

ESCUELA POLITÉCNICA NACIONAL

**FACULTAD DE INGENIERÍA ELÉCTRICA Y
ELECTRÓNICA**

**SIMULACIÓN DE UN SISTEMA DE CONTROL DE UNA SILLA DE
RUEDAS BASADO EN VISIÓN ARTIFICIAL MEDIANTE
MOVIMIENTOS DE CABEZA, PARA PERSONAS
CUADRIPLÉJICAS**

**TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE
INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES**

JHON ANTHONY JIRÓN MUZHA

DIRECTOR: PhD. ROBIN GERARDO ÁLVAREZ RUEDA

Quito, marzo 2022

AVAL

Certifico que el presente trabajo fue desarrollado por Jhon Anthony Jirón Muzha, bajo mi supervisión.

PHD. ROBIN ÁLVAREZ RUEDA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Jhon Anthony Jirón Muzha, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

JHON ANTHONY JIRÓN MUZHA

DEDICATORIA

A mis abuelos, por cuidarme y guiarme en mi infancia, en especial a la memoria de mi abuelito José, pilar fundamental en mi formación personal y demostrarme que las cosas importantes se consiguen con dedicación y esmero; a mi padre Franco, por ser mi apoyo, mi amigo, saberme orientar en los malos momentos, por haber sido comprensivo y por hacerme sentir orgulloso de ser su hijo; a mi madre María, por su dedicación en las situaciones más adversas, por su confianza, apoyo en todas las etapas de mi vida y quien con su esfuerzo me enseñó a que siempre se puede hacer más aun estando cansado. Dedico también el presente trabajo a mi hermano menor Guillermo, porque aún sin saberlo me incentiva a lograr mis metas; y finalmente, a mi familia por siempre alegrar mis días y ayudarme en todo momento.

AGRADECIMIENTO

Agradezco en primer lugar a Dios, por brindarme salud y bendecirme con unos abuelos y padres maravillosos que siempre me cuidan y se preocupan por mi bienestar. También quiero agradecer a mis padres, por haberme sabido guiar en mi infancia, enseñar que las metas se las consigue con sacrificio y honestidad, por no haberme permitido rendirme cuando sentía que no podía dar más en lo académico y personal. A mis abuelos quienes, aunque fueron estrictos en cuanto a mi formación académica, me permitieron forjar los valores y actitudes que he llegado a desarrollar actualmente. Así también, agradezco a mi hermano por motivarme a ser una persona dedicada en todo ámbito, esperando poder ser su ejemplo a seguir.

A mi director PhD. Robin Álvarez un agradecimiento especial por guiarme oportunamente y darme su confianza en la realización de este trabajo de titulación.

A mis amigos de la Universidad por haber convertido el tiempo de esta difícil etapa académica en muchos momentos amenos.

A mi familia en general, a Ramiro y Verónica por brindarme su apoyo en situaciones muy difíciles de mi vida y ayudarme a no sentirme solo en una ciudad completamente lejos de mi familia.

Finalmente, quiero agradecer a la Escuela Politécnica Nacional por brindarme la oportunidad de haber cursado sus instalaciones y a quienes en su momento fueron mis profesores por impartirme sus conocimientos y sobre todo complementar mi formación con valores éticos y responsables.

ÍNDICE DE CONTENIDO

AVAL	II
DECLARACIÓN DE AUTORÍA	III
DEDICATORIA.....	IV
AGRADECIMIENTO.....	V
ÍNDICE DE CONTENIDO	VI
RESUMEN.....	VIII
ABSTRACT.....	IX
1.....	INTRODUCCIÓN
.....	1
1.1 OBJETIVOS.....	2
1.2 ALCANCE.....	3
1.3 MARCO TEÓRICO.....	4
1.3.1 ANTECEDENTES	4
1.3.2 VISIÓN ARTIFICIAL.....	5
1.3.3 ALGORITMO DE DETECCIÓN DE OBJETOS VIOLA-JONES.....	10
1.3.4 ALGORITMO DE SEGUIMIENTO DE OBJETOS KANADE-LUCAS-TOMASI15	
1.3.5 ALGORITMO DE ESTIMACIÓN GEOMÉTRICA.....	19
2.....	METODOLOGÍA
.....	23
2.1 DISEÑO DEL SISTEMA DE SIMULACIÓN DE CONTROL DE LA SILLA DE RUEDAS	23
2.1.1 ETAPAS DEL SISTEMA.....	23
2.1.2 RECURSOS USADOS PARA LA SIMULACIÓN DEL DISEÑO.....	24
2.1.3 CONDICIONES DE FUNCIONAMIENTO	25
2.2 ETAPA 1: ADQUISICIÓN Y PROCESAMIENTO DE IMÁGENES	26
2.2.1 CONFIGURACIÓN DEL OBJETO PARA LA CÁMARA.....	26
2.3 ETAPA 2: DETECCIÓN DE ROSTRO.....	30
2.3.2 CREACIÓN DEL DETECTOR.....	30
2.3.3 VALIDACIÓN DE ROSTRO EN IMÁGENES	34
2.4 ETAPA 3: OBTENCIÓN DEL RECTÁNGULO DE LA NARIZ	38
2.5 ETAPA 4: SEGUIMIENTO DE ROSTRO Y NARIZ	39
2.5.1 CREACIÓN DEL OBJETO SEGUIDOR.....	40
2.6 ETAPA 5: DETECCIÓN DE LA DIRECCIÓN DE GIRO DE LA CABEZA.....	53
2.6.1 INICIALIZACIÓN DE PUNTOS DE REFERENCIA DE LA NARIZ	55
2.6.2 OBTENCIÓN DE LA DIRECCIÓN DE GIRO	56

2.7	ETAPA 6: OBTENCIÓN DE PARÁMETROS PARA EL MOVIMIENTO DE LA SILLA 59		
2.7.1	INICIALIZACIÓN DE CARACTERÍSTICAS PARA MOVIMIENTO.....	61	
2.7.2	DETERMINACIÓN DEL MOVIMIENTO DE LA SILLA EN BASE A LAS CARACTERÍSTICAS DEL ROSTRO EN MOVIMIENTO.....	61	
2.8	ETAPA 7: INTERFAZ GRÁFICA PARA PRESENTACIÓN DEL MOVIMIENTO DE LA SILLA	68	
3.	RESULTADOS	Y	DISCUSIÓN
		72
3.1	ETAPA 1 (PRUEBAS): ADQUISICIÓN Y PROCESAMIENTO DE IMÁGENES 74		
3.2	ETAPA 2 (PRUEBAS): DETECCIÓN DE ROSTRO.....	76	
3.2.1	AUMENTO DE LA VENTANA DE DETECCIÓN (SCALEFACTOR).....	76	
3.2.2	NÚMERO DE DETECCIONES MÍNIMAS EN UNA REGIÓN (MERGETHRESHOLD).....	77	
3.3	ETAPA 3 (PRUEBAS): OBTENCIÓN DEL RECTÁNGULO DE LA NARIZ....	77	
3.4	ETAPA 4 (PRUEBAS): SEGUIMIENTO DEL ROSTRO Y NARIZ.....	78	
3.4.1	ELECCIÓN DEL ALGORITMO DE DETECCIÓN DE CARACTERÍSTICAS 79		
3.4.2	PARÁMETROS DEL OBJETO SEGUIDOR.....	79	
3.4.3	PARÁMETROS DEL OBJETO ESTIMADOR.....	82	
3.5	ETAPA 5 Y 6 (PRUEBAS): ENTRADA, SALIDA Y PRUEBA DEL MODO DE COMANDOS.....	84	
3.5.1	POSICIÓN IDEAL DE LA CÁMARA.....	85	
3.5.2	RESOLUCIÓN DE LA CÁMARA.....	86	
3.5.3	ENTRADA/SALIDA DEL MODO DE COMANDOS EN BASE AL NÚMERO DE PARPADEOS.....	87	
3.5.4	NIVELES DE ILUMINACIÓN EN AMBIENTES INTERIORES Y EXTERIORES.....	89	
4.	CONCLUSIONES	Y	RECOMENDACIONES
		92
4.1	CONCLUSIONES.....	92	
4.2	RECOMENDACIONES.....	93	
5.	REFERENCIAS BIBLIOGRÁFICAS.....	94	
	ANEXOS.....	100	
	ANEXO A: CÓDIGO DEL PROGRAMA PRINCIPAL.....	101	
	ANEXO B: MANUAL DE USUARIO	107	

RESUMEN

Las personas con cuadriplejia dependen de otra persona para desplazarse, por la falta de movilidad en sus 4 extremidades, usando sillas de ruedas manuales. Usar sillas de ruedas motorizadas les otorga cierta independencia, necesitando métodos para controlarlas. Por ejemplo, se han desarrollado tesis controlando los motores mediante voz o movimientos de la cabeza detectados por acelerómetros. Esta última usa una diadema con el chip que contiene los acelerómetros, pudiendo ser molesto para el usuario. El presente trabajo busca ser menos invasivo, por ello se ha desarrollado un sistema de detección de movimientos de la cabeza mediante visión artificial evitando colocarse algún dispositivo y detectando dichos movimientos mediante una cámara ubicada estratégicamente.

El sistema inicia detectando el rostro con el algoritmo Viola-Jones, luego se extrae la sección de la nariz para su seguimiento mediante el algoritmo Kanade-Lucas-Tomasi y se obtienen las nuevas regiones del rostro y nariz con la estimación geométrica. Para el ingreso/salida del modo de comandos, el usuario gira la cabeza de izquierda a derecha en menos de máximo 3 segundos. Mientras esto no suceda, el usuario puede mover su cabeza libremente sin activar los motores. Finalmente, si el usuario ingresa al sistema e inclina su cabeza hacia la izquierda, derecha, adelante o atrás los motores se activarán en la dirección detectada. Este prototipo, por la situación de pandemia, solo ha sido simulado y no se interactúa realmente con una silla de ruedas, lo cual no es problema ya que esta parte ha sido resuelta en aquellas tesis anteriores.

PALABRAS CLAVE: cuadriplejia, detección de rostros, seguimiento de objetos, estimación geométrica, movimientos de cabeza.

ABSTRACT

People with quadriplegia depend on another person to move, by the lack of mobility in their 4 limbs, using manual wheelchairs. To use motorized wheelchairs gives them some independence, needing methods to control them. For example, these have been developed controlling motors by voice or head movements detected by accelerometers. The latter uses a headband with the chip that contains the accelerometers, which can be annoying for the user. The present work seeks to be less invasive, for this reason a system for detecting movements of the head has been developed through artificial vision avoiding placing any device and detecting these movements through a strategically located camera.

The system starts by detecting the face with the Viola-Jones algorithm, then the section of the nose is extracted for monitoring using the Kanade-Lucas-Tomasi algorithm and the new regions of the face and nose are obtained with the geometric estimate. To enter/exit command mode, user turns head from left to right in less than 3 seconds maximum. As long as this does not happen, the user can move his head freely without activating the motors. Finally, if the user enters the system and tilts his head to the left, right, forward or backward the motors will be activated in the detected direction. This prototype, due to the pandemic situation, has only been simulated and does not really interact with a wheelchair, which is not a problem since this part has been solved in those previous theses.

KEYWORDS: quadriplegia, face detection, object tracking, geometric estimation, head movements.

1. INTRODUCCIÓN

La cuadriplejia es la falta de sensibilidad en las partes del cuerpo que son tronco, zona pélvica, ambos brazos y ambas piernas debido a una lesión o tumor en la zona cervical provocando un estado de ausencia de cualquier movimiento de las 4 extremidades del cuerpo. Aunque actualmente hay diversas opciones para tratar la cuadriplejia donde mejoran la postura y producen algunos cambios funcionales para realizar actividades básicas [1], esto no es suficiente para mejorar la calidad de vida de las personas que sufren de esta condición y son necesarios dispositivos basados en tecnología de asistencia para que puedan vivir de forma productiva e independiente.

El adaptar una silla para desplazarse también representa un impacto psicológico, ya que minimiza síntomas depresivos [2]. Por lo mencionado, se plantea realizar la simulación de un sistema de control de una silla de ruedas controlada por medio de movimientos específicos de la cabeza. Este sistema tiene como principales beneficiarios a las personas con discapacidad física mayor al 75%, que representa personas con falta de movilidad completa y engloba la cuadriplejia [3]. En este sistema, el movimiento de la cabeza será detectada a través de visión artificial y podrá ser usado en ambientes tanto interiores con poca iluminación como exteriores en condiciones diurnas como nocturnas con ciertas condiciones por medio de una cámara infrarroja.

Inicialmente se pensó en una forma para entrar o salir del modo de comandos, basada en el conteo de un determinado número de parpadeos; sin embargo, este método presenta varios inconvenientes al momento de definir una cantidad de parpadeos adecuada, ya que la frecuencia de parpadeos depende de muchos factores [4]. Además, la clasificación correcta en ambientes de poca iluminación es bastante baja, más aún con el uso de lentes. De todas maneras, dicho sistema (Figura 1.1) fue implementado y más adelante en el capítulo 3 se evidencia los problemas de funcionamiento. Por tal motivo, se plantea otra forma de entrar o salir del modo de comandos basado en el seguimiento de la nariz, como se muestra en el diagrama de bloques de la Figura 1.2, el cual resultó ser mucho más robusto.

El proyecto será solo simulación por las restricciones actuales de la pandemia. Esto dificulta acceder a las instalaciones de la Universidad, donde se cuenta con una silla de ruedas ya adaptada y, por lo tanto, restringe realizar una implementación física del sistema. Este

prototipo no existe en el mercado nacional y sería importante continuar perfeccionándolo debido a que es cero invasivo.

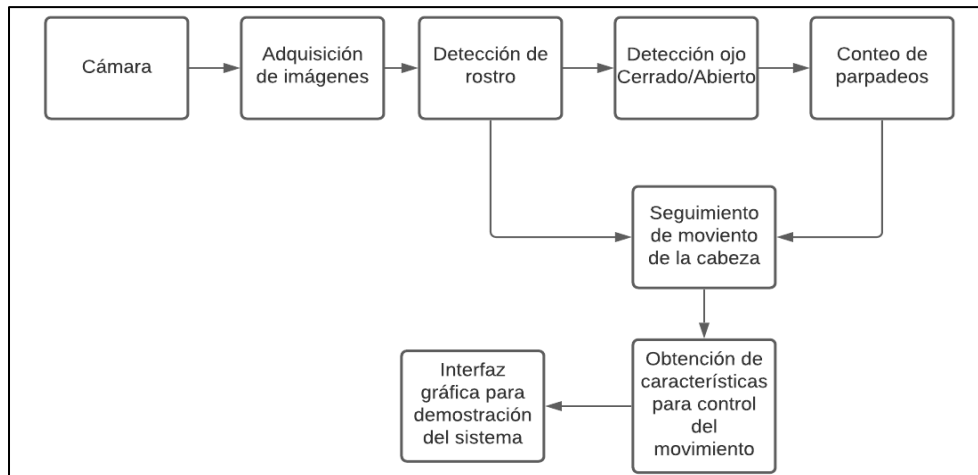


Figura 1.1. Diagrama inicialmente propuesto basado en parpadeos para entrar y salir del modo de comandos.

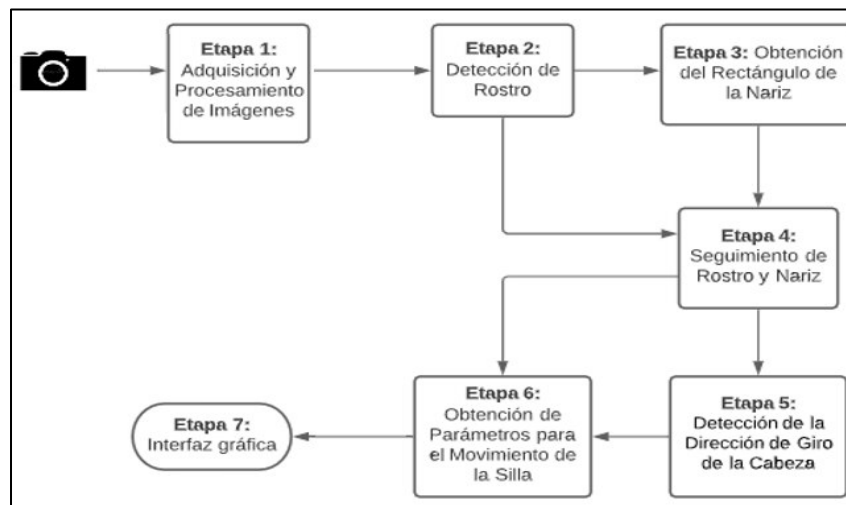


Figura 1.2. Diagrama de bloques del sistema basado en el seguimiento de la nariz para entrar y salir del modo de comandos.

1.1 OBJETIVOS

El objetivo general de este Proyecto Técnico es realizar un sistema simulado en tiempo real del control de una silla de ruedas para personas cuádruplégicas basado en visión artificial.

Los objetivos específicos del Proyecto Técnico son los siguientes:

- i) Estudiar los algoritmos de detección y seguimiento del rostro, basados en visión artificial.

- ii) Analizar algunas técnicas de clasificación mediante redes neuronales.
- iii) Implementar una base de datos de imágenes de ojos cerrados y abiertos adquiridas a través de una cámara infrarroja empleando diferentes niveles de iluminación.
- iv) Entrenar una red neuronal para clasificar el ojo como abierto o cerrado.
- v) Implementar una interfaz gráfica que simule el movimiento de una silla mediante el seguimiento de la cabeza.

1.2 ALCANCE

El proyecto simula un sistema de control de una silla de ruedas no invasivo para personas cuádruplégicas. La simulación del sistema se implementa en el software Matlab donde se detecta el rostro y se realiza el seguimiento. Todos los parámetros de objetos de Matlab que se usan para la simulación serán probados en tiempo diferido para determinar qué valor es el adecuado. La detección del rostro se realiza con la obtención de imágenes frontales de una persona por medio de una cámara infrarroja las cuales serán sometidas al procesamiento de imágenes. El algoritmo de Viola-Jones detecta automáticamente el rostro, por lo que no será necesario crear una base de datos. Para entrar y salir del modo de comandos se emplea el giro de la cabeza haciendo un movimiento específico en base al seguimiento de la nariz, el cual es rotar la cabeza a la izquierda y luego a la derecha en un tiempo determinado de máximo 3 segundos.

El sistema de control mueve la imagen de una silla en la pantalla del computador a través de una interfaz gráfica implementada en Matlab en base a los movimientos que se detecten con la cámara. El sistema propuesto tiene 5 comandos: 'Adelante', 'Atrás', 'Izquierda', 'Derecha' o 'Alto'. Para el movimiento vertical, la persona se acerca o aleja de la cámara respecto al punto donde tuvo inicialmente la cabeza por lo que se establece un rango de ángulos de enfoque de la cámara que cumplan con este propósito. Por otro lado, para mover la silla a la izquierda o la derecha, el usuario deberá inclinar la cabeza hacia el lado correspondiente y para el comando 'alto' la persona deberá estar dentro de los rangos establecidos como reposo bien sea para el eje vertical u horizontal.

La inclinación de la cabeza para la detección estará restringida a un ángulo máximo, por lo que se establece un intervalo de ángulos de inclinación. Además, se considera que el sistema detectará los movimientos de la cabeza de una sola persona, la cual debe estar justo frente al lente de la cámara. Por lo tanto, si existen varias personas frente a la cámara, se hará la detección de la persona que esté más cerca.

De este modo, en el presente trabajo se tiene como producto final demostrable una interfaz gráfica desarrollada en Matlab, la cual mostrará el movimiento de la imagen de una silla en base a los comandos emitidos por el movimiento de la cabeza según la dirección en la que el usuario desee moverse.

1.3 MARCO TEÓRICO

1.3.1 ANTECEDENTES

Se han desarrollado varias tecnologías de asistencia, las cuales “... comprenden la investigación, fabricación, uso de equipos, recursos o estrategias para potenciar las habilidades funcionales de personas con deficiencia, incapacidad o desventaja ...” [5]. La finalidad de todo este proceso es mejorar la calidad de vida y procurar la reinserción de personas con problemas de movilidad como cuadriplejía.

Actualmente se tienen varios dispositivos para ayudar a la ocupación e independencia funcional, tanto así que en un estudio realizado en el 2018 a 100 personas con cuadriplejía el 57 % usa tecnología de asistencia, 13 de ellos con más de 5 dispositivos para este fin [6]. Los dispositivos usados varían desde aparatos ortopédicos para estabilizar puño, mano y dedos, touch mouse y pantallas capacitivas. Además, se tienen dispositivos que permiten movilizar a las personas mediante sillas de ruedas autónomas controladas por distintos tipos de implementaciones.

En la literatura se tienen diferentes formas para controlar la silla tales como: detección invasiva de movimientos de la cabeza con el uso de una diadema con acelerómetro [7], control no invasivo mediante comandos de voz [8]. Otras tecnologías por ejemplo emplean, señales de encefalograma que pueden ser adquiridas por una EPOC Neuroheadset [9] o seguimiento de los ojos [6].

Tecnologías que ya existen en el mercado desde hace varios años se fundamentan en el uso de elementos mínimamente invasivos. Una de estas es Tongue Drive System que usa un piercing magnético que detecta movimientos de la lengua a través de precisamente campos magnéticos para controlar una silla de ruedas con 6 comandos [10] o un sensor de movimiento de cabeza extremadamente sensible *Gyroset*[11]. Otra tecnología ya usada en hospitales o aeropuertos es Self-Driving Wheelchairs que es una silla de ruedas eléctrica inteligente, aunque no es del todo autónoma para personas cuadripléjicas, con solo seleccionar el lugar destino a través de una pantalla táctil elige la ruta y se mueve evitando

obstáculos [12]. Además, se tienen sistemas para el control de una silla de ruedas comandada por voz como Easy Chair Voice [13] o Clever Chear [14].

En relación a trabajos que usen visión artificial se tiene [15] que usa un dispositivo comercial para el seguimiento de objetos y en base a los parámetros otorgados por el dispositivo se definen distintos movimientos, con la desventaja de que no es tan asequible en el mercado ecuatoriano y es costoso en comparación con la economía de nuestro país. Así también, se tiene una tecnología donde se realiza la detección y seguimiento a un conjunto de leds pegados en la parte superior de la cabeza para determinar el ángulo de inclinación y mover la silla [16]. En cuanto a operación del sistema, se tienen diferentes métodos como: empleando la forma de la boca para detener los motores de la silla y no se dispone del comando para retroceso [17] y otros donde el movimiento no se basa en la inclinación del rostro, sino en reconocer exclusivamente la dirección de movimiento de la nariz [18].

1.3.1.1 Personas con cuádrupleja en el Ecuador

En Ecuador, hasta junio del 2021, existen 471.322 personas con discapacidad registradas de las cuales el 45.78% es del tipo física y solo el 16.25% de este grupo tiene discapacidad mayor o igual al 75%, es decir, tienen cuádrupleja con afectación total de miembros superiores e inferiores [3]. En total hay 39 342 personas con esta condición, lo cual representa un número considerable de personas que están restringidas a vivir con independencia reducida [3].

Los avances tecnológicos y políticas de inclusión desplegadas a nivel mundial han favorecido a que las personas con falta de movilidad en sus extremidades puedan incursionar en oportunidades que antes resultaban impensables [19]. Una de estas oportunidades tiene que ver en el ámbito laboral pues, aunque en Ecuador solo el 4% de los trabajadores totales de una empresa debe ser personal discapacitado [20]. Este porcentaje se puede incrementar considerando que avances tecnológicos como teletrabajo y dispositivos controlados con el movimiento de los ojos o de la cabeza permiten un mayor desenvolvimiento de actividades [19].

1.3.2 VISIÓN ARTIFICIAL

La Inteligencia Artificial (IA) tiene varios aportes en distintos ámbitos con el fin de automatizar tareas o mejorar el desempeño al momento de realizarlas. Al estar desarrollada en distintas aplicaciones, cuando esta se enfoca en imágenes y videos se conoce como visión artificial [21]. Las tareas relacionadas con la visión artificial pueden ser detección de

objetos, clasificación de imágenes, seguimiento de objetos, segmentación de imágenes, predicción de movimiento y más.

El término artificial hace referencia a la capacidad que se les otorga a las computadoras de una inteligencia similar a la de los seres humanos, específicamente al procesamiento de datos visuales. Mientras que, la palabra visión se entiende por el hecho de que se trabaja con datos visuales como imágenes y videos los cuales se obtienen mediante una cámara. Este dispositivo detecta la intensidad de la luz para luego digitalizar el plano tridimensional capturado y poder representarlo en un plano de dos dimensiones.

El plano de dos dimensiones es la imagen digitalizada, es decir, son un arreglo o matriz bidimensional que están compuestas por muestras denominadas pixeles [21]. Las dos dimensiones (x, y) especifican la posición del pixel, donde cada pixel está formado por bits y los valores más comunes que puede tomar son entre 0 y 255. La representación de las imágenes capturadas es del tipo RGB, donde mediante la combinación 3 canales (colores) rojo (R, red), verde (G, green) y azul (B, blue) se consigue el espacio de colores [22]. El siguiente diagrama relaciona los componentes que se puede considerar en cuanto a video e imágenes.

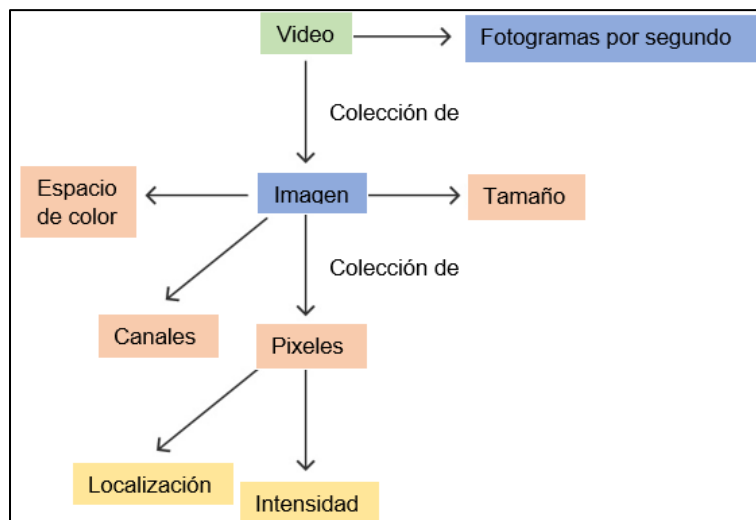


Figura 1.3. Relación de los componentes del video.

1.3.2.1 Formatos de presentación de imágenes

Los tipos de imágenes que soporta Matlab son los siguientes [23]:

Escala de grises: este tipo de imágenes está representada por una matriz de $M \times N$ pixeles, cuyos datos son del tipo double y denotan la intensidad de escala de grises desde 0 (negro) hasta 255 (blanco).

RGB: se representa la imagen como una matriz de datos tipo double tridimensional $M \times N \times 3$. Cada pixel tiene 3 componentes, los cuales se denotan como $(m,n,1)$ =rojo, $(m,n,2)$ =verde y $(m,n,3)$ =azul.

Indexada: estas imágenes constan de una matriz de índices $M \times N$ y una matriz del espacio de colores $k \times 3$, donde k es el número de colores que necesita la imagen para ser presentada.

Binaria: las imágenes se representan mediante una matriz $M \times N$ que contiene solo dos valores, 1 (blanco) y 0 (negro).

Uint8: la configuración predeterminada para almacenar información de matrices en Matlab es doble precisión, por lo que utiliza 64 bits para almacenar cualquier número. Sin embargo, 8 bits son suficientes para describir la intensidad del pixel y así se usa menos memoria [24].

1.3.2.2 Técnicas para la detección de objetos

El sistema detecta un rostro mediante la identificación de objetos de imágenes, por lo que se analizan los algoritmos más comunes enfocados en las características faciales. En la literatura existen varios métodos para cumplir con este propósito, algunos basados en imágenes o en las características. Para esto se resumen algunos de estos algoritmos:

1.3.2.2.1 Modelo ASM (Active Shape Model) [25]

El modelo de forma activa (ASM) se centra en características complejas no rígidas, como la apariencia física real y de nivel superior de las características. El objetivo principal de ASM es localizar automáticamente puntos de referencia que definen la forma de cualquier objeto modelado estadísticamente en una imagen. Por ejemplo, en una imagen del rostro del ser humano, se extraen rasgos como los ojos, los labios, la nariz, la boca y las cejas.

1.3.2.2.2 Algoritmo de Viola-Jones [25]

Este método se basa en el uso de características simples codificadas por diferentes métodos que se evalúan rápidamente mediante el uso de una nueva representación de imagen. Se genera un gran conjunto de características y utiliza el algoritmo de impulso AdaBoost para reducir el número de características y entrenar los clasificadores. Este algoritmo de detección de rostros es capaz de procesar imágenes extremadamente rápido mientras logra altas tasas de detección.

1.3.2.2.3 Redes neuronales convolucionales (CNN)

Se han desarrollado y demostrado varios métodos de aprendizaje profundo para la detección de rostros. Quizás uno de los enfoques más populares es 'Red neuronal

convolucional en cascada multitarea', o MTCNN por sus siglas en inglés [26]. La red utiliza una estructura en cascada con tres redes donde primero se cambia la escala de la imagen. “Las CNN propuestas constan de tres etapas. En la primera etapa, produce ventanas candidatas rápidamente a través de una CNN superficial. Luego, refina las ventanas para rechazar una gran cantidad de ventanas no caras a través de una CNN más compleja. Finalmente, utiliza una CNN más poderosa para refinar el resultado y emitir posiciones de puntos de referencia faciales” [27].

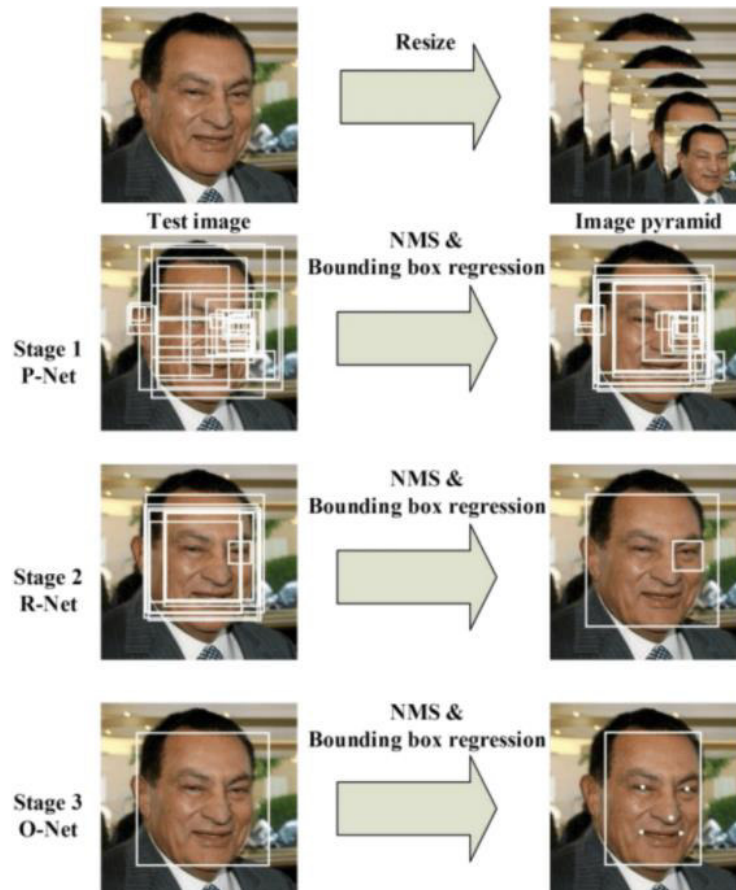


Figura 1.4. Funcionamiento de las redes neuronales convolucionales para la detección de rostros [27].

En base a las técnicas analizadas, se elige el método de Viola-Jones debido a su alta precisión, bajo consumo de recursos computacionales, facilidad de implementación y por la variedad de codificación de características que permiten trabajar en ambientes de baja iluminación. Además, el software Matlab tiene un clasificador de rostros pre entrenado basado en Viola-Jones. Por lo tanto, en la sección 1.3.3 se procede a explicar cómo se codifican las características de la imagen y el proceso de clasificación para determinar si

existe el objeto de interés. Una vez elegido el algoritmo de detección, se debe analizar cómo se logra seguir un objeto determinado.

1.3.2.3 Algoritmos de seguimientos de objetos

El algoritmo de Viola-Jones es uno de los algoritmos más rápidos y potentes en cuanto a la detección de objetos. Sin embargo, la detección de rostros en todos los frames de video implica una alta carga computacional [28]. Además, dependiendo del clasificador, el detector fallará si la persona inclina la cabeza o se mueve rápido. Por lo tanto, se necesita de un algoritmo alternativo y surgen los algoritmos de seguimiento. Sin embargo, la mayoría están enfocados en la aplicación para ámbitos específicos. Es por eso, que se analizan los algoritmos con mejor desempeño para el seguimiento del rostro.

1.3.2.3.1 Algoritmo Camshift [29]

El algoritmo Camshift se utiliza para mantener adaptativamente el tamaño y la ubicación adecuada de la ventana de búsqueda necesarios para diversos fines de experimentación. El principio del algoritmo Camshift es la distribución de probabilidad de color, la cual varía cada vez que las secuencias de fotogramas de vídeo cambian con el tiempo, por lo que se usa la imagen de distribución de probabilidad. Se calcula el centro objetivo de la imagen empleada por el algoritmo Camshift, es decir, la imagen de distribución de probabilidad.

1.3.2.3.2 Algoritmo Kanade-Lucas-Tomasi (KLT) [29]

El algoritmo rastrea el movimiento de los objetos en los frames de vídeo. La restricción de este modelo es una constancia de brillo y un movimiento de la imagen pequeño. Este algoritmo detecta un conjunto de puntos de objeto a través de los fotogramas de vídeo. Una vez completada la detección de rostros, los puntos de característica facial deben identificarse y que se pueden rastrear constantemente. El rastreador de puntos rastrea el punto a lo largo del número de fotogramas uno por uno haciendo referencia al fotograma anterior. Después de la detección de rostros, el siguiente proceso es extraer información de las expresiones faciales que están presentes allí. Varios rasgos faciales permanentes de los que dependen los diversos enfoques son las cejas, los ojos, la boca, etc.

De acuerdo con los algoritmos más robustos para rostros, se elige el KLT. Pues en [29] se concluye que el algoritmo Camshift incluye el seguimiento de otros objetos y parte del rostro es excluida. Aunque, el método KLT presenta restricciones de desplazamiento entre frames, para los propósitos del sistema es suficiente puesto que no se necesita el seguimiento a grandes velocidades. La solución al problema del seguimiento de objeto se

explica en la sección 1.3.4. Además, con un estimador geométrico aplicado después del seguimiento, este tiene un mejor desempeño, cuyo proceso se detalla en la sección 1.3.5.

1.3.3 ALGORITMO DE DETECCIÓN DE OBJETOS VIOLA-JONES

Para la detección del rostro en el sistema se usa el objeto de Matlab *vision.CascadeObjectDetector*. Para el funcionamiento de este objeto se debe especificar el modelo clasificación, los cuales pueden ser basados en características LBP o Haar; sin embargo en [30] se encontró que el usar estas características la detección es menos confiable y más lenta. Además, necesita de la configuración de los parámetros: *ScaleFactor*, define en cuánto incrementa el tamaño de la ventana de búsqueda; y *MergeThreshold*, determina cuántas detecciones mínimas deben haber en dicha ventana para decretar la existencia del objeto. Estos parámetros del objeto de Matlab explican cómo se codifican las características ni cómo se determina que la ventana contiene el objeto de interés, por tal motivo a continuación se explican dichos procesos acorde con el algoritmo Viola Jones que tiene 3 características principales:

1. Una nueva representación de imágenes llamada Imagen Integral.
2. Algoritmo de aprendizaje basado en AdaBoost.
3. Un método para combinar clasificadores en forma de cascada.

