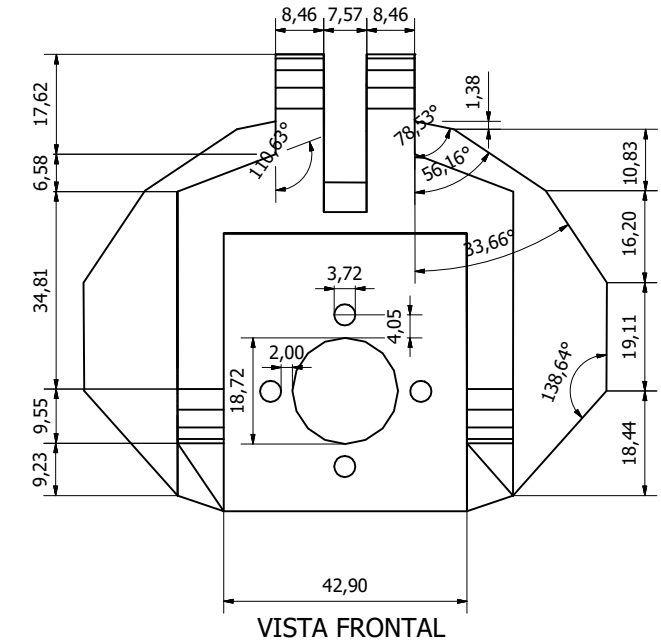


VISTA FRONTAL

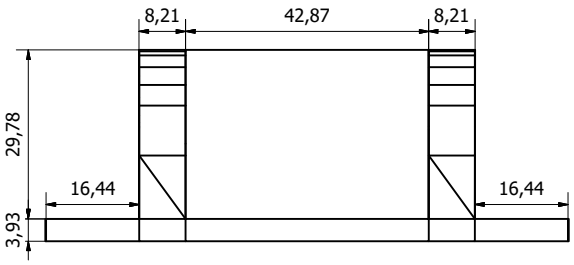


VISTA GENERAL

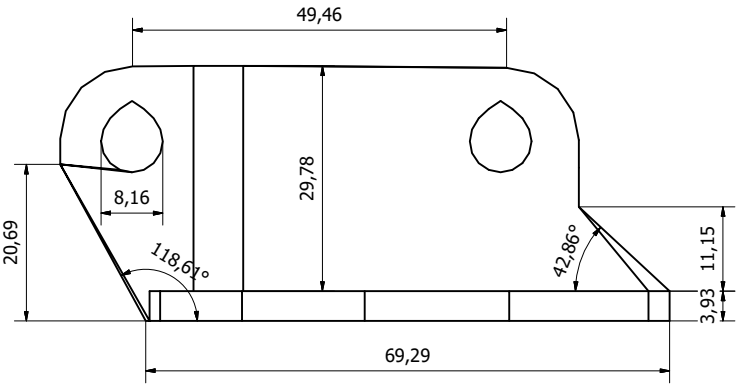
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	24.ipt	Escala 5: 1	21



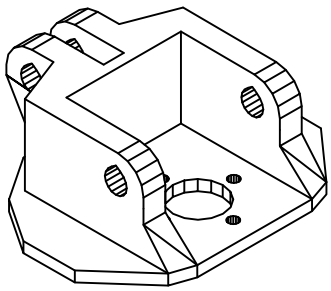
VISTA FRONTAL



VISTA INFERIOR

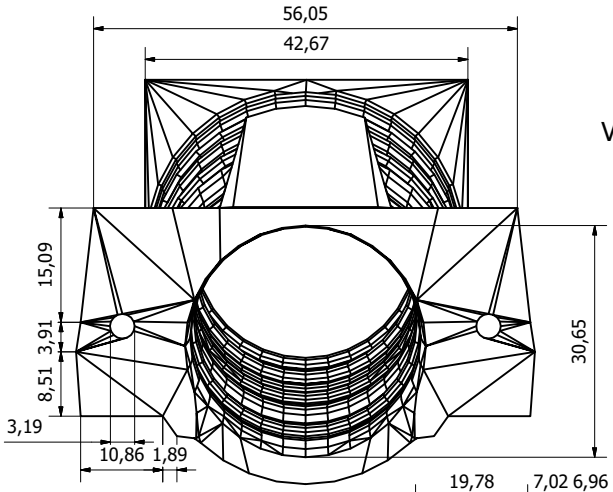


VISTA IZQUIERDA

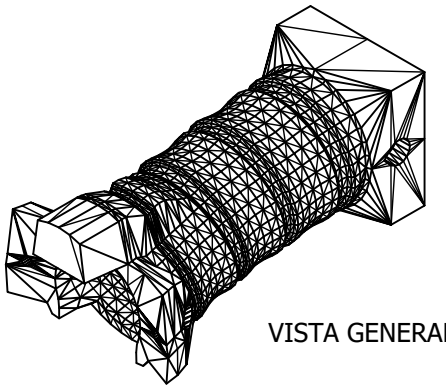


VISTA GENERAL

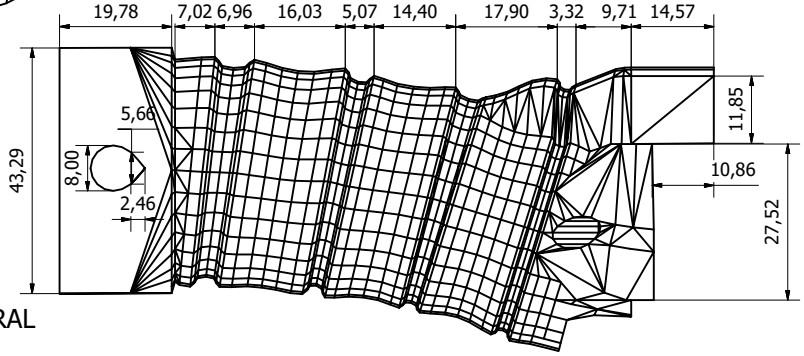
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	25.ipt	Escala 1.5 : 1	22