1.3.3.1 Extracción de características

En este paso se hace una introducción de una nueva representación de imágenes, denominada 'Imagen Integral'. Sin embargo, en este sistema se usa el algoritmo LBP (*Local Binary Pattern*) para la codificación de las características debido a su mayor robustez.

1.3.3.1.1 Imagen integral [31]

Es una representación intermedia de la imagen, la cual contiene la suma de los pixeles del lado superior izquierdo desde un punto específico (x,y) como en la Figura 1.5.

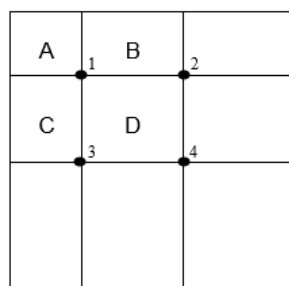


Figura 1.5. Suma de pixeles de una Imagen Integral [31]. El valor en el punto 1 es la suma de pixeles del rectángulo A; el del punto 2, es A+B; el punto 3, es A+C; y el punto 4, es A+B+C+D. La suma dentro de D se puede obtener como $4 + 1 - 2 - 3$.

1.3.3.1.2 Patrón binario local (LBP, Local Binary Pattern)

Esta técnica se enfoca en obtener texturas que no sean propensas a la rotación ni a la escala de grises para caracterizar imágenes y toma como referencia los pixeles vecinos. LBP es bastante robusto contra la variación de la posición o de la iluminación en comparación con otros métodos [32]. La textura tiene dos propiedades: estructura espacial y contraste. La estructura espacial es el conjunto de pixeles que conforman el patrón y el contraste es la diferencia entre la intensidad de los pixeles, la estructura espacial es afectada por la rotación y el contraste por la escala de grises [33].

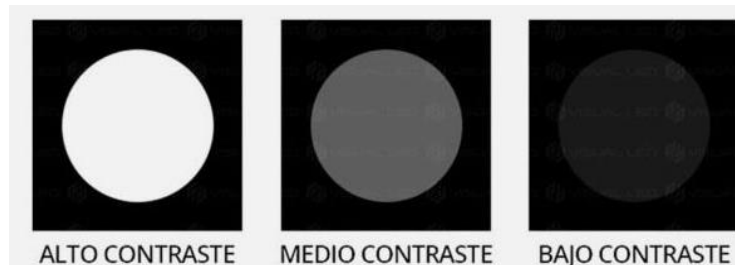


Figura 1.6. Ejemplo de diferentes niveles de contraste [34].

Para un caso general, se define el operador de textura $LBP_{p,R}^{riu2}$ dentro de un conjunto simétrico de P miembros en un círculo con radio R . P controla la cuantificación del espacio angular y R determina la resolución espacial.

Se define una textura T en un vecindario local con la intensidad de P ($P > 1$) pixeles.

$$T = t(g_c, g_0, \dots, g_{P-1}) \quad (1.1)$$

Donde:

g_c : intensidad de la escala de grises del pixel central.

g_p ($p = 0, \dots, P - 1$): intensidad de P pixeles.

El pixel central está ubicado en $(0,0)$ y la numeración de los p pixeles se realiza en sentido contrario a las manecillas del reloj tomando el pixel $p=0$ en la posición $(0,R)$ (Figura 1.7).

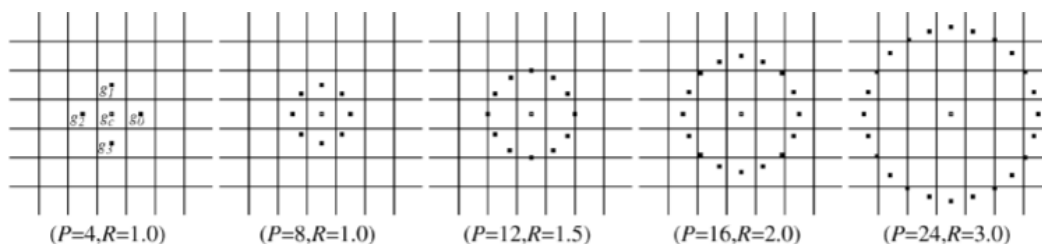


Figura 1.7. Conjuntos de vecinos circularmente simétricos para diferentes $(P; R)$ [33].

Como primer paso se resta el valor de la intensidad del pixel central a cada pixel p . Luego, se asume que $g_p - g_c$ es independiente de g_c . La distribución $t(g_c)$ describe la luminancia general de la imagen y no proporciona información útil para la textura [33]. Por lo tanto:

$$T \approx t(g_0 - g_c, \dots, g_{P-1} - g_c) \quad (1.2)$$

Los signos de los resultados de las restas en (1.2) no se ven afectados por el cambio de g_c . Por lo que solo se consideran los signos de estas diferencias.

$$T \approx t(s(g_0 - g_c), \dots, s(g_{P-1} - g_c)) \quad (1.3)$$

Donde:

$$s(x) \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

Se transforma el operador (1.3) en un número $LBP_{P,R}$ único asignando un factor 2^p a cada signo $s(g_p - g_c)$. Este número caracteriza la estructura espacial local.

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c) 2^p \quad (1.4)$$

Se define ahora el término patrón uniforme como la estructura circular que contiene muy pocas transiciones espaciales. Para determinar si un patrón es uniforme se define también una medida de uniformidad U , la cual corresponde al número de transiciones de 0 a 1 o viceversa en el patrón. Para ejemplificar:

$$\text{Patrón (Secuencia de bits)} = b_0b_1b_2b_3b_4b_5b_6b_7$$

Donde:

b_0 : bit menos significativo

b_7 : bit más significativo

El patrón binario 00000000 no presenta ninguna transición, por lo tanto, $U = 0$.

El patrón binario 00001000 presenta dos transiciones: entre b_4 y b_5 hay una transición de 0L a 1L y entre b_5 y b_6 hay una segunda transición de 1L a 0L, $U = 2$.

El patrón binario 00101001 presenta 6 transiciones: b_1 - b_2 , b_2 - b_3 , b_3 - b_4 , b_4 - b_5 , b_6 - b_7 y b_7 - b_0 . La última transición b_7 - b_0 se considera ya que el patrón representa un círculo y el último bit está enlazado con el primero, $U = 6$.

En [33] se define como patrón uniforme si U es menor o igual a 2 y se propone el siguiente operador invariante a la escala de grises y a la rotación:

$$LBP_{P,R}^{riu2} = \begin{cases} \sum_{p=0}^{P-1} s(g_p - g_c) 2^p & \text{si } U(LBP_{P,R}) \leq 2 \\ P + 1 & \text{otros casos} \end{cases} \quad (1.5)$$

Finalmente, el histograma de cada muestra se concatena en uno solo describiendo la textura global (Figura 1.8).

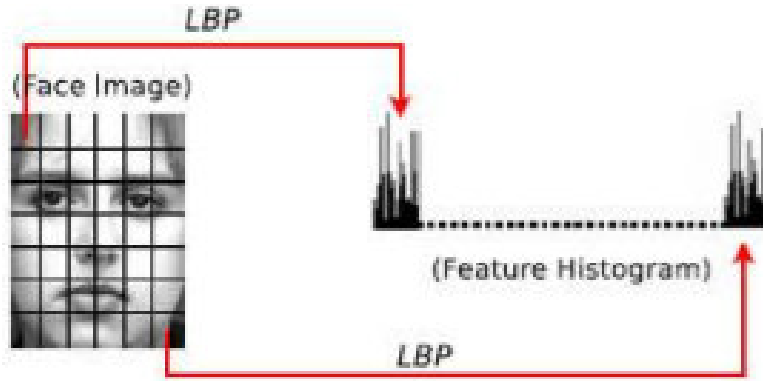


Figura 1.8. Ejemplo de la concatenación de histogramas de cada característica LBP [35].

1.3.3.2 Clasificación de características

La resolución mínima en la que trabaja el algoritmo es de 24x24, dando como resultado un número muy alto de características. Por lo que se busca un conjunto pequeño de características para que al combinarlas se pueda formar un clasificador fuerte [31]. Un algoritmo de aprendizaje débil selecciona el único intervalo del histograma LBP que separa mejor las muestras positivas y negativas [35]. El clasificador débil $h_j(x)$ determina la función óptima de clasificación de umbral.

1.3.3.2.1 Algoritmo AdaBoost para el aprendizaje de clasificadores [31]

Dadas las imágenes de muestra $(x_1, y_1), \dots, (x_n, y_n)$ donde y_i representa la salida que debería tener la imagen de muestra x_i correspondiente. La variable y_i puede tener un valor de 0 para ejemplos negativos (no rostros) y 1 para positivos (rostros).

El algoritmo cambia los pesos utilizados para el cálculo del error de clasificación respecto a los pesos de los clasificadores débiles. Se inicializan los pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0|1$ respectivamente, donde m es el número total de muestras negativas y l el número total de muestras positivas.

El proceso de cálculo de pesos se repite T veces y se describe a continuación:

Se normaliza los pesos:

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}} \quad (1.6)$$

Donde w_t es una probabilidad de distribución.

Para cada característica j , se entrena el clasificador h_j , el cual está restringido para usar una sola característica. El error es evaluado con respecto a w_t , $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.

Donde ϵ_j es el error de clasificación.

Se elige el clasificador h_t de la repetición t con menor error ϵ_t y se actualizan los pesos:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \quad (1.7)$$

donde $e_i = 0$ si el ejemplo x_i es clasificado correctamente, $e_i = 1$ si no se clasifica la muestra de forma correcta y $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$. Esto hace que un pequeño error se pondere más y asegura que las 2 mejores características no sean similares como en la Figura 1.9. Al final el clasificador fuerte queda expresado de la siguiente forma:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{en otros casos} \end{cases} \quad (1.8)$$

donde $\alpha_t = \log\left(\frac{1}{\beta_t}\right)$

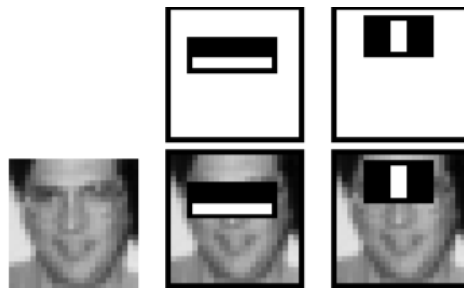


Figura 1.9. Ejemplo de las dos mejores características elegidas por AdaBoost [31]. La primera característica es la diferencia de intensidad entre la región de los ojos y la región de la parte superior de las mejillas. La segunda característica compara las intensidades en las regiones de los ojos con la intensidad del entrecejo.

1.3.3.3 Entrenamiento de clasificadores en cascada

Los clasificadores fuertes resultantes del algoritmo AdaBoost son llamados etapas y se estructuran en forma de cascada etiquetando la región actual de la sub-ventana. Si la etiqueta es negativa, la clasificación de esa sub-ventana está completa y se pasa a analizar la siguiente región. Por otro lado, si la etiqueta es positiva, el clasificador pasa la región de esa sub-ventana a la siguiente etapa. Así se determina la existencia de un objeto en esa región solo si la etapa final la clasifica como positiva. De esta manera se enfocan en las regiones etiquetadas como positivas y descartan las muestras negativas lo más rápido posible [36].

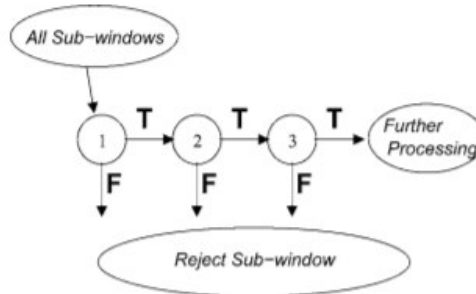


Figura 1.10. Funcionamiento de los clasificadores en cascada [37].

1.3.4 ALGORITMO DE SEGUIMIENTO DE OBJETOS KANADE-LUCAS-TOMASI

Para el seguimiento del rostro detectado se usa el objeto de Matlab *vision.PointTracker*, el cual tiene de entrada un conjunto de características determinado por el algoritmo de Matlab *detectMinEigenFeatures*. La extracción de características que sigue esta función es propia del algoritmo KLT para así poder rastrear dichas características con el objeto seguidor mencionado. Este objeto necesita de la configuración de los parámetros: *NumPyramidLevels*, niveles de resolución, *MaxBidirectionalError*, error máximo permitido luego hacer el seguimiento; *BlockSize*, región de la imagen donde ocurre el algoritmo KLT; y *MaxIterations*, número de intentos hasta converger. La función de extracción de características nos devuelve varios valores y los parámetros del objeto de Matlab definen ciertas partes del algoritmo de manera aislada, pero no dan una idea clara de cómo se logra el seguimiento del objeto en una región determinada; por lo que a continuación, se explica dicho proceso.

El algoritmo KLT basa su funcionamiento en obtener adecuadas características para realizar el seguimiento diferentes a las que han propuesto en [38][39][40] como una 'característica buena' independientemente del algoritmo de seguimiento [41]. El método KLT se resume en 3 pasos [42]:

- 1) Se extraen características de la región de interés, realizado por el objeto detector.
- 2) Se rastrea el fotograma actual a partir del anterior.
- 3) El rastreador se utiliza para estimar la escala, la rotación y la traslación.

1.3.4.1 Modelo de movimiento de imágenes [41][43]

En la ventana donde se realice el seguimiento se debe poder obtener características con buenas propiedades de textura. Una imagen está definida por los patrones de intensidad como una función $I(x, y, t)$ de tres variables discretas tanto en espacio como en tiempo. Las

imágenes tomadas entre tiempos cortos están fuertemente correlacionadas, ya que es la misma escena desde ángulos ligeramente diferentes. La función $I(x, y, t)$ satisface la siguiente propiedad.

$$I(x, y, t + \tau) = I(x - \xi, y - \eta, t) \quad (1.9)$$

Donde:

$I(x, y, t + \tau)$: imagen actual después de un tiempo $t + \tau$.

$I(x - \xi, y - \eta, t)$: imagen anterior desplazada ξ en x y η en el eje y al tiempo t .

(ξ, η) : cantidad de movimiento desde la imagen actual hasta la siguiente.

En base a la cantidad de pixeles (movimiento) entre los patrones de intensidad dos imágenes se define el vector desplazamiento como:

$$\delta = (\xi, \eta)$$

δ : desplazamiento del punto $x = (x, y)$ entre los tiempos t y τ .

El problema con los patrones de intensidad es que un solo pixel no puede ser seguido, ya que puede cambiar por el ruido. Como solución a eso se puede rastrear a una ventana de pixeles que contenga suficiente textura; sin embargo, en una ventana se tienen muchos puntos. Al tener muchos puntos, estos se pueden mover a diferentes velocidades y puede aparecer y desaparecer nuevos puntos presentando los siguientes inconvenientes:

- ¿Cómo sabemos que estamos siguiendo la misma ventana?
- ¿Cómo se combinan las diferentes velocidades para dar un solo vector resultante de desplazamiento?

La solución al primer problema es sencilla, se monitorea la apariencia de la ventana. En cambio, el segundo problema es un poco más complejo, para esto se estima un vector desplazamiento para ventanas pequeñas de tal forma que se minimice el error residual. La forma más fácil de esto es considerar como error cuando se tiene discrepancia que no pueda ser explicada en la traslación dos fotogramas sucesivos.

Se redefinen las funciones sin tomar en cuenta la variable temporal modificando (1.9):

$$J(x) = I(x - \delta) + n(x) \quad (1.10)$$

Donde:

$J(x)$: imagen actual.

$I(x - \delta)$: imagen anterior desplazada una cantidad δ .

$n(x)$: ruido.

El error residual está definido por la siguiente integral de superficie sobre una ventana W .

$$\epsilon = \iint_W [I(x - \delta) - J(x)]^2 \omega dx \quad (1.11)$$

Donde:

ϵ : error residual entre dos tramas consecutivas.

ω : función de pesos indicar qué parte de la ventana tiene mayor peso ponderad.

Cuando el vector desplazamiento es mucho más pequeño que W , para reducir ϵ es más eficiente usar el método de linealización propuesto en [44] explicado en la siguiente sección.

1.3.4.2 Resolviendo el desplazamiento de imágenes [34]

Si el vector desplazamiento δ entre dos frames es suficientemente pequeño, la función de intensidad puede ser aproximada por su serie truncada de Taylor a términos lineales:

$$I(x - \delta) = I(x) - g \cdot \delta \quad (1.12)$$

g : gradiente de la función de intensidad de la imagen expresado como $g = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$.

Se escribe la ecuación (1.11) que define el error residual como:

$$\epsilon = \iint_W [I(x) - g \cdot \delta - J(x)]^2 \omega dx = \iint_W (h(x) - g \cdot \delta)^2 \omega dx \quad (1.13)$$

Donde:

$$h(x) = I(x) - J(x)$$

Se puede obtener el valor mínimo del error residual diferenciando la ecuación (1.13) con respecto a δ e igualando el resultado a cero:

$$\iint_W (h(x) - g \cdot \delta) g \omega dx = 0 \quad (1.14)$$

Si δ es constante dentro de W , sabiendo que $(g \cdot \delta)g = (gg^T)\delta$ y teniendo en cuenta que $\delta = (\xi, \eta)$, se tiene un sistema de dos ecuaciones con dos incógnitas:

$$G\delta = e \quad (1.15)$$

Donde:

G : representa la matriz de coeficientes y es calculada a partir de un fotograma:

$$G = \iint_W gg^T \omega dx \quad (1.16)$$

e : define el vector de dos dimensiones y se puede obtener a partir de la diferencia entre los dos fotogramas junto con el gradiente calculado anteriormente:

$$e = \iint_W (I(x) - J(x))g \omega dx \quad (1.17)$$

Por lo tanto, el vector desplazamiento δ es la solución del sistema. En la práctica se necesitan de muy pocas iteraciones para llegar a converger.

1.3.4.3 Selección de características [34]

En una imagen no todas las regiones contienen información adecuada, escogiendo las que su sistema (1.15) presente buenas mediciones y pueda ser resuelto confiablemente.

El sistema presenta buenas condiciones si los elementos de la matriz de coeficientes G son mayores al nivel de ruido de la imagen y está bien condicionada, es decir, no pueden diferir en varios órdenes de magnitud. Los valores propios λ_1 y λ_2 característicos de la matriz G pueden presentarse 3 casos: ambos valores propios son pequeños, esto significa que el perfil de intensidad es constante dentro de una ventana; un valor propio es grande y el otro pequeño, lo que indica un patrón de intensidad unidireccional; y λ_1 y λ_2 grandes, esto representan esquinas, texturas *salt-and-pepper* como la Figura 1.11 o cualquier otro patrón.



Figura 1.11. Ejemplos de imágenes con texturas *salt-and-pepper* [45][46] definidas como imágenes entremezcladas de blanco y negro.

En la práctica, el valor propio más pequeño que está por encima del nivel de ruido, se puede decir que la matriz G está bien condicionada. Esto debido a que el valor del pixel máximo está limitado y, por lo tanto, el mayor λ no es arbitrariamente grande en comparación con el menor valor propio. Entonces, la ventana es aceptada como buena para seguir si:

$$\min(\lambda_1, \lambda_2) > \lambda \quad (1.25)$$

Donde:

λ : umbral predefinido por el nivel de ruido.

El algoritmo de seguimiento funciona adecuadamente a corto plazo. Esto debido a que conforme pasa el tiempo las características elegidas pueden perderse por la variación de la iluminación, movimientos o rotación fuera del plano. Por lo que se debe adquirir los puntos periódicamente para tener un buen rastreador a largo plazo.

1.3.5 ALGORITMO DE ESTIMACIÓN GEOMÉTRICA

Los algoritmos de seguimiento necesitan una región de interés para realizar el proceso respectivo y al evitar hacer la detección del rostro en todos los fotogramas, se necesita de algún método para obtener dichas regiones. Por lo tanto, se debe considerar perspectivas geométricas como líneas rectas y planos que engloben todas las características en forma de puntos en dos dimensiones. Con este fin se usan estimadores de estas perspectivas [47].

En Matlab se tiene un objeto estimador de planos *estimateGeometricTransform2D*. Este estimador se basa en el algoritmo MSAC (*M-estimator Sample Consensus*). El estimador de Matlab necesita de parámetros para su funcionamiento: *MaxNumTrials*, número máximo de muestreos aleatorios; *Confidence*, confianza de encontrar una nueva característica; y *MaxDistance*, distancia máxima entre un punto de la primera imagen y su proyección en la siguiente. Todos los parámetros mencionados permiten identificar los *inliers* (puntos coincidentes) que describen la superficie; sin embargo, es necesario entender cómo se logran identificar dichos puntos coincidentes. Por tal motivo se explica el proceso del algoritmo en el cual se basa el objeto de Matlab.

1.3.5.1 MODELO RANSAC

Para explicar las posibles relaciones sobre el movimiento de puntos entre dos vistas considere el movimiento entre dos imágenes puntuales de un objeto. Después del movimiento, el conjunto de puntos x_i de la primera imagen se transforma al conjunto x'_i en la segunda imagen como se muestra en la Figura 1.12. Si todos los puntos observados se encuentran en un plano, entonces las correspondencias se encuentran en una proyectividad dada por la matriz H mediante la relación:

$$x' = Hx \quad (1.26)$$

Donde:

x' : conjunto de puntos de la primera imagen.

x : conjunto de puntos de la segunda imagen.

H : matriz de proyectividad.

Cada par de puntos correspondientes x, x' bidimensional define un solo punto en un espacio de medición R^4 . Este espacio es el “espacio de imagen conjunta” [48]. Por lo que se asume que las dos imágenes tienen el mismo modelo de ruido en; por lo tanto, se puede utilizar la misma medida de distancia para minimizar el error en cada imagen.

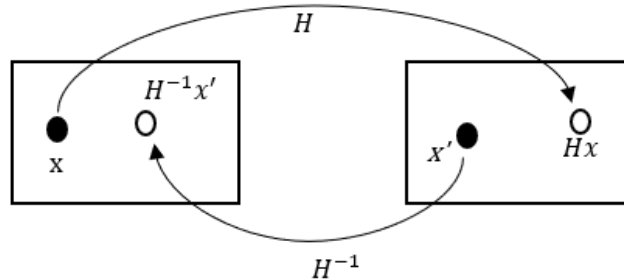


Figura 1.12. Proyección de los puntos x en la segunda imagen y x' en la primera imagen. El punto Hx es la proyección del punto x en la segunda imagen. Por otro lado, el punto $H^{-1}x'$ es la proyección del punto x' en la primera imagen. La distancia mínima se toma entre los puntos originales (negros) y los estimados (blancos).

Para cada correspondencia $x_i \leftrightarrow x'_i$ se debe encontrar la estimación de los puntos de máxima coincidencia $\hat{x}_i \leftrightarrow \hat{x}'_i$ tal que minimicen el error e_i^2 , dado por:

$$e_i^2 = \sum_{j=1,2} (\hat{x}_i^j - x_i^j)^2 + (\hat{y}_i^j - y_i^j)^2 \quad (1.27)$$

Donde:

e_i^2 : error de Estimación de Máxima Verosimilitud

\hat{x}_i^j : coordenada en x del i -ésimo punto proyectado en la imagen j .

\hat{y}_i^j : coordenada en y del i -ésimo punto proyectado en la imagen j .

Por lo tanto, la ecuación (1.27) proporciona la función de error para los datos puntuales. Este cálculo requiere una coincidencia inicial de puntos (esquinas) sobre los pares de imágenes. Las esquinas se detectan utilizando el algoritmo de detección de características planteado en la sección 1.3.4.3.

Las coincidencias correctas obedecerán a la geometría epipolar como se muestra en la Figura 1.13. La geometría epipolar es el conjunto de puntos proyectados en un plano diferente al que pertenecen y está definida por la matriz fundamental F . Las supuestas matrices fundamentales se calculan a partir de conjuntos aleatorios de siete correspondencias de esquinas formando la matriz de datos Z :

$$Z = \begin{pmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_7x_7 & x'_7y_7 & x'_7 & y'_7x_7 & y'_7y_7 & y'_7 & x_7 & y_7 & 1 \end{pmatrix}$$

La solución para F se puede obtener del espacio nulo de Z. Sea Z una matriz de dimensión $m \times n$. De acuerdo con [49] el espacio nulo de una matriz $m \times n$ presenta la restricción $n > m$ y su dimensión ha de ser al menos $n - m$. Es así como, la matriz de datos Z, donde $m = 7$ y $n = 9$, presenta una dimensión de espacio nulo bidimensional que cumple:

$$Zf = 0$$

Donde f está formada por $[f_1 \ f_2]$. f_1 y f_2 son los vectores singulares del espacio nulo bidimensional de Z.

Para las proyectividades, se usa la matriz H que proporciona las proyecciones para todos los puntos. Así se determina el número de correspondencias con error por debajo de un cierto umbral determinado por (1.29).

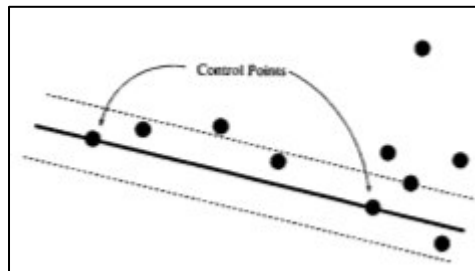


Figura 1.13. Ejemplo de plano epipolar [48].

En la Figura 1.13 se muestra el resultado de MSAC ajustando una línea a un conjunto de puntos, la línea en negrita es la ajustada, las líneas de trazos finos son los umbrales para determinar si los puntos son adyacentes.

Los *inliers* (coincidencias) se utilizan para ajustar un plano. Uno de los problemas de RANSAC es que, si el umbral T de la función de costo para los *inliers* se establece demasiado alto, la estimación robusta no es eficiente [50]. Entonces, para esta situación se obtiene una nueva función de costo [48].

$$C_2 = \sum_i p_2(e_i^2) \quad (1.28)$$

Donde:

C_2 : nueva función de costo o el error cuadrático medio.

e : función de error y σ es una desviación estándar.

p_2 : nuevo término de error robusto que se representa como:

$$p(e_i^2) = \begin{cases} e^2 & e^2 < T^2 \\ T^2 & e^2 \geq T^2 \end{cases} \quad (1.29)$$

2. METODOLOGÍA

En este capítulo se detallan las etapas y los pasos en el diseño del sistema de control de la silla de ruedas con la opción de ingreso/salida del modo de comandos que presenta los mejores resultados como es el seguimiento de la nariz. El sistema está basado en visión artificial que detecta los movimientos de la cabeza y los interpreta en función del modo de comandos. Se desarrolla este sistema con el fin de no ser invasivo, es decir, sin tener que ubicar algún elemento adicional en la cabeza como requisito para su funcionamiento.

2.1 DISEÑO DEL SISTEMA DE SIMULACIÓN DE CONTROL DE LA SILLA DE RUEDAS

El sistema se diseña para que funcione mediante un modo de comandos, el cual tiene 5 opciones de movimiento de la cabeza: posición normal, inclinación a la derecha, a la izquierda, hacia adelante y a hacia atrás. Estos movimientos se ilustran en la Figura 2.1.



Figura 2.1. Movimientos de la cabeza para el sistema de control de la silla.

2.1.1 ETAPAS DEL SISTEMA

El sistema se diseña por etapas para facilitar su comprensión y desarrollo, de esta forma se tienen 7 etapas interconectadas entre ellas como se indica en la Figura 1.2. Algunas etapas se desarrollan en fases debido a que se realizan varios procesos, estas fases con sus entradas y salidas se detallan al inicio de cada etapa en la correspondiente de este capítulo.

1. Adquisición y procesamiento de imágenes
2. Detección de rostro
3. Obtención del rectángulo de la nariz
4. Seguimiento de rostro y nariz
5. Detección de la dirección de giro de la cabeza
6. Obtención de parámetros para el movimiento de la silla
7. Presentación del movimiento de la Silla mediante una Interfaz Gráfica.

2.1.2 RECURSOS USADOS PARA LA SIMULACIÓN DEL DISEÑO

Para implementar el sistema se dispone de recursos tanto de hardware como de software:

- Cámara de visión nocturna de retroceso de vehículos con conectores RCA.
- Batería de 12V DC.
- Adaptador RCA macho-macho.
- Capturadora de video EasyCap USB 2.0.
- Laptop Lenovo Legion Y-540
- Software Matlab versión R2021A.
- Paquetes de Matlab: Matlab support Package for USB Webcams, Image Processing Toolbox y Computer Vision System Toolbox.

La forma en que se interconectan los recursos se muestra en la Figura 2.2. El hardware corresponde a una cámara infrarroja para el retroceso de vehículos, una batería, adaptador RCA, capturadora de video EasyCap y una laptop. Mientras que, el software usado es Matlab versión 2021A, el cual se ha decidido usar debido al licenciamiento que se dispone, la cantidad de documentación oficial y la facilidad de usar los objetos de visión artificial como en [36]. Sin embargo, la licencia que se necesita es una desventaja sino se dispone de una debido a su costo.

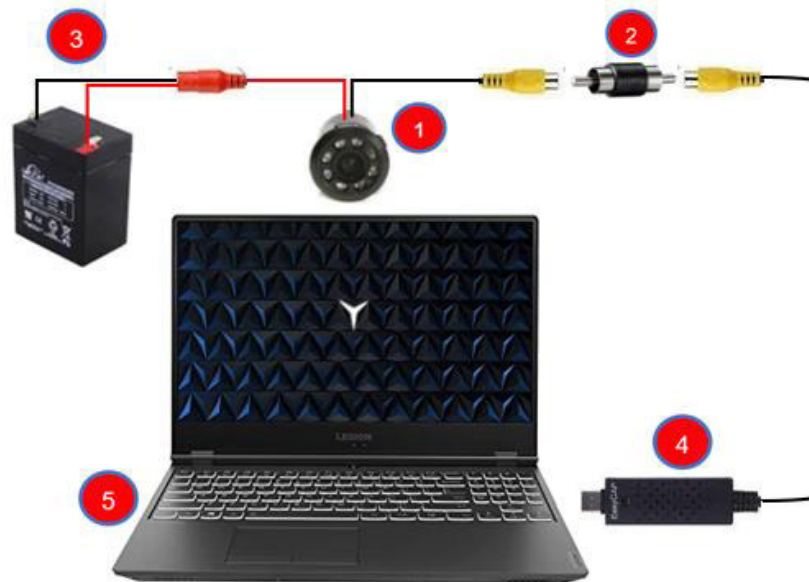


Figura 2.2. Esquema de conexión de los recursos. 1) Cámara de Visión Nocturna. 2) Adaptador RCA Macho-Macho. 3) Batería de 12V. 4) Capturadora de Video EasyCap USB 2.0. 5) Computador con Matlab R2021A.

2.1.3 CONDICIONES DE FUNCIONAMIENTO

Para cumplir con la detección inicial del rostro se debe cumplir las siguientes condiciones :

- La cámara debe estar en ángulo entre 25° y 45° enfocando al rostro.
- La persona debe estar frente a la cámara procurando mantener la cabeza recta y con la menor inclinación posible.
- Para la distancia de la cámara al rostro, se debe guiar por la sección verde de la cámara de retroceso de vehículo, la cual debe estar al nivel de las cejas como se muestra en la Figura 2.3 para tener el mejor desempeño. Sin embargo, es suficiente con que alguna parte del rostro esté ubicada entre las líneas verdes con la condición de que el rostro no se mueva fuera del plano de visión de la cámara.
- En ambientes con ausencia completa de iluminación se debe evitar el uso de lentes de cualquier tipo. Para ambientes con baja y buena iluminación se puede hacer el uso de lentes con lunas transparentes.

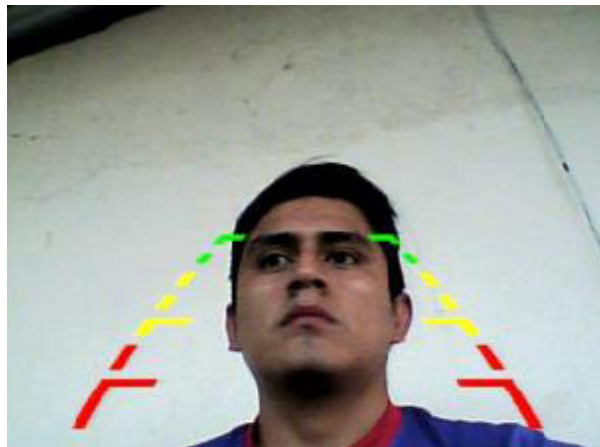


Figura 2.3. Posición inicial del rostro.

Para el movimiento del rostro se deben cumplir las siguientes condiciones:

- El movimiento de la cabeza debe ser lento y el fondo de color uniforme.
- Después del movimiento de la cabeza para entrar o salir del modo de comandos, el usuario debe volver a la posición inicial antes de que transcurran 3 segundos.
- El sistema presenta mejor desempeño cuando la cabeza está delante de fondos uniformes. Por lo que no es viable el uso en ambientes exteriores.
- Para ingresar o salir del modo de comandos, los dos lados del rostro deben estar uniformemente iluminados o no debe haber una diferencia significativa.
- Para detener el movimiento de la silla en el eje horizontal se debe tener la cabeza en un rango de inclinación máximo de 10° ya sea a la derecha o a la izquierda. En

el eje vertical para el comando 'Alto' se debe volver la posición de la cabeza pudiendo estar un 10% más cerca o más lejos respecto a la inicial.

2.2 ETAPA 1: ADQUISICIÓN Y PROCESAMIENTO DE IMÁGENES



Figura 2.4. Entrada y salida de la etapa 1. La entrada es el objeto de cámara creado y la salida es la imagen capturada que pasa por un proceso ecualización de histograma.

En esta etapa se capturan imágenes mediante un dispositivo de detección de energía [23]. El sistema propuesto adquiere fotogramas en tiempo real del rostro de una persona a través de una cámara de visión nocturna. Se usa este tipo de cámara con el fin de que el sistema soporte su utilización en ambientes con malas condiciones de iluminación. Las imágenes capturadas pueden tener una resolución de 640x480 o menor, se escoge para la presentación del trabajo escrito 480x320 (480 pixeles de ancho y 320 pixeles de alto), esta es la mínima resolución posible de la capturadora de video.

Se conecta la cámara web mediante la función *webcam*, para luego adquirir una imagen con la función *snapshot* [51]. Las imágenes capturadas pueden no tener un buen contraste por lo que se realiza la ecualización de histograma. Finalmente se muestran las imágenes en una de las figuras de la interfaz gráfica presentada.

2.2.1 CONFIGURACIÓN DEL OBJETO PARA LA CÁMARA

Se dispone de dos cámaras conectadas a la computadora donde se desarrolla el software. Por lo tanto, para saber el nombre de la cámara que se usará se usa la función *webcamlist*.

```
2x1 cell array
      {'Integrated Camera'}
      {'AV TO USB2.0'      }
```

Figura 2.5. Nombres de las cámaras disponibles en la computadora.