VISTA FRONTAL

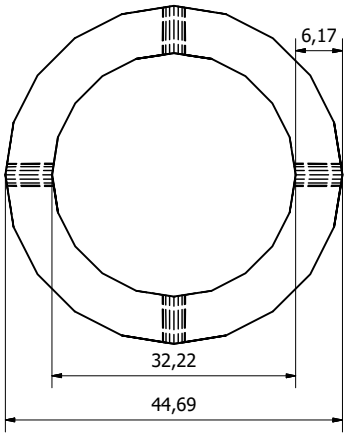


VISTA GENERAL

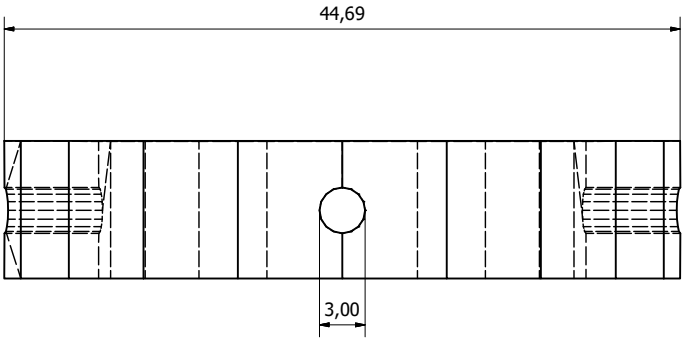


VISTA LATERAL

Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	26.ipt	Escala 2: 1	23

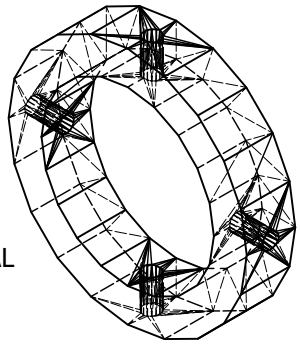


VISTA FRONTAL

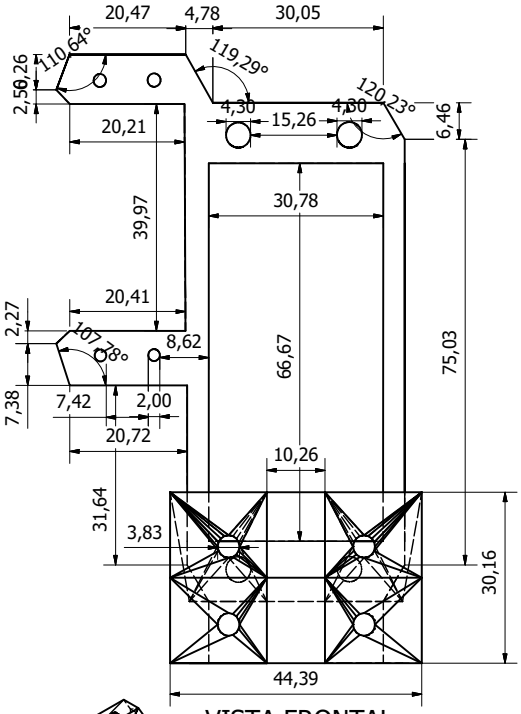


VISTA LATERAL

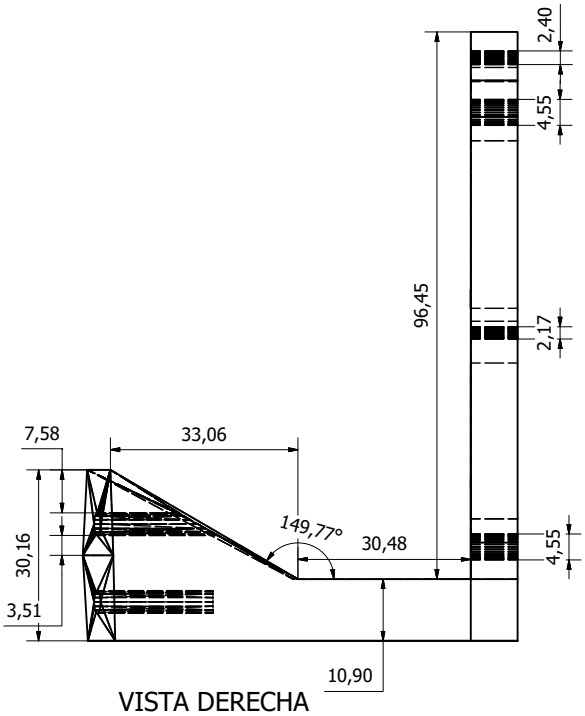
VISTA GENERAL



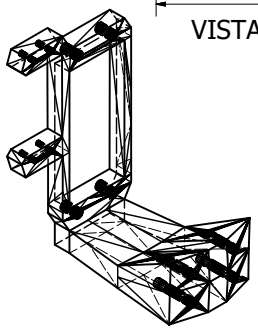
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	27.ipt	Escala 2: 1	24



VISTA FRONTAL

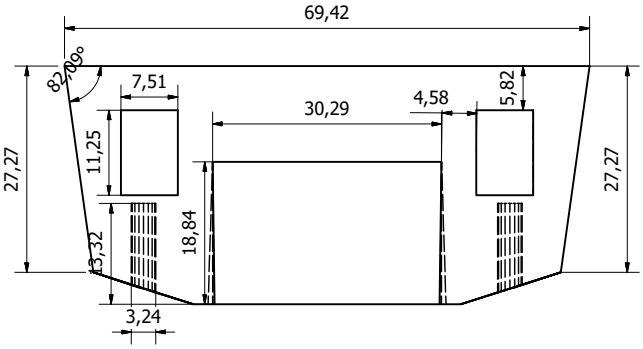


VISTA DERECHA

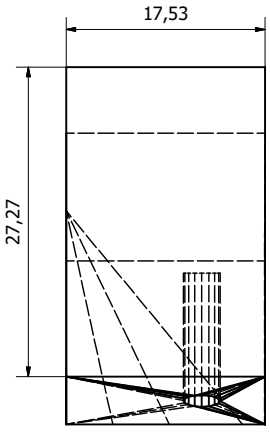


VISTA GENERAL

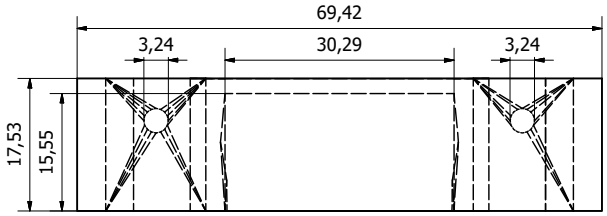
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	29.ipt	Escala 1.5 : 1	25



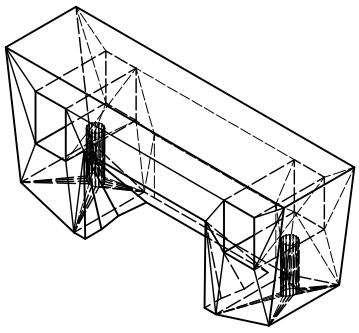
VISTA FRONTAL



VISTA DERECHA

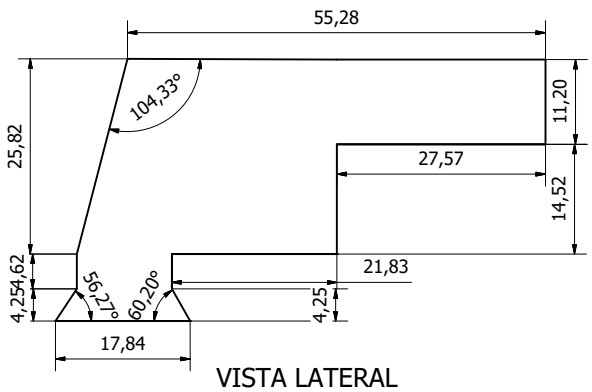


VISTA SUPERIOR

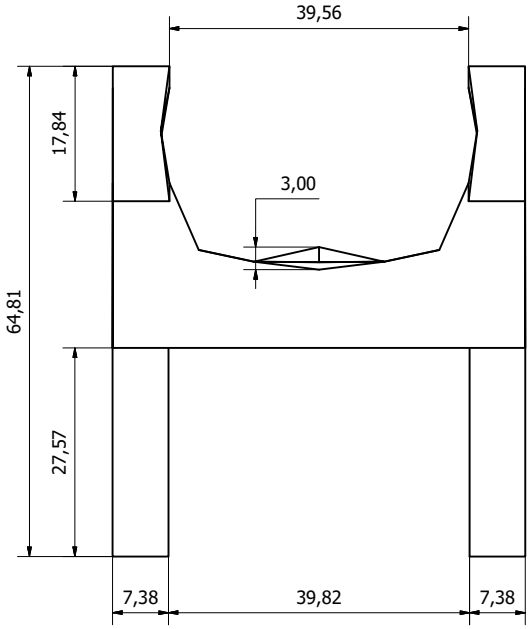


VISTA GENERAL

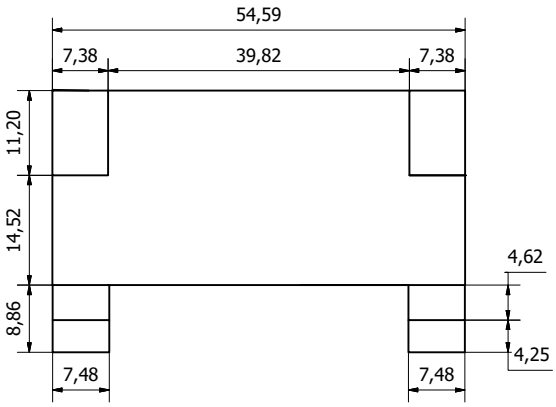
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	30.ipt	Escala 2: 1	26