2.2.1.1 Creación del objeto

Una vez determinado el nombre de la cámara a usar, el cual es 'AV TO USB2.0' que tendrá como argumento la función *webcam*. Se escoge la menor resolución que permite el dispositivo elegido, la cual es 480x320. Esto debido a que se busca reducir el procesamiento necesario para llevar a cabo el software.

- **webcam:** crea un objeto de adquisición de imágenes *cam* estableciendo una conexión con una cámara web *deviceName*.
 - **Sintaxis:** `cam = webcam('deviceName')`
 - **Propiedades:**
 - Name: nombre de la cámara web.
 - Resolution: resolución del fotograma adquirido.
 - AvailableResolutions: lista de resoluciones disponibles en la cámara.

```
cam=webcam('AV TO USB2.0')
cam.Resolution='480x320';
```

2.2.2. CAPTURA DE FOTOGRAMAS

Las imágenes se empiezan a capturar con la función *snapshot* y se almacena como *camFrame* para luego mostrarse en la interfaz gráfica hasta que esta se cierre. Esto se consigue evaluando si el tamaño de la interfaz gráfica es mayor a 0. Los nuevos fotogramas se van mostrando con la función *imshow*.

- **snapshot:** adquiere una sola imagen *camFrame* del objeto cámara *cam*.
 - **Sintaxis:** `camFrame = snapshot(cam)`
- **imshow:** muestra una imagen *I* con los ejes y propiedades optimizadas.
 - **Sintaxis:** `imshow(camFrame)`

```
camFrame = snapshot(cam);
imshow(camFrame);
```

2.2.3. CONVERSIÓN A ESCALA DE GRISES

La imagen capturada por la cámara está en formato RGB (Rojo, Verde y Azul) para conformar el espacio de colores que representen a una determinada imagen. Para esto se tienen 3 matrices de dimensión MxN píxeles, donde cada una representa a un color en específico. Sin embargo, se puede reducir el procesamiento en las siguientes etapas si se caracteriza la imagen con una sola matriz del espacio de colores. Esto se puede realizar extrayendo un canal de los 3 que captura la cámara, pero dependiendo de la iluminación una imagen podría estar representada mejor por un canal que por los demás. Por tal motivo, se escoge la opción de convertirla a una escala de grises con la función *im2gray*, que representa un fotograma en base a una sola matriz de píxeles con intensidad de escala de grises desde 0 (negro) hasta 255 (blanco). Para fines de presentación en la interfaz gráfica se guarda la imagen *camFrame* en su formato original RGB *camFrameRGB*, pero esta no se usa en las siguientes etapas de diseño.

```
camFrameRGB = camFrame;  
camFrame = im2gray(camFrame);
```



(a)

(b)

Figura 2.6. Resultado de la conversión de una imagen en formato RGB a escala de grises. (a) Imagen en formato Original. (b) Imagen en escala de grises.

2.2.4. ECUALIZACIÓN DE HISTOGRAMA

El histograma es una representación gráfica de la distribución de la intensidad de los píxeles de una imagen [22] como se muestra en la Figura 2.7.

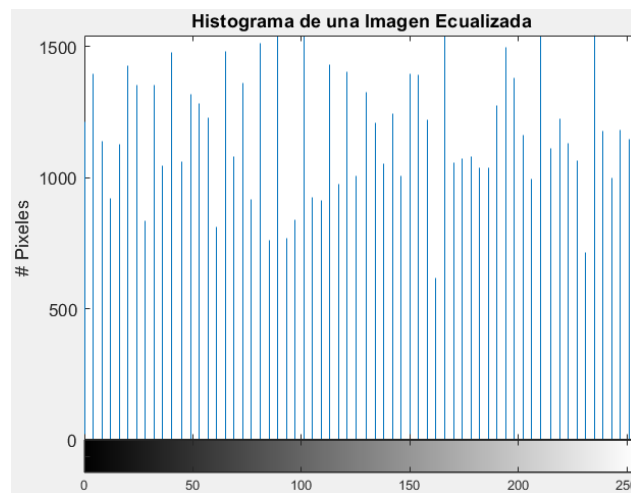


Figura 2.7. Histograma de una imagen en formato escala de grises.

La ecualización de histograma tiene como objetivo mejorar el contraste de la imagen modificando su histograma. La modificación que realiza es que, si el histograma está concentrado en una intensidad específica, este se esparce en todos los valores de intensidad logrando una distribución uniforme y como resultado una imagen de alto contraste como se aprecia en la Figura 2.8.

En Matlab, se tiene la función *histeq* para realizar la ecualización de histograma.

- **histeq**: devuelve una imagen *camFrame* con el contraste mejorado a través la ecualización de histograma de la imagen capturada.
 - **Sintaxis**: `camFrame = histeq(camFrame)`



Figura 2.8. Resultado de la ecualización de histograma. (a) Imagen Original. (b) Imagen ecualizada.

El diagrama de flujo de la Figura 2.9 muestra el proceso de adquisición y ecualización de histograma de imágenes.

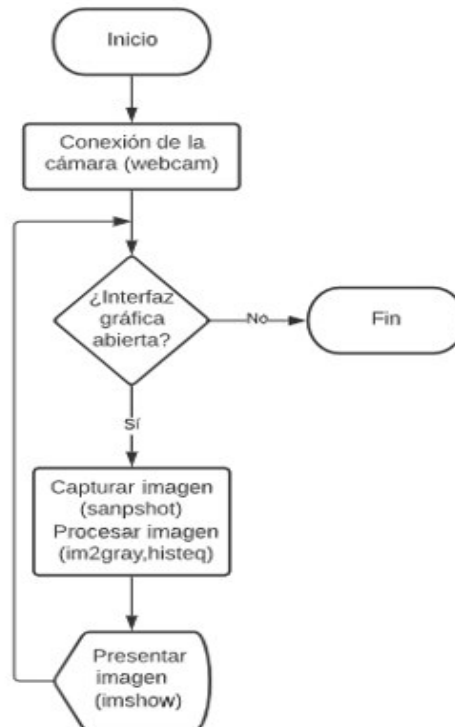


Figura 2.9. Diagrama de flujo para la adquisición y procesamiento de las imágenes.

El código completo de esta etapa se muestra a continuación.

```
cam=webcam('AV TO USB2.0')
cam.Resolution='480x320';
while size(InterfazMovimientoSilla)>0%Se adquieren imágenes hasta
cerrar el guide
    camFrame = snapshot(cam);% Se captura un frame
    camFrameRGB = camFrame;
    camFrame = im2gray(camFrame);%Se convierte una imagen RGB a escala
de grises
    camFrame = histeq(camFrame);% Se realiza la ecualización de
histrograma del frame
    axes(handles.axes2)%Se especifica el objeto axes donde se presenta
la imagen
    imshow(camFrameRGB);%Se presenta el frame con los rectángulos y
los puntos
end
```

2.3 ETAPA 2: DETECCIÓN DE ROSTRO



Figura 2.10. Entrada y salida de la etapa 2. La entrada es el fotograma capturado y procesado de la etapa 1 y la salida es el rectángulo detectado donde se encuentra el rostro.

En esta etapa se detecta el rostro de una sola persona usando el algoritmo de Viola Jones. Este algoritmo es utilizado en la función *vision.CascadeObjectDetector* de Matlab.

2.3.2 CREACIÓN DEL DETECTOR

Se elige el objeto de sistema *vision.CascadeObjectDetector*, el cual tiene algunos clasificadores pre-entrenados para detectar rostros de frente, rostros de perfil, nariz, ojos, y la parte superior del cuerpo dividiendo la imagen en rectángulos. El tamaño de los rectángulos varía para detectar objetos a diferentes escalas, pero su relación de aspecto es constante. Por lo tanto, debido a que la relación de aspecto en la mayoría de los objetos de tres dimensiones cambia, este detector es muy sensible a la rotación [36].

Para la creación del objeto se especifica un umbral de detección denotado en Matlab como *MergeThreshold*. Este umbral es un escalar entero y permite definir criterios para una detección final en un rectángulo donde hay varias detecciones alrededor del objeto. Se crea un cuadro delimitador con los grupos de detecciones que cumplen el umbral. Si se aumenta

el valor del umbral, mejora la detección del objeto debido a que reduce el número de detecciones falsas, pero si el valor es muy alto se es demasiado restrictivo y el tiempo en detectar un rostro aumenta.

El objeto de sistema usado en Matlab es el siguiente:

- **vision.CascadeObjectDetector:** crea un detector para un objeto especificado.
 - **Sintaxis:** `faceDetector = vision.CascadeObjectDetector('modelo de clasificación')`
 - **Argumento de entrada:**
ClassificationModel: se especifica el modelo de clasificación del objeto, el cual determina el tipo de característica a usar y el objeto a detectar. Los modelos se detallan en la Tabla 2.1.
 - **Propiedades:**
MinSize: el tamaño en pixeles de la región más pequeña que puede contener un objeto definido por un ancho y un alto: [*height width*], por defecto no están definidos ninguno de los dos valores, es decir, un vector sin elementos.
MaxSize: el tamaño en pixeles de la región más grande que puede contener un objeto, por defecto es está vacío. Esta propiedad junto con *MinSize* no ayudan a mejorar el proceso de clasificación, sino que permiten reducir la carga computacional.
ScaleFactor: valor mayor que 1.0001 usado para incrementar la escala de la sub-ventana a utilizar como se muestra en la Figura 2.11. Por defecto, este valor es 1.1.
MergeThreshold: número entero que permite controlar el número de detecciones necesarias antes de combinar o rechazar las detecciones como se muestra en la Figura 2.12.

Tabla 2.1. Modelos de clasificación de *vision.CascadeObjectDetector*.

Modelo de clasificación (<i>ClassificationModel</i>)	Descripción del Modelo
'FrontalFaceCART' (Default)	Detecta rostros que están en posición vertical y mirando hacia al frente codificando los detalles del rostro con características Haar.
'FrontalFaceLBP'	Detecta rostros que están en posición vertical y mirando hacia al frente codificando los detalles del rostro con patrones binarios locales (LBP).
'UpperBody'	Detecta la parte superior del cuerpo (cabeza y hombros) codificando esta región con características Haar.
'EyePairBig' 'EyePairSmall'	Detecta un par de ojos, donde el modelo 'EyePairSmall' detecta ojos más pequeños que el modelo 'EyePairBig'.
'LeftEye' 'RightEye'	Detecta el ojo izquierdo y derecho por separado codificando estas regiones con características Haar.
'LeftEyeCART' 'RightEyeCART'	Detecta el ojo izquierdo y derecho por separado.
'ProfileFace'	Detecta perfiles faciales verticales codificando los rasgos faciales con características Haar.
'Mouth'	Detecta bocas codificando sus detalles con características Haar.
'Nose'	Detecta narices codificando sus rasgos con características Haar.

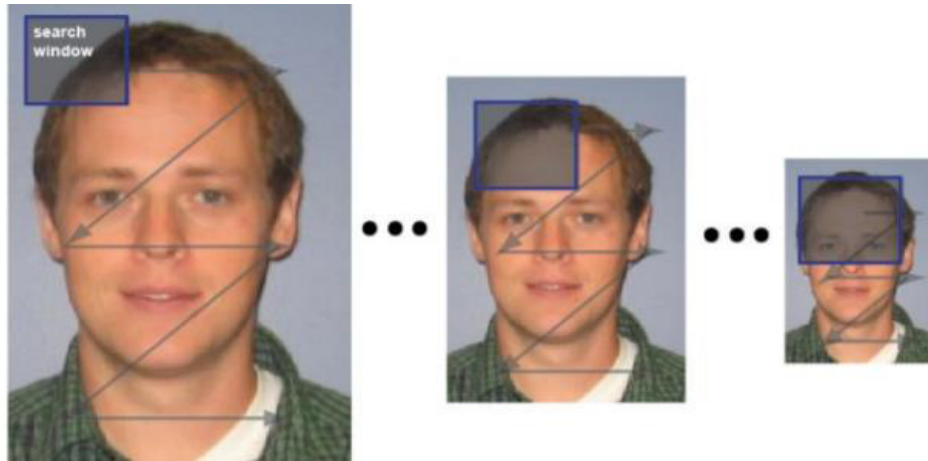


Figura 2.11. Ejemplo del incremento de la ventana de búsqueda definido por el ScaleFactor [36].

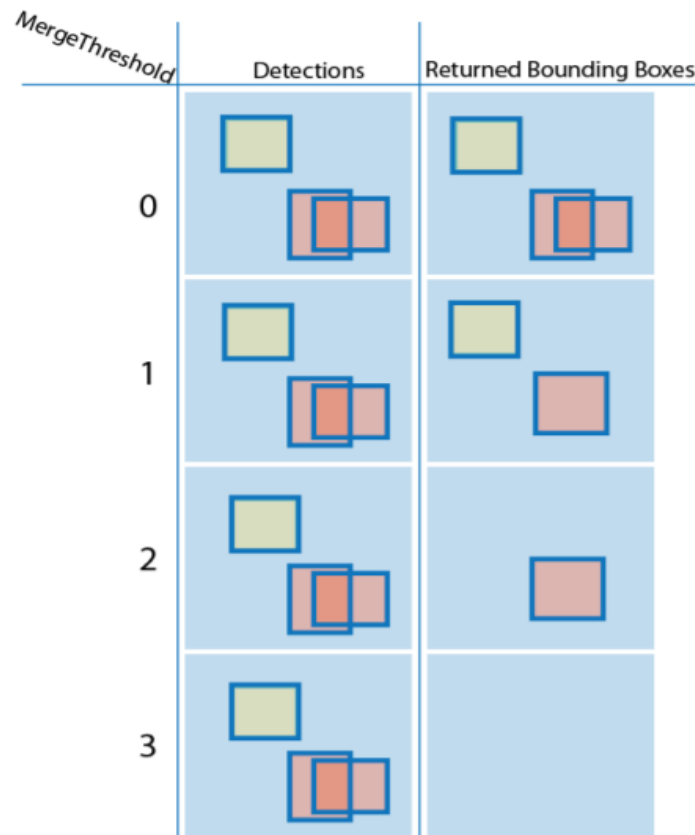


Figura 2.12. Ejemplo de la utilización de la propiedad *MergeThreshold* [36]. El cuadro delimitador final es un promedio de los cuadros delimitadores para las detecciones individuales.

El código para los parámetros modificados se indica a continuación:

```
faceDetector = vision.CascadeObjectDetector('FrontalFaceLBP');  
faceDetector.MergeThreshold = 11;
```

2.3.3 VALIDACIÓN DE ROSTRO EN IMÁGENES

La imagen capturada y procesada en la etapa 1 sirve como entrada para el objeto detector creado. Existe la posibilidad de que no se haya detectado ningún rostro en la imagen, por lo que se usará una variable *NewDetection* para determinar si se ha encontrado un rostro, cuyo valor es *1* si no se ha detectado un rostro o *-1* si ya se realizó la detección. Además, permite una pausa de 3 segundos hasta colocar la cabeza en la posición correspondiente cuando se necesite una nueva detección, más adelante se especifica cuándo se debe hacer esta nueva detección. La pausa se realiza mediante los comandos *tic* y *toc* en un bucle *while* donde se usan las líneas de código de la etapa 1 para presentar los fotogramas.

```
tic  
while toc < 3  
    camFrameRGB = snapshot(cam);  
    imshow(camFrameRGB);  
end  
toc
```

Para que no se procesen imágenes en las cuales no haya rostros, se valida que exista por lo menos un objeto detectado, lo cual se realiza con la función *isempty*, resultando el siguiente código:

```
NewDetection = 1;%Bandera para determinar una detección o volver a  
detectar el rostro cuando se pierden las características seguidas  
if NewDetection == 1  
    % Extracción de los 4 puntos de la caja del rostro  
    face = faceDetector(camFrame);% Se almacenan los rectángulos de  
los rostros detectados en camFrame  
    if isempty(face)%Si no se detecta un rostro  
        continue%pasa al siguiente frame  
    else  
        %Acciones a realizar si se detecta un rostro  
        NewDetection = -1;%Se cambia el valor de la Bandera de  
detección  
    end
```

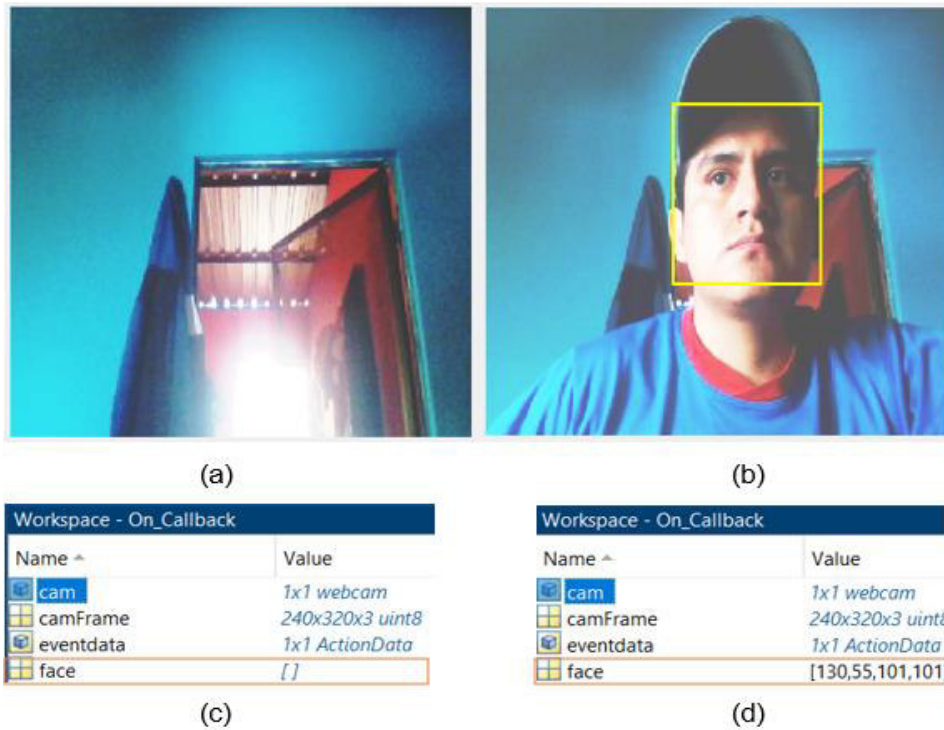


Figura 2.13. Casos posibles en la validación del rostro. (a) Sin un rostro para detectar. (b) Con un rostro detectado. (c) Al no haber ningún rostro la variable *face* está vacía. (d) La variable *face* contiene los elementos correspondientes del rectángulo.

Los rectángulos obtenidos en la detección están definidos en un vector con los elementos $[x_0 \ y_0 \ w_0 \ h_0]$.

x_0 : posición del eje x en pixeles donde empieza el rostro de derecha a izquierda.

y_0 : posición del eje y en pixeles donde empieza el rostro de arriba hacia abajo.

w_0 : ancho del rostro.

h_0 : altura del rostro.

También se pueden tener varios rostros en la imagen. Por lo que, el detector devolverá las coordenadas de los rectángulos donde se hayan encontrado rostros. Los cuales se almacenan en *face*; sin embargo, el sistema está diseñado para que solo funcione con un rostro. Por lo tanto, se escoge el rectángulo del rostro detectado más cercano a la cámara, para lo cual se determina el índice k del ancho mayor M de todos los rostros detectados con la función *max* y se asignan solo los valores de esa fila k . Además, se establece el número de fotograma *NumFrame* donde se detectó el rostro como el primero para uso en las siguientes etapas.

```

%face(:,3): El ancho de todos los rostros detectados
[M,k]=max(face(:,3));
face = face(k,:);
NumFrame=1;

```



(a)



(b)

camFrameRGB	1152x864x3 uint8
face	[676,721,147,147;310,222,178,178]
faceDetector	1x1 CascadeObjectDetector

(c)

camFrameRGB	1152x864x3 uint8
face	[310,222,178,178]
faceDetector	1x1 CascadeObjectDetector

(d)

Figura 2.14. Ejemplo de detección de más de un rostro. (a) Dos rostros detectados. (b) Se selecciona el rectángulo del rostro más prominente. (c) El vector *face* tiene una dimensión 2x4, una fila por cada rostro. (d) Se almacena el rostro más cercano.

El diagrama de flujo para la validación de la existencia de un rostro y selección del más prominente se muestra en la Figura 2.15.

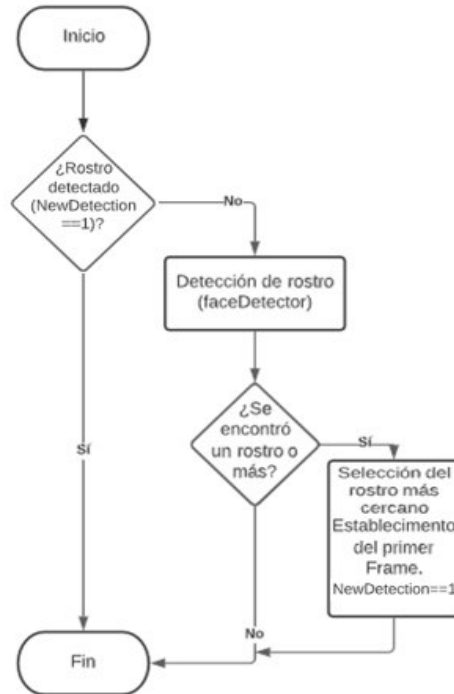


Figura 2.15. Diagrama de flujo para la validación y elección de un solo rostro.

El código completo para la detección y validación de rostros en un fotograma queda de la siguiente forma:

```

NewDetection = 1;%Bandera para determinar una detección o volver a
detectar el rostro cuando se pierden las características seguidas
if NewDetection == 1
    tic%Pausa de tres segundos después del giro o para una nueva
detección hasta volver a la posición inicial
    while toc < 3
        camFrameRGB = snapshot(cam);% Se captura un frame
        axes(handles.axes2)
        imshow(camFrameRGB);%Se presenta el frame con los rectángulos
y los puntos
    end
    toc
    % Extracción de los 4 puntos de la caja del rostro
    face = faceDetector(camFrame);% Se almacenan los rectángulos de
los rostros detectados en camFrame
    if isempty(face)%Si no se detecta un rostro
        continue
    else
        %face(:,3): El ancho de todos los rostros detectados
        [M,k]=max(face(:,3));%Se determina la fila k del ancho más
grande M.
        [face = face(k,:); % Selección del rectángulo del rostro más
prominente
        NumFrame=1;%Se establece el primer frame
        NewDetection = -1;%Se cambia el valor de la Bandera
    end
end
end
  
```

2.4 ETAPA 3: OBTENCIÓN DEL RECTÁNGULO DE LA NARIZ

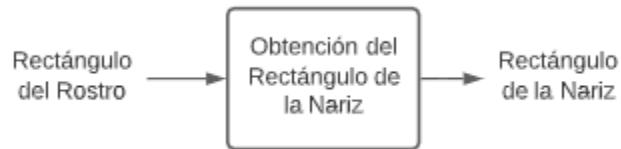


Figura 2.16. Entrada y salida de la etapa 3. La entrada es el rectángulo de la etapa 3 y la salida es la porción del rectángulo del rostro donde se encuentra la nariz.

En esta etapa se obtiene la sección donde se encuentra la nariz, la cual se adquiere a través de las proporciones estéticas comunes de la cara [52]. El eje horizontal de la cara se divide en cinco partes iguales equivalentes al ancho de la nariz. Mientras que, la altura facial se divide en tres partes iguales, estas divisiones se muestran en la Figura 2.17.

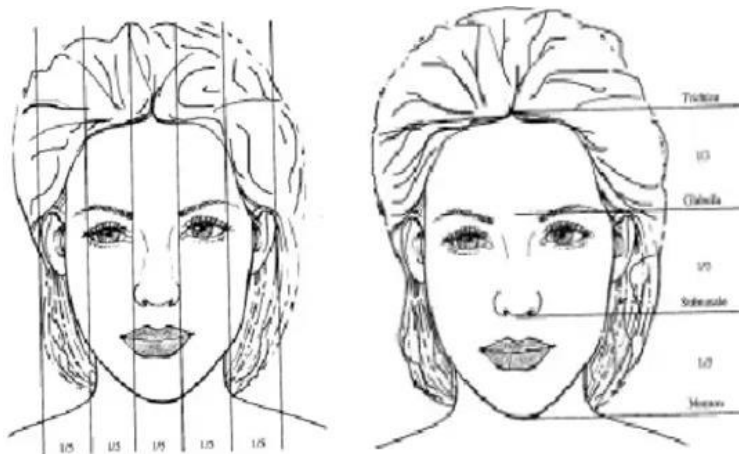


Figura 2.17. Proporciones normales del rostro [52].

En el sistema a simular, las divisiones mencionadas en el eje horizontal y vertical se realizan sobre el rectángulo adquirido en la etapa 2. Para obtener el rectángulo de la nariz se usa la forma realizada en [53], donde el vector $face = [x_0 \ y_0 \ w_0 \ h_0]$ corresponde a las coordenadas de la cara y el vector $nose = [x_1 \ y_1 \ w_1 \ h_1]$ es el recuadro que se obtiene de la nariz.

- x_1 es dos veces un quinto del ancho del rostro más la coordenada en x donde inicia el rostro. La ecuación es la siguiente: $x_1 = x_0 + \frac{2 \cdot w_0}{5}$
- y_1 es el un tercio del alto del rostro más la coordenada en y donde inicia el rostro. La ecuación es la siguiente: $y_1 = y_0 + \frac{h_0}{3}$
- w_1 es el un quinto del ancho del rostro. La ecuación es la siguiente: $w_1 = \frac{w_0}{5}$
- h_1 es el un tercio del alto del rostro. La ecuación es la siguiente: $h_1 = \frac{h_0}{3}$

```
nose = [ face (1)+2*face (3)/5 face (2)+face (4)/3 face (3)/5 face (4)/3 ] ;
```

Name ^	Value
cam	1x1 webcam
camFrame	240x320x3 uint8
eventdata	1x1 ActionData
face	[107,28,98,98]

Figura 2.18. Ejemplo del rectángulo del rostro como entrada para la etapa 3.

Aplicando las fórmulas para obtener el rectángulo de la nariz en el rectángulo del rostro se muestra un ejemplo de los valores en la Figura 2.19.

Name v	Value
nose	[146.2000,60.6667,19.6000,32.6667]

Figura 2.19. Ejemplo del rectángulo obtenido de la nariz en base a las fórmulas según las proporciones normales del rostro.

Para observar los rectángulos del rostro y de la nariz como se muestra en la Figura 2.20 se usa la función *insertShape*, detallada más adelante en la etapa 7.

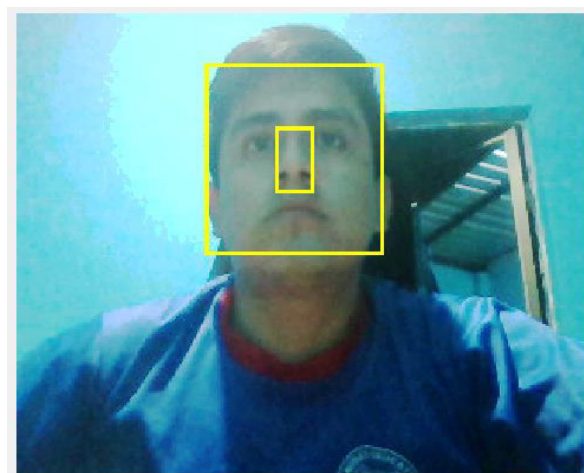


Figura 2.20. Rectángulos del rostro y nariz en la imagen adquirida por la cámara.

2.5 ETAPA 4: SEGUIMIENTO DE ROSTRO Y NARIZ

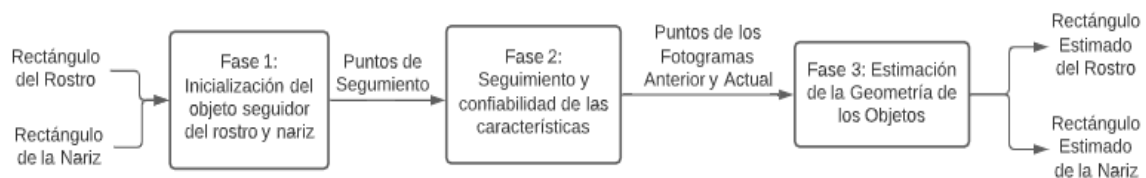


Figura 2.21. Entradas y salidas de la etapa 4. Las entradas son los rectángulos del rostro de la etapa 2 y de la nariz de la etapa 3. En la Fase 1, se inicializa el objeto rastreador; en

la Fase 2, se realiza el seguimiento de los puntos y se determina su confiabilidad; y en la Fase 3, se obtienen los nuevos rectángulos del rostro y nariz.

En esta etapa se realiza el seguimiento los objetos detectados de rostro y nariz usando el algoritmo KLT. Este algoritmo es usado por el objeto *vision.PointTracker* de Matlab para realizar el seguimiento de puntos, mejor especificados como características, en una fuente de video. Este objeto seguidor funciona bien particularmente para objetos que no cambien de forma.

2.5.1 CREACIÓN DEL OBJETO SEGUIDOR

Para la etapa de seguimiento se usa el objeto de Matlab *vision.PointTracker*, el cual crea un objeto seguidor para validar el movimiento de las características que se encuentran dentro de los rectángulos de interés. Dado que se realiza dos veces el mismo proceso, se establecen funciones para evitar repetir líneas de código. Estas funciones se aplican según la fase del seguimiento: en la fase 1 se inicializan los puntos a seguir y en la fase 2 se realiza el seguimiento de las características encontradas en la fase 1.

2.5.1.1 Fase 1: inicializando el objeto seguidor

La función creada para esta fase se denomina *TrackerInit* y su función es crear el objeto seguidor, identificar las características del objeto que se desea seguir e inicializa los puntos de las características en la imagen presentada. Los parámetros de entrada y salida de la función se especifican a continuación:

Entradas:

- *obj*: rectángulo del objeto a seguir especificado de la forma $[x_0 \ y_0 \ width \ height]$.
- *camFrame*: imagen donde se encuentra el objeto requerido.

Salidas:

- *objTracker*: objeto seguidor *objTracker*.
- *Box*: coordenadas (x,y) de los vértices del rectángulo donde se encuentra el objeto.
- *oldPoints*: puntos encontrados para realizar el seguimiento en el siguiente frame.

En el objeto seguidor creado se especifican las propiedades analizadas en la sección 1.3.4; sin embargo, no se encarga de obtener las características apropiadas para el seguimiento. Sino que hace uso de las características de otros algoritmos como: *Features from accelerated segment test (FAST)*, *Minimum eigenvalue*, *Harris-Stephens*, etc. El algoritmo elegido es *Minimum eigenvalue* explicado en la sección 1.3.4.3. Este algoritmo es el que

presenta mejor rendimiento para el seguimiento del rostro obteniendo más características buenas y siendo más estable ante el movimiento del objeto de interés.

La función usada para esta fase es *detectMinEigenFeatures* que devuelve entre otros parámetros la posición de los puntos como se muestra en la Figura 2.22. Los parámetros se almacenan en *points*; sin embargo, es de interés solo la localización de todas las características definidas en la matriz *Location*. Esta matriz es precisamente la salida *oldPoints* de la función *TrackerInit*.

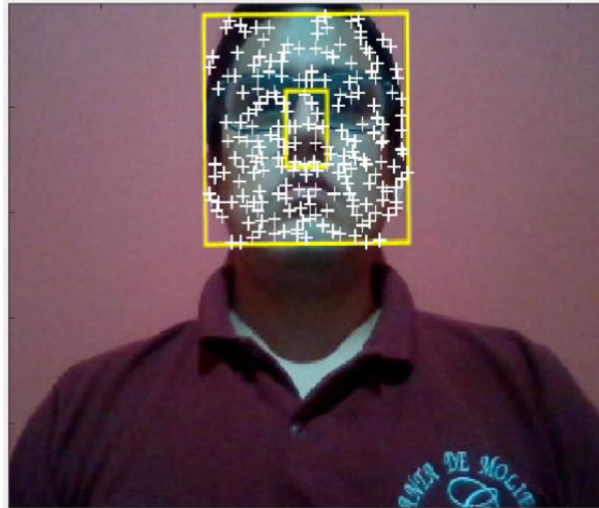


Figura 2.22. Características para el seguimiento encontradas en la región del rostro y nariz.

- **detectMinEigenFeatures:** devuelve un objeto *points* que contiene información de los puntos característicos detectados en una imagen en formato escala de grises.
 - **Sintaxis:** `points = detectMinEigenFeatures(camFrame)`
 - **Propiedades:**
 - MinQuality: calidad mínima aceptada de las esquinas (características) especificado dentro del rango [0,1], donde 1 es una calidad del 100%. Un mayor valor elimina esquinas erróneas. Por defecto, este valor es 0.01.
 - FilterSize: dimensión del filtro Gaussiano, este filtro suaviza el gradiente de la imagen de entrada. Por defecto este valor es 5.
 - ROI: región rectangular para la detección de las características especificado como un vector de la forma $[x_0 \ y_0 \ width \ height]$.

```
points = detectMinEigenFeatures(im2gray(camFrame), 'ROI', Obj);  
oldPoints = points.Location;
```

Una vez obtenidas las características que se seguirán entre fotogramas de video se crea el objeto rastreador *objTracker* con el sistema de objeto *vision.PointTracker* especificado por Matlab, los valores de los parámetros se justifican en el capítulo 3.

- **vision.PointTracker:** devuelve un objeto de seguimiento de puntos utilizando el algoritmo de seguimiento de características de Kanade-Lucas-Tomasi (KLT).
 - **Sintaxis:** `objTracker = vision.PointTracker('Name', value)`. 'Name' y *value* son el nombre de la propiedad y su respectivo valor.
 - **Propiedades:**
 - NumPyramidLevels: valor entero que especifica el número de niveles de la pirámide. Esta imagen piramidal como se observa en la Figura 2.23 rastrea los puntos en múltiples niveles de resolución. Por defecto este valor es 3.
 - MaxBidirectionalError: número entero que define un umbral de error bidireccional. Se interpreta como la distancia en píxeles desde la localización del punto original hasta la localización final después del *backward track* como se puede ver en la Figura 2.24. Este valor se usa eliminar puntos que no se pudieron seguir de manera confiable y por defecto no está configurado.
 - BlockSize: vector de dos elementos impares que especifica el alto y ancho de la sub-ventana para el cálculo del gradiente. Por defecto tiene un valor de [31 31].
 - MaxIterations: valor entero positivo que establece el máximo número iteraciones en la búsqueda de la ubicación de cada punto. Por defecto este valor es 30.

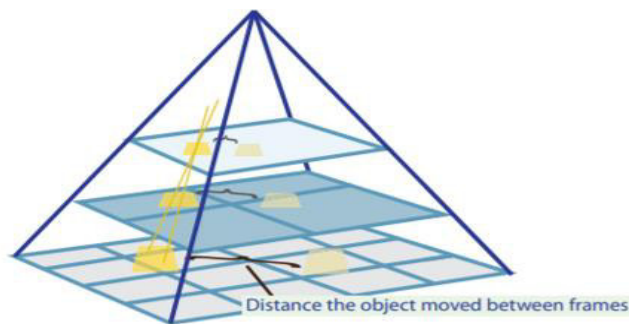


Figura 2.23. Imagen piramidal del objeto seguidor con 3 niveles de resolución. Cada nivel en forma descendente se incrementa en un factor de 2.

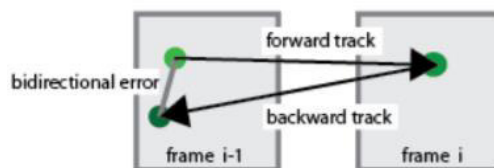


Figura 2.24. Error bidireccional después de hacer un doble seguimiento de la característica. Se rastrea el punto del frame *i-1* en el *i* (*forward track*) y luego ese mismo

punto en la imagen i se sigue en la imagen anterior $i-1$ (*backward track*). Un error mayor a este valor, invalida esa característica.

Teniendo el objeto seguidor y las posiciones de los puntos, se debe enlazar estos parámetros con el primer fotograma donde se detectó el rostro y del cual se extrajeron las características. Para este propósito se usa la función *initialize*.

- **initialize:** inicializa el frame de video y los puntos para el seguimiento.
 - **Sintaxis:** initialize(objTracker, oldPoints, camFrame)

```
objTracker = vision.PointTracker('MaxBidirectionalError',4,...  
    'BlockSize',[41 41],'MaxIterations',40,'NumPyramidLevels',4);  
initialize(objTracker, oldPoints, camFrame);
```

Se ha detallado el proceso para la obtención de dos salidas de la función para inicializar el objeto; sin embargo, no se ha tomado en cuenta la salida *Box*. Este rectángulo es el mismo que se obtiene en la etapa de detección del rostro y obtención de la zona de la nariz. La única diferencia es que no está especificado de la forma $[x_0, y_0, width, height]$, sino que ahora se representa en forma de los vértices $[x_0, y_0; x_1, y_1; x_2, y_2; x_3, y_3]$ del rectángulo. Se necesita cambiar la forma de la representación debido a que la función *transformPointsForward* usada y detallada más adelante, no acepta la definida por el sistema detector. Para facilitar el cambio de representación se usa la función *bbox2points*

- **bbox2points:** convierte un rectángulo $[x_0, y_0, width, height]$ en una lista de 4 puntos de esquina.
 - **Sintaxis:** points = bbox2points(rectangle).

```
Box = bbox2points(Obj);
```

La función *TrackerInit* queda de la siguiente manera:

```
function [objTracker, Box, oldPoints] = TrackerInit(camFrame,Obj)  
%Etapa 4  
%Fase 1: Inicialización del Objeto de Seguimiento  
%Cambio de representación del rectángulo  
Box = bbox2points(Obj);  
%Identificación de Características  
points = detectMinEigenFeatures(im2gray(camFrame), 'ROI', Obj);  
oldPoints = points.Location;  
%Inicialización de los puntos  
objTracker = vision.PointTracker('MaxBidirectionalError',4,...  
    'BlockSize',[41 41],'MaxIterations',40,'NumPyramidLevels',4);  
%Se establecen los puntos a seguir en la imagen  
initialize(objTracker, oldPoints, camFrame);  
end
```

La información que proporciona el objeto detector de características luego de ejecutar la línea de código correspondiente es: *Location*, localización de la característica; *Count*, número de puntos; y *Metric*, fuerza de la característica detectada.

Property ^	Value
Location	375x2 single
Metric	375x1 single
Count	375

Figura 2.25. Parámetros proporcionados por el objeto que detecta las características.

El objeto seguidor *objTracker* queda establecido como se muestra en la Figura 2.26.

```
faceTracker =
    vision.PointTracker with properties:
        MaxBidirectionalError: 4
            BlockSize: [41 41]
        NumPyramidLevels: 4
        MaxIterations: 40
```

Figura 2.26. Valores de las propiedades con los que funcionará el objeto seguidor.

Name ^	Value
Box	[100,16;220,16;220,136;100,136]
Obj	[100,16,120,120]

Figura 2.27. Ejemplo del cambio de representación de un rectángulo *Obj* definido por su altura y ancho a uno *Box* especificado por la posición de sus vértices.

La fase 1 con la función *TrackerInit* detallada se aplica para tanto para el objeto del rostro como para el de la nariz, resultando las siguientes líneas de código:

```
%Fase 1
%Iniciando el objeto seguidor del Rostro
[faceTracker, faceBox, oldFacePoints] = TrackerInit(camFrame,face);
%faceBox: Coordenadas (x,y) de los 4 vértices del rectángulo de la
cara
%faceTracker: Objeto seguidor del rostro
%oldFacePoints: Puntos para el seguimiento del rostro en el siguiente
Frame

%Iniciando el objeto seguidor de la nariz
[noseTracker, noseBox, oldNosePoints] = TrackerInit(camFrame,nose);
%noseBox: Coordenadas (x,y) de los 4 vértices del rectángulo de la
nariz
%noseTracker: Objeto seguidor de la nariz
%oldNosePoints: Puntos para el seguimiento de la nariz en el siguiente
Frame
```

2.5.1.2 Fase 2: seguimiento y confiabilidad de las características

Para esta fase y la siguiente de la etapa 4 se crea la función *Tracker* que realiza propiamente el seguimiento de las características del objeto entre los frames de video consecutivos y obtiene el rectángulo que las contiene. Los parámetros de entrada y salida de esta función son:

Entradas

- *camFrame*: imagen capturada por la cámara.
- *objTracker*: objeto seguidor de las características.
- *oldPoints*: puntos donde se encuentran las características del frame anterior.
- *Box*: rectángulo del fotograma anterior donde está el objeto de interés.

Salidas

- *oldPoints*: puntos actualizados para el siguiente frame.
- *Box*: rectángulo estimado donde se encuentra el objeto en el frame actual.
- *NewDetection*: variable que determina si se debe realizar una nueva detección.

2.5.1.2.1 Validación de la confiabilidad de las características.

Luego de haber inicializado el objeto seguidor con el primer fotograma, se debe comparar en el siguiente si las características correspondientes aún son confiables. Si las características siguen siendo confiables se las propaga frame tras frame, sino simplemente se las descarta. Para esta tarea el objeto seguidor *objTracker* devuelve, además de la localización de los puntos seguidos, un parámetro denominado *validity* para confiabilidad. Este parámetro es del tipo lógico devuelto como una matriz $M \times 1$, donde M es la cantidad de puntos; sin embargo, solo los que sean confiables estarán marcados por 1 y los que serán descartados por 0 en la posición acorde con la matriz de puntos.

Este procedimiento filtra todas las características *oldPoints* del frame anterior y todas las características *points* del actual, de tal manera que se haga el seguimiento solo a las que aún sean confiables. Una característica deja de ser confiable si: queda fuera del plano que puede captar la cámara y el error bidireccional es mayor que *MaxBidirectionalError*. El número de características confiables *visiblePoints* seguidas en el frame actual se puede reducir luego de realizar su estimación de transformación geométrica.

El código para esta fase queda de la siguiente manera:

```
[points, validity] = objTracker(camFrame);
visiblePoints = points(validity, :);
oldPoints = oldPoints(validity, :);
```

Los puntos confiables de la imagen actual se los guarda en una nueva matriz *visiblePoints* y los de la imagen anterior en la misma matriz *oldPoints* como se muestra en la Figura 2.28.

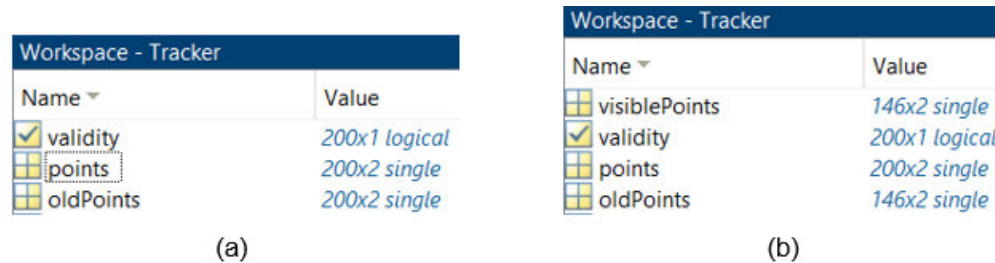


Figura 2.28. Ejemplo de la validación de confiabilidad de los puntos entre dos fotogramas consecutivos. (a) La matriz *points* y *oldPoints* contienen todas las 200 características. (b) Las matrices *oldPoints* y *visiblePoints* tienen 146 características confiables según *validity*.

2.5.1.3 Fase 3: estimación de la geometría del objeto

Para obtener la transformación geométrica a partir de pares de puntos de dos imágenes, Matlab se basa en el algoritmo mencionado en la sección 1.3.5, donde se obtiene la matriz de transformación H.

2.5.1.3.1 Transformación de similitud no reflectante (Transformation nonreflective similarity)

Esta transformación soporta traslación, rotación y escala isotrópica, es decir, que cambio de tamaño igual en todas las direcciones. Dado un pixel localizado en $[x', y']$ de la siguiente imagen y el valor de un pixel localizado en $[x, y]$ de la primera imagen, la posición de ambos pixeles está relacionado mediante las siguientes ecuaciones [54]:

$$\begin{cases} x' = xh_1 - yh_2 + h_3 \\ y' = xh_2 + yh_1 + h_4 \end{cases} \quad (2.1)$$

Donde h_1, h_2, h_3, h_4 son los coeficientes de transformación. La matriz de transformación H está dada por:

$$H = \begin{bmatrix} h_1 & -h_2 \\ h_2 & h_1 \\ h_3 & h_4 \end{bmatrix}$$

2.5.1.3.2 Transformación Afín (Affine Transformation)

A diferencia de la transformación de similitud no reflectante, esta soporta cambios de escala no isotrópicos. La posición de ambos pixeles está relacionada mediante la siguiente ecuación:

$$\begin{cases} x' = xh_1 + yh_2 + h_3 \\ y' = xh_4 + yh_5 + h_6 \end{cases} \quad (2.2)$$

Donde h_1, h_2, \dots, h_6 son los coeficientes de transformación. La matriz H debe ser convertida en un arreglo de 3×2 para ser usada en una transformación:

$$H = \begin{bmatrix} h_1 & h_4 \\ h_2 & h_5 \\ h_3 & h_6 \end{bmatrix}$$

2.5.1.3.3 Transformación Proyectiva (Projective Transformation)

En la transformación proyectiva, además lo que permite la afín, soporta la inclinación. La relación entre la localización de los puntos está dada por las siguientes ecuaciones:


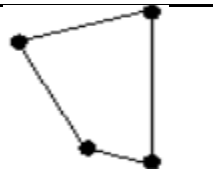

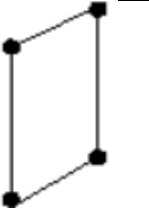
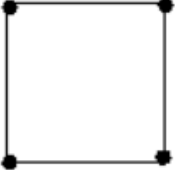
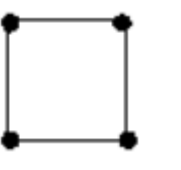
$$\begin{cases} x' = \frac{xh_1 + yh_2 + h_3}{xh_7 + yh_8 + h_9} \\ y' = \frac{xh_4 + yh_5 + h_6}{xh_7 + yh_8 + h_9} \end{cases} \quad (2.3)$$

Donde h_1, h_2, \dots, h_9 son los coeficientes de transformación. La matriz H debe ser convertida en un arreglo de 3×3 para ser usada en una transformación:

$$H = \begin{bmatrix} h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \\ h_3 & h_6 & h_9 \end{bmatrix}$$

Algunos ejemplos de estas transformaciones se pueden observar en la Tabla 2.2.

Tabla 2.2. Tipos de transformaciones con sus respectivos ejemplos.

Transformación	Antes	Después
Proyectiva		
Afín		
Similitud no Reflectante		

Para encontrar la H , el algoritmo usado en Matlab tiene el siguiente procedimiento:

- 1) Se inicializa la matriz de transformación H con ceros.
- 2) Se establece el contador de muestras aleatorias, $count = 0$.
- 3) Mientras $count < k$, donde K es el número total de muestreos aleatorios a realizar:
 - a. Se incrementa en 1 el valor de $count$.
 - b. Selecciona al azar pares de puntos de las dos imágenes (2 pares para *Nonreflective similarity*). Se escogen 2 pares de puntos para *Nonreflective similarity* debido a que se tiene 4 incógnitas (h_1, h_2, h_3, h_4) y por lo tanto se necesitan 4 ecuaciones según (2.1).
 - c. Calcular una matriz de transformación a partir de los puntos seleccionados.
 - d. Si la matriz de transformación tiene una métrica de distancia menor que la anterior matriz, se reemplaza la matriz H anterior por la actual H .
- 4) Se utiliza todos los pares de puntos en las dos imágenes que pueden ser mapeados por H para calcular una matriz de transformación refinada H .
- 5) Se marcan todos los pares de puntos como *inliers* en base a (1.32), donde el umbral T es definido por el usuario mediante el parámetro *MaxDistance*.

En el sistema propuesto se usa el tipo de transformación *similarity*, ya que la transformación realizada proporciona un rectángulo que cambia uniformemente la escala en todas las direcciones. Para el diseño del movimiento de la silla se utilizan características simples como pendiente y diferencia de altura entre rectángulos de imágenes sucesivas. Es así como, no es conveniente que un vértice del rectángulo varíe independientemente de los demás como se muestra en la Figura 2.29..

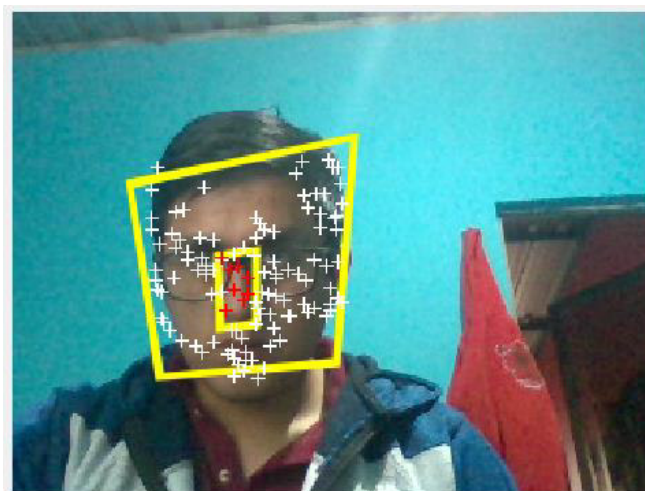


Figura 2.29. Obtención del rectángulo del rostro usando el tipo de transformación proyectiva.

Al igual que en la validación de los puntos de seguimiento, se deben almacenar solo los puntos considerados como *inliers*. Para este propósito la función *estimateGeometricTransform2D* que proporciona el objeto estimador de la geometría *tform*, también devuelve una matriz para determinar cuáles son los puntos coincidentes. Esta matriz es *inlierIndex*, la cual contiene valores de tipo lógicos definiendo 1L para los puntos determinados como *inlier* y 0L para los puntos no coincidentes (*outliers*).

- **estimateGeometricTransform2D:** estima la transformación geométrica 2-D a partir de pares de puntos coincidentes.
 - **Sintaxis:** [tform, inlierIndex] = estimateGeometricTransform2D(Points1, Points2, transformType, 'Name', value). 'Name' y value son el nombre de la propiedad y su respectivo valor.
 - **Argumentos de Entrada:**
 - Points1: puntos confiables para seguimiento de la imagen anterior, representado por la matriz *oldPoints*.
 - Points2: puntos confiables para seguimiento de la imagen actual, representado por la matriz *visiblePoints*.
 - transformType: tipo de transformación especificado como 'similarity' para la del tipo similitud no reflectante, 'affine' para la transformación afín y 'projective' para la proyectiva.
 - **Propiedades:**
 - 'MaxNumTrials': número entero que representa el máximo número de muestreos aleatorios *k* a realizar. Por defecto se establece en 1000.
 - 'Confidence': número escalar entre 0 y 100. Representa la confianza de encontrar el número máximo de *inliers*. Aumentar este valor mejora la robustez de los resultados. Por defecto está configurado en 99.
 - 'MaxDistance': distancia máxima en píxeles desde un punto y su proyección correspondiente. Por defecto establecido en 1.5.

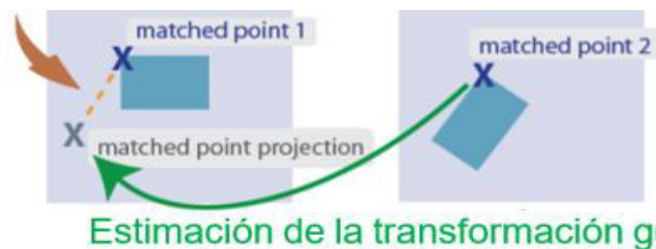


Figura 2.30. Ejemplo de la estimación de la transformación geométrica inversa [36]. El punto de la segunda imagen *matched point 2* es proyectado en la primera imagen y la

distancia se calcula desde el punto proyectado hasta el punto correspondiente de la primera imagen *matched point 1*.

```
[tform, inlierIdx] = estimateGeometricTransform2D(...  
    oldPoints, visiblePoints, 'similarity', 'MaxDistance', 15);
```

Además, para la creación objeto estimador se debe especificar el umbral que dividirá a los puntos como coincidentes o no coincidentes. Este umbral es establecido por la propiedad *MaxDistance*, cuyo valor es definido en pixeles. Así, los puntos por debajo este umbral serán almacenados en la matriz *visiblePoints*.

```
%inlierIdx: Puntos coincidentes correctos  
visiblePoints = visiblePoints(inlierIdx, :);
```

De momento se ha obtenido la matriz de transformación *H* como un parámetro del objeto estimador *tform* como se muestra en el ejemplo de la Figura 2.32. Entonces, se debe aplicar este objeto estimador de transformación geométrica a los puntos que corresponden a los vértices de los rectángulos del rostro *faceBox* y de la nariz *noseBox*. Existen dos tipos de esta aplicación: hacia adelante (*forward*) y hacia atrás (*backward*). La primera realiza la transformación de los puntos proporcionados y la segunda transformación realiza el proceso inverso por lo que se proporcionan los puntos ya transformados. En este caso se necesita la primera opción, por lo que se usa la función *transformPointForward* de Matlab.

- **transformPointsForward:** aplica la transformación geométrica hacia adelante.
 - **Sintaxis:** $X = \text{transformPointsForward}(tform, U)$.
 - **Argumentos de entrada:**
 - tform: objeto estimador
 - U: matriz de coordenadas $n \times 2$, donde n es el número de puntos a transformar.

```
Box = transformPointsForward(tform, Box);
```

Las coordenadas del anterior rectángulo *Box* se reemplazan por las actuales luego de aplicar la transformación. Por lo que se deben actualizar los puntos *oldPoints*, ahora los actuales *visiblePoints* serán los puntos anteriores para el siguiente frame. Por último, se establecen los nuevos puntos a seguir con la función *setPoints*.

```
oldPoints = visiblePoints;  
setPoints(objTracker, oldPoints);
```

El diagrama de flujo para llevar a cabo esta fase se muestra en la Figura 2.31.

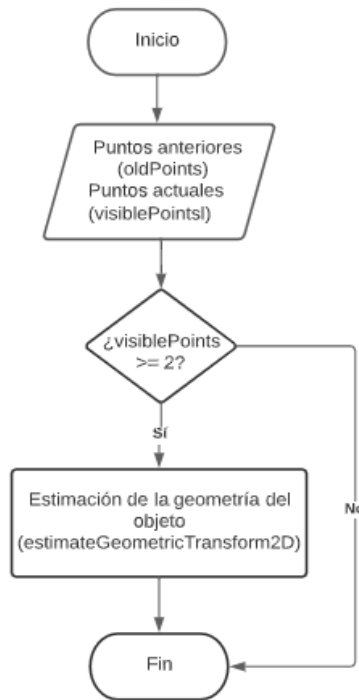


Figura 2.31. Diagrama de flujo para realizar la estimación de la geometría de un objeto.

El código completo para la obtención de la geometría es el siguiente:

```

if size(visiblePoints, 1) >= 2 %Se necesitan al menos dos puntos para
la estimación
    %Se crea el objeto para encontrar la geometría a partir de pares
    %de puntos coincidentes
    [tform, inlierIdx] = estimateGeometricTransform2D(...
        oldPoints, visiblePoints, 'similarity', 'MaxDistance', 15);
    %inlierIdx: Puntos coincidentes correctos
    visiblePoints = visiblePoints(inlierIdx, :);

%Se aplica el objeto de transformación geométrica a las coordendas del
%rectángulo del objeto a seguir
Box = transformPointsForward(tform, Box);
% Reset the points
oldPoints = visiblePoints;
setPoints(objTracker, oldPoints);
end
  
```

La información que proporciona el objeto *tform* luego de ejecutar código anterior es: *T*, matriz de transformación de los puntos; y *Dimensionality*, dimensión de la transformación geométrica para los puntos de entrada y salida, especificada como el valor 2.

Property ^	Value
T	[1.0019,-0.0011,0;0.0011,1.0019,0;-0.8710,0.2536,1]
Dimensionality	2

Figura 2.32. Parámetros proporcionados por el objeto de transformación geométrica. La matriz *T* es de dimensión 3x3, la última columna es 0,0,1 para la transformación de similitud.

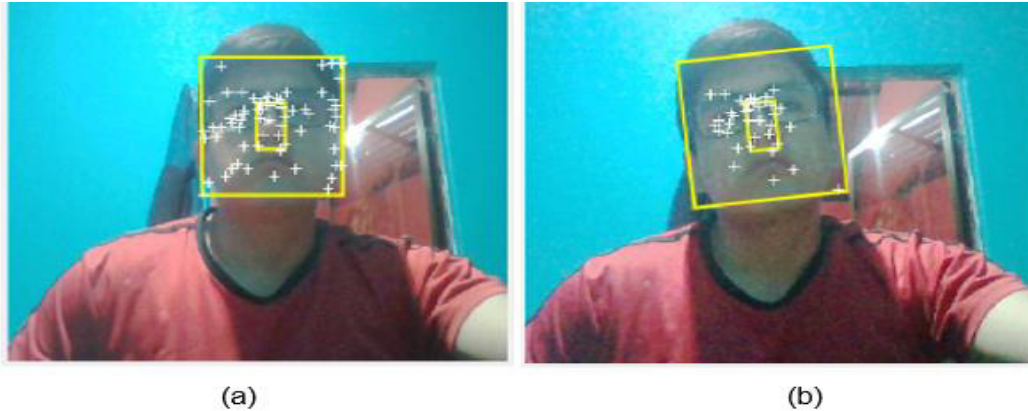


Figura 2.33. Ejemplo de la transformación geométrica del rectángulo del rostro *faceBox*.
 (a) Rectángulo del frame 1. (b) Rectángulo del frame 3 después inclinar el rostro.

La función *Tracker* queda conformada por las siguientes líneas de código.

```
function [oldPoints, Box, NewDetection] =
Tracker(camFrame,objTracker,oldPoints,Box)
%Etapa 4
%Fase 2: Seguimiento de las características
%Validación de los puntos seguidos
[points,validity] = objTracker(camFrame);
visiblePoints = points(validity, :);
oldPoints = oldPoints(validity, :);
%Estimación de la transformación Geométrica
if size(visiblePoints, 1) >= 2 %Se necesitan al menos dos puntos
    NewDetection = -1;
    %Se crea el objeto para encontrar la geometría a partir de pares
    %de puntos coincidentes
    [tform, inlierIdx] = estimateGeometricTransform2D(...
        oldPoints, visiblePoints, 'similarity', 'MaxDistance', 15);
    %inlierIdx: Puntos coincidentes correctos
    visiblePoints = visiblePoints(inlierIdx, :);

    %Se aplica el objeto de transformación geométrica a las coordendas del
    %rectángulo del objeto a seguir
    Box = transformPointsForward(tform, Box);
    % Reset the points
    oldPoints = visiblePoints;
    setPoints(objTracker, oldPoints);
else
    NewDetection = 1;
end
end
```

La función aplicada a cada objeto (rostro y nariz) queda de la siguiente forma. Donde *oldFacePoints* y *oldNosePoints* representan los puntos confiables para seguimiento y que definen la transformación geométrica de la cara y nariz respectivamente. Las salidas *faceBox* y *noseBox* son la geometría que engloba las características del rostro y nariz correspondiente.

```

[oldFacePoints, faceBox, NewDetection] =
Tracker(camFrame,faceTracker,oldFacePoints,faceBox);
[oldNosePoints, noseBox, NewDetection] =
Tracker(camFrame,noseTracker,oldNosePoints,noseBox);
%Se valida que hayan al menos 2 puntos para seguimiento
if NewDetection == 1
    InitMov = -1;%No se inicializa de nuevo las variables de movimiento
    continue%Se vuelve a detectar el rostro en el siguiente frame
end

```

2.6 ETAPA 5: DETECCIÓN DE LA DIRECCIÓN DE GIRO DE LA CABEZA



Figura 2.34. Entradas y salidas de la etapa 5. La entrada es el rectángulo de la nariz de la etapa 4 y la salida es un indicador de entrada o salida del modo de comandos.

Para ingresar o salir del modo de comandos donde se simula el movimiento de la silla es necesario definir un movimiento específico del rostro. Este movimiento no tiene que verse afectado en gran medida por la inclinación de la cabeza por lo que se elige el movimiento de la nariz. Aunque, los puntos ubicados en la zona de la nariz están dentro de los que identifican al rostro, estos tienen un mayor desplazamiento al rotar la cabeza. Para evidenciar esto se toma arbitrariamente el quinto punto del rostro y de la nariz y se evalúan sus posiciones al girar la cabeza como se muestra en la Figura 2.35.

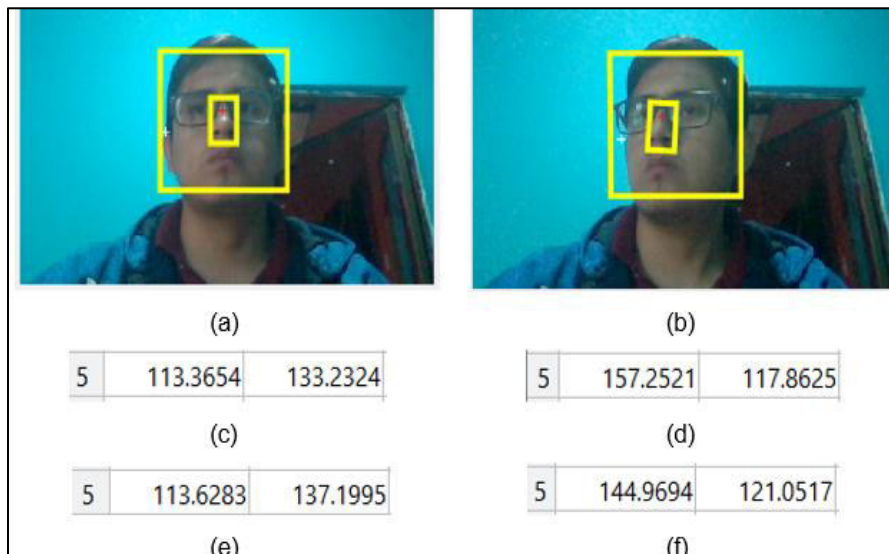


Figura 2.35. Movimiento de los puntos del rostro y nariz al girar la cabeza hacia la izquierda. (a) Punto del rostro y de la nariz al mirar hacia el frente. (b) Punto del rostro y

de la nariz al girar la cabeza. (c) Posición inicial del punto 5 del rostro. (d) Posición inicial del punto 5 de la nariz. (e) Posición del punto 5 del rostro al rotar la cabeza. (f) Posición del punto 5 de la nariz al rotar la cabeza.

En la Figura 2.35 se observa que existe mayor desplazamiento del punto de la nariz, la diferencia es de 12.2827 píxeles; mientras que, para el punto del rostro es de solo 0.2633 píxeles. Esto define entonces una geometría que contiene los puntos de la nariz diferente a la que engloba todo el rostro, es decir, el rectángulo de la nariz se desplazará más píxeles que el rectángulo del rostro al girar hacia algún lado, esto se comprueba en la Figura 2.36. Entonces, ahora tiene sentido el obtener dos objetos estimadores de transformación geométrica: el del rostro, para movimientos de la silla; y el de la nariz, para entrada y salida del modo de comandos.

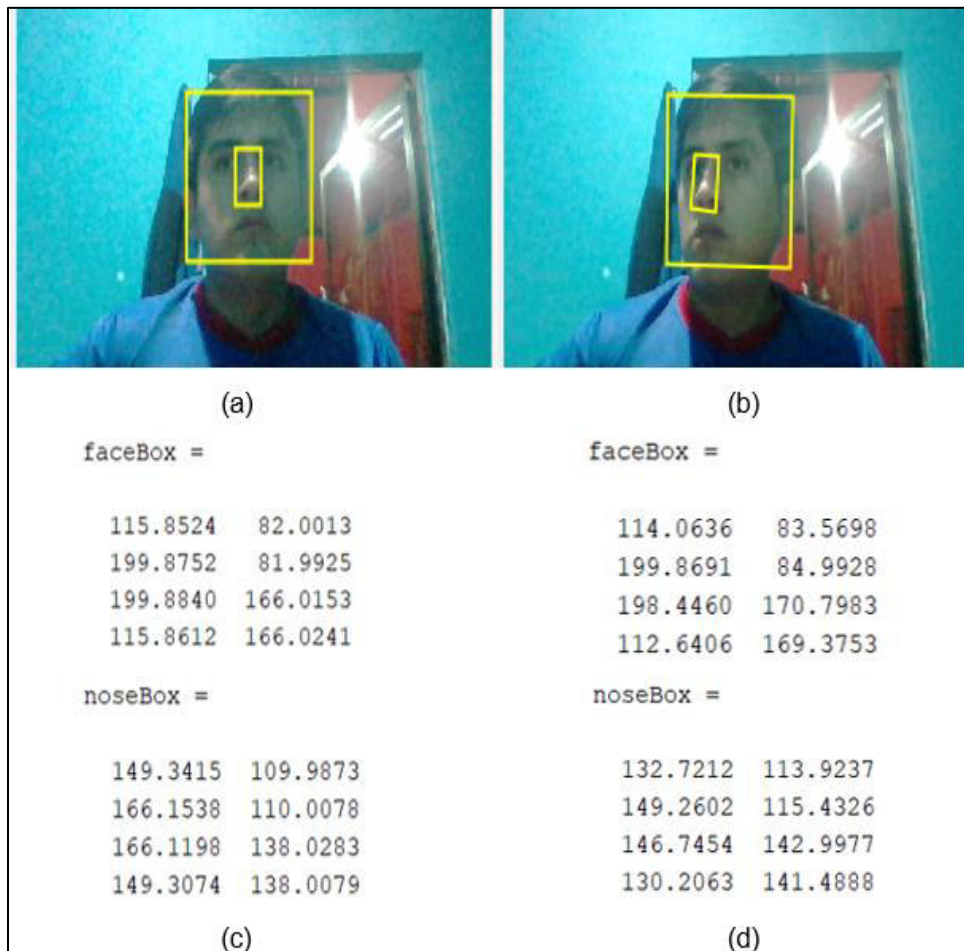


Figura 2.36. Movimiento de los rectángulos del rostro y nariz conforme se gira la cabeza hacia la izquierda. (a). Rectángulo de la nariz inicial. (b) Rectángulo de la nariz final al rotar la cabeza. (c) Posiciones iniciales de los vértices de los rectángulos. (d) Posiciones de los vértices de los rectángulos luego de rotar la cabeza.

De acuerdo con el ejemplo mostrado en la Figura 2.36 los vértices del rectángulo de la nariz se movieron en promedio 17.9974 píxeles. Por otro lado, los vértices del rectángulo del rostro se movieron en promedio 1.6138 píxeles. Esto da cerca de 11 veces más desplazamiento de la nariz que del rostro en general al girar la cabeza, lo cual proporciona una forma de desplazamiento independiente en gran medida a los que se usan para controlar el movimiento de la silla.

2.6.1 INICIALIZACIÓN DE PUNTOS DE REFERENCIA DE LA NARIZ

Al tener ya la referencia de qué parámetro se usará para la entrada y salida de comandos, se debe especificar cuál de los sentidos de rotación de la cabeza tendrá una determinada función. El parámetro usado será el desplazamiento en píxeles de los vértices del rectángulo de la nariz y para esto debe haber un punto de referencia. Los puntos de referencia de la nariz y la altura inicial de la cara se establecen en la primera imagen donde se haya hecho la primera detección del rostro para lo cual se establece la bandera *InitMov*. Esta validación se realiza para evitar que ante una nueva detección donde la cabeza esté fuera de la posición de reposo se inicialicen de forma incorrecta estas variables.

La coordenada de los vértices que se usa es la correspondiente al eje horizontal pues la rotación de la cabeza se hace en ese eje. Se inicializan los vértices del lado izquierdo superior *Pref1* e inferior *Pref2* del rectángulo *noseBox* obtenido en la etapa 4, ya que presenta de forma más práctica las coordenadas en x de los vértices. Se usa solo el lado izquierdo, ya que el desplazamiento de ambos lados del rectángulo es significativamente similar, esto se evidencia en la Tabla 2.3 organizada en base a la Figura 2.36.

Tabla 2.3. Desplazamientos en el eje x de los vértices del rectángulo de la nariz. Los vértices de ambos lados en la parte superior e inferior tienen un desplazamiento en píxeles similar.

Vértice	Posición Inicial	Posición Final	Desplazamiento
Superior izquierdo	149.3415	132.7212	16.8936
Superior derecho	166.1538	149.2602	16.8936
Inferior derecho	166.1198	146.7454	19.3744
Inferior izquierdo	149.3074	130.2063	19.1011

La rotación de la cabeza debe tener un umbral en el cual se determine que se haya girado la cabeza hacia una dirección específica. Este umbral se define en la siguiente fase, pero se basa en el ancho de la nariz por lo que es necesario de igual manera inicializar esta característica en función del rectángulo que se obtiene en el primer frame. Del rectángulo

de la nariz obtenido en la etapa 3, el cual es *nose*, se obtiene el ancho específicamente de la posición 3 de ese vector. Este parámetro se almacena en la variable *WidthNose* y será usado para definir en qué proporción del mismo será determinado el umbral.

```
InitMov = 1;
Pref1 = noseBox(1,1);
Pref2 = noseBox(4,1);
WidthNose = nose(3);
```

2.6.2 OBTENCIÓN DE LA DIRECCIÓN DE GIRO

En esta fase se define el umbral para el cual se determina si se ha girado la cabeza bien sea a la izquierda o a la derecha. Debido a que el umbral está definido en función del ancho de la nariz, se determina el desplazamiento máximo de los vértices cuando se inclina la cabeza para mover la silla a la izquierda o a la derecha. En función de este desplazamiento se determina un valor mayor para no interpretar de forma errónea una rotación hacia uno de los lados. Un ejemplo de este desplazamiento se puede observar en la Figura 2.37.

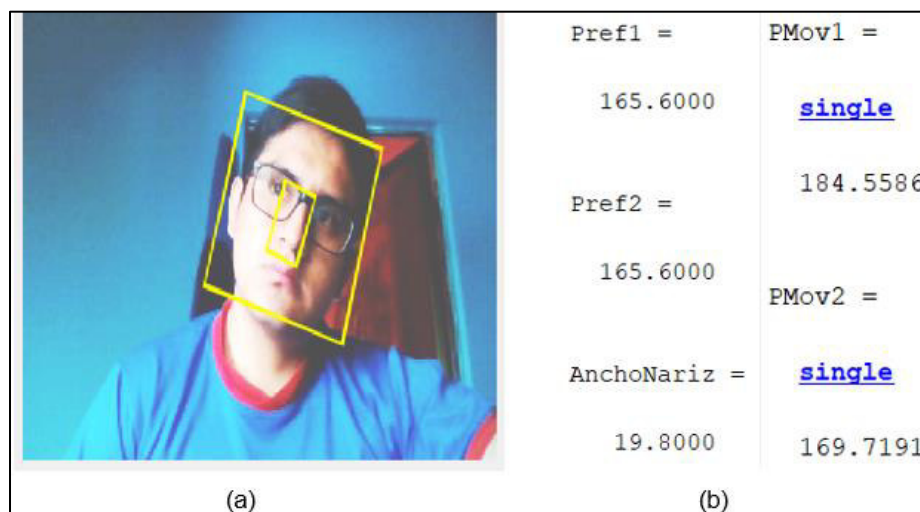


Figura 2.37. Desplazamiento de los vértices del rectángulo con inclinación a la izquierda. (a) Fotograma con inclinación. (b). Posiciones inicial y final después del desplazamiento.