VISTA LATERAL

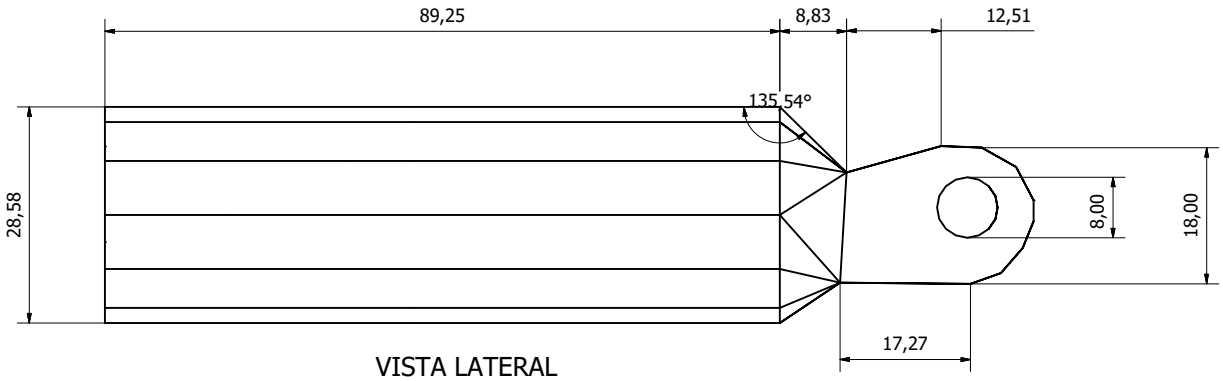


VISTA FRONTAL

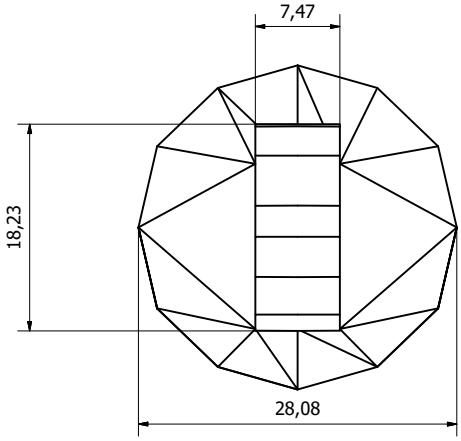


VISTA INFERIOR

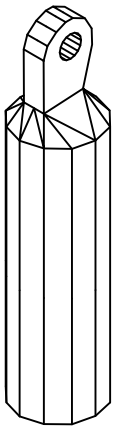
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	31.ipt	Escala 2: 1	27



VISTA LATERAL

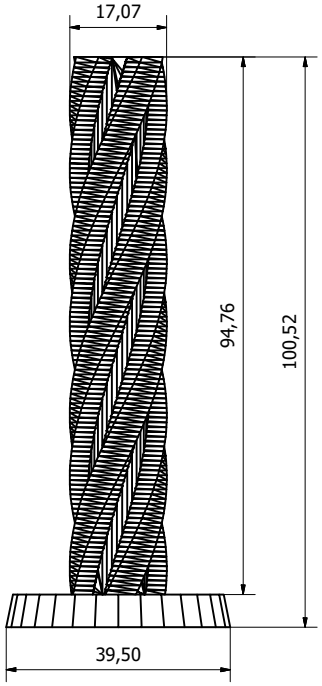


VISTA SUPERIOR

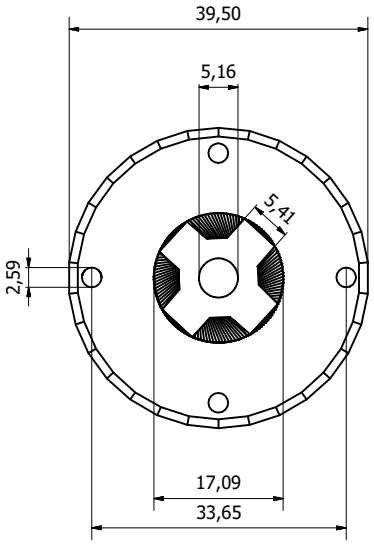


VISTA GENERAL

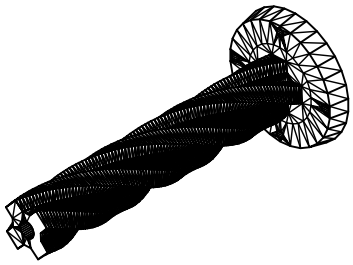
Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	32.ipt	Escala 2: 1	28



VISTA LATERAL



VISTA INFERIOR



VISTA GENERAL

Facultad	Dibujado por:	Revisado por:	Aprobado por:	Fecha
Ingenieria Mecatronica	L Vargas	N Ortiz	M Jinete	
Universidad Piloto de Colombia	Universidad Piloto de Colombia	33.ipt	Escala 1.5 : 1	29

C. Códigos en C++

C.1. Librerías

C.1.1. Algoritmo SB

```
/*This library was created by Nicolas Ortiz and the research group
    on Image Processing of the
Universidad Piloto de Colombia.
*/

#include <opencv2/core/utility.hpp>
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>
#include <ctype.h>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <math.h>
#include <cmath>
#include <sstream>
#include <unistd.h>

using namespace cv;
using namespace std;

void InverseKinematics(double VMx, double VMy){

  /*Inverse Kinematics*/
  double alphai=0.0722;
  double alphaj=0.0708;
  double L=0.12445; // meters
  double phi=(90-15.37)*3.1415/180;
  double Beta,Lambda;
  double theta1,theta2;
  double t1, t2, t;

  Beta=alphai*VMx+phi;
  Lambda=alphaj*VMy;

  theta1=acos(cos(Beta)*cos(Lambda))-phi;

  if(phi==-theta1){
    theta2=0;
  }
  else{
    theta2=acos(sin(Beta)*cos(Lambda)/sin(phi+theta1));
  }
  t1=int(theta1*180/3.1415);
```

```
46    t2=int(theta2*180/3.1415);
47
48    if(t1<=t2){
49      t=t1/375000;
50    }
51    else{
52      t=t2/375000;
53    }
54
55    int data[] ={t1,t2};
56    FILE *file;
57    file = fopen("/dev/ttyACM0","w");  //Opening device file
58    int i = 0;
59        for(i = 0 ; i < 2 ; i++){
60            fprintf(file,"%d",data[i]); //Writing to the file
61      if(i==0) fprintf(file,"%c",','); //To separate digits
62      cout << "theta" << i << ": " << data[i] << endl;
63            waitKey(t+300);
64        }
65        fclose(file);
66    /*End*/
67  }
68
69  Mat Subtraction(Mat actu, Mat prev, Mat subs, Mat frame){
70
71    Mat subs1, subs2;
72    Mat kernel= Mat::ones(3, 3, CV_8UC1);
73    imwrite("frame.png", subs);
74    cvtColor(subs,subs,COLOR_RGB2GRAY);
75    cvtColor(prev,prev,COLOR_RGB2GRAY);
76    cvtColor(actu,actu,COLOR_RGB2GRAY);
77    subs1=prev-subs;
78    subs2=actu-prev;
79    adaptiveThreshold( subs1, subs1, 1, CV_ADAPTIVE_THRESH_MEAN_C,
       CV_THRESH_BINARY_INV,51,5);
80    dilate(subs1, subs1, kernel);
81    multiply(prev,subs1,prev);
82    adaptiveThreshold( subs2, subs2, 1, CV_ADAPTIVE_THRESH_MEAN_C,
       CV_THRESH_BINARY_INV,51,5);
83    dilate(subs2, subs2, kernel);
84    multiply(actu,subs2,actu);
85
86    double s = cv::sum( actu )[0];
87    if(s>=3500){
88    double s=cv::sum(prev)[0];
89    if(s>=3500){
90    //fastNlMeansDenoising(prev,prev,3,17,21);
91    //fastNlMeansDenoising(actu,actu,3,17,21);
92    double posx, posy, posxo, posyo; int i=0; double h,l; int Yo, Yf
       , Xo, Xf;
93    posxo=0; posyo=0; posx=0; posy=0; int x,y;
94
```

```
95  for (y = 0; y < prev.rows; ++y) {
96        for (x = 0; x < prev.cols; ++x) {
97      if(prev.at<uchar>(y,x)>=30){
98        posxo += x; posyo +=y;
99        i += 1;
100     }
101   }
102 }
103
104 posxo=int(posxo/i); posyo=int(posyo/i);
105 i=0;
106
107 for (y = 0; y < actu.rows; ++y) {
108       for (x = 0; x < actu.cols; ++x) {
109     if(actu.at<uchar>(y,x)>=30){
110       posx += x; posy +=y;
111       if(i==0){
112         Xo=x;Yo=y;
113       }
114       i += 1; Yf=y; Xf=x;
115     }
116   }
117 }
118
119 posx=int(posx/i); posy=int(posy/i);
120
121 //cout << int(posx-posxo) << setw(10) << int(posy-posyo) << setw
      (10) << i << endl;
122 //cout << posx << setw(10) << posy << endl;
123 //cout << int(posxo) << setw(10) << int(posyo) << endl;
124
125 //rectangle(frame,cvPoint(Xo,Yo),cvPoint(Xf,Yf),Scalar(255,0,0)
      ,2,8,0);
126 circle( frame, cvPoint(posxo,posyo), 8, Scalar(255,0,0), -1, 8);
127 circle( frame, cvPoint(posx,posy), 8, Scalar(0,0,255), -1, 8);
128 line(frame,cvPoint(posxo,posyo),cvPoint(posx,posy),Scalar
      (0,255,0),2,8,0);
129
130 double VM=sqrt((posxo-posx)*(posxo-posx)+(posyo-posy)*(posyo-
      posy));
131 double Phi, VMx, VMy;
132
133 if(posxo-posx!=0){
134   Phi=atan((posy-posyo)/(posx-posxo));
135 }
136 else if(posy-posyo>0){
137   Phi=3.1415/2;
138 }
139 else if(posy-posyo<0){
140   Phi=-3.1415/2;
141 }
142
```

```
143   VMx=posx-posxo;
144   VMy=posy-posyo;
145
146   InverseKinematics(VMx,VMy);
147
148   std::ostringstream strs;
149   strs << VM;
150   std::string str = strs.str();
151   std::ostringstream strs2;
152   strs2 << Phi;
153   std::string str2 = strs2.str();
154   putText(frame, "Magnitude: "+str, Point(posx,posy),
        FONT_HERSHEY_PLAIN, 2, CV_RGB(255,0,0), 2);
155   putText(frame, "Angle: "+str2, Point(posx,posy-30),
        FONT_HERSHEY_PLAIN, 2, CV_RGB(255,0,0), 2);
156
157   }
158   }
159   return frame;
160
161 }
```

C.1.2. Algoritmo LK

```
1  /*This library was created by Nicolas Ortiz and the research group
       on Image Processing of the
2  Universidad Piloto de Colombia.
3  */
4
5  #include "opencv2/video/tracking.hpp"
6  #include "opencv2/imgproc/imgproc.hpp"
7  #include "opencv2/highgui/highgui.hpp"
8  #include <iostream>
9  #include <ctype.h>
10 #include <stdio.h>
11 #include <fstream>
12 #include <iomanip>
13 #include <math.h>
14 #include <cmath>
15 #include <sstream>
16
17 using namespace cv;
18 using namespace std;
19
20 void ArduinoSerial(double VM, double Phi){
21
22   int data[] ={VM,Phi};
23   FILE *file;
24   file = fopen("/dev/ttyACM0","w");  //Opening device file
25   int i = 0;
26       for(i = 0 ; i < 2 ; i++){
27           fprintf(file,"%d",data[i]); //Writing to the file
28           if(i==0) fprintf(file,"%c",','); //To separate digits
```

```
            //sleep(1);
        }
        fclose(file);

}

String convert(int i, String name){
  std::stringstream sstm;
  String result;
  sstm << i << name;
  result = sstm.str();
  return result;
}

/*Transform the image to the XYZ
 */

Mat RGBtoXYZ(Mat image){

  double minR, maxR, minG, maxG, minB, maxB, intensityR,
    intensityG, intensityB, m, n;
  Mat imageR, imageG, imageB, imageXYZ;
  //vector<Mat> channels(3);
  Mat channels[3];
  split(image,channels);
  imageR=channels[0];
  imageG=channels[1];
  imageB=channels[2];
  for(n=1;n==image.cols;n++){
    for(m=1;m==image.rows;m++){

      intensityR=imageR.at<uchar>(n,m);
      intensityG=imageG.at<uchar>(n,m);
      intensityB=imageB.at<uchar>(n,m);
      imageR.at<uchar>(n,m)=intensityR/(intensityR+intensityG+
    intensityB);
      imageG.at<uchar>(n,m)=intensityG/(intensityR+intensityG+
    intensityB);
      imageB.at<uchar>(n,m)=intensityB/(intensityR+intensityG+
    intensityB);

    }
  }
  minMaxLoc(imageR,&minR,&maxR);
  minMaxLoc(imageG,&minG,&maxG);
  minMaxLoc(imageB,&minB,&maxB);
  imageR=(255/maxR)*imageR;
  imageG=(255/maxG)*imageG;
  imageB=(255/maxB)*imageB;
  channels[0]=imageR;
  channels[1]=imageG;
  channels[2]=imageB;
```

```
77   merge(channels,3,imageXYZ);
78   return imageXYZ;
79 }
80
81 /* Lucas Kanade Optical Flow mixed with the subtraction Algorythm
82 */
83
84 Mat LK(Mat subs, Mat prevGray, Mat gray, Mat image){
85
86   Mat subs1, subs2, kernel;
87   kernel = Mat::ones(51, 51, CV_8UC1);
88   TermCriteria termcrit(TermCriteria::COUNT|TermCriteria::EPS
       ,100,0.03);
89   Size subPixWinSize(7,7), winSize(31,31);
90   Point2f point;
91   const int MAX_COUNT=1000;
92   double min, max;
93   vector<Point2f> points[2];
94
95   subs=RGBtoXYZ(subs);
96   cvtColor(subs, subs, COLOR_BGR2GRAY);
97   prevGray=RGBtoXYZ(prevGray);
98   cvtColor(prevGray, prevGray, COLOR_BGR2GRAY);
99   gray=RGBtoXYZ(gray);
100  cvtColor(gray, gray, COLOR_BGR2GRAY);
101  subs1=prevGray-subs;
102  subs2=gray-prevGray;
103  adaptiveThreshold( subs1, subs1, 1, CV_ADAPTIVE_THRESH_MEAN_C,
       CV_THRESH_BINARY_INV,81,5);
104      dilate(subs1, subs1, kernel);
105  multiply(prevGray,subs1,prevGray);
106  minMaxLoc(prevGray, &min, &max);
107  double s = cv::sum( prevGray )[0];
108  if(s>=2000){
109  //fast.detect(prevGray,points[0],kernel);
110  goodFeaturesToTrack(prevGray, points[0], MAX_COUNT, 0.01, 1, Mat
       (), 3, 0, 0.04);
111  cornerSubPix(prevGray, points[0], subPixWinSize, Size(-1,-1),
       termcrit);
112  adaptiveThreshold( subs2, subs2, 1, CV_ADAPTIVE_THRESH_MEAN_C,
       CV_THRESH_BINARY_INV,81,5);
113      dilate(subs2, subs2, kernel);
114  multiply(gray,subs2,gray);
115  minMaxLoc(gray,&min,&max);
116  double s = cv::sum( gray )[0];
117  if(s>=2000){
118  //fast.detect(prevGray,points[1],kernel);
119  goodFeaturesToTrack(gray, points[1], MAX_COUNT, 0.01, 1, Mat(),
       3, 0, 0.04);
120  cornerSubPix(gray, points[1], subPixWinSize, Size(-1,-1),
       termcrit);
121  vector<uchar> status;
```

```
122         vector<float> err;
123  calcOpticalFlowPyrLK(prevGray, gray, points[0], points[1],
      status, err, winSize,3, termcrit, 0, 0.01);
124  size_t i, k;
125  double VM, Phi, X1, X0, Y1, Y0, X, Y, L;
126 VM=0; Phi=0; X=0; Y=0;
127
128         for( i = k = 0; i < points[1].size(); i++ ){
129
130    points[1][k++] = points[1][i];
131                 circle( image, points[1][i], 3, Scalar(0,0,0), -1,
       8);
132    X1=points[1][i].x; Y1=points[1][i].y;
133    X0=points[0][i].x; Y0=points[0][i].y;
134    CvPoint p0 =cvPoint(cvRound(points[0][i].x),cvRound(points[0][
      i].y));
135    CvPoint p1 =cvPoint(cvRound(points[1][i].x),cvRound(points[1][
      i].y));
136    if(sqrt((X1-X0)*(X1-X0)+(Y1-Y0)*(Y1-Y0))<25){
137        VM=VM + sqrt((X1-X0)*(X1-X0)+(Y1-Y0)*(Y1-Y0));
138        if(X1-X0!=0){
139          Phi=Phi+atan((Y1-Y0)/(X1-X0));
140        }
141        else if(Y1-Y0>0){
142          Phi=3.1415/2;
143        }
144        else if(Y1-Y0<0){
145          Phi=-3.1415/2;
146        }
147        X=X+X1; Y=Y+Y1;
148        line(image,p0,p1,CV_RGB(0,255,0),0.1,8);
149    }
150  }
151
152 VM=VM/points[1].size(); Phi=Phi/points[1].size(); ArduinoSerial(
      VM,Phi);
153 X=X/points[1].size(); Y=Y/points[1].size();
154  double Xnew=X+VM*cos(Phi); double Ynew=Y+VM*sin(Phi);
155  CvPoint p0 =cvPoint(cvRound(X),cvRound(Y));
156  CvPoint p1 =cvPoint(cvRound(Xnew),cvRound(Ynew));
157  circle( image, p0, 8, Scalar(255,0,0), -1, 8);
158  circle( image, p1, 8, Scalar(0,0,255), -1, 8);
159  std::ostringstream strs;
160  strs << VM;
161  std::string str = strs.str();
162  std::ostringstream strs2;
163  strs2 << Phi;
164  std::string str2 = strs2.str();
165  putText(image, "Magnitude: "+str, Point2f(X,Y),
      FONT_HERSHEY_PLAIN, 2,  Scalar(0,0,0,0), 2);
166  putText(image, "Angle: "+str2, Point2f(X,Y-30),
      FONT_HERSHEY_PLAIN, 2,  Scalar(0,0,0,0), 2);
```

```
167   line(image,p0,p1,CV_RGB(0,0,0),5,8);
168
169   points[1].resize(k);
170   if( points[1].size() < (size_t)MAX_COUNT ){
171
172     vector<Point2f> tmp;
173     tmp.push_back(point);
174           cornerSubPix( gray, tmp, winSize, Size(-1,-1), termcrit)
      ;
175                 points[0].push_back(tmp[1]);
176
177         }
178   }
179   }
180
181   return image;
182
183 }
```