Se han definido las variables *PMov1* y *Pmov2* para guardar la nueva posición de los vértices de la nariz y así comparar con el umbral. En la Figura 2.37 se aprecia que la diferencia de la posición desplazada *PMov1* con respecto a la inicial *Pref1* es de 19.9586 píxeles, lo cual es aproximadamente el ancho de la nariz. De esta manera, se define una rotación cuando hay un desplazamiento positivo (derecha) o negativo (izquierda) mayor a 1.5 veces el ancho de la nariz.

```
PMov1 = noseBox(1,1);
PMov2 = noseBox(4,1);
```

En la Figura 2.38 se muestra un giro hacia la derecha aumentando la posición de vértice. Por otro lado, cuando se realice un giro a la izquierda la posición del vértice disminuye. Por lo tanto, si los nuevos vértices $Pmov1$ y $Pmov2$ aumentan o disminuyen $1.5 * WidthNose$ respecto a la posición inicial $Pref1$ y $Pref2$, se considera rotación a la derecha o izquierda respectivamente.

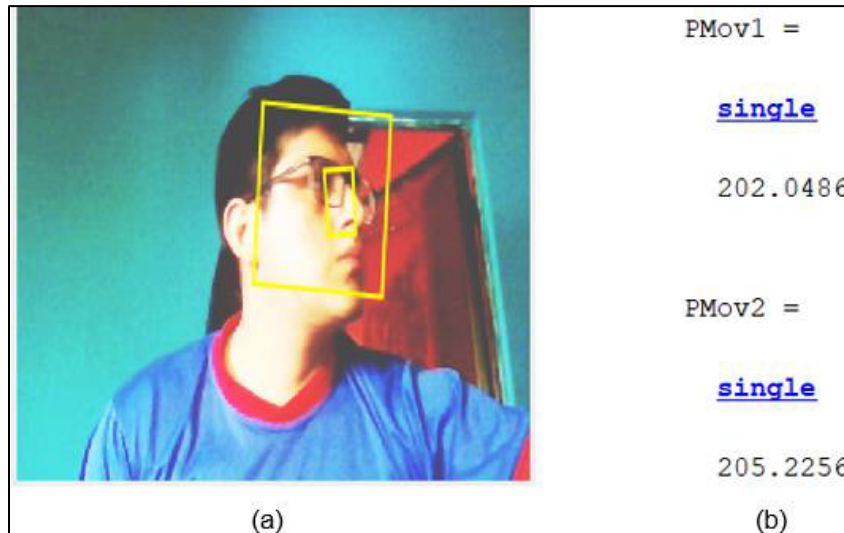


Figura 2.38. Desplazamiento de los vértices del rectángulo con rotación a la derecha. (a) Fotograma con rotación de la cabeza. (b). Posición final después de la rotación. La variable $Pmov2=205.2256$ es mayor que $Pref2+1.5 * WidthNose=195.3$, por lo tanto, corresponde a un giro hacia la derecha.

Se ingresa o sale del modo de comandos si el giro es primero a la izquierda y luego se gira inmediatamente a la derecha. Para lo cual se ha definido una bandera *IzqDetect* que toma el valor de -1 si existe un giro a la izquierda. Además, se debe restringir el tiempo dentro del cual se puede hacer el posterior giro a la derecha definido en 3 segundos, por lo que se hace uso de las funciones *tic* y *toc*.

```
if PMov1 <= Pref1 - 1.5*AnchoNariz && PMov2 <= Pref2 - 1.5*AnchoNariz
    IzqDetect = - 1
    tic
    continue%Pasa al siguiente Frame para detección del rostro
end
```

Se comprueba que primero se ha girado la cabeza a la izquierda y no se ha pasado de los 3 segundos para hacer el giro a la derecha. Si se cumple esta condición, se valida que

exista una rotación a la derecha. Finalmente, cuando se realice el movimiento especificado se la variable *InMode* tomará un valor de -1 si se entra al modo de comandos o 1 si se sale del mismo.

```

if IzqDetect == -1 && toc < 3
    if PMov1 >= Pref1 + 1.5*AnchoNariz && PMov2 >= Pref2 + 1.5*AnchoNariz
        inMode = inMode*-1;%Bandera para entrar/salir del sistema
        continue%Pasa al siguiente Frame para detección del rostro
    end
else
    IzqDetect = 1;
end

```

Luego del movimiento para entrar o salir del sistema se pierden varias características y el seguimiento no es confiable para controlar la silla, por lo que se hace una nueva detección del rostro en las condiciones iniciales. Por lo tanto, dentro del *if* que valida el giro a la derecha se cambia el valor de *NewDetection* a 1. Dentro del mismo *if* se determina que no se inicialicen de nuevo las variables de movimiento de la fase 1 de esta etapa mediante *InitMov = -1* y también se establece la variable *IzqDetect* a su valor inicial.

```

NewDetection = 1;%Bandera para empezar una nueva detección
InitMov = -1;%No se inicializa de nuevo las variables de movimiento
IzqDetect = 1;%Valor de la bandera al inicial

```

El proceso realizado para esta fase se presenta en el diagrama de flujo de la Figura 2.39.

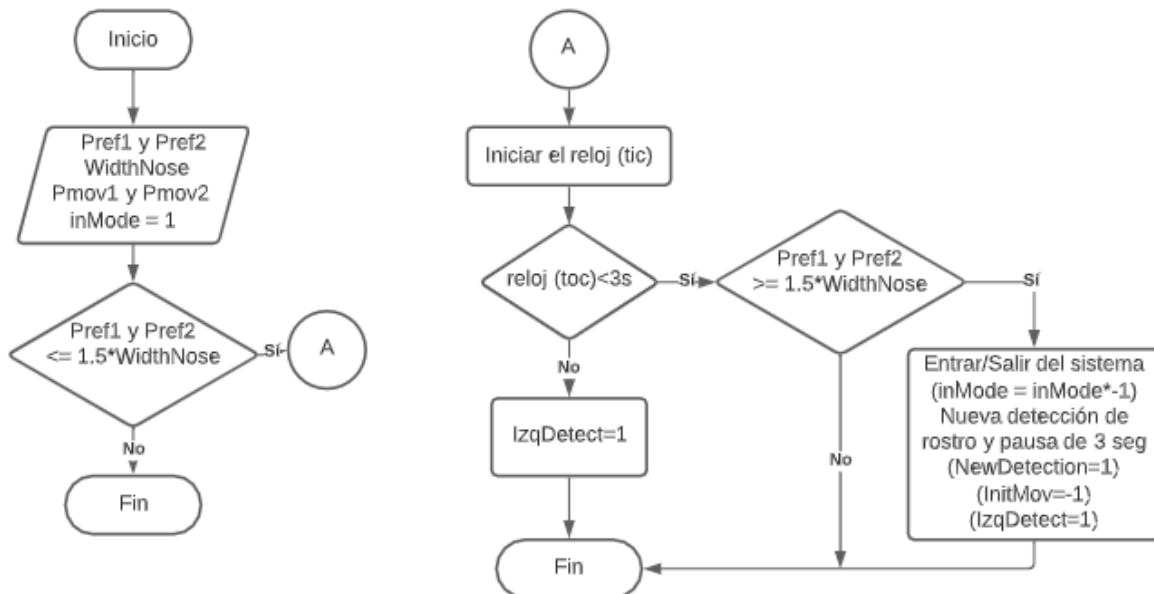


Figura 2.39. Diagrama de flujo para determinar en qué sentido se realizó la rotación de la cabeza.

Finalmente, el código para esta fase queda de la siguiente forma:

```

%%ETAPA 5 - Fase 2: Obtención de la dirección de giro
%noseBox: rectángulo de la nariz de la etapa 4, fase 2
%Puntos movibles de la nariz en los siguientes Frames
PMov1 = noseBox(1,1);% Coordenada x del Vértice superior izquierdo de
noseBox
PMov2 = noseBox(4,1);% Coordenada x del Vértice inferior izquierdo de
noseBox

%Si los puntos movibles se mueven el doble del ancho de noseBox desde
los puntos
%de referencia hacia la izquierda se determina como giro hacia ese
lado
if PMov1 <= Pref1 - 1.5*AnchoNariz && PMov2 <= Pref2 - 1.5*AnchoNariz
    IzqDetect = - 1;%Primer paso para entrar/salir del modo de
comandos
    tic %se limita el tiempo para cumplir el siguiente paso
entrar/salir del modo de comandos
    continue%Pasa al siguiente Frame para detección del rostro
end

if IzqDetect == -1 && toc < 3
    %Puntos movibles se mueven el doble del ancho de noseBox hacia
%la derecha -> giro hacia la derecha
    if PMov1 >= Pref1 + 1.5*AnchoNariz && PMov2 >= Pref2 +
1.5*AnchoNariz
        inMode = inMode*-1;%Bandera para entrar/salir del sistema
        NewDetection = 1;%Bandera para empezar una nueva detección
        InitMov = -1;%No se inicializa de nuevo las variables de
movimiento
        IzqDetect = 1;%Valor de la bandera al inicial
        continue%Pasa al siguiente Frame para detección del rostro
    end
else
    IzqDetect = 1;
end
%%FIN ETAPA 5 - Fase 2

```

2.7 ETAPA 6: OBTENCIÓN DE PARÁMETROS PARA EL MOVIMIENTO DE LA SILLA



Figura 2.40. Entradas y salidas de la etapa 6. La primera entrada es el rectángulo del rostro de la etapa 4 y la segunda entrada es el indicador de la etapa 5. En la Fase 1, se inicializa la altura del rostro en base al primer fotograma. En la Fase 2, se procesan los parámetros del rectángulo y se determina la dirección del movimiento de la silla.

En esta etapa se definen las características que se extraen del rectángulo del rostro para adaptarlas al movimiento de la silla y en función de la variación de estas características se tiene mayor rapidez en cuanto al movimiento. Las características que se extraen del rectángulo son la pendiente del lado izquierdo y la altura del rectángulo, las cuales se inicializan en la fase 1 en función del primer fotograma. La pendiente es transformada a grados para facilitar la lectura de la inclinación del rostro y la altura inicial se compara con la de los fotogramas donde hay movimiento. En la fase 2 se detalla cómo se usa la inclinación del rostro en grados y la variación de la altura con respecto a la inicial para mover la imagen de la silla de ruedas en la interfaz gráfica.

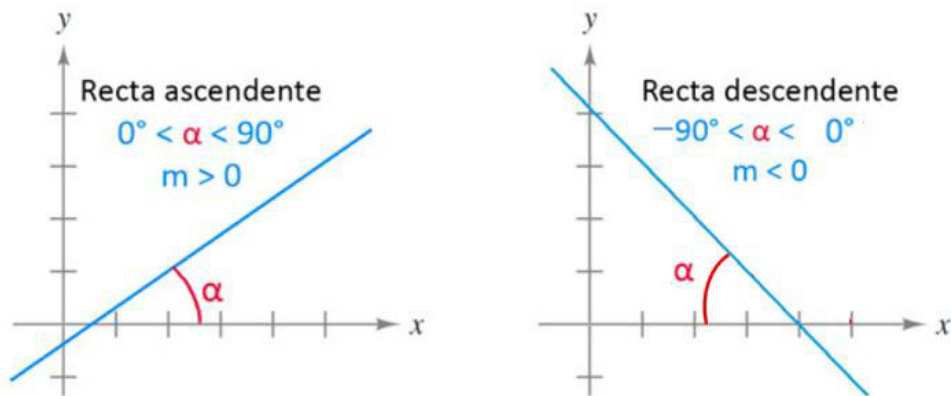
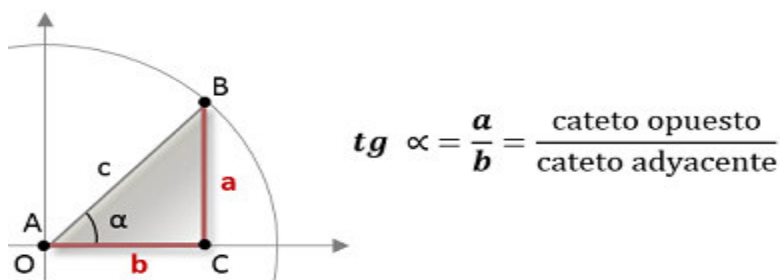


Figura 2.41. Tipos de pendientes de rectas consideradas [55]. Para la obtención de los de grados de inclinación se usa $\alpha = \tan^{-1} m$.

Se usa la función inversa de la función tangente, ya que para obtener la tangente de un ángulo α se usa la relación dada por (2.4). Donde se divide el cateto opuesto para el cateto adyacente como se muestra en la Figura 2.42. El lado opuesto es la altura del del triángulo dada por la diferencia $Y2-Y1$; mientras que, el lado adyacente es el ancho del triángulo recto definida por la diferencia $X2-X1$.

$$\tan(\alpha) = \frac{a}{b} \tag{2.4}$$



$$tg \alpha = \frac{a}{b} = \frac{\text{cateto opuesto}}{\text{cateto adyacente}}$$

Figura 2.42. Relación de la tangente de un ángulo [56].

2.7.1 INICIALIZACIÓN DE CARACTERÍSTICAS PARA MOVIMIENTO

En esta fase se obtienen los valores iniciales que servirán como referencia para los demás fotogramas, donde se tiene una variación de la altura ya sea al acercamiento o alejamiento del rostro en la cámara. Se establece la altura inicial h_0 del rectángulo del rostro *faceBox*, específicamente se utiliza la diferencia entre las coordenadas verticales del vértice superior izquierdo Y_1 e inferior izquierdo Y_2 para realizar las comparaciones con la altura del rostro en las siguientes imágenes. Para los valores de las pendientes de los siguientes fotogramas no es necesario inicializar un valor en base al primer fotograma, ya que el signo de dicho ángulo es que indica el sentido de inclinación. Es así como, para el movimiento en el eje horizontal y en el eje vertical de la imagen se la silla de ruedas se tiene una determinada característica de movimiento. Así, la fase 1 se simula mediante:

```
%Inicialización de la posición (x,y) de la silla
X=0;
Y=0;
Y1 = faceBox(1,2);%Coordenada 'y' del Vértice superior izquierdo
Y2 = faceBox(4,2);%Coordenada 'y' del Vértice inferior izquierdo
h0 = Y2-Y1;%Altura inicial del rostro
```

2.7.2 DETERMINACIÓN DEL MOVIMIENTO DE LA SILLA EN BASE A LAS CARACTERÍSTICAS DEL ROSTRO EN MOVIMIENTO

Para el movimiento en el eje 'x' se tienen los grados de inclinación; sin embargo, no se toma directamente todo el rango del ángulo α establecido en la Figura 2.41 sino que se deja un intervalo que se considera como reposo/alto. En cambio, para el movimiento en el eje 'y' se considera cuánto varía la altura h con respecto a la inicial h_0 del rectángulo del rostro *faceBox*.

```
h = Y2-Y1;%altura actual del rostro.
```

De igual forma se considera las coordenadas de los vértices del lado izquierdo, almacenados en X_1, Y_1, X_2, Y_2 , para el cálculo de la pendiente con la ecuación (2.5).

```
X1 = faceBox(1,1);%Coordenada en x del Vértice superior izquierdo del
rectángulo del rostro
Y1 = faceBox(1,2);%Coordenada en y del Vértice superior izquierdo del
rectángulo del rostro
X2 = faceBox(4,1);%Coordenada en x del Vértice inferior izquierdo del
rectángulo del rostro
Y2 = faceBox(4,2);%Coordenada en y del Vértice inferior izquierdo del
rectángulo del rostro
```

Finalmente, se convierte el valor de la pendiente a grados a través de la función *atand* guardándola en la variable *Ang* para compararla en la siguiente fase.

$$m = \frac{Y2 - Y1}{X2 - X1} \quad (2.5)$$

```
m = (Y2-Y1)/(X2-X1);%Pendiente de la recta
Ang = atand(m);%Conversión a grados
```

Una vez definidos todos los parámetros a usar para el movimiento de la imagen de la silla de ruedas, se procede a establecer qué valor de los parámetros determinarán una dirección específica. En base a esto se tienen 5 comandos, es decir, 5 tipos de movimientos especificados anteriormente en la Figura 2.1. Estos movimientos involucran solo el movimiento de la cabeza, que es la parte del cuerpo que una persona con cuádrupleja puede mover. Así mismo, estos movimientos solo tienen influencia en el movimiento de la silla cuando se accede al modo de comandos.

Como se mencionó anteriormente tanto para el eje vertical como para el horizontal se tiene un rango en el cual no se considera algún movimiento de la silla y esta permanece inmóvil, este rango es usado para el modo de comando *Reposo/Alto*. En el eje *y* el intervalo usado está relacionado con la altura, en donde el rango de reposo desde una disminución del 10% de la altura inicial hasta un incremento del 10% de la altura inicial, es decir, el rango abarca $[0.9 * h0 \ 1.1 * h0]$. Por otro lado, para el eje *x* el espacio considerado involucra el grado de inclinación de la cabeza, el cual contiene 10° a la izquierda y 10° a la derecha, es decir, un movimiento entre -80° y 80° como se muestra en la Figura 2.43.

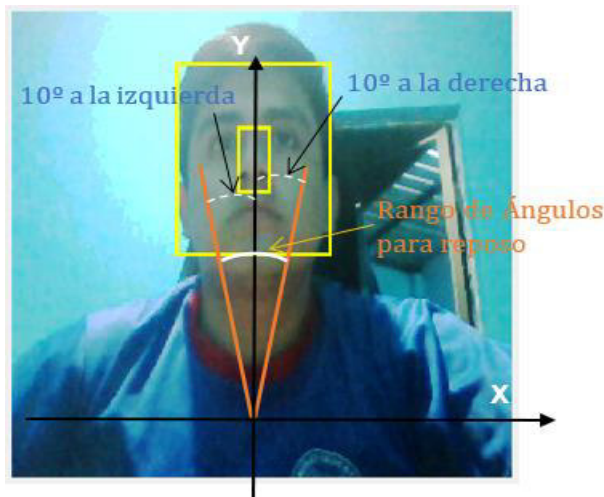


Figura 2.43. Intervalo de inclinación para posición horizontal estática. Rango dentro del cual se considera la cabeza en estado de reposo y activa el modo de comando *Alto*.

Al definir los valores para los cuales se considera el estado de reposo, se determina hacia qué dirección se moverá la simulación de la silla en el eje vertical. Si el valor de la altura del rectángulo del rostro es menor al 90% de la altura inicial, la imagen de la silla se moverá hacia arriba. En cambio, si la altura de ese rectángulo es mayor al 110% de la altura inicial h_0 , entonces la silla se moverá hacia abajo.

```

if h < h0*0.9
elseif h > h0*1.1
else%Estado de reposo
end

```

Para el eje vertical existe una velocidad determinada a mayor diferencia de la altura del rostro actual con la de la inicial. Si la persona inclina la cabeza hacia atrás, la silla se moverá hacia atrás; sin embargo, en la interfaz gráfica presentada en la etapa 7 se muestra la vista superior del movimiento de la silla, lo que se refleja como un movimiento hacia arriba o hacia abajo si la persona inclina su cabeza hacia adelante. Entonces, para el movimiento hacia arriba se resta la altura actual al límite inferior $0.9 * h_0$, este resultado se suma a la posición anterior Y de la silla inicializada en 0. Por último, para el movimiento hacia abajo (adelante) se resta la altura actual al límite superior de reposo $1.1 * h_0$ y este valor se resta a la posición anterior Y . En general, el movimiento vertical está dado por (2.6).

$$\begin{cases} Y = Y - (h - h_{max}) \\ Y = Y + (h_{min} - h) \end{cases} \quad (2.6)$$

Donde:

Y : posición de la silla en el eje vertical.

h_{max} : máxima altura del estado de reposo definida como $h_{max} = 1.1 * h_0$.

h_{min} : mínima altura del estado de reposo definida como $h_{min} = 0.9 * h_0$.

h : altura actual del rectángulo del rostro.

h_0 : altura inicial del rectángulo del rostro.

Debido a que teóricamente el número de puntos que identifican un objeto se van perdiendo conforme la rotación y el movimiento se deben obtener nuevos puntos cada cierto tiempo. Esto se demuestra en las pruebas realizadas, cuyos resultados se muestran en Tabla 3.17 y Tabla 3.19 donde se procesan 13.13 fotogramas por segundo para la resolución utilizada y la distorsión del rectángulo más temprana se da en el frame 458. En base a lo interior se calcula que la primera distorsión ocurre a 34.88 segundos. Por lo tanto, se realiza una nueva detección cada 30 segundos en la posición de reposo vertical, que es donde se debe definir la altura inicial del rostro.


```

if h < h0*0.9%Hacia atrás
    Y = Y + floor(h0*0.9-h);
elseif h > h0*1.1%Hacia adelante
    Y = Y - floor(h- h0*1.1);
else
    Y = Y;%Reposo en el eje Y
    if toc>30
        InitMov = -1;
        NewDetection=1;%Nueva detección
        tic
    end
end
end

```

En el eje horizontal, si el ángulo de inclinación está entre 0° y -80° la silla se moverá hacia la derecha; mientras que, si Ang está entre 0° y 80° la silla se moverá a la izquierda. Para el movimiento hacia la izquierda, se resta del límite superior de reposo θ_{max} el valor de inclinación actual θ , este resultado se resta a la posición X anterior de la silla que inicialmente está definida en 0. Por otro lado, para que la silla se dirija hacia la derecha, se resta a la inclinación actual en grados el límite inferior de reposo θ_{min} , luego se suma a la posición anterior X ese desplazamiento. Así, entre más diferencia haya entre el respectivo límite superior o inferior con la inclinación actual, la silla se desplazará un mayor valor de pixeles acorde con (2.7).

$$\begin{cases} X = X - (\theta_{max} - \theta) & 0 < \theta < 80 \\ X = X + (\theta - \theta_{min}) & 0 > \theta > -80 \end{cases} \quad (2.7)$$

Donde:

X : posición de la silla en el eje horizontal.

θ_{max} : máxima inclinación hacia izquierda del estado de reposo. Definido como

$$\theta_{max} = 80^\circ.$$

θ_{min} : mínima inclinación hacia la derecha del estado de reposo. Definido como

$$\theta_{min} = -80^\circ.$$

θ : altura actual del rectángulo del rostro.

```

if Ang < 80 && Ang > 0%hacia la izquierda
    X=X-floor(80-Ang);
elseif Ang > -80 && Ang < 0% hacia la derecha
    X=X+floor(Ang- (-80));
else
    X = X;%Reposo en el eje X
end
end

```

En la Figura 2.44 se detallan los movimientos posibles desplazamientos de la silla según el modo de comandos especificados en esta etapa. Se especifica además un alejamiento y

acercamiento del rostro en donde la altura del rectángulo del mismo se reduce o aumenta respectivamente para determinar la dirección del desplazamiento vertical. Cabe recalcar que se pueden combinar movimientos horizontales y verticales para hacer un movimiento diagonal de la silla.

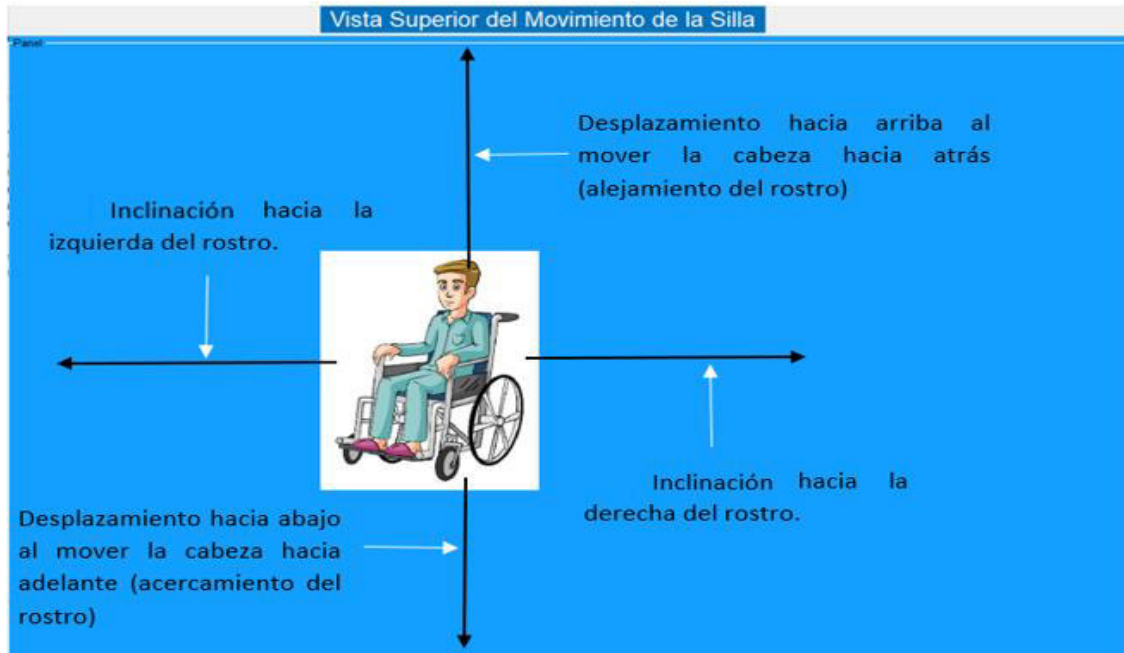


Figura 2.44. Posibles movimientos de la silla en la interfaz gráfica.

La sección de la interfaz gráfica donde se mueve la imagen de la silla simula una habitación cerrada rectangular sin obstáculos; por lo que, tiene límites de movimiento en los 4 lados. La habitación simulada tiene una dimensión de 840 pixeles de ancho y 395 pixeles de alto debido al máximo tamaño posible para que no interfiera con otros objetos de la interfaz, donde el vértice inferior izquierdo es el origen (0,0) del plano (habitación) como se muestra en la

Figura 2.45. Al mover la cabeza hacia una determinada dirección se puede exceder dicho límite, por lo que se hacen 4 consideraciones y una vez excedido el límite correspondiente se mantiene una posición X o Y de la silla en un valor fijo tal como se indica a continuación.

```

if X < 0%Eje X
    X=0;%Se mantiene en 0 si la posición x es negativa
elseif X > 840
    X=840;%Se establece 840px máximo
end
if Y < 0%Eje Y
    Y=0;%Se mantiene en 0 si la posición y es negativa
elseif Y > 395
    Y=395;%Se establece 395px máximo
end

```



Figura 2.45. Plano de la simulación de habitación para el movimiento de la silla.

El diagrama de flujo que detalla el procedimiento para llevar a cabo la simulación de esta etapa se detalla en la Figura 2.46.

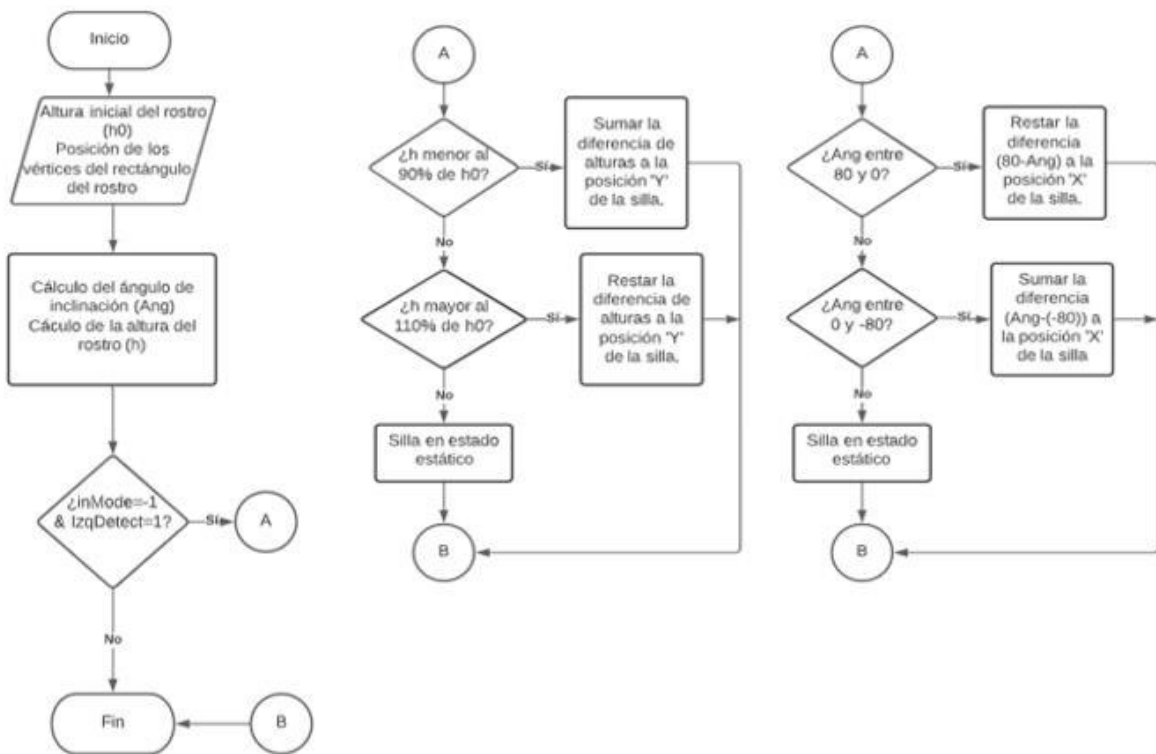


Figura 2.46. Diagrama de flujo de la obtención de parámetros del rectángulo del rostro para su interpretación en movimiento de la silla.

El código completo de la fase 2 de la etapa 6 queda de la siguiente forma:

```

% Se obtiene el ángulo de rotación para el movimiento del eje X
X1 = faceBox(1,1);%Coordenada en x del Vértice superior izquierdo del
rectángulo del rostro
Y1 = faceBox(1,2);%Coordenada en y del Vértice superior izquierdo del
rectángulo del rostro
X2 = faceBox(4,1);%Coordenada en x del Vértice inferior izquierdo del
rectángulo del rostro
Y2 = faceBox(4,2);%Coordenada en y del Vértice inferior izquierdo del
rectángulo del rostro
m = (Y2-Y1)/(X2-X1);%Pendiente de la recta del lado izquierdo de
faceBox
Ang = atand(Pendiente);%Conversión a grados
% Se obtiene la diferencia de alturas para el movimiento del eje Y
h = Y2-Y1;%Y2 altura actual, Y1 altura inicial
if InMode == -1
    %Movimiento en el eje Y
    LedIndicator='green';%Indicador de estado ON del sistema
    IzqDetect == 1
        if h < h0*0.9%Hacia atrás si se reduce al 90% de la altura
inicial
            %Se suma la diferencia de alturas en forma de pixeles
            Y = Y + floor(h0*0.9-h);
        elseif h > h0*1.1%Hacia adelante si la altura aumenta 10%
            %Se resta la diferencia de alturas en forma de pixeles
            Y = Y - floor(h-h0*1.1);
        else
            Y = Y;%Reposo en el eje Y
        end

        %Movimiento en el eje X
        if Ang < 80 && Ang > 0%Ángulo entre 0 y 80 grados se mueve
hacia la izquierda
            %Se resta la diferencia de ángulos en forma de pixeles
            X=X-floor(80-Ang);
        elseif Ang > -80 && Ang < 0 %Ángulo entre -45 y -80 grados se
mueve hacia la derecha
            %Se suma la diferencia de ángulos en forma de pixeles
            X=X+floor(Ang-(-80));
        else
            X = X;%Reposo en el eje X
        end
        %Establecimiento de límites de movimiento
        %Eje X
        if X < 0 %Si posición menor a 0 en cuanto a pixeles
            X=0;
        elseif X > 840 %Si posición mayor a 600 en cuanto a pixeles
            X=840;
        end
        %Eje Y
        if Y < 0 %Si posición menor a 0 en cuanto a pixeles
            Y=0;
        elseif Y > 395 %Si posición mayor a 400 en cuanto a pixeles
            Y=395;
        end
    else
        LedIndicator='red';%Indica que se ha salido del modo de comandos
    end
end
end

```

2.8 ETAPA 7: INTERFAZ GRÁFICA PARA PRESENTACIÓN DEL MOVIMIENTO DE LA SILLA



Figura 2.47. Entrada y salida de la etapa 7. La entrada es la posición de la silla obtenida en la etapa 6 y la salida es la imagen de la silla en movimiento mostrada en la interfaz gráfica.

En esta etapa se presentan todas las características obtenidas del rostro y nariz, los rectángulos correspondientes, la imagen de la silla y su movimiento de acuerdo con los parámetros de posición obtenidos en la etapa anterior. La interfaz gráfica con sus respectivas secciones donde se mostrarán los resultados de la simulación se muestra en la Figura 2.48. En total se tienen 3 secciones para distribuir adecuadamente los resultados de las diferentes etapas: en la sección 3, las salidas de la etapa 1, 2, 3 y 4; en la sección 2, las de la etapa 6; y en la sección 1, los resultados de la etapa 5.

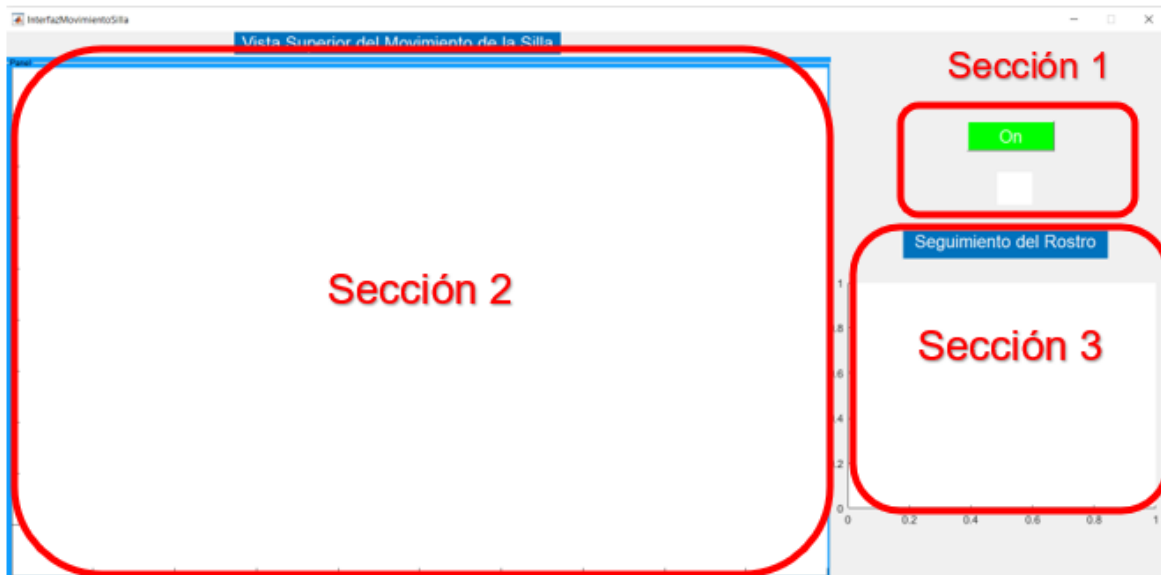


Figura 2.48. Secciones de la interfaz gráfica diseñada para la presentación de los resultados.