C.1.3. Algoritmo FB

```
1  /*This library was created by Nicolas Ortiz and the research group
       on Image Processing of the
2  Universidad Piloto de Colombia.
3  */
4
5  #include "opencv2/video/tracking.hpp"
6  #include "opencv2/imgproc/imgproc.hpp"
7  //#include "opencv2/videoio/videoio.hpp"
8  #include "opencv2/highgui/highgui.hpp"
9  #include <iostream>
10 #include <ctype.h>
11 #include <stdio.h>
12 #include <fstream>
13 #include <iomanip>
14 #include <math.h>
15 #include <cmath>
16 #include <sstream>
17
18 using namespace cv;
19 using namespace std;
20
21 void ArduinoSerial(double VM, double Phi){
22
23   int data[] ={VM,Phi};
24   FILE *file;
25   file = fopen("/dev/ttyACM0","w");  //Opening device file
26   int i = 0;
27       for(i = 0 ; i < 2 ; i++){
28           fprintf(file,"%d",data[i]); //Writing to the file
29           if(i==0) fprintf(file,"%c",','); //To separate digits
30           //sleep(1);
31       }
```

```
32        fclose(file);
33
34  }
35
36  String convert(int i, String name){
37    std::stringstream sstm;
38    String result;
39    sstm << i << name;
40    result = sstm.str();
41    return result;
42  }
43
44  /*Transform the image to the XYZ
45   */
46
47  Mat RGBtoXYZ(Mat image){
48
49    double minR, maxR, minG, maxG, minB, maxB, intensityR,
       intensityG, intensityB, m, n;
50    Mat imageR, imageG, imageB, imageXYZ;
51    //vector<Mat> channels(3);
52    Mat channels[3];
53    split(image,channels);
54    imageR=channels[0];
55    imageG=channels[1];
56    imageB=channels[2];
57    for(n=1;n==image.cols;n++){
58      for(m=1;m==image.rows;m++){
59
60        intensityR=imageR.at<uchar>(n,m);
61        intensityG=imageG.at<uchar>(n,m);
62        intensityB=imageB.at<uchar>(n,m);
63        imageR.at<uchar>(n,m)=intensityR/(intensityR+intensityG+
       intensityB);
64        imageG.at<uchar>(n,m)=intensityG/(intensityR+intensityG+
       intensityB);
65        imageB.at<uchar>(n,m)=intensityB/(intensityR+intensityG+
       intensityB);
66
67      }
68    }
69    minMaxLoc(imageR,&minR,&maxR);
70    minMaxLoc(imageG,&minG,&maxG);
71    minMaxLoc(imageB,&minB,&maxB);
72    imageR=(255/maxR)*imageR;
73    imageG=(255/maxG)*imageG;
74    imageB=(255/maxB)*imageB;
75    channels[0]=imageR;
76    channels[1]=imageG;
77    channels[2]=imageB;
78    merge(channels,3,imageXYZ);
79    return imageXYZ;
```

```
}

/* Drawing Method for the Farnerback Algorythm Mixed with the
    substraction Algorythm
*/

Mat drawOptFlowMap (const Mat& flow, Mat& cflowmap, int step,
    const Scalar& color) {

  double VM, Phi, X1, X0, Y1, Y0, X, Y, L, i;
  VM=0; Phi=0; X=0; Y=0; i=0;

  for(int y = 0; y < cflowmap.rows; y += step){

          for(int x = 0; x < cflowmap.cols; x += step){

      const Point2f& fxy = flow.at< Point2f>(y, x);
                line(cflowmap, Point(x,y), Point(cvRound(x+fxy.x),
    cvRound(y+fxy.y)),color);
                circle(cflowmap, Point(cvRound(x+fxy.x), cvRound(y
   +fxy.y)), 3, color, -1);
      X1=x+fxy.x; Y1=y+fxy.y;
      X0=x; Y0=y;

      if(X1!=X0 && Y1!=Y0){
        if(sqrt((X1-X0)*(X1-X0)+(Y1-Y0)*(Y1-Y0))<25){
          VM=VM + sqrt((X1-X0)*(X1-X0)+(Y1-Y0)*(Y1-Y0));
          if(X0-X1!=0){
            Phi=Phi+atan((Y1-Y0)/(X1-X0));
          }
          else if(Y1-Y0>0){
            Phi=3.1415/2;
          }
          else if(Y1-Y0<0){
            Phi=-3.1415/2;
          }
          X=X+X1; Y=Y+Y1;
          i=i+1;
        }
      }

          }

  }

  VM=VM/i; Phi=Phi/i; ArduinoSerial(VM,Phi);
  X=X/i; Y=Y/i;
  double Xnew=X+VM*cos(Phi); double Ynew=Y+VM*sin(Phi);
  CvPoint p0 =cvPoint(cvRound(X),cvRound(Y));
  CvPoint p1 =cvPoint(cvRound(Xnew),cvRound(Ynew));
  circle( cflowmap, p0, 8, Scalar(255,0,0), -1, 8);
  circle( cflowmap, p1, 8, Scalar(0,0,255), -1, 8);
```

```
128   std::ostringstream strs;
129   strs << VM;
130   std::string str = strs.str();
131   std::ostringstream strs2;
132   strs2 << Phi;
133   std::string str2 = strs2.str();
134   putText(cflowmap, "Magnitude: "+str, Point2f(X,Y),
        FONT_HERSHEY_PLAIN, 2,  Scalar(0,0,0,0), 2);
135   putText(cflowmap, "Angle: "+str2, Point2f(X,Y-30),
        FONT_HERSHEY_PLAIN, 2,  Scalar(0,0,0,0), 2);
136   line(cflowmap,p0,p1,CV_RGB(0,0,0),5,8);
137
138   return cflowmap;
139
140 }
141
142 /* Farnerback Optical Flow mixed with the subtraction Algorythm
143 */
144
145 Mat FB(Mat subs, Mat prevGray, Mat gray, Mat image){
146
147   Mat kernel= Mat::ones(51, 51, CV_8UC1);
148   TermCriteria termcrit(TermCriteria::COUNT|TermCriteria::EPS
        ,100,0.03);
149   Size subPixWinSize(7,7), winSize(31,31);
150   Point2f vel;
151   const int MAX_COUNT=1000;
152   double min, max;
153   Mat flow, subs1, subs2;
154   subs=RGBtoXYZ(subs);
155   cvtColor(subs, subs, COLOR_BGR2GRAY);
156   prevGray=RGBtoXYZ(prevGray);
157   cvtColor(prevGray, prevGray, COLOR_BGR2GRAY);
158   gray=RGBtoXYZ(gray);
159   cvtColor(gray, gray, COLOR_BGR2GRAY);
160   subs1=prevGray-subs;
161   subs2=gray-prevGray;
162   adaptiveThreshold( subs1, subs1, 1, CV_ADAPTIVE_THRESH_MEAN_C,
        CV_THRESH_BINARY_INV,51,5);
163       dilate(subs1, subs1, kernel);
164   multiply(prevGray,subs1,prevGray);
165   adaptiveThreshold( subs2, subs2, 1, CV_ADAPTIVE_THRESH_MEAN_C,
        CV_THRESH_BINARY_INV,51,5);
166       dilate(subs2, subs2, kernel);
167   multiply(gray,subs2,gray);
168
169   double s = cv::sum( gray )[0];
170   if(s>=2000){
171   calcOpticalFlowFarneback(prevGray, gray, flow, 0.5, 2, 21, 2, 7,
        1.5, 1);
172   image=drawOptFlowMap(flow, image, 15, CV_RGB(0, 255, 0));
173   }
```

```
174
175   return image;
176 }
```

C.1.4. Algoritmo PC

```
1  /*This library was created by Nicolas Ortiz and the research group
       on Image Processing of the
2  Universidad Piloto de Colombia.
3  */
4
5  #include "opencv2/core.hpp"
6  #include "opencv2/videoio.hpp"
7  #include "opencv2/highgui.hpp"
8  #include "opencv2/imgproc.hpp"
9  #include <iostream>
10 #include <ctype.h>
11 #include <stdio.h>
12 #include <fstream>
13 #include <iomanip>
14 #include <math.h>
15 #include <cmath>
16 #include <sstream>
17
18 using namespace cv;
19 using namespace std;
20
21 void ArduinoSerial(double VM, double Phi){
22
23   int data[] ={VM,Phi};
24   FILE *file;
25   file = fopen("/dev/ttyACM0","w");  //Opening device file
26   int i = 0;
27       for(i = 0 ; i < 2 ; i++){
28           fprintf(file,"%d",data[i]); //Writing to the file
29           if(i==0) fprintf(file,"%c",','); //To separate digits
30           //sleep(1);
31       }
32       fclose(file);
33
34 }
35
36 String convert(int i, String name){
37   std::stringstream sstm;
38   String result;
39   sstm << i << name;
40   result = sstm.str();
41   return result;
42 }
43
44 /*Transform the image to the XYZ
45  */
46
```

```
47 Mat RGBtoXYZ(Mat image){
48
49   double minR, maxR, minG, maxG, minB, maxB, intensityR,
      intensityG, intensityB, m, n;
50   Mat imageR, imageG, imageB, imageXYZ;
51   //vector<Mat> channels(3);
52   Mat channels[3];
53   split(image,channels);
54   imageR=channels[0];
55   imageG=channels[1];
56   imageB=channels[2];
57   for(n=1;n==image.cols;n++){
58     for(m=1;m==image.rows;m++){
59
60       intensityR=imageR.at<uchar>(n,m);
61       intensityG=imageG.at<uchar>(n,m);
62       intensityB=imageB.at<uchar>(n,m);
63       imageR.at<uchar>(n,m)=intensityR/(intensityR+intensityG+
      intensityB);
64       imageG.at<uchar>(n,m)=intensityG/(intensityR+intensityG+
      intensityB);
65       imageB.at<uchar>(n,m)=intensityB/(intensityR+intensityG+
      intensityB);
66     }
67   }
68   minMaxLoc(imageR,&minR,&maxR);
69   minMaxLoc(imageG,&minG,&maxG);
70   minMaxLoc(imageB,&minB,&maxB);
71   imageR=(255/maxR)*imageR;
72   imageG=(255/maxG)*imageG;
73   imageB=(255/maxB)*imageB;
74   channels[0]=imageR;
75   channels[1]=imageG;
76   channels[2]=imageB;
77   merge(channels,3,imageXYZ);
78   return imageXYZ;
79 }
80
81 /*Phase Correlation Algorithm
82 */
83
84 Mat PC(Mat subs, Mat prevGray, Mat gray, Mat image){
85
86   Mat kernel= Mat::ones(51, 51, CV_8UC1);
87   Mat subs1, subs2;
88   Mat curr64f, prev64f, hann;
89   subs=RGBtoXYZ(subs);
90   cvtColor(subs, subs, COLOR_BGR2GRAY);
91   prevGray=RGBtoXYZ(prevGray);
92   cvtColor(prevGray, prevGray, COLOR_BGR2GRAY);
93   gray=RGBtoXYZ(gray);
94   cvtColor(gray, gray, COLOR_BGR2GRAY);
```

```
95  subs1=prevGray-subs;
96  subs2=gray-prevGray;
97  adaptiveThreshold( subs1, subs1, 1, CV_ADAPTIVE_THRESH_MEAN_C,
     CV_THRESH_BINARY_INV,51,5);
98      dilate(subs1, subs1, kernel);
99  multiply(prevGray,subs1,prevGray);
100 adaptiveThreshold( subs2, subs2, 1, CV_ADAPTIVE_THRESH_MEAN_C,
     CV_THRESH_BINARY_INV,51,5);
101     dilate(subs2, subs2, kernel);
102 multiply(gray,subs2,gray);
103
104 double s = cv::sum( gray )[0];
105 if(s>=2000){
106
107 createHanningWindow(hann, gray.size(), CV_64F);
108
109 prevGray.convertTo(prev64f, CV_64F);
110       gray.convertTo(curr64f, CV_64F);
111
112 Point2d shift = phaseCorrelate(prev64f, curr64f, hann);
113       double radius = std::sqrt(shift.x*shift.x + shift.y*shift.
     y);
114
115 Point center(gray.cols >> 1, gray.rows >> 1);
116       circle(image, center, (int)radius, Scalar(0, 255, 0), 3,
     LINE_AA);
117       line(image, center, Point(center.x + (int)shift.x, center.
     y + (int)shift.y), Scalar(0, 255, 0), 3, LINE_AA);
118
119 double VM=sqrt((int)shift.x*(int)shift.x+(int)shift.y*(int)shift
     .y);
120 double Phi;
121 if((int)shift.x!=0){
122   Phi=atan((int)shift.y/(int)shift.x);
123 }
124 else if((int)shift.y>0){
125   Phi=3.1415/2;
126 }
127 else if((int)shift.y<0){
128   Phi=-3.1415/2;
129 }
130
131 ArduinoSerial(VM,Phi);
132
133 std::ostringstream strs;
134 strs << VM;
135 std::string str = strs.str();
136 std::ostringstream strs2;
137 strs2 << Phi;
138 std::string str2 = strs2.str();
139 putText(image, "Magnitude: "+str, Point(center.x + (int)shift.x,
     center.y + (int)shift.y),FONT_HERSHEY_PLAIN, 2,  Scalar
```

```
    (0,0,0,0), 2);
140   putText(image, "Angle: "+str2, Point(center.x + (int)shift.x,
      center.y + (int)shift.y-30),FONT_HERSHEY_PLAIN, 2,  Scalar
      (0,0,0,0), 2);
141
142   }
143
144   return image;
145
146 }
```