La sección 1 contiene 2 elementos: un pushbutton *On* usado para poner en marcha del código del sistema y un *statictext* en blanco para indicar con un color rojo que está fuera del modo de comandos o verde si se ha ingresado al modo de comandos. En la sección 2, se tiene el *axes1* que muestra la imagen de la silla y el espacio por donde se moverá desde

una vista superior. Finalmente, en la sección 3 se tiene el objeto *axes2* que presenta los fotogramas capturados son los rectángulos y puntos respectivos superpuestos.

Para la sección 2 se configura la posición de la silla y se establecen las unidades del plano donde se presenta la imagen, para lo cual se establecen los pixeles como unidades con la función *set*. El objeto al cual se configura es el *axes1* a través de la propiedad '*Units*' con el valor '*pixels*'. Una vez establecida la unidad, se debe configurar la posición y tamaño de la imagen, lo cual se hace también con la función *set*, la propiedad *Position* y cuyo valor se establece como un vector $[x_0 \ y_0 \ width \ height]$. Finalmente se presenta la imagen de la silla con todas las propiedades configuradas con la función *imshow*.

```
axes(handles.axes1)
img = imread('silla.jpg');
set(handles.axes1, 'Units', 'pixels');
set(handles.axes1, 'Position', [X Y 200 250]);
imshow(img);
```

Para la sección 1 se configura el objeto *Control* que indicará si está o no en el modo de comandos con la función *set*, propiedad '*BackgroundColor*' y valor '*red*' o '*green*' según corresponda. El color se define en función de la dirección del giro de la cabeza, obtenido en la etapa 5.

```
set(handles.Control, 'BackgroundColor', LedIndicator);
```

En la sección 3 se presentan los resultados de 4 etapas. De la etapa 1, se devuelven los fotogramas adquiridos con la función *imshow*. Para entregar los rectángulos de la etapa 2, 3 y 4 se utiliza la función *insertShape* que tiene como entrada *nosePolygon* y *facePolygon*. Estas entradas son los rectángulos *noseBox* y *faceBox* respectivamente, pero con cambio de dimensión de la matriz a un vector Lx1 con la función *reshape*. Este cambio de matriz a vector se realiza porque es la forma que acepta la función *insertShape* como entrada.

- **insertShape:** devuelve una imagen *camFrameRGB* con una forma *shape* insertada en una imagen utilizando opciones adicionales especificadas por una propiedad *Name* y argumento *Value*.
 - **Sintaxis:** *camFrame* = *insertShape*(*camFrame*, *shape*, *position*, *Name*, *Value*)
 - **Argumentos de entrada:**
 - camFrame*: imagen de entrada a la que se le insertará una figura.
 - Shape*: tipo de forma, especificada como vector de caracteres. El vector puede ser 'Rectangle', 'Line', 'Polygon' y 'Circle'.

Position: posición de la forma especificada de acuerdo con el tipo de forma. En este caso se usa la forma polígona dado como un vector $L \times 1$ $[X_1 Y_1 X_2 Y_2 \dots X_L Y_L]$ con L número de vértices.

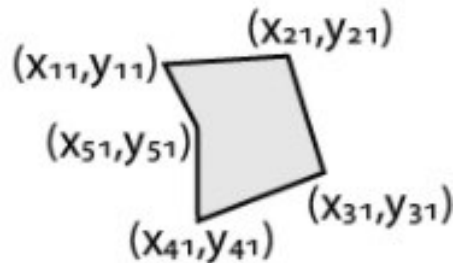


Figura 2.49. Polígono formado por $L=5$ vértices [57].

- **Propiedades:**

LineWidth: ancho de la línea del borde de la forma a insertar. Por defecto es 1 pixel.

Color: color de la forma. Por defecto es 'yellow'.

```
facePolygon = reshape(faceBox', 1, []);
nosePolygon = reshape(noseBox', 1, []);
camFrameRGB = insertShape(camFrameRGB, 'Polygon', facePolygon,
'LineWidth', 4);
camFrameRGB = insertShape(camFrameRGB, 'Polygon', nosePolygon,
'LineWidth', 4);
```

Finalmente, se insertan en el fotograma los puntos que representan las características de cada objeto, para lo cual se usa la función *insertMarker* que recibe las entradas *oldNosePoints* y *oldFacePoints* obtenidas de la etapa 4. Para presentar el frame actual con todas las formas superpuestas se usa la función *imshow*.

- **insertMarker:** devuelve una imagen *camFrame* con un marcador (punto) *marker* en la posición *position* en una imagen utilizando opciones adicionales especificadas por una propiedad *Name* y argumento *Value*.
 - **Sintaxis:** `camFrame = insertShape(camFrame, position, marker, Name, Value)`
 - **Argumentos de entrada:**
 - camFrame:* imagen de entrada a la que se le insertará el marcador.
 - position:* posición del marcador especificado como una matriz $M \times 2$ con M puntos de coordenadas (x,y) .
 - marker:* tipo de marcador especificado como un vector de caracteres.

- **Propiedades:**

Size: número escalar del tamaño del marcador. Por defecto es 3.

Color: color del marcador. Por defecto es 'green'.

```
camFrameRGB= insertMarker(camFrameRGB, oldFacePoints, '+', 'Color',  
'white');  
camFrameRGB = insertMarker(camFrameRGB, oldNosePoints, '+', 'Color',  
'red');  
axes(handles.axes2)  
imshow(camFrameRGB);
```

El código completo que cumple con los objetivos de la etapa 7 es el siguiente:

```
%%ETAPA 7 (cont):Demostración del movimiento de la silla desde una  
vista superior  
  
%Presentación de la imagen de la silla  
axes(handles.axes1)  
img = imread('silla.jpg');%Se lee la imagen  
set(handles.axes1,'Units', 'pixels');%Se configura la unidad de los  
ejes  
set(handles.axes1,'Position', [X Y 200 250]);%Se establece la posición  
y tamaño de la imagen de la silla.  
imshow(img);%se presenta la imagen  
  
%Presentación de los rectángulos del rostro y nariz  
facePolygon = reshape(faceBox', 1, []);%Vectorización del rectángulo  
del rostro  
nosePolygon = reshape(noseBox', 1, []);%Vectorización del rectángulo  
de la nariz  
camFrameRGB = insertShape(camFrameRGB, 'Polygon', facePolygon,  
'LineWidth', 2);  
camFrameRGB = insertShape(camFrameRGB, 'Polygon', nosePolygon,  
'LineWidth', 2);  
%Presentación de los puntos de seguimiento  
camFrameRGB = insertMarker(camFrameRGB, oldFacePoints, '+', 'Color',  
'white');  
camFrameRGB = insertMarker(camFrameRGB, oldNosePoints, '+', 'Color',  
'red');  
axes(handles.axes2)  
imshow(camFrameRGB);%Se presenta el frame con los rectángulos y los  
puntos  
  
%Se establece un indicador para saber si está dentro o fuera del modo  
de  
%comandos  
set(handles.Control, 'BackgroundColor', LedIndicator);%Control es el  
objeto statictext  
%%FIN ETAPA 7
```


3. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados de las pruebas se realizan en tiempo diferido y en tiempo real del sistema diseñado. Las pruebas en tiempo diferido se realizan para las etapas (1, 2, 3 y 4) de creación de los objetos de detección, seguimiento y estimación geométrica para garantizar que se trabaje en las mismas condiciones. Mientras que, las pruebas en tiempo real se realizan para las etapas (5 y 6) de movimiento de la silla para probar el desempeño de los parámetros seleccionados en base al tiempo diferido. Además, se presentan los resultados de la simulación del sistema basado en el diagrama planteado inicialmente de la Figura 1.1, explicando los inconvenientes que tiene esta versión del sistema. En la última etapa no se realiza ninguna prueba, solo se realiza la presentación de los resultados (puntos y rectángulos) de las pruebas.

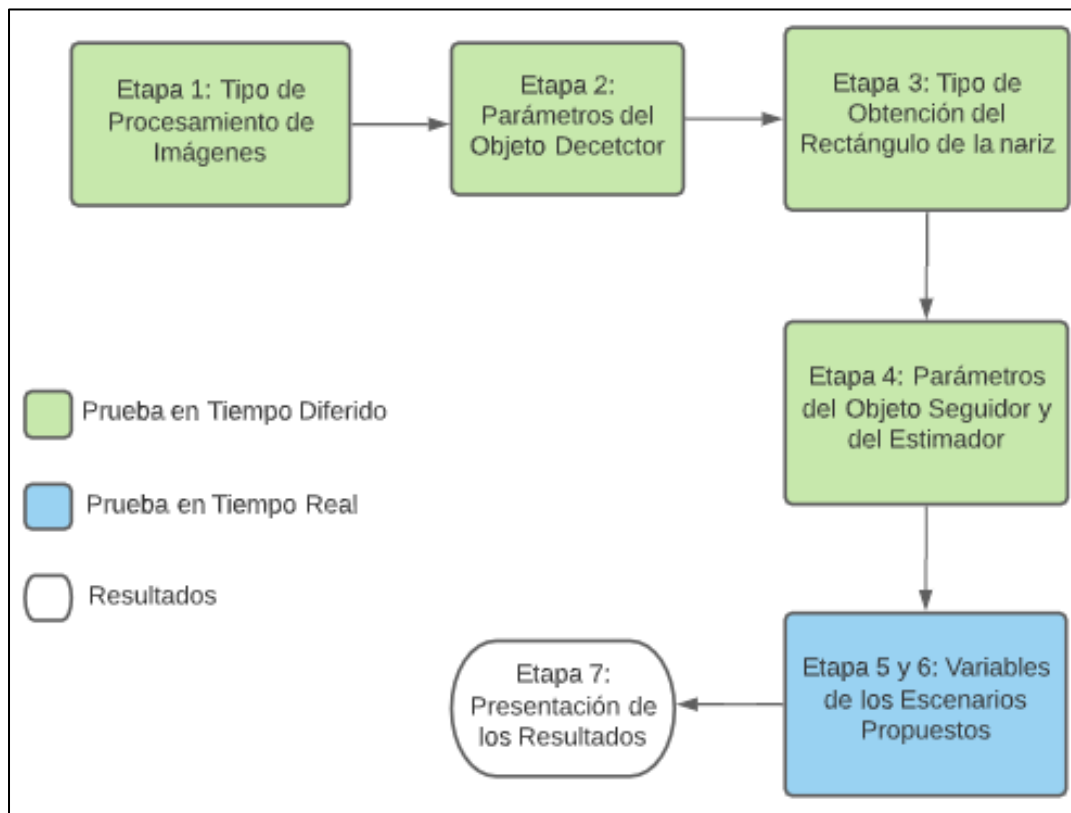


Figura 3.1. Esquema general para las pruebas realizadas. Los recuadros en verde son las etapas donde se usa videos grabados y cargados a Matlab para probar cada variable. El recuadro en azul representa las etapas probadas en tiempo real con la cámara infrarroja.

Los parámetros de los objetos usados para diseñar el sistema se especifican en la Tabla 3.1. Así como las formas en que se pueden desarrollar las distintas etapas.

Tabla 3.1. Variables, parámetros o formas que influyen en el desarrollo de cada etapa.

Etapas	Variables para Prueba
1. Adquisición y Procesamiento de Imágenes	Tipo de Procesamiento de Imagen
2. Detección de Rostro	'Scale Factor' 'Merge Threshold'
3. Obtención del Rectángulo de la Nariz	Forma de Obtención
4. Seguimiento del Rostro y Nariz	MaxBidirectionalError 'BlocSize' 'MaxIterations' 'NumPyramidLevels' 'MaxNumTrials' 'Confidence' MaxDistance'
5. Detección de la Dirección de Giro de la Cabeza	Nivel de Iluminación Tipo de Ambiente
6. Obtención de Parámetros para el Movimiento de la Silla	Resolución de Imagen Posición de la Cámara

Para las pruebas en tiempo diferido se tienen 6 videos grabados mediante una cámara infrarroja con una resolución de 480x320 a 12 fps (fotogramas por segundo) en un ambiente exterior con diferentes niveles de iluminación y donde se realizan movimientos de acuerdo con la característica a evaluar. Sin embargo, el video 6 no se usa debido a que no se logra detectar el rostro bajo esas condiciones de iluminación. Con el objetivo de obtener una mejor medida de las diferencias al escoger un valor en específico, se procesaron los videos capturados aumentando la resolución a 1920x1080 a 30 fps. Para las pruebas en tiempo real se trabajará con 480x320 para reducir el tiempo de procesamiento y poder trabajar efectivamente en tiempo real.

- Video 1 'Mov Horizontal - 1920x1080': video grabado con 412 fotogramas y el sol de frente al rostro a 3015 luxes realizando rotaciones en ambos sentidos.
- Video 2 'Mov Vertical – 1920x1080': video grabado con 188 fotogramas y el sol de frente al rostro con 3002 luxes realizando inclinaciones hacia atrás y adelante.
- Video 3 'Sol de Frente – 1920x1080: video grabado con 1350 fotogramas y el sol de frente al rostro con 3024 luxes realizando rotaciones e inclinaciones verticales.
- Video 4 'Sol lado derecho – 1920x1080': video grabado con 1265 frames, el lado derecho hacia el solo con 2981 luxes y el lado izquierdo iluminado con 43 luxes.

- Video 5 'Sol lado izquierdo – 1920x1080': video con 1351 frames, el lado izquierdo hacia el sol con 2996 luxes y el lado derecho 49 luxes.
- Video 6 'Sol de Atrás – 1920x1080': video con 282 frames y 30 luxes de iluminación, la fuente de luz (sol) estaba en dirección a la cámara y el rostro se veía solo como una sombra por lo que no se pudo detectarlo.

 Mov Horizontal - 1920X1080		Archivo MP4	13.449 KB
 Mov Vertical - 1920x1080		Archivo MP4	6.099 KB
 Sol de Frente - 1920x1080		Archivo MP4	43.957 KB
 Sol lado derecho - 1920x1080		Archivo MP4	41.182 KB
 Sol lado izquierdo - 1920x1080		Archivo MP4	43.929 KB
 Sol de Atrás - 1920x1080		Archivo MP4	9.305 KB

Figura 3.2. Base de datos de videos grabados con la cámara de visión nocturna para las pruebas en tiempo diferido.

En la etapa 1, se usaron los videos 3, 4 y 5 para determinar el número de características detectadas en función del procesamiento. Por otro lado, en la etapa 2, se utilizaron los videos 3, 4 y 5 para definir el tiempo de detección del rostro variando los parámetros del objeto detector. Mientras que, para la etapa 3, con los videos 1 y 2 se encuentra la mejor forma de obtención del rectángulo de la nariz. Finalmente, para la etapa 4, se resolvió a usar los videos del 2 al 5 y en función de la variación de los diferentes parámetros de los objetos seguidor y estimador se precisan los puntos encontrados.

Las pruebas en la etapa 1 y 3 son formas en los que se puede realizar el sistema, pero no representan una variable con un valor específico. Por lo tanto, de acuerdo con la Tabla 3.1 se tienen 9 parámetros a evaluar en tiempo diferido y 3 variables a considerar en las pruebas en tiempo real. En base a las pruebas realizadas en este capítulo se define qué variables y parámetros son los que más influyen en el sistema.

3.1 ETAPA 1 (PRUEBAS): ADQUISICIÓN Y PROCESAMIENTO DE IMÁGENES

Los parámetros de los objetos están configurados inicialmente en su valor por defecto para posteriormente cambiar su valor en base a las pruebas realizadas. Los parámetros y sus valores se muestran a continuación:

Tabla 3.2. Valores iniciales de los parámetros configurables de los objetos.

Objeto Detector	Objeto Seguidor	Objeto Estimador
ScaleFactor: 1.1	MaxBidirectionalError: 2	MaxNumTrials: 1000
MergeThreshold: 4	BlockSize: [31 31]	Confidence: 99
	MaxIterations: 30	MaxDistance: 1.5
	NumPyramidLevels: 3	

El procesamiento propuesto para las imágenes de entrada del sistema propuesto son la mejora de contraste y reducción de la matriz de representación de la imagen. El aumento de contraste permite aumentar la diferencia entre niveles de intensidad parecidos de una imagen, para esto se usa la ecualización de histograma. En cambio, para reducir la matriz de representación de la imagen capturada por la cámara de $M \times N \times 3$ a $M \times N$ y mejorar el tiempo de procesamiento se utiliza la conversión a escala de grises.

Tomando lo anterior, se realizaron pruebas para determinar si era necesario o no el procesamiento en función del número de características obtenidas. Adicionalmente, se considera el orden en que se hacen los dos procesamientos para determinar si influye en el rendimiento de la detección. La Tabla 3.3 presenta los resultados obtenidos de 10 pruebas con el tipo de procesamiento y el orden en que se realizaron.

Tabla 3.3. Número de características encontradas para el Rostro y Nariz. El procesamiento de escala de grises y luego ecualización de histograma presenta 904 puntos más respecto a solo realizar la ecualización de histograma.

Tipo de Procesamiento	Puntos (Rostro)	Puntos (Nariz)
Ninguno	1647	272
Ecualización de Histograma	2586	323
Escala de Grises	1536	294
Ecualización de Histograma y Escala de Grises	2327	375
Escala de Grises y Ecualización de Histograma	3490	517

La Tabla 3.3 muestra que convirtiendo una imagen RGB en formato Escala de Grises y luego ecualizándola en base a su histograma se tiene una cantidad de puntos detectados considerablemente mayor a los demás procesamientos. Por lo tanto, se elige este orden de

procesamiento de los fotogramas capturados. Un ejemplo de esta cantidad de puntos se muestra en la Figura 3.3.

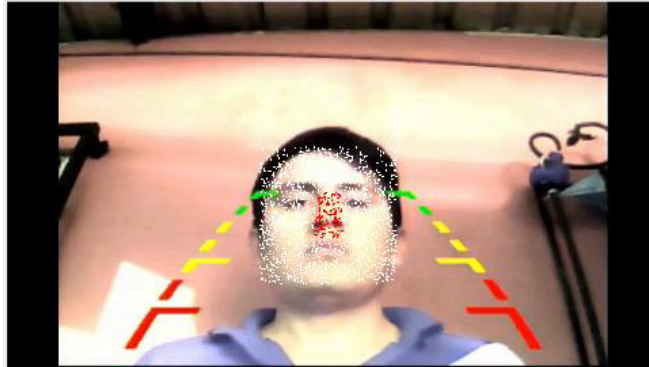


Figura 3.3. Puntos encontrados en el Rostro y Nariz con el tipo de procesamiento de imágenes elegido.

3.2 ETAPA 2 (PRUEBAS): DETECCIÓN DE ROSTRO.

Las pruebas en esta etapa consideran el tiempo que se tarda el algoritmo en detectar un rostro al iniciar el sistema o al girar y volver a realizar una detección. También se toma en cuenta el número de detecciones correctas e incorrectas para determinar el desempeño del algoritmo. El número de detecciones correctas del rostro que se deben realizar son 11: 4 del video 3, 4 del video 4 y 3 del video 5.

3.2.1 AUMENTO DE LA VENTANA DE DETECCIÓN (SCALEFACTOR)

El primer parámetro que se varia es el factor de aumento (*ScaleFactor*) de la ventana donde se realiza la detección y los demás parámetros de los objetos se mantienen.

Tabla 3.4. Tiempo de detección del rostro y número de detecciones correctas/incorrectas en función del factor de escala.

ScaleFactor	1.01	1.05	1.1	1.2	1.3	1.4
Tiempo en detectar un rostro [s]	1.150	0.194	0.114	0.061	0.045	0.032
Número de Detecciones Correctas	5	9	11	9	7	5
Número de Detecciones Incorrectas	26	5	0	0	0	0

En la Tabla 3.4 se presentan los resultados de las pruebas realizadas. El factor de escala de valor 1.1 es el único valor donde se logra detectar de forma correcta los 11 rostros a lo

largo de los 3 videos, por lo tanto, se escoge este valor. Con un factor muy pequeño, se tuvieron demasiadas detecciones incorrectas y con factores mayores que 1.1 no se tuvieron detecciones incorrectas, pero a medida que incrementa el número rostros detectados disminuye.

3.2.2 NÚMERO DE DETECCIONES MÍNIMAS EN UNA REGIÓN (MERGETHRESHOLD)

El segundo parámetro del objeto detector por variar es el umbral de unión (MergeThreshold) de la región para decretar la detección y los demás parámetros permanecen fijos.

En la Tabla 3.5 se presentan los resultados de las pruebas realizadas. El rango 7 al 11 del umbral logra detectar de forma correcta los 11 rostros; mientras que, por debajo y por encima de este valor no se logra y se tienen detecciones incorrectas. Un valor alto de umbral requiere más tiempo para detectar un objeto, pues debe haber más coincidencias en una determinada región de la imagen para decretarlo como existente lo cual es restrictivo. Por otro lado, un valor bajo requiere menos tiempo, pero al existir bajas coincidencias se cometen más errores y se tiene falsos positivos. Considerando lo expuesto, se define el valor de umbral en 11, ya que dentro del rango de detecciones correctas no hay mucha diferencia en cuanto al tiempo de detección y al necesitar más coincidencias es más robusto.

Tabla 3.5. Tiempo de detección del rostro y número de detecciones correctas/incorrectas en función del umbral de unión.

MergeThreshold	40	30	25	15	11	10	9	7	4
Tiempo en detectar un rostro [s]	0.092	0.105	0.095	0.081	0.079	0.081	0.083	0.085	0.078
Número de Detecciones Correctas	5	7	7	10	11	11	11	11	7
Número de Detecciones Incorrectas	0	0	0	0	0	0	0	0	6

3.3 ETAPA 3 (PRUEBAS): OBTENCIÓN DEL RECTÁNGULO DE LA NARIZ

El desarrollo de esta etapa se mostró en el capítulo anterior; sin embargo, se puede realizar de una forma diferente. El objetivo de la etapa 3 es obtener el rectángulo de la nariz, para lo cual el objeto *vision.CascadeObjectDetector*, que usa el algoritmo de Viola-Jones, dispone

de un modelo para detectar narices como se muestra en la Tabla 2.1. Este modelo puede ser usado para cumplir este objetivo, pero en la Tabla 3.6 se detallan los resultados que justifican la obtención acorde con las proporciones normales del rostro detectado en la etapa 2.

Tabla 3.6. Tiempo de obtención y tamaño del rectángulo con el método de obtención .

Parámetro	Obtención mediante el Algoritmo Viola-Jones	Obtención en base a las Proporciones del Rostro
Tiempo Obtención del Rectángulo de la Nariz	0.617	0.137
Tamaño del Rectángulo	190x157	130.6x218.9

Usando el algoritmo de viola Jones se demora 4.5 veces más que en base a las proporciones del rostro, de igual forma el ancho es mayor para el primer caso. Como se muestra en la Figura 3.4 el rectángulo por Viola-jones no abarca todo el centro de la nariz y esto disminuye la diferencia de movimiento respecto a los puntos del rostro .

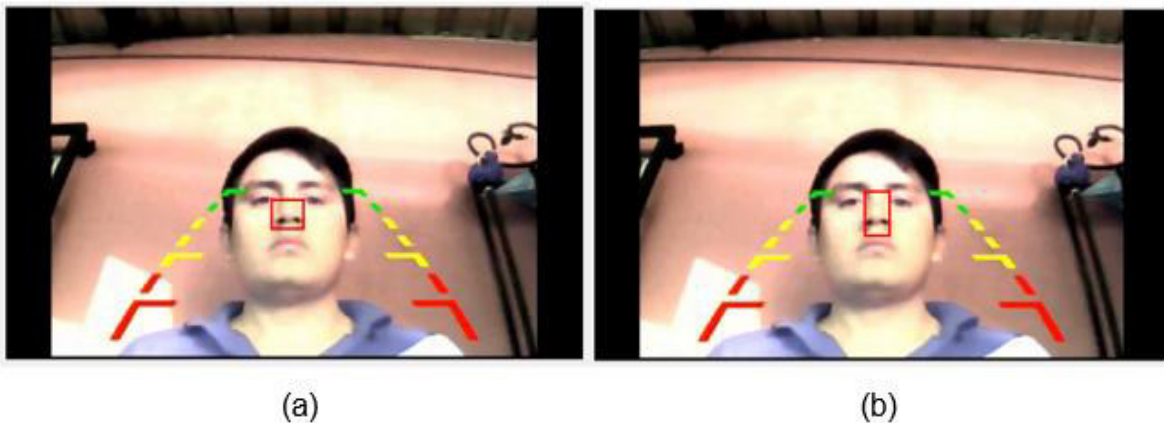


Figura 3.4. Rectángulos de la nariz en base a la forma de su obtención. (a) Mediante el algoritmo de Viola-Jones. (b) Mediante las proporciones normales del rostro.

3.4 ETAPA 4 (PRUEBAS): SEGUIMIENTO DEL ROSTRO Y NARIZ

Las pruebas correspondientes a esta etapa justifican la elección del tipo de características elegidas para el seguimiento y los valores de los parámetros de cada objeto. La elección de cada parámetro está basada en el número de punto que se detectan y cómo estos se logran mantener conforme transcurre el video. Para esto se usan los videos 3, 4 y 5 en la prueba del tipo de características y el video 2 en las pruebas de seguimiento y estimación geométrica.

3.4.1 ELECCIÓN DEL ALGORITMO DE DETECCIÓN DE CARACTERÍSTICAS

Estos algoritmos funcionan dentro de una región previamente definida, para lo cual se usa el detector de la etapa 2 configurado con los parámetros analizados $ScaleFactor=1.1$ y $MergeThreshold=11$.

En la Tabla 3.7 se pueden observar los algoritmos que permite usar Matlab para encontrar características en una región de Interés. También se detalla la función que permite implementar cada algoritmo y el número de características correspondientes encontradas. Así, se aprecia que el algoritmo *Minimum eigenvalue* funciona mejor para la detección de características en rostros, encontrando más puntos que los demás algoritmos una cantidad mayor a 10 veces. Por tal motivo, es escoge el algoritmo del Valor Propio Mínimo.

Tabla 3.7. Número de puntos encontrados de acuerdo con el algoritmo de detección de características usado.

Algoritmo de Detección de Características	Función en Matlab	Número de características
Features from Accelerated Segment Test (FAST)	detectFASTFeatures	1
Minimum eigenvalue	detectMinEigenFeatures	3490
Harris-Stephens	detectHarrisFeatures	324
Binary Robust Invariant Scalable Keypoints (BRISK)	detectBRISKFeatures	36
Speeded-up robust features (SURF)	detectSURFFeatures	69
Maximally stable extremal regions (MSER)	detectMSERFeatures	55

3.4.2 PARÁMETROS DEL OBJETO SEGUIDOR.

Los parámetros del objeto *vision.CascadeObjectDetector* fueron explicados y presentados anteriormente, pero no se justificó la elección de un valor específico. Este objeto funciona a la par con el objeto *estimateGeometricTransform2D*, el cual retorna la región de interés para el algoritmo de detección de características en el siguiente fotograma y así no es necesaria realizar la detección del rostro en todos los frames. Por tal razón, se especifican los valores de los parámetros de todos los objetos cada que se vaya a variar alguno de ellos.

3.4.2.1 Máximo Error Bidireccional en el Seguimiento de una Característica

El parámetro que se varía es *MaxBidirectionalError*, mientras que, los demás se mantienen fijos.

La Tabla 3.8 indica que en el primer fotograma del video 2 se encontraron 2999 puntos y conforme el video transcurre estas se van perdiendo por el ruido. El ruido es un cambio de intensidad del pixel por la variación de iluminación, rotación del objeto o movimiento del mismo. Mientras más se incremente el error máximo se es más permisivo en cuanto al movimiento del objeto, pero al permite más error la confiabilidad de esa característica disminuye. Si la confiabilidad baja, el objeto estimador no puede hacer una buena representación de la geometría del objeto. Por lo tanto, se debe tener un equilibrio en el máximo error. El valor de 4 pixeles de error es el único que permite obtener 4 puntos en el fotograma 150, por lo que se escoge ese valor.

Tabla 3.8. Número de Puntos a lo largo del video según el Máximo Error Bidireccional.

'MaxBidirectionalError'	Número de Puntos Detectados			
	Frame 1	Frame 50	Frame 100	Frame 150
0	2999	0	0	0
2	2999	58	9	0
3	2999	69	10	0
4	2999	58	10	4
5	2999	41	19	0

3.4.2.2 Tamaño del Bloque donde se realiza el algoritmo KLT.

El parámetro que se varía es *BlockSize*; mientras que, los demás se mantienen fijos en los valores ya analizados.

El tamaño de ventana establece la vecindad de pixeles o región de la imagen donde se realiza todo el proceso que necesita el algoritmo KLT. Esta región no debe ser muy grande por lo expuesto en la sección 1.3.4.1, que es una restricción para que el algoritmo funcione adecuadamente y el procesamiento no se incremente. Por otro lado, si este valor es muy pequeño se tiene una buena cantidad de pixeles que permitan caracterizar esa región y el vector desplazamiento no converge. En la Tabla 3.9 se muestra que tamaño [41 41] y [61 61] ofrecen puntos en el frame 150; sin embargo, se escoge el menor para disminuir el procesamiento.

Tabla 3.9. Número de Puntos a lo largo del video según el Tamaño de Bloque.

'BlockSize'	Número de Puntos Detectados			
	Frame 1	Frame 50	Frame 100	Frame 150
[5 5]	2999	0	0	0
[11 11]	2999	0	0	0
[21 21]	2999	16	8	0
[31 31]	2999	51	13	0
[41 41]	2999	160	31	4
[61 61]	2999	289	52	4

3.4.2.3 Iteraciones para encontrar el Desplazamiento de un Punto

El parámetro que se varía es *MaxIterations*. Acorde con la Tabla 3.10 un número bajo de iteraciones no logra que el algoritmo llegue a converger y por ende se pierden más puntos. Por otro lado, si el número es mayor a 50 se tienen demasiadas iteraciones y aunque se logre detectar más puntos en los siguientes frames, el tiempo que toma calcular todos los puntos se incrementa. Para fines del sistema propuesto no es tan necesario obtener todos los puntos, sino que se pueda implementar en tiempo real. Así que, se escoge un número de 40 iteraciones, pues por encima de este valor se obtiene resultados similares.

Tabla 3.10. Número de Puntos a lo largo del video según la máxima cantidad de iteraciones para encontrar el vector desplazamiento.

'MaxIterations'	Número de Puntos Detectados			
	Frame 1	Frame 50	Frame 100	Frame 150
5	2999	128	19	0
10	2999	140	12	2
20	2999	127	27	3
30	2999	160	26	3
40	2999	162	30	5
50	2999	160	27	5

3.4.2.4 Niveles de Resolución para el acercamiento o alejamiento de objetos

El parámetro por evaluar es *NumPyramidLevels*. La Tabla 3.11 indica que 4 niveles de resolución son idóneos para los movimientos de acercamiento/alejamiento del rostro que una persona puede hacer sentada delante de la cámara. A pesar de que, hay más niveles

de resolución donde se logran encontrar puntos en el frame 150, un mayor número incrementa la carga computacional y para fines de tiempo real no es conveniente.

Tabla 3.11. Número de Puntos a lo largo del video según los niveles de resolución.

'NumPyramidLevels'	Número de Puntos Detectados			
	Frame 1	Frame 50	Frame 100	Frame 150
0	2999	23	128	0
1	2999	140	18	0
3	2999	144	13	2
4	2999	163	27	5
5	2999	160	32	5
6	2999	160	28	4

3.4.3 PARÁMETROS DEL OBJETO ESTIMADOR.

Los parámetros del objeto *estimateGeometricTransform2D* fueron explicados y presentados en el capítulo 2, pero no se justificó la elección de un valor específico. Como se mencionó anteriormente, se usa junto con el objeto seguidor. Por lo que, se especifican los valores de los parámetros de todos los objetos cada que se vaya a variar alguno.

3.4.3.1 Máximo Número de Intentos para Encontrar un Nuevo Punto.

El parámetro que se varía es *MaxNumTrials*. Este valor es usado por Matlab para encontrar la Matriz de Transformación. La cual se obtiene escogiendo aleatoriamente 2 puntos. Por lo que, si se escoge aleatoriamente dos puntos buenos en los primeros intentos no sería necesario establecer un valor alto. Sin embargo, si los mejores puntos para describir la Matriz se escogen después de muchos intentos, si es necesario un valor alto. Como no hay una relación del número de puntos al incrementar o disminuir *MaxNumTrials* se deja en su valor por defecto, el cual es 1000.

Tabla 3.12. Número de Puntos a lo largo del video según la máxima cantidad de intentos para encontrar un nuevo punto.

'MaxNumTrials'	Número de Puntos Detectados			
	Frame 1	Frame 50	Frame 100	Frame 150
60	2999	154	32	5
100	2999	122	8	2
500	2999	147	31	6
1000	2999	160	14	2

3.4.3.2 Confianza de una Característica.

El parámetro que se evalúa es *Confidence*. La influencia de la confianza de una característica en los siguientes frames se muestra en la Tabla 3.13. Un valor de confianza bajo no permite estimar adecuadamente la geometría del objeto y se pierden más características. Un valor alto permite obtener más características y el tiempo que involucra una mayor o menor confianza no es significativo. Por este motivo, se escoge la mayor confianza posible, 99%.

Tabla 3.13. Número de Puntos a lo largo del video según la confianza de la característica.

'Confidence'	Número de Puntos Detectados			
	Frame 1	Frame 50	Frame 100	Frame 150
80		111	18	0
85	2999	103	17	0
90	2999	135	19	1
95	2999	146	21	2
99	2999	157	20	3

3.4.3.3 Máxima Distancia entre un Punto y su Proyección.

El último parámetro por determinar su valor correspondiente es *MaxDistance*. La distancia entre la posición de un punto y su proyección en el siguiente fotograma influye directamente en la cantidad de puntos como se indica en la Tabla 3.14. Así, con una mayor distancia se permite más movilidad, pero los puntos pueden no representar adecuadamente la geometría del objeto. En la tabla se muestra que con valores bajos de distancia máxima se tienen muchos puntos que no cumplen con este criterio y conforme transcurre el video se pierden. Por lo tanto, se escoge el valor 15 pixeles como distancia máxima ya que contiene varios puntos y por encima de este valor la diferencia no es significativa en cuanto a la forma del rectángulo del rostro como se muestra en la Figura 3.5.

Tabla 3.14. Número de Puntos según la máxima distancia entre puntos proyectados.