C.2. Códigos par la Validación de los Algoritmos

C.2.1. Algoritmo SB

```
1  #include "opencv2/video/tracking.hpp"
2  #include "opencv2/imgproc/imgproc.hpp"
3  #include "opencv2/highgui/highgui.hpp"
4  #include "opencv2/core/core.hpp"
5  #include <iostream>
6  #include <ctype.h>
7  #include <stdio.h>
8  #include <fstream>
9  #include <iomanip>
10 #include <math.h>
11 #include <cmath>
12 #include "SUB.h"
13
14 using namespace cv;
15 using namespace std;
16
17 int  main(){
18
19   VideoCapture cap(0);
20   double C, VelF;
21   VelF=0;
22   Mat gray, prevGray, image, frame, subs;
23   namedWindow( "Algorythm", 0 );
24
25   subs=imread("frame1.png");
26   prevGray=imread("frame2.png");
27   gray=imread("frame3.png");
28
29   image=Subtraction(subs, prevGray, gray, subs);
30
31   imwrite("imageSUB.png",image);
32
33   return 0.0;
34 }
```

C.2.2. Algoritmo LK

```
1  #include "opencv2/video/tracking.hpp"
```

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <ctype.h>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <math.h>
#include <cmath>
#include "LK.h"

using namespace cv;
using namespace std;

int  main(){

  VideoCapture cap(0);
  double C, VelF;
  VelF=0;
  Mat gray, prevGray, image, frame, subs;
  namedWindow( "Algorythm", 0 );

  subs=imread("frame1.png");
  prevGray=imread("frame2.png");
  gray=imread("frame3.png");

  image=LK(subs, prevGray, gray, subs);

  imwrite("imageLK.png",image);

  return 0.0;

}
```

C.2.3. Algoritmo FB

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <ctype.h>
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <math.h>
#include <cmath>
#include "FB.h"

using namespace cv;
using namespace std;

int  main(){

```

```
18   VideoCapture cap(0);
19   double C, VelF;
20   VelF=0;
21   Mat gray, prevGray, image, frame, subs;
22   namedWindow( "Algorythm", 0 );
23
24   subs=imread("frame1.png");
25   prevGray=imread("frame2.png");
26   gray=imread("frame3.png");
27
28   image=FB(subs, prevGray, gray, subs);
29
30   imwrite("imageFB.png",image);
31
32   return 0.0;
33
34 }
```

C.2.4. Algoritmo PC

```
1  #include "opencv2/core.hpp"
2  #include "opencv2/videoio.hpp"
3  #include "opencv2/highgui.hpp"
4  #include "opencv2/imgproc.hpp"
5  #include <iostream>
6  #include <ctype.h>
7  #include <stdio.h>
8  #include <fstream>
9  #include <iomanip>
10 #include <math.h>
11 #include <cmath>
12 #include <sstream>
13 #include "PC.h"
14
15 using namespace cv;
16 using namespace std;
17
18 int  main(){
19
20   VideoCapture cap(0);
21   double C, VelF;
22   VelF=0;
23   Mat gray, prevGray, image, frame, subs;
24   namedWindow( "Algorythm", 0 );
25
26   subs=imread("frame1.png");
27   prevGray=imread("frame2.png");
28   gray=imread("frame3.png");
29
30   image=PC(subs, prevGray, gray, subs);
31
32   imwrite("imagePC.png",image);
33
```

```
34   return 0.0;
35
36 }
```

D. Algoritmo SB implementado

D.1. Programa Principal

```
1  #include "opencv2/video/tracking.hpp"
2  #include "opencv2/imgproc/imgproc.hpp"
3  #include "opencv2/highgui/highgui.hpp"
4  #include "opencv2/core/core.hpp"
5  #include <iostream>
6  #include <ctype.h>
7  #include <stdio.h>
8  #include <fstream>
9  #include <iomanip>
10 #include <math.h>
11 #include <cmath>
12 #include "SUB.h"
13
14 using namespace cv;
15 using namespace std;
16
17 int  main(){
18
19   VideoCapture cap(0);
20   double C, VelF, t;
21   VelF=0;
22   Mat gray, prevGray, image, frame, subs;
23   namedWindow( "Algorythm", 0 );
24
25   /*subs=imread("x/frame0.png");
26   prevGray=imread("x/frame2.png");
27   gray=imread("x/frame4.png");*/
28
29   while(1){
30
31     cap >> frame; imshow("Algorythm",frame);
32     frame.copyTo(subs);
33     cap >> frame; imshow("Algorythm",frame);
34     frame.copyTo(prevGray);
35     cap >> frame; imshow("Algorythm",frame);
36     frame.copyTo(gray);
37
38     image=Subtraction(subs, prevGray, gray, subs);
39
40     imshow("Algorythm",image);
41     //waitKey();
42
43     C= cvWaitKey(10);
44       if (C == 27) break;
```

```
45
46    }
47    return 0.0;
48 }
```

D.2. Libreria

```
1  #include <opencv2/core/utility.hpp>
2  #include "opencv2/video/tracking.hpp"
3  #include "opencv2/imgproc.hpp"
4  #include "opencv2/videoio.hpp"
5  #include "opencv2/highgui.hpp"
6  #include <iostream>
7  #include <ctype.h>
8  #include <stdio.h>
9  #include <fstream>
10 #include <iomanip>
11 #include <math.h>
12 #include <cmath>
13 #include <sstream>
14 #include <unistd.h>
15
16 using namespace cv;
17 using namespace std;
18
19 void InverseKinematics(double VMx, double VMy){
20
21   /*Inverse Kinematics*/
22   double alphai=0.0722;
23   double alphaj=0.0708;
24   double L=0.12445; // meters
25   double zeta=15.37;
26   double phi=(90-zeta);
27   double k=3.1415/180;
28   double Beta,Lambda;
29   double theta1,theta2;
30   double t1, t2, t;
31
32   std::cout.precision(4);
33
34   Beta=k*(alphai*VMx+phi);
35   Lambda=k*(alphaj*VMy);
36
37   theta1=acos(cos(Beta)*cos(Lambda))-phi*k;
38
39   if(phi==-theta1){
40     theta2=0;
41   }
42   else{
43     theta2=asin(sin(Lambda)/sin(phi*k+theta1));
44   }
45
46   t1=(theta1*180/3.1415);
```

```
47    t2=(theta2*180/3.1415);
48
49    if(t1<=t2){
50      t=t1/375000;
51    }
52    else{
53      t=t2/375000;
54    }
55
56    if(sqrt(VMx*VMx) > 100 || sqrt(VMy*VMy) > 100){
57      t1=0;
58      t2=0;
59    }
60
61    cout << t1 << setw(10) << t2 << endl;
62
63    double data[] ={t1,t2};
64    int i = 0;
65
66    FILE *file;
67    file = fopen("/dev/ttyACM0","w");  //Opening device file
68
69        for(i = 0 ; i < 2 ; i++){
70      if(i==1) fprintf(file,"%c",'F');
71      fprintf(file,"%f",data[i]);
72      //usleep(50000);
73        }
74        fclose(file);
75    usleep(400000);
76    /*End*/
77  }
78
79  Mat RGBtoXYZ(Mat image){
80
81    double minR, maxR, minG, maxG, minB, maxB, intensityR,
       intensityG, intensityB, m, n;
82    Mat imageR, imageG, imageB, imageXYZ;
83    //vector<Mat> channels(3);
84    cvtColor(image,image,COLOR_BGR2XYZ);
85    Mat channels[3];
86    split(image,channels);
87    imageR=channels[0];
88    imageG=channels[1];
89    imageB=channels[2];
90    minMaxLoc(imageR,&minR,&maxR);
91    minMaxLoc(imageG,&minG,&maxG);
92    minMaxLoc(imageB,&minB,&maxB);
93    imageR=(255/maxR)*imageR;
94    imageG=(255/maxG)*imageG;
95    imageB=(255/maxB)*imageB;
96    channels[0]=imageR;
97    channels[1]=imageG;
```

```
98    channels[2]=imageB;
99    merge(channels,3,imageXYZ);
100   return imageXYZ;
101
102
103 }
104
105 Mat Subtraction(Mat actu, Mat prev, Mat subs, Mat frame){
106
107   Mat subs1, subs2;
108   Mat kernel= Mat::ones(3, 3, CV_8UC1);
109   subs=RGBtoXYZ(subs);
110   prev=RGBtoXYZ(prev);
111   actu=RGBtoXYZ(actu);
112   cvtColor(subs,subs,COLOR_RGB2GRAY);
113   cvtColor(prev,prev,COLOR_RGB2GRAY);
114   cvtColor(actu,actu,COLOR_RGB2GRAY);
115   subs1=prev-subs;
116   subs2=actu-prev;
117   adaptiveThreshold( subs1, subs1, 1, CV_ADAPTIVE_THRESH_MEAN_C,
       CV_THRESH_BINARY_INV,51,5);
118   dilate(subs1, subs1, kernel);
119   multiply(prev,subs1,prev);
120   adaptiveThreshold( subs2, subs2, 1, CV_ADAPTIVE_THRESH_MEAN_C,
       CV_THRESH_BINARY_INV,51,5);
121   dilate(subs2, subs2, kernel);
122   multiply(actu,subs2,actu);
123
124   double posx, posy, posxo, posyo; int i=0; double h,l; int Yo, Yf
       , Xo, Xf;
125   posxo=0; posyo=0; posx=0; posy=0; int x,y;
126
127   for (y = 0; y < prev.rows; ++y) {
128         for (x = 0; x < prev.cols; ++x) {
129       if(prev.at<uchar>(y,x)>=10){
130         posxo += x; posyo +=y;
131         i += 1;
132       }
133     }
134   }
135
136   posxo=int(posxo/i); posyo=int(posyo/i);
137   i=0;
138
139   for (y = 0; y < actu.rows; ++y) {
140         for (x = 0; x < actu.cols; ++x) {
141       if(actu.at<uchar>(y,x)>=10){
142         posx += x; posy +=y;
143         if(i==0){
144           Xo=x;Yo=y;
145         }
146         i += 1; Yf=y; Xf=x;
```

```
147         }
148       }
149     }
150
151     posx=int(posx/i); posy=int(posy/i);
152
153     rectangle(frame,cvPoint(Xo,Yo),cvPoint(Xf,Yf),Scalar(255,0,0)
        ,2,8,0);
154     line(frame,cvPoint(posxo,posyo),cvPoint(posx,posy),Scalar
        (0,255,0),2,8,0);
155     posyo=frame.rows-posyo;
156     posy=frame.rows-posy;
157
158     /*double VM=sqrt((posxo-posx)*(posxo-posx)+(posyo-posy)*(posyo-
        posy));
159     double Phi;
160
161     if(posxo-posx!=0){
162       Phi=atan((posyo-posy)/(posx-posxo));
163     }
164     else if(posy-posyo>0){
165       Phi=3.1415/2;
166     }
167     else if(posy-posyo<0){
168       Phi=-3.1415/2;
169     }*/
170
171     double VMx=posx-posxo;
172     double VMy=-posy+posyo;
173
174     InverseKinematics(VMx,VMy);
175
176     std::ostringstream strs;
177     strs << VMx;
178     std::string str = strs.str();
179     std::ostringstream strs2;
180     strs2 << VMy;
181     std::string str2 = strs2.str();
182     putText(frame, "VMx: "+str, Point(posx,posy),FONT_HERSHEY_PLAIN,
          2, CV_RGB(255,0,0), 2);
183     putText(frame, "VMy: "+str2, Point(posx,posy-30),
        FONT_HERSHEY_PLAIN, 2, CV_RGB(255,0,0), 2);
184
185     return frame;
186
187 }
```

E. Algoritmo Arduino

```
1 #include <Servo.h>
2
3 double Theta1=70; //Almacena el Valor de Beta
```

```
4  double Theta2=60; //Almacena el Valor de Lambda
5  double t1=0; String T1;
6  double t2=0; String T2;
7  boolean check=false;
8  String inputString="";
9  char inChar="";
10 int index;
11
12 Servo servo1; //Giro alrededor de z
13 Servo servo2; //Giro alrededor de x
14
15 void setup()
16 {
17     Serial.begin(9600);
18     servo1.attach(9);
19     servo2.attach(12);
20     servo1.write(70);
21     servo2.write(15);
22 }
23
24 void loop()
25 {
26     while(Serial.available()>1){
27       delay(10);
28       if(Serial.available()>0){
29         inChar = Serial.read();
30         inputString += inChar;
31         check=true;
32       }
33     }
34     if(check){
35       Serial.println(inputString);
36       index=inputString.indexOf('F');
37       T1=inputString.substring(1,index);
38       T2=inputString.substring(index+1);
39       t1=T1.toDouble();
40       t2=T2.toDouble();
41       //Serial.println(t1);
42       //Serial.println(t2);
43
44       Theta1=Theta1+t1;
45       Theta2=Theta2+t2;
46
47       if(Theta1>20 && Theta1<160 && Theta2>30 && Theta2<150 ){
48         servo1.write(Theta1);
49         servo2.write(Theta2);
50       }
51       else{
52         Theta1=Theta1-t1;
53         Theta2=Theta2-t2;
54       }
55
```

```
56          Serial.println(Theta1);
57          Serial.println(Theta2);
58
59          inputString="";
60          check=false;
61      }
62  }
```