'MaxDistance'	Número de Puntos Detectados			
	Frame 1	Frame 50	Frame 100	Frame 150
2	2999	254	38	3
4	2999	659	340	119
6	2999	1230	956	407
10	2999	1557	1263	694
15	2999	2123	1388	828
20	2999	2307	1817	995
25	2999	2319	2075	1411

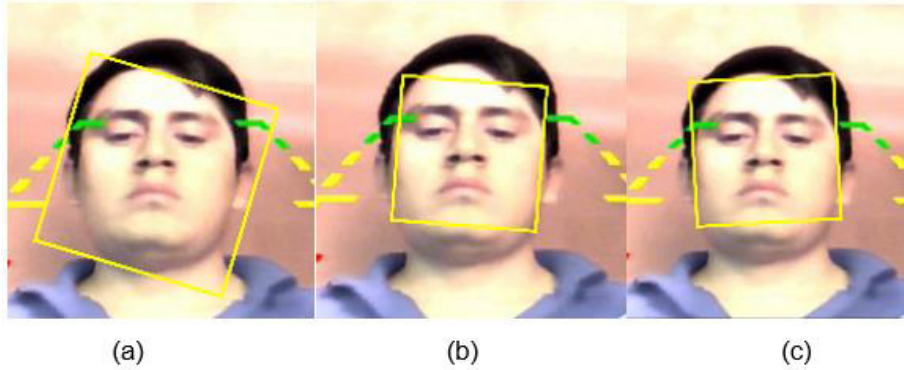


Figura 3.5. Rectángulos del Rostro en el Fotograma 100 con diferentes valores de distancia máxima. (a) *MaxDistance=2*. (b) *MaxDistance=15*. (c) *MaxDistance=25*.

3.5 ETAPA 5 Y 6 (PRUEBAS): ENTRADA, SALIDA Y PRUEBA DEL MODO DE COMANDOS

Los parámetros de lo objeto analizados en tiempo diferido han cambiado con respecto a su valor inicial establecidos en la Tabla 3.2. Así, al haber analizado etapa por etapa se definen los parámetros con sus valores finales y con los cuales se realizarán las pruebas en tiempo real según la Tabla 3.15.

Tabla 3.15. Valores finales de los Parámetros Configurables de los Objetos.

Objeto Detector	Objeto Seguidor	Objeto Estimador
ScaleFactor: 1.1	MaxBidirectionalError: 4	MaxNumTrials: 1000
MergeThreshold: 11	BlockSize: [41 41]	Confidence: 99
	MaxIterations: 40	MaxDistance: 15
	NumPyramidLevels: 4	

Las variables de los objetos que más influyen en el desempeño del sistema son en orden de importancia son: *MaxDistance*, *MergeThreshold* y *MaxBidirectionalError*.

Se probarán diferentes ángulos de enfoque de la cámara y distintas resoluciones de imagen para determinar su influencia en el funcionamiento en tiempo real del sistema. Luego, se presentan los resultados del uso de un número determinado de parpadeos para entrar y salir del modo de comandos inicialmente propuesto para el diseño del sistema. Adicionalmente para determinar qué variables del entorno del usuario se plantean 6 escenarios en un ambiente interior y 5 escenarios en un ambiente exterior para determinar si el nivel de iluminación influye en el desarrollo del sistema.

3.5.1 POSICIÓN IDEAL DE LA CÁMARA.

Para el modo de comandos se detallan una serie de movimientos de la cabeza para controlar la silla de ruedas. Dentro de estos movimientos están inclinación hacia adelante y hacia atrás, así que, se asume que al mover la cabeza hacia adelante habrá un acercamiento a la cámara y cuando se la mueva hacia atrás se alejará de ella. Al no cumplir estos criterios para cualquier posición de la cámara se establece un rango de ángulos en los que la cámara debe enfocar el rostro usando el sensor de orientación del celular móvil.

Para esto se definen los parámetros que tendrá esta prueba. La resolución se establece en 480x320, una fuente de luz que ilumine la parte frontal de rostro y se prueba en un ambiente interior con iluminación de 112 luxes. Los ángulos de enfoque se miden respecto a la línea de vista del rostro en una posición vertical como se muestran en Figura 3.6. En la figura también se muestran los ángulos para los cuales el sistema funciona adecuadamente.

En la Tabla 3.16 se muestran los resultados de las pruebas realizadas. Se especifica el ángulo de enfoque de la cámara. Para esta prueba se hicieron 20 movimientos de la cabeza hacia adelante y 20 hacia atrás. A partir de estos movimientos se determinó si la imagen de la silla se movía efectivamente en la dirección correspondiente. Antes de los 20° y después de los 45° el sistema no interpreta bien el movimiento de la cabeza. En el rango de 20° a 45° el sistema interpreta correctamente todos 40 movimientos acercamiento/alejamiento de la cabeza.

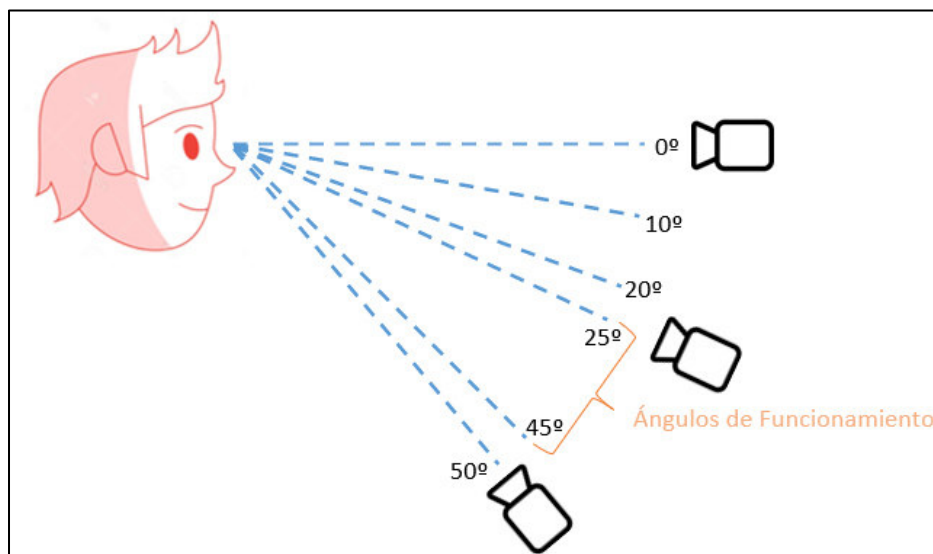


Figura 3.6. Posiciones de prueba de la cámara y el rango de ángulos de funcionamiento.

Tabla 3.16. Interpretaciones correctas del movimiento vertical de la cabeza según el ángulo de posicionamiento de la cámara.

Posición de la cámara	Movimientos hacia el Frente Interpretados Correctamente	Movimientos hacia atrás Interpretados Correctamente	Porcentaje de Interpretaciones Correctas
0°	0	0	0%
10°	6	10	40%
20°	18	15	82.5%
25°	20	18	95%
30°	20	20	100%
35°	20	20	100%
45°	20	20	100%
50°	10	16	65%

3.5.2 RESOLUCIÓN DE LA CÁMARA

Las pruebas realizadas en cuanto a la resolución de la cámara determinan si el aumentar este parámetro influye en el funcionamiento en tiempo real del sistema. Las resoluciones disponibles del capturador EasyCap son 480x320, 640x480 y 720x480. En la siguiente tabla se muestra los resultados de las pruebas realizadas.

Tabla 3.17. Características que varían con el cambio de Resolución.

Resolución	FPS [Frame/s]	Tiempo de Procesamiento por Frame [s]	% de Uso del CPU
480x320	13.13	0.076	29.6%
640x480	12.4	0.081	31.1%
720x480	11.9	0.084	33.2%

En base a la Tabla 3.17 se observa que cuando se sube la resolución de la cámara se procesan capturan menos imágenes por segundo, debido a que aumenta el tiempo de procesamiento de cada fotograma. Esto ayuda a que a pesar del aumento de resolución se pueda seguir usando el sistema en tiempo real. Por lo tanto, las resoluciones disponibles de la capturadora no influyen en el desarrollo en tiempo real del sistema.

3.5.3 ENTRADA/SALIDA DEL MODO DE COMANDOS EN BASE AL NÚMERO DE PARPADEOS

Este modo de entrar y salir del modo de comandos de la silla de ruedas se planteó inicialmente para ingresar cuando se detecten 3 parpadeos consecutivos. Para ser consecutivos estos deben suceder en menos de 3 segundos, tiempo elegido debido a que ocurren en promedio entre 10 a 20 parpadeos por minutos. Esto refleja que entre dos parpadeos consecutivos existen 3 y 6 segundos. Por otro lado, el tiempo promedio de parpadeo es de 0.4 segundos en condiciones normales [4]. En base al tiempo promedio y convirtiendo este tiempo al número de fotogramas durante el cual se tienen los ojos cerrados se tiene la cantidad necesaria de estos fotogramas para ingresar al modo de comandos o salir del mismo de ser el caso.

La recolección de imágenes de rostro capturadas por la cámara infrarroja se realizó en ambientes de iluminación con 1976, 14, 93750,4 y 0 luxes. La base de datos consiste en 6510 fotos de ojos cerrados y 6510 fotos de ojos abiertos de 10 personas. Las 10 personas son 6 de sexo femenino y 4 sexo masculino, rango de edad entre 11 y 73 años, diferentes tonos de piel y con y sin lentes. Una parte de la base de datos de ojos abiertos se aprecia en la Figura 3.7. Obteniendo una clasificación correcta del 99.1% usando una red con 1 capa oculta y 10 neuronas en dicha capa.

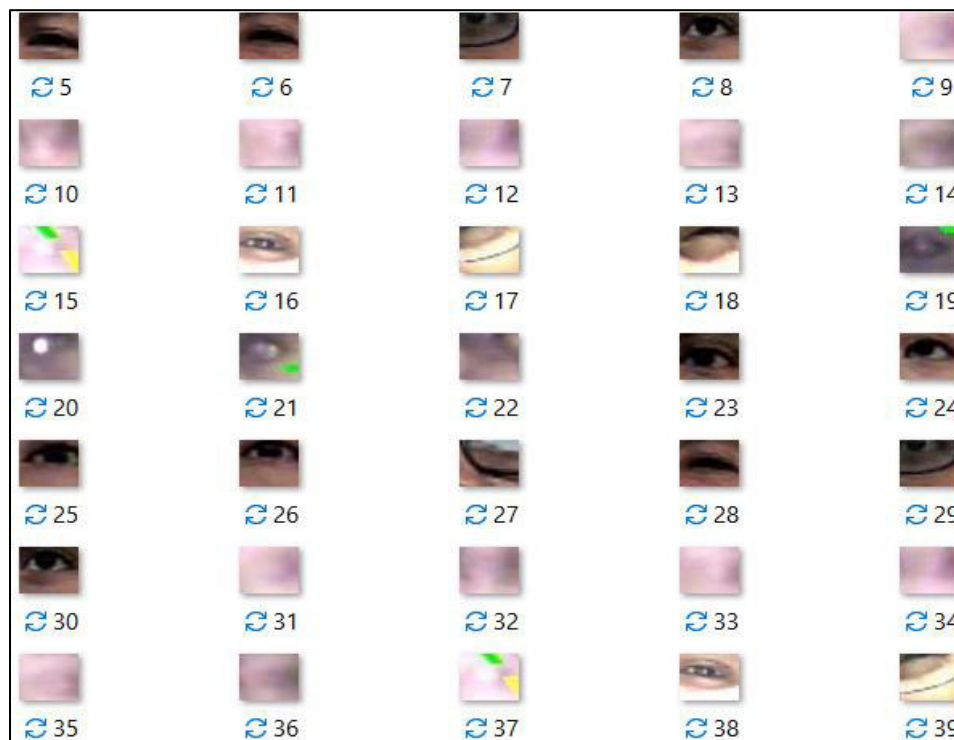


Figura 3.7. Parte de la base de datos implementada.

3.5.3.1 Variación de la frecuencia de parpadeos y de los fotogramas procesados

El inconveniente que se tiene con los tiempos promedios de parpadeos es que estos pueden cambiar considerablemente, pues la frecuencia de parpadeos depende de factores como del cansancio de la mirada, edad, el ambiente y la acción que se realiza. Por ejemplo, en una lectura entre 4.6 y 11.2 parpadeos por minuto o durante una conversación de 15.9 a 27.1 parpadeos [4]. Es así como, los usuarios del sistema podrían entrar o salir del sistema sin darse cuenta y sin tener la intención de hacerlo. Además, la frecuencia de fotogramas procesados varía como se muestra en la Tabla 3.18, debido al comportamiento aleatorio de los algoritmos usados para seguimiento y estimación geométrica.

Tabla 3.18. Variación del número de fotogramas procesados por segundo.

Número de Fotogramas	Tiempo de Adquisición [s]	Fotogramas por segundo
133	11.07	12.01
62	13.72	4.51
65	13.82	4.7
71	10.35	6.86
108	14.03	7.70
102	10.22	9.98

Los valores presentados en la Tabla 3.18 son los valores más comunes en las pruebas realizadas e indican una desviación estándar de 2.95 frames por segundo y un promedio de 7.63. Entonces, el rango de fotogramas estaría dado por $(7.63[fps] \pm 2.95[fps]) \times 0.4[s]$ cuyo resultado da un rango entre 2 y 4 fotogramas. El problema se presenta cuando se toma en cuenta los 3 segundos para considerar los parpadeos consecutivos, pues el rango sería entre 6 y 12 fotogramas para salir o entrar del modo de comandos. Presentado varios inconvenientes al momento de probarlo con diferentes usuarios. Para esto se realizaron pruebas en 3 sujetos, obteniendo que en frecuencia de parpadeos normales se reconoció un ingreso y salida del modo de comandos el 42.3% del tiempo de uso.

3.5.3.2 Funcionamiento en ambientes sin iluminación

Para determinar si efectivamente reconoce un ojo como cerrado o abierto se realizan pruebas con dos sujetos en ambiente de 0 luxes de iluminación. Para lo cual, se tuvo un tiempo de uso promedio del sistema de 120 segundos, en donde se clasificó correctamente la condición del ojo un aproximado de 5.7%. Esto nos permite concluir que en ambiente de

iluminación extremadamente bajos el número de parpadeos no es una buena alternativa para entrar o salir del modo de comandos. Hay que considerar también que con el uso de lente se clasificó correctamente el ojo solo el 1.2% del tiempo. Por tal motivo, se dispone a usar el seguimiento de la nariz para según un movimiento específico de rotación de la cabeza se ingrese o salga del modo de comandos.

3.5.4 NIVELES DE ILUMINACIÓN EN AMBIENTES INTERIORES Y EXTERIORES.

Para las pruebas de iluminación se usó el sensor de luz del celular. Se plantean 6 escenarios dentro de una habitación y 5 escenarios en un ambiente exterior considerados los más comunes, en los cuales el rostro de una persona puede estar iluminada por distintos ángulos de una fuente de luz. Las variables por considerar son:

- Número del frame donde se distorsionan los rectángulos del rostro o nariz.
- Número de puntos existentes en determinado número de fotogramas.
- Nivel de iluminación de cada escenario.
- Correcta interpretación del giro de la cabeza.

Se considera que un rectángulo se ha distorsionado si en posición de reposo, este sobrepasa el umbral de inclinación o de altura respectivo.

Los escenarios propuestos son los siguientes y se pueden ilustrar en la Figura 3.8.

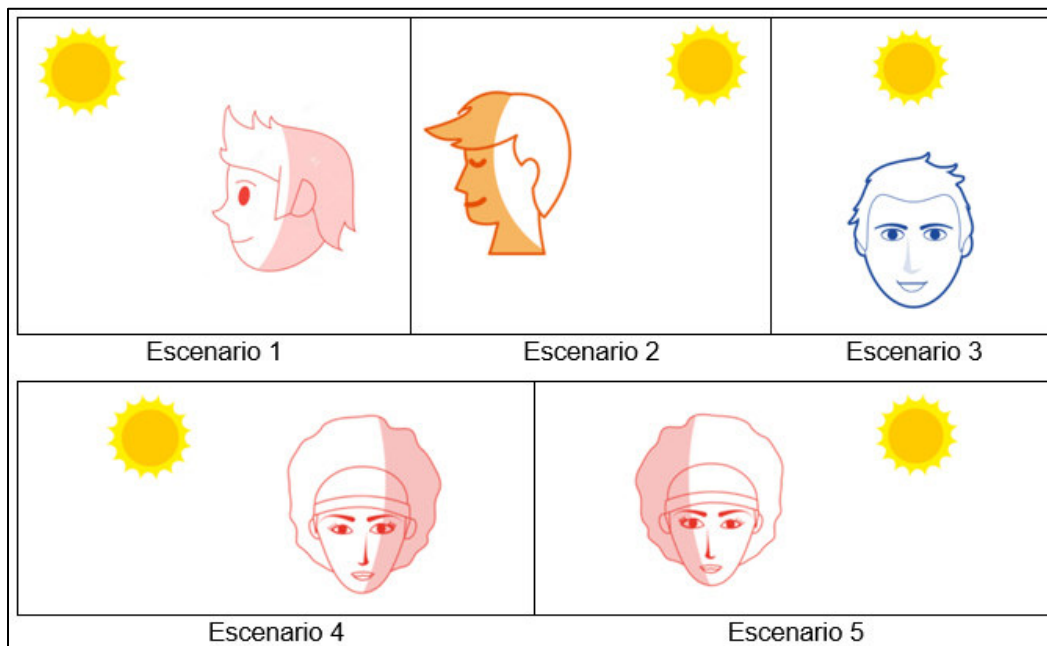


Figura 3.8. Escenarios comunes propuestos para pruebas de funcionamiento.

Escenario 1: fuente de Luz que ilumina la parte frontal del rostro.

Escenario 2: fuente de Luz ubicada en la parte posterior de la cabeza.

Escenario 3: fuente de Luz ubicada encima de la cabeza.

Escenario 4: fuente de Luz que ilumina el lado izquierdo del rostro.

Escenario 5: fuente de Luz que ilumina el lado derecho del rostro.

Escenario 6: habitación sin Iluminación.

Donde la fuente de luz considerada es un foco para interiores y el solo para exteriores.

La siguiente tabla muestra los resultados de las pruebas de los primeros 3 escenarios para ambientes interiores y exteriores y para una habitación sin iluminación. Se muestra la relación que existe en la iluminación respecto al número de puntos; sin embargo, aunque el número de puntos aumenta con la iluminación la diferencia no es significativa, llegando incluso a disminuir cuando la iluminación es alta como en ambientes exteriores. Para ambientes con iluminación bajo beneficia el uso de la cámara de visión nocturna, pues incluso con 0 luxes de iluminación, se encuentra una gran cantidad de puntos, no así para más de 10000 luxes. Además, el primer fotograma donde se distorsionan los rectángulos no es inconveniente en ambientes interiores debido a que se realiza una nueva detección antes del fotograma 458. De la Tabla 3.17 se obtuvo que se procesan 13.13 frames por segundo, entonces para el fotograma 458 habrán pasado 34.88 segundos. Por este motivo se realiza una nueva detección en la posición de reposo cada 30 segundos. Por otro lado, cuando el sol está detrás de la cabeza, no se logra detectar el rostro, así que se concluye que el uso del sistema no es recomendable para ambientes exteriores.

Tabla 3.19. Resultados de las pruebas de funcionamiento de los 3 primeros escenarios y el escenario 6 (sin iluminación).

Escenario	Frame con distorsión	Número de Puntos en el Frame				Nivel de Iluminación [luxes]
		1	100	200	300	
1 (Interior)	602	341	331	327	298	35
1 (Exterior)	474	546	486	290	251	99350
2 (Interior)	486	288	249	232	213	5
2 (Exterior)	-	-	-	-	-	9936
3 (Interior)	583	299	279	187	173	12
3 (Exterior)	353	461	256	150	141	44963
6 (Interior)	458	213	191	142	128	0

De acuerdo con la Tabla 3.20 existe relación entre el lado del rostro más iluminado y la interpretación del sistema relacionado a la dirección del giro de la cabeza. Es así como, el giro hacia el lado más iluminado se interpreta siempre de forma correcta. Mientras que, el giro de la cabeza hacia el lado menos iluminado se detecta entre el 66.67% y el 88.89% de las veces.

Tabla 3.20. Resultados de las pruebas de funcionamiento de los 2 últimos escenarios propuestos para ambientes interiores.

Escenario	Iluminación		Giros correctos/incorrectos	
	lado izquierdo	lado derecho	a la izquierda	a la derecha
4 (Interior)	24 luxes	2 luxes	9 / 9	6 / 9
4 (Exterior)	8946	81126	10 / 10	8 / 10
5 (Interior)	4 luxes	25 luxes	8 / 9	9 / 9
5 (Exterior)	9221	79895	9 / 10	10 / 10

Al considerar tanto las pruebas en tiempo diferido y tiempo real se determina que existen 3 variables propias de los algoritmos que usa el sistema y 3 variables que dependen de la cámara y del ambiente donde se ejecute el sistema. Todas estas variables que influyen en el desempeño correcto del sistema se indican en la Tabla 3.21.

Tabla 3.21. Variables que más influyen en el funcionamiento adecuado del control de la silla mediante movimientos de la cabeza.

Variables propias de los Algoritmos del Sistema	Variables externas al Software del Sistema
Número de coincidencias en una región para decretar la detección de un objeto (MergeThreshold).	Angulo de enfoque de la cámara.
Error permitido en el doble seguimiento (forward y bakward) del punto (MaxBidirectionalError).	Resolución de la Cámara.
Máxima distancia entre un punto y su proyección en el siguiente frame (MaxDistance).	Nivel de iluminación en las partes laterales del Rostro.

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- El movimiento de la silla en el eje vertical depende del ángulo con que la cámara apunte al rostro. Pues, dentro del rango de funcionamiento establecido (25° a 45°) se obtuvo un 98.75% de interpretación correcta del movimiento de la cabeza en ese eje. Sin embargo, fuera de este rango solo se logró un 46.79% de acierto.
- Para ambientes de iluminación donde una parte del rostro está más iluminada que la otra, se tiene un 97.22% de acierto en la detección de giro si la relación de iluminación es menor a 10 veces la cantidad de luxes. Mientras, que si se excede esta relación el porcentaje de acierto baja a 88.89%.
- La resolución de trabajo no incide en el funcionamiento en tiempo real del sistema. Pues, con la menor resolución (480x320) se procesan 13.13 fps y con la mayor resolución (720x480) se obtienen 11.9 fps. Evidenciando que se reduce la cantidad de fotogramas procesados con el fin de tener una mejor resolución.
- La resolución de la cámara está directamente relacionada con el número de puntos que logran detectarse. Un incremento de 2.17 veces en la resolución permite detectar 2.24 veces más características. Esto favorece al desempeño del sistema, ya que con una mayor cantidad de puntos se tiene una mejor estimación de los rectángulos del rostro y nariz.
- El giro hacia el lado del rostro menos iluminado se logra interpretar correctamente el 83.89 % de las veces. Mientras que, el giro hacia el lado más iluminado se detectó adecuadamente el 100% de las veces.
- En ambientes exteriores con más de 40000 luxes de iluminación se consigue detectar 1.6 veces más características que en ambientes interiores con menos de 100 luxes. Sin embargo, estos se pierden 1.2 veces más rápido conforme se mueve el rostro a partir del fotograma 200.
- El desempeño en ambientes exteriores es crítico, ya que los rectángulos del rostro y de la nariz se distorsionan 1.46 veces más rápido que en ambientes interiores. Esto ocasiona que la silla no se mueva acorde con el movimiento de la cabeza.

- La detección del rostro en ambientes con poca o nula iluminación se demora en promedio un aproximado de 10 veces más que en condiciones de buena iluminación, esto sin usar lentes. Mientras que, con el uso de lentes se detectó el rostro solo el 10% de las veces y tomando hasta 20 segundos para la detección.
- En ambientes interiores con buena iluminación se ingresa y sale del modo de comandos el 97.6% de las veces y en ambientes con poca iluminación el 89.42%. El movimiento de la cabeza es interpretado correctamente en la interfaz un 98.75% veces y ante poca iluminación el 89.65% de las veces.

4.2 RECOMENDACIONES

- Las secciones del rostro que presentan mejores características son la boca, los ojos y nariz. Por lo tanto, se puede enfocar en obtener el rectángulo de estas facciones en lugar de todo el rostro, que tiende a abarcar zonas que no pertenecen al mismo e influirán en la estimación del rectángulo del rostro.
- Para encontrar una mayor cantidad de características, el fondo detrás del rostro puede ser implementado de color blanco o negro para tener mayor contraste con respecto al contorno del rostro.
- Los rectángulos del rostro y de la nariz presentan distorsión a mayor tiempo cuando el rostro está iluminado uniformemente, por lo que para una futura implementación se debe tener en cuenta el uso de fuente de luz que ilumine el rostro de frente en caso de ser necesario.
- Para uso en ambientes exteriores con sol donde la iluminación presenta valores en órdenes de las decenas de miles de luxes, se debe considerar la implementación de una forma de crear sombra en todo el rostro y así evitar la iluminación directa.
- Para ambientes exteriores con ausencia de sol, no se necesita crear sombra. Sin embargo, independientemente del tipo de ambiente, se debe implementar un fondo uniforme un poco más grande que el rostro para evitar seguir características que no pertenecen al rostro como árboles, muebles, edificios, etc.

5. REFERENCIAS BIBLIOGRÁFICAS

- [1] “Efecto de la fisioterapia en un paciente con cuadriplejía por ependimoma medular”. [Online]. Disponible en: http://www.scielo.org.pe/scielo.php?script=sci_arttext&pid=S1018-130X2014000300007. [Accedido: 01-ago-2021]
- [2] V. Baldassin, C. Lorenzo, y H. E. Shimizu, “Tecnología de asistencia y calidad de vida en la cuadriplejía: abordaje bioético”, *Rev. Bioét.*, vol. 26, pp. 574-586, dic. 2018, doi: 10.1590/1983-80422018264276.
- [3] “Estadísticas de Discapacidad – Consejo Nacional para la Igualdad de Discapacidades”. [Online]. Disponible en: <https://www.consejodiscapacidades.gob.ec/estadisticas-de-discapacidad/>. [Accedido: 01-ago-2021]
- [4] L. H. González Sandoval y J. Caicedo Pedrozo, “Estudio comparativo entre dos ambientes de aprendizaje de señales electroencefalográficas del parpadeo con bio-retroalimentación para el desarrollo de la destreza motriz en el control de un exoesqueleto virtual.”, *ReponameRepositorio Inst. Univ. Pedagógica Nac.*, 2018 [Online]. Disponible en: <http://repository.pedagogica.edu.co/handle/20.500.12209/10345>. [Accedido: 23-nov-2021]
- [5] “Introducción a los trastornos de la médula espinal - Enfermedades cerebrales, medulares y nerviosas”, *Manual MSD versión para público general*. [Online]. Disponible en: <https://www.msdmanuals.com/es-ec/hogar/enfermedades-cerebrales,-medulares-y-nerviosas/trastornos-de-la-m%C3%A9dula-espinal/introducci%C3%B3n-a-los-trastornos-de-la-m%C3%A9dula-espinal>. [Accedido: 01-ago-2021]
- [6] J. D. Cabrera Gómez y J. A. Sierra Ariza, “Dispositivo de seguimiento de los ojos para el control de movimiento de una silla bipedestadora (DCOMSB)”, UniPiloto, Bogotá, 2019 [Online]. Disponible en: <http://repository.unipiloto.edu.co/handle/20.500.12277/6516>. [Accedido: 01-ago-2021]
- [7] V. Sisa y E. Fernando, “Diseño e implementación de un sistema de control de una silla de ruedas haciendo uso del movimiento de cabeza, para personas con cuadriplejía”, dic. 2019 [Online]. Disponible en: <http://bibdigital.epn.edu.ec/handle/15000/20612>. [Accedido: 02-ago-2021]
- [8] P. Robalino y E. Andrea, “Diseño e implementación de un sistema de control de la linealidad de la trayectoria de una silla de ruedas manejada mediante comandos de voz, para personas cuadripléjicas”, dic. 2019 [Online]. Disponible en: <http://bibdigital.epn.edu.ec/handle/15000/20670>. [Accedido: 02-ago-2021]

- [9] F. Freire Carrera, O. Chadrina, E. Maila Andrango, y V. Drozdov, “Diseño de sistema para controlar una silla de ruedas mediante señales eléctricas cerebrales”, *MediSur*, vol. 17, n.º 5, pp. 650-663, oct. 2019.
- [10] “Un piercing magnético en la lengua para conducir la silla de ruedas de forma más rápida, cómoda e intuitiva”, *abc*, 27-nov-2013. [Online]. Disponible en: <https://www.abc.es/sociedad/20131127/abci-silla-ruedas-conducir-lengua-201311271900.html>. [Accedido: 22-nov-2021]
- [11] “GyroSet™ Vigo | The future of wheelchair head control”, *NOW technologies*. [Online]. Disponible en: <https://nowtech.hu/gyroset-vigo/>. [Accedido: 28-nov-2021]
- [12] “Self-Driving Wheelchairs Debut in Hospitals and Airports”, *IEEE Spectrum*, 17-ago-2017. [Online]. Disponible en: <https://spectrum.ieee.org/selfdriving-wheelchairs-debut-in-hospitals-and-airports>. [Accedido: 22-nov-2021]
- [13] A. Cerón, “Diseñan prototipo de silla de ruedas guiada por voz”, *México Ciencia y Tecnología*. [Online]. Disponible en: <http://www.cienciamx.com/index.php/tecnologia/tic/10795-disenan-prototipo-de-silla-de-ruedas-guiada-por-voz>. [Accedido: 22-nov-2021]
- [14] C. E. E. Tiempo, “La silla de ruedas que se puede controlar con la voz”, *El Tiempo*, 03-ago-2016. [Online]. Disponible en: <https://www.eltiempo.com/tecnosfera/novedades-tecnologia/silla-de-ruedas-controladas-con-la-voz-35151>. [Accedido: 22-nov-2021]
- [15] L. F. Manta, D. Cojocar, I. C. Vladu, A. Dragomir, y A. M. Mariniuc, “Wheelchair control by head motion using a noncontact method in relation to the patient”, en *2019 20th International Carpathian Control Conference (ICCC)*, Krakow-Wieliczka, Poland, 2019, pp. 1-6, doi: 10.1109/CarpathianCC.2019.8765982 [Online]. Disponible en: <https://ieeexplore.ieee.org/document/8765982/>. [Accedido: 28-nov-2021]
- [16] D. J. Kupetz, S. A. Wentzell, y B. F. BuSha, “Head motion controlled power wheelchair”, en *Proceedings of the 2010 IEEE 36th Annual Northeast Bioengineering Conference (NEBEC)*, New York, NY, USA, 2010, pp. 1-2, doi: 10.1109/NEBEC.2010.5458224 [Online]. Disponible en: <http://ieeexplore.ieee.org/document/5458224/>. [Accedido: 28-nov-2021]
- [17] J. S. Ju, Y. Shin, y E. Y. Kim, “Vision based interface system for hands free control of an intelligent wheelchair”, *J. NeuroEngineering Rehabil.*, vol. 6, p. 33, ago. 2009, doi: 10.1186/1743-0003-6-33.
- [18] T. Lu, “Head Gesture Recognition for Hands-free Control of an Intelligent Wheelchair”.

- [19] "Una economía digital inclusiva para las personas con discapacidad | Disability Hub". [Online]. Disponible en: <https://disabilityhub.eu/es/outcomes/una-economia-digital-inclusiva-para-las-personas-con-discapacidad>. [Accedido: 22-nov-2021]
- [20] Ecuador, "Acuerdo Ministerial, 2018-0175, de instructivo que regula el porcentaje de inclusión laboral de personas con discapacidad".
- [21] H. Asad, V. Ravi Shrimali, y N. Singh, "Basics of Image Processing", en *The Computer Vision Workshop*, 1.^a ed., Birmingham, UK: Packt Publishing Ltd., 2020, pp. 1-52.
- [22] B. Furht, E. Akar, y W. A. Andrews, *Digital Image Processing: Practical Approach*. Cham: Springer International Publishing, 2018 [Online]. Disponible en: <http://link.springer.com/10.1007/978-3-319-96634-2>. [Accedido: 22-nov-2021]
- [23] V. Tyagi, *Understanding Digital Image Processing*, 1.^a ed. Raghogarh, Guna (MP), India: CRC Press, 2018.
- [24] "double vs. uint8 input using imshow function - MATLAB Answers - MATLAB Central". [Online]. Disponible en: <https://www.mathworks.com/matlabcentral/answers/22785-double-vs-uint8-input-using-imshow-function>. [Accedido: 22-nov-2021]
- [25] A. Kumar, A. Kaur, y M. Kumar, "Face detection techniques: a review", *Artif. Intell. Rev.*, vol. 52, n.º 2, pp. 927-948, ago. 2019, doi: 10.1007/s10462-018-9650-2.
- [26] J. Brownlee, "How to Perform Face Detection with Deep Learning", *Machine Learning Mastery*, 02-jun-2019. [Online]. Disponible en: <https://machinelearningmastery.com/how-to-perform-face-detection-with-classical-and-deep-learning-methods-in-python-with-keras/>. [Accedido: 24-nov-2021]
- [27] K. Zhang, Z. Zhang, Z. Li, y Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks", *IEEE Signal Process. Lett.*, vol. 23, n.º 10, pp. 1499-1503, oct. 2016, doi: 10.1109/LSP.2016.2603342.
- [28] R. J. Mstafa y K. M. Elleithy, "A novel video steganography algorithm in the wavelet domain based on the KLT tracking algorithm and BCH codes", en *2015 Long Island Systems, Applications and Technology*, Farmingdale, NY, USA, 2015, pp. 1-7, doi: 10.1109/LISAT.2015.7160192 [Online]. Disponible en: <http://ieeexplore.ieee.org/document/7160192/>. [Accedido: 22-nov-2021]
- [29] D. Chatterjee y S. Chandran, "Comparative study of camshift and KLT algorithms for real time face detection and tracking applications", en *2016 Second International Conference on Research in Computational Intelligence and Communication Networks*

- (ICRCICN), Kolkata, India, 2016, pp. 62-65, doi: 10.1109/ICRCICN.2016.7813552 [Online]. Disponible en: <http://ieeexplore.ieee.org/document/7813552/>. [Accedido: 24-nov-2021]
- [30] K. Kadir, M. K. Kamaruddin, H. Nasir, S. I. Safie, y Z. A. K. Bakti, "A comparative study between LBP and Haar-like features for Face Detection using OpenCV", en *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*, Kuala Lumpur, Malaysia, 2014, pp. 335-339, doi: 10.1109/ICE2T.2014.7006273 [Online]. Disponible en: <http://ieeexplore.ieee.org/document/7006273/>. [Accedido: 24-nov-2021]
- [31] P. Viola y M. Jones, "Rapid object detection using a boosted cascade of simple features", en *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 2001, vol. 1, p. I-511-I-518, doi: 10.1109/CVPR.2001.990517 [Online]. Disponible en: <http://ieeexplore.ieee.org/document/990517/>. [Accedido: 22-nov-2021]
- [32] A. Brunetti, D. Buongiorno, G. F. Trotta, y V. Bevilacqua, "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey", *Neurocomputing*, vol. 300, pp. 17-33, jul. 2018, doi: 10.1016/j.neucom.2018.01.092.
- [33] T. Ojala, M. Pietikainen, y T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, n.º 7, pp. 971-987, jul. 2002, doi: 10.1109/TPAMI.2002.1017623.
- [34] "El contraste en pantallas led", *Visual Led*, 31-jul-2018. [Online]. Disponible en: <https://visualled.com/glosario/contraste-pantallas/>. [Accedido: 22-nov-2021]
- [35] J. Chang-yeon, "Face Detection using LBP features". 2008.
- [36] MathWorks, *Computer Vision System Toolbox: Getting Started Guide*. 2016.
- [37] Hongjin Zhu, Shisong Zhu, y Toshio Koga, "Face detection based on AdaBoost algorithm with differential images", en *2008 International Conference on Audio, Language and Image Processing*, Shanghai, China, 2008, pp. 718-722, doi: 10.1109/ICALIP.2008.4590254 [Online]. Disponible en: <http://ieeexplore.ieee.org/document/4590254/>. [Accedido: 22-nov-2021]
- [38] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover", ene. 1980, doi: 10.1184/R1/6557033.v1. [Online]. Disponible en: https://kithub.cmu.edu/articles/journal_contribution/Obstacle_avoidance_and_navigation_in_the_real_world_by_a_seeing_robot_rover/6557033/1. [Accedido: 22-nov-2021]
- [39] D. Marr, S. Ullman, y T. Poggio, "Bandpass channels, zero-crossings, and early visual information processing", *J. Opt. Soc. Am.*, vol. 69, n.º 6, p. 914, jun. 1979, doi: 10.1364/JOSA.69.000914.

- [40] L. Dreschler y H.-H. Nagel, "Volumetric model and 3D trajectory of a moving car derived from monocular TV frame sequences of a street scene", *Comput. Graph. Image Process.*, vol. 20, n.º 3, pp. 199-228, nov. 1982, doi: 10.1016/0146-664X(82)90081-8.
- [41] C. Tomasi y T. Kanade, "Detection and Tracking of Point Features", p. 22, 1991.
- [42] N. H. Barnouti, M. H. N. Al-Mayyahi, y S. S. M. Al-Dabbagh, "Real-Time Face Tracking and Recognition System Using Kanade-Lucas-Tomasi and Two-Dimensional Principal Component Analysis", en *2018 International Conference on Advanced Science and Engineering (ICOASE)*, Duhok, 2018, pp. 24-29, doi: 10.1109/ICOASE.2018.8548818 [Online]. Disponible en: <https://ieeexplore.ieee.org/document/8548818/>. [Accedido: 22-nov-2021]
- [43] Jianbo Shi y Tomasi, "Good features to track", en *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, Seattle, WA, USA, 1994, pp. 593-600, doi: 10.1109/CVPR.1994.323794 [Online]. Disponible en: <http://ieeexplore.ieee.org/document/323794/>. [Accedido: 22-nov-2021]
- [44] B. Lucas y T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", *Proceedings DARPA Image Understanding Workshop*, pp. 121-130, 1981.
- [45] "Slab salt and pepper granite texture seamless 02221", *Sketchuptexture - Textures*. [Online]. Disponible en: <http://www.sketchuptexturedub.com/textures/architecture/marble-slabs/granite/slab-salt-and-pepper-granite-texture-seamless-02221>. [Accedido: 22-nov-2021]
- [46] M. T. Niknejad, "Salt and pepper sign (skull) | Radiology Reference Article | Radiopaedia.org", *Radiopaedia*. [Online]. Disponible en: <https://radiopaedia.org/articles/salt-and-pepper-sign-skull-1>. [Accedido: 22-nov-2021]
- [47] P. F. U. Gotardo, O. R. P. Bellon, K. L. Boyer, y L. Silva, "Range Image Segmentation Into Planar and Quadric Surfaces Using an Improved Robust Estimator and Genetic Algorithm", *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 34, n.º 6, pp. 2303-2316, dic. 2004, doi: 10.1109/TSMCB.2004.835082.
- [48] P. H. S. Torr y A. Zisserman, "MLE-SAC: A New Robust Estimator with Application to Estimating Image Geometry", *Comput. Vis. Image Underst.*, vol. 78, n.º 1, pp. 138-156, abr. 2000, doi: 10.1006/cviu.1999.0832.
- [49] R. López Camino, "Espacios Vectoriales". Universidad de Granada, 2003.
- [50] K. Pleansamai, "M-ESTIMATOR SAMPLE CONSENSUS PLANAR EXTRACTION FROM IMAGE-BASED 3D POINT CLOUD FOR BUILDING INFORMATION MODELLING", *Int. J. GEOMATE*, vol. 17, pp. 69-76, nov. 2019, doi: 10.21660/2019.63.09667.

- [51] MathWorks, "MATLAB Support Package for USB Webcams User's Guide", p. 62, 2021.
- [52] J. Burgué, "La Cara, sus proporciones estéticas" [Online]. Disponible en: https://www.academia.edu/21685151/LA_CARA_Y_SUS_PROPORCIONES_ESTETICAS_LA_HABANA. [Accedido: 22-nov-2021]
- [53] A. S. Revelo Álava, "Implementación de un algoritmo de detección de somnolencia humana, en tiempo real basado en visión artificial", oct. 2019 [Online]. Disponible en: <http://bibdigital.epn.edu.ec/handle/15000/20522>. [Accedido: 22-nov-2021]
- [54] "Projective 2 D geometry course 2 Multiple View". [Online]. Disponible en: <https://slidetodoc.com/projective-2-d-geometry-course-2-multiple-view-2/>. [Accedido: 22-nov-2021]
- [55] "Pendiente de una recta, ejercicios resueltos | Matemóvil", 10-jul-2017. [Online]. Disponible en: <https://matemovil.com/pendiente-de-una-recta-ejercicios-resueltos/>. [Accedido: 22-nov-2021]
- [56] "Ejemplos de Tangente". [Online]. Disponible en: <https://www.matematicas10.net/2015/12/ejemplos-de-tangente.html>. [Accedido: 22-nov-2021]
- [57] "Insert shapes in image or video - MATLAB insertShape". [Online]. Disponible en: <https://www.mathworks.com/help/vision/ref/insertshape.html>. [Accedido: 22-nov-2021]

ANEXOS

ANEXO A. CÓDIGO DEL PROGRAMA PRINCIPAL PARA LA SIMULACIÓN DEL CONTROL DE LA SILLA DE RUEDAS MEDIANTE MOVIMIENTOS DE LA CABEZA.

ANEXO B. MANUAL DE USUARIO

ANEXO A: CÓDIGO DEL PROGRAMA PRINCIPAL

```
%-----***** INICIO DEL PROGRAMA*****-----%
% - - - - ### ESTUDIANTE:
% - - - - #####JHON JIRÓN
%(cont): Significa continuación de una etapa previamente planteada
% ETAPA 1:Adquisición de Imágenes
cam=webcam('AV TO USB2.0');%Creación del objeto para la cámara
cam.Resolution='480x320';%Resolución de la cámara

% ETAPA 2: Detección de Rostro
NewDetection = 1;%Bandera para volver a detectar el rostro cuando se
pierden las características seguidas
faceDetector = vision.CascadeObjectDetector('FrontalFaceLBP');%Objeto
detector de rostro
faceDetector.MergeThreshold = 11;% umbral establecido para la
detección
% ETAPA 5: Detección de la dirección de giro de la cabeza
InitMov = 1;%Bandera para habilitar la inicialización de variables de
la etapa 5 y 6
IzqDetect = 1;%Bandera para indicar un giro a la izquierda y evaluar
el movimiento para entrar o salir del modo de comandos
inMode = 1;%Bandera para habilitar la entrada/salida del sistema en
base al seguimiento de la nariz
% ETAPA 6: Obtención de Características para el movimiento
%Inicialización arbitraria de la posición (x,y) de la silla
X=0;
Y=0;
% ETAPA 7:Demostración del movimiento de la silla desde una vista
superior
%Presentación de la imagen de la silla
axes(handles.axes1)
img = imread('silla.jpg');%Se lee la imagen
set(handles.axes1,'Units','pixels');%Se configura la unidad de los
ejes
set(handles.axes1,'Position',[X Y 200 250]);%Se establece la posición
y tamaño
imshow(img);%se presenta la imagen
%Se establece un indicador para saber si está dentro o fuera del modo
de
%comandos
LedIndicator='red';
set(handles.Control,'BackgroundColor',LedIndicator);%Control es el
objeto statictext

% Etapa 1(cont):Adquisición de Imágenes
try
    while size(InterfazMovimientoSilla)>0%Se adquieren imágenes hasta
    cerrar el guide
        camFrame = snapshot(cam);% Se captura un frame
        camFrameRGB = camFrame;%Imagen en formato RGB para
presentación
        camFrame = im2gray(camFrame);%Se convierte una imagen RGB a
escala de grises
        camFrame = histeq(camFrame);% Se realiza la ecualización del
frame de histograma
```

```

%%ETAPA 2(cont): Detección de Rostro
    if NewDetection == 1
        tic
        %Pausa de tres segundos después del giro o para una nueva
detección hasta volver a la posición inicial
        while toc < 3
            camFrameRGB = snapshot(cam); % Se captura un frame
            axes(handles.axes2)
            imshow(camFrameRGB); %Se presenta el frame con los
rectángulos y los puntos
        end
        toc

        % Extracción de los 4 puntos de la caja del rostro
        face = faceDetector(camFrame); % Se almacenan los
rectángulos de los rostros detectados en camFrame
        if isempty(face) %Si no se detecta un rostro
            axes(handles.axes2) %Se especifica el objeto axes donde
se presenta la imagen
            imshow(camFrameRGB); % Se presenta el frame
            continue
        else
            %face(:,3): El ancho de todos los rostros detectados
            [M,k]=max(face(:,3)); %Se determina la fila k del ancho
más grande M.
            face = face(k, :); % Selección del rectángulo del
rostro más prominente
            NumFrame=1; %Se establece el primer frame con rostro
detectado como el primero
            NewDetection = -1; %Se cambia el valor de la Bandera de
detección
        end
    end
    %%FIN ETAPA 2

    if NumFrame == 1 % Primer frame con rostro detectado
        %%ETAPA 3: Obtención del rectángulo de la nariz
        %face=[X0 Y0 w0 h0]
        nose = [face(1)+2*face(3)/5 face(2)+face(4)/3 face(3)/5
face(4)/3]; %rectángulo de la nariz
        %X0: Posición del eje x donde empieza el rostro de derecha
a izquierda
        %Y0: Posición del eje y en pixeles donde empieza el rostro
de arriba hacia abajo.
        %w0: Ancho del Rostro
        %h0: Altura del Rostro
        %%FIN ETAPA 3

        % ETAPA 4: Seguimiento Rostro
        %%ETAPA 4 - Fase 1: Inicializando el objeto seguidor del
rostro y nariz
        %Inicializando el objeto seguidor de la nariznción de
inicialización del objeto seguidor
        [faceTracker, faceBox, oldFacePoints] =
TrackerInit(camFrame,face);
        %faceBox: Coordenadas (x,y) de los 4 vértices del
rectángulo de la cara

```

```

        %faceTracker: Objeto seguidor del rostro
        %oldFacePoints: Puntos para el seguimiento del rostro en
el siguiente Frame
        [noseTracker, noseBox, oldNosePoints] =
TrackerInit(camFrame,nose);
        %noseBox: Coordenadas (x,y) de los 4 vértices del
rectángulo de la nariz
        %noseTracker: Objeto seguidor de la nariz
        %oldNosePoints: Puntos para el seguimiento de la nariz en
el siguiente Frame
        %%FIN ETAPA 4 - Fase 1

        if InitMov == 1%En base a la detección inicial cuando
arranca el programa
        % ETAPA 5 (cont): Detección de la dirección de giro de la
cabeza
        %%ETAPA 5 - Fase 1: Inicialización de puntos de referencia
de la nariz
        %noseBox: rectángulo de la nariz de la etapa 4, fase 1
        %Puntos de referencia de la nariz obtenidos en el primer
Frame
        Pref1 = noseBox(1,1);% Coordenada x del Vértice superior
izquierdo de noseBox
        Pref2 = noseBox(4,1);% Coordenada x del Vértice inferior
izquierdo de noseBox
        AnchoNariz = nose(3);% Ancho de noseBox
        %%FIN ETAPA 5 - Fase 1

        %ETAPA 6 (cont): Obtención de Características para el
movimiento
        %%ETAPA 6 - Fase 1: Inicialización de características para
movimiento
        %Posiciones en el eje 'Y' iniciales
        Y1 = faceBox(1,2);%Coordenada 'y' inicial del Vértice
superior izquierdo del rostro
        Y2 = faceBox(4,2);%Coordenada 'y' inicial del Vértice
inferior izquierdo del rostro
        h0 = Y2-Y1;%Altura inicial del rostro
        %%FIN ETAPA 6 - Fase 1
        end

        else % Sigüientes Frames después de la detección del Rostro
        %%ETAPA 4 - Fase 2 y Fase 3: Función de Seguimiento del
rostro y nariz
        [oldFacePoints, faceBox, NewDetection] =
Tracker(camFrame,faceTracker,oldFacePoints,faceBox);
        [oldNosePoints, noseBox, NewDetection] =
Tracker(camFrame,noseTracker,oldNosePoints,noseBox);
        %Se valida que hayan al menos 2 puntos para seguimiento
        if NewDetection == 1
            InitMov = -1;%No se inicializa de nuevo las variables
de movimiento
            continue%Se vuelve a detectar el rostro en el
siguiente frame
        end
        %%FIN ETAPA 4 - Fase 2

```



```

%%ETAPA 5 - Fase 2: Obtención de la dirección de giro

    %noseBox: rectángulo de la nariz de la etapa 4, fase 2
    %Puntos movibles de la nariz en los siguientes Frames

    PMov1 = noseBox(1,1);% Coordenada x del Vértice superior
    izquierdo de noseBox

    PMov2 = noseBox(4,1);% Coordenada x del Vértice inferior
    izquierdo de noseBox

    %Si los puntos movibles se mueven el doble del ancho de
    noseBox desde los puntos
    %de referencia hacia la izquierda se determina como giro
    hacia ese lado

    if PMov1 <= Pref1 - 1.5*AnchoNariz && PMov2 <= Pref2 -
    1.5*AnchoNariz
        IzqDetect = - 1;%Primer paso para entrar/salir del
        modo de comandos
        tic %se limita el tiempo para cumplir el siguiente
        paso entrar/salir del modo de comandos
        end

    if IzqDetect == -1 && toc < 3
        %Puntos movibles se mueven el doble del ancho de
        noseBox hacia
        %la derecha -> giro hacia la derecha
        if PMov1 >= Pref1 + 1.5*AnchoNariz && PMov2 >= Pref2 +
        1.5*AnchoNariz
            inMode = inMode*-1;%Bandera para entrar/salir del
            sistema
            NewDetection = 1;%Bandera para empezar una nueva
            detección
            InitMov = -1;%No se inicializa de nuevo las
            variables de movimiento
            IzqDetect = 1;%Valor de la bandera al inicial
            continue%Pasa al siguiente Frame para detección
            del rostro
        end
    else
        IzqDetect = 1;
    End

    %%FIN ETAPA 5 - Fase 2

```

```

%%ETAPA 6 - Fase 2: Determinación del movimiento de la silla en base
al movimiento del rostro
    % Se obtiene el ángulo de rotación para el movimiento del eje X
    X1 = faceBox(1,1);%Coordenada en x del Vértice superior
izquierdo del rectángulo del rostro
    Y1 = faceBox(1,2);%Coordenada en y del Vértice superior
izquierdo del rectángulo del rostro
    X2 = faceBox(4,1);%Coordenada en x del Vértice inferior
izquierdo del rectángulo del rostro
    Y2 = faceBox(4,2);%Coordenada en y del Vértice inferior
izquierdo del rectángulo del rostro
    m = (Y2-Y1)/(X2-X1);%Pendiente de la recta del lado
izquierdo de faceBox
    Ang = atand(m);%Conversión a grados
    % Se obtiene la altura actual del rostro para el
movimiento del eje Y
    h = Y2-Y1;%Y1:Coordenada 'y' actual del Vértice superior
izquierdo del rostro
                                %Y2:Coordenada 'y' actual del Vértice
inferior izquierdo del rostro

    % Se valida la variable inMode determinada según la
dirección del giro de la cabeza
    if inMode == -1
        LedIndicator='green';%Indicador de estado ON del
sistema
        if IzqDetect == 1%Si existe un giro no se considera el
movimiento
            %Movimiento en el eje Y
            if h < h0*0.9%Hacia atrás si se reduce al 90% de
la altura inicial
                %Se suma la diferencia de alturas en forma de pixeles
                Y = Y + floor(h0*0.9-h);
            elseif h > h0*1.1%Hacia adelante si la altura
aumenta 10%
                %Se resta la diferencia de alturas en forma de pixeles
                Y = Y - floor(h-h0*1.1);
            else
                Y = Y;%Reposo en el eje Y
                if toc>30
                    InitMov = -1;
                    NewDetection=1;
                    tic
                end
            end
            %Movimiento en el eje X
            if Ang < 80 && Ang > 0%Ángulo entre 0 y 80 grados
se mueve hacia la izquierda
                %Se resta la diferencia de ángulos en forma de pixeles
                X=X-floor(80-Ang);
            elseif Ang > -80 && Ang < 0%Ángulo entre 0 y -80
grados se mueve hacia la derecha
                %Se suma la diferencia de ángulos en forma de pixeles
                X=X+floor(Ang-(-80));
            else
                X = X;%Reposo en el eje X
            end
        end
    end
end

```

```

%Establecimiento de límites de movimiento
    %Eje X
    if X < 0%Mínima posición 0 en cuanto a pixeles
        X=0;
    elseif X > 840%Posición máxima a 840 en cuanto a
pixeles
        X=840;%Para ocupar todo el lugar establecido para
desplazamiento en la interfaz eje 'Y'
    end
    %Eje Y
    if Y < 0%Mínima posición 0 en cuanto a pixeles
        Y=0;
    elseif Y > 395%Posición máxima a 395 en cuanto a
pixeles
        Y=395;%Para ocupar todo el lugar establecido para
desplazamiento en la interfaz eje 'X'
    End
    else
        LedIndicator='red';%Indica que se ha salido del modo
de comandos
    end
    %%FIN ETAPA 6 - Fase 2

    %%ETAPA 7 (cont):Demostración del movimiento de la silla
desde una vista superior
    %Presentación de los rectángulos del rostro y nariz
    facePolygon = reshape(faceBox', 1, []);%Vectorización del
rectángulo del rostro
    nosePolygon = reshape(noseBox', 1, []);%Vectorización del
rectángulo de la nariz
    camFrameRGB = insertShape(camFrameRGB, 'Polygon',
facePolygon, 'LineWidth', 4);
    camFrameRGB = insertShape(camFrameRGB, 'Polygon',
nosePolygon, 'LineWidth', 4);
    %Presentación de los puntos de seguimiento
    camFrameRGB = insertMarker(camFrameRGB, oldFacePoints,
'+', 'Color', 'white');
    camFrameRGB = insertMarker(camFrameRGB, oldNosePoints,
'+', 'Color', 'red');
    axes(handles.axes2)
    imshow(camFrameRGB);%Se presenta el frame con los
rectángulos y los puntos
    set(handles.axes1, 'Position', [X Y 200 250]);%Se configura
la posición de la silla

    %Se establece un indicador para saber si está dentro o
fuera del modo de
    %comandos

set(handles.Control, 'BackgroundColor', LedIndicator);%Control es el
objeto statictext

    %%FIN ETAPA 7
    end
    NumFrame = NumFrame + 1;%Se actualiza el número de Frame
end
catch
end

```

ANEXO B: MANUAL DE USUARIO

ESCUELA POLITÉCNICA NACIONAL TRABAJO DE TITULACIÓN

Autor: Jhon Jirón

Director: PhD. Robin Álvarez

Tema:

SIMULACIÓN DE UN SISTEMA DE CONTROL DE UNA SILLA DE RUEDAS BASADO EN VISIÓN ARTIFICIAL MEDIANTE MOVIMIENTOS DE CABEZA, PARA PERSONAS CUADRIPLÉJICAS

MANUAL DE USUARIO

El proceso que un nuevo usuario necesita para poder realizar la ejecución la simulación del sistema de control de la silla de ruedas se detalla de manera resumida en este manual de usuario. Así como la forma en que se controla el sistema, elementos a instalar en Matlab, los recursos necesarios y una alternativa en caso de que no se disponga de la cámara especificada.

RECURSOS UTILIZADOS

- 1) Cámara de visión nocturna de retroceso de vehículos con conectores RCA.
- 2) Adaptador RCA macho-macho.
- 3) Batería de 12V DC.
- 4) Capturadora de video EasyCap USB 2.0.
- 5) Laptop con el software Matlab instalado.

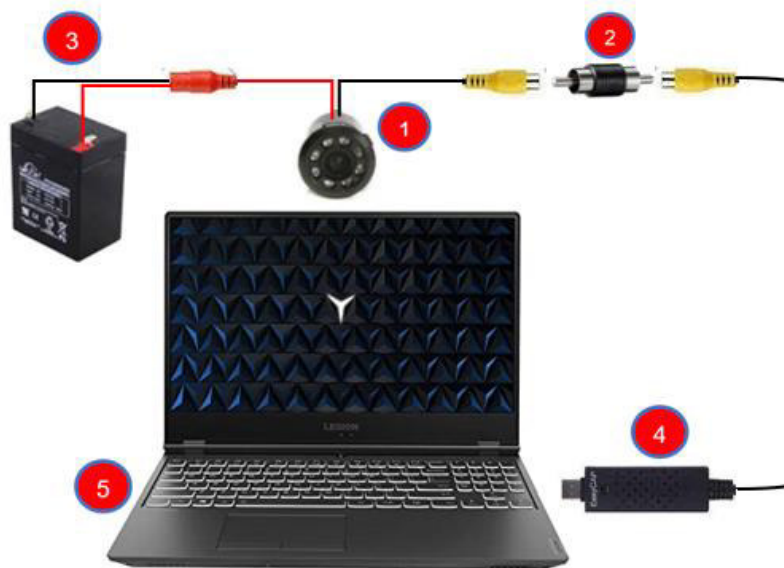


Figura A. 1. Diagrama de los recursos usados con sus respectivas conexiones

INSTALACIÓN DE PAQUETES Y TOOLBOX NECESARIOS

1. Tener abierto Matlab y en la pestaña *Home* desplegar las opciones de *Add-Ons* y dar clic en *Get Add-Ons*.

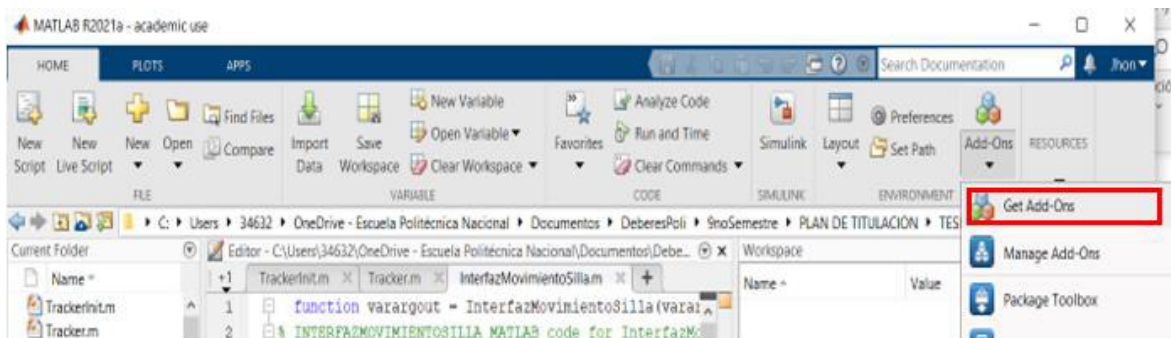


Figura A. 2. Menú para la instalación de paquetes y toolbox de Matlab

2. Buscar e instalar los siguientes elementos: Matlab support Package for USB Webcams, Image Processing Toolbox y Computer Vision System Toolbox.

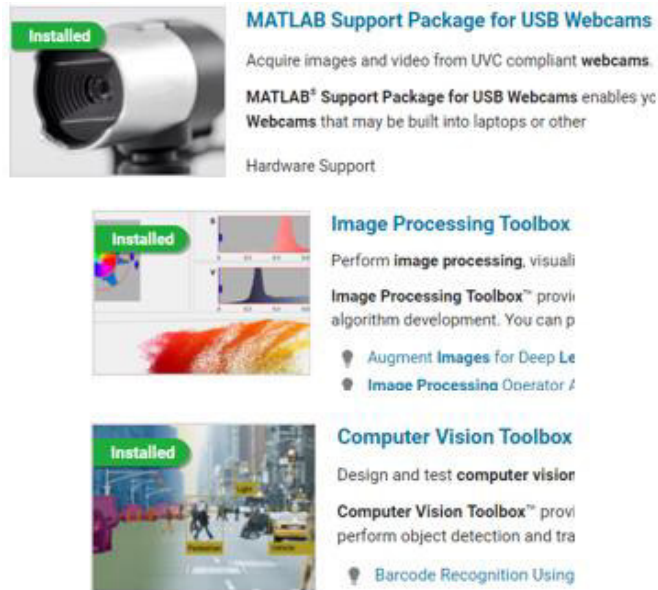


Figura A. 3. Paquetes y toolbox a instalar

ELECCIÓN DE LA CÁMARA Y RESOLUCIÓN

Si la cámara usada es una cámara de visión nocturna para retroceso de vehículos, la conexión será como se muestra en la Figura A. 1 siguiendo los pasos:

1. La cámara de visión nocturna se conecta mediante su terminal de salida rojo a la batería de 12 V DC y mediante su terminal RCA (amarillo) para video a un extremo del adaptador RCA macho-macho.
2. El otro extremo del adaptador se conecta el terminal RCA hembra (amarillo) del capturador EasyCap.
3. El capturador se conecta mediante un puerto USB al computador.

Los pasos mencionados a continuación se pueden omitir si se llevó a cabo la conexión de la cámara de visión nocturna satisfactoriamente y dicha cámara tiene una resolución disponible de '640x480'. Si no se dispone de lo mencionado anteriormente, se puede hacer uso de otra cámara ya sea de visión nocturna o no. Entonces, para determinar qué cámaras y con qué resolución están disponibles se tienen los siguientes pasos:

1. Escribir el comando `webcamlist` en la ventana de comandos de Matlab, desplegándose una lista con los nombres de las cámaras que están disponibles como se muestra en la Figura A. 4.

```
2x1 cell array

{'Integrated Camera'}
{'AV TO USB2.0' }
```

Figura A. 4. Ejemplo de cámara disponibles en un computador

2. Elegir una de las cámaras disponibles y copiar el nombre incluyendo las comillas simples.
3. Escribir el comando `webcam(CamaraName).AvailableResolutions` reemplazando `CamaraName` por el nombre copiado de la cámara elegida en el paso anterior, desplegándose las resoluciones que pueden ser usadas como se muestra en el ejemplo de la Figura A. 5.

```
>> webcam('Integrated Camera').AvailableResolutions

ans =

1x9 cell array

Columns 1 through 5

    {'1280x720'}    {'320x180'}    {'320x240'}    {'352x288'}    {'424x240'}

Columns 6 through 9

    {'640x360'}    {'640x480'}    {'848x480'}    {'960x540'}
```

Figura A. 5. Resoluciones disponibles en la cámara integrada del computador

4. Se recomienda usar una resolución igual o menor a '640x480', pero si los recursos de su computador lo permiten puede escoger un valor mayor.

PREPARACIÓN DE LAS CONDICIONES DEL LUGAR DE EJECUCIÓN

Para una ejecución óptima del software a simular se debe:

- Tener una iluminación uniforme en los dos lados del rostro, de preferencia que la fuente de la luz ilumine al rostro de frente.
- Posicionar la cámara con un ángulo de enfoque de la cámara entre 25° y 45 ° apuntando al rostro como en la Figura A. 6.

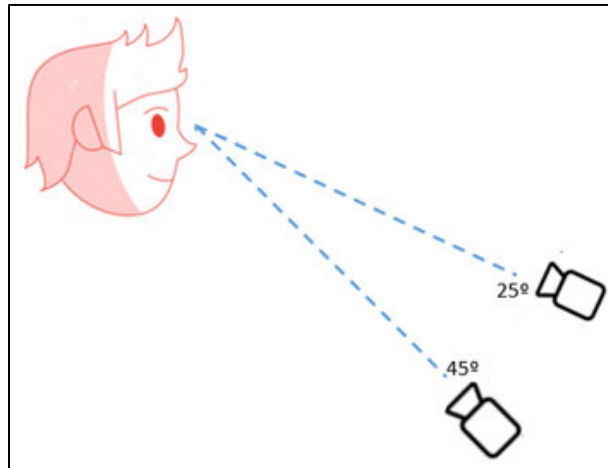


Figura A. 6. Rango de posiciones ideales de la cámara

- Estar de frente hacia el lente de la cámara con un fondo de color uniforme manteniendo la cabeza recta como en la Figura A. 7 al momento de iniciar el programa sin salir de plano de enfoque de la cámara.

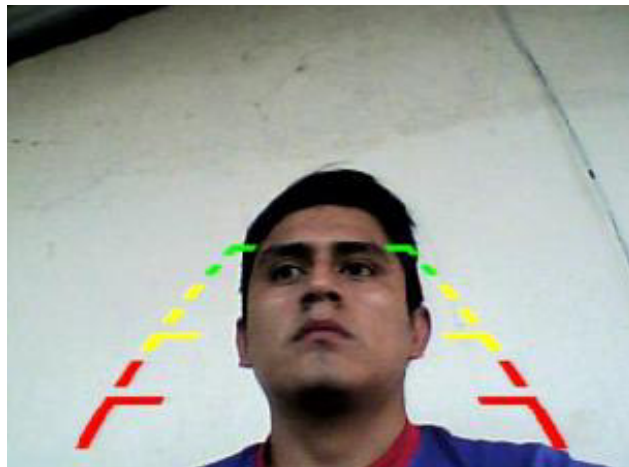


Figura A. 7. Posición ideal de la cabeza al iniciar el programa

EJECUCIÓN DEL SCRIPT

Los scripts que se deben ejecutar se encuentran en la carpeta SoftwareMovSilla, por lo que al descargar esta carpeta los archivos que se tendrán en el mismo directorio son:

- InterfazMovimientoSilla.m
- InterfazMovimientoSilla.fig
- TrackerInit.m
- Tracker.m

Para la ejecución del programa:

1. Se abre el archivo *InterfazMovimientoSilla.m* de la carpeta *SoftwareMovSilla*.
2. Se modifica la siguiente sección del código del script *InterfazMovimientoSilla.m* acorde con el nombre de la cámara elegida (línea 86) y su resolución (línea 87). Este paso es necesario solo en caso de que no se disponga de los recursos para hacer la conexión de la cámara de retroceso para vehículos.

```
85 % ETAPA 1:Adquisición de Imágenes
86 - cam=webcam('AV TO USB2.0');%Creación del objeto para la cámara
87 - cam.Resolution='640x480';%Resolución de la cámara
```

Figura A. 8. Líneas de código a modificar si no se cuenta con los recursos necesarios para conectar la cámara de visión nocturna

3. Se ejecuta el archivo abierto en el paso 1 con el editor de Matlab.
4. Finalmente se da clic en el botón ON de la interfaz gráfica.

CONTROL DE LA IMAGEN DE LA SILLA

El programa empieza mostrando la imagen capturada por la cámara elegida, la imagen de la silla en la posición inferior izquierda del rectángulo azul (zona de movimiento de la silla) y un cuadro de color rojo lo que indica que se inicia fuera del modo de comandos del sistema de control. Cuando se realice la detección del rostro después de al menos 3 segundos ejecutado el programa se muestran los rectángulos de color amarillo y las características del rostro (azul) y de la nariz (rojo).

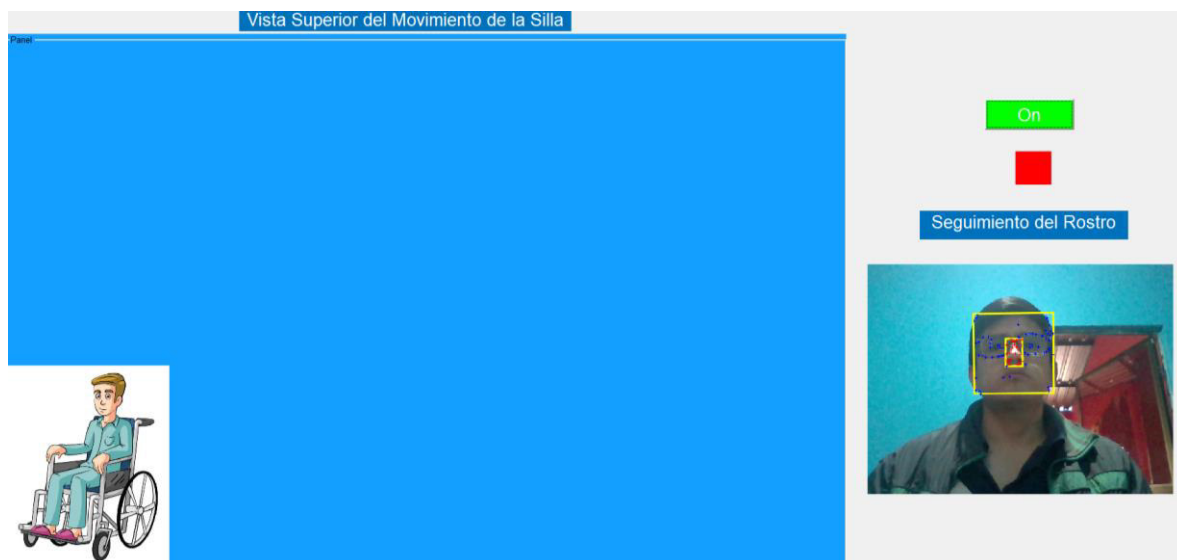


Figura A. 9. Ejemplo del programa luego de haberse detectado el rostro.

Si realiza un movimiento de la cabeza, este no se verá reflejado en la silla, por lo que, para ingresar o salir del modo de comandos, se debe hacer un giro de la cabeza que se muestre en la interfaz hacia el lado izquierdo y luego inmediatamente un giro hacia el lado contrario (derecho). Para este movimiento específico (posición inicial -> rotación a la izquierda -> rotación a la derecha) se dispone de un tiempo máximo de 3 segundos. Si se excede este tiempo, no cuenta como rotación de izquierda a derecha inmediata y la entrada/salida del modo de comandos no se ve afectada. Con esto se busca evitar que accidentalmente se ingrese al modo de comandos y la silla se mueva sin conocimiento del usuario.

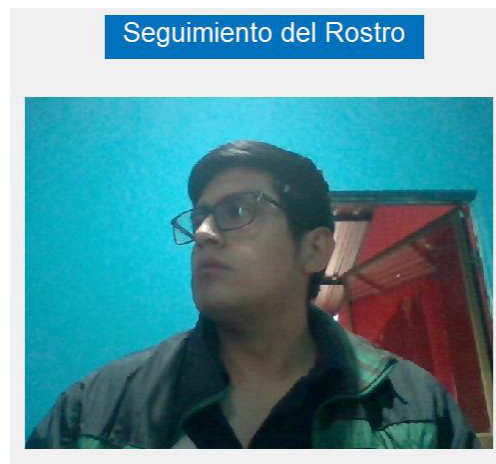


Figura A. 10. Ejemplo de giro hacia el lado izquierdo de la interfaz

Una vez dentro del modo de comandos, el cuadro rojo cambia a un color verde indicando que se ha ingresado correctamente.



Figura A. 11. Ingreso al modo de comandos

Los movimientos disponibles para manejar la silla se muestran en la Figura A. 12. Es importante recalcar que los movimientos de la cabeza no deben ser rápidos, sino el sistema no logra seguirlos.



Figura A. 12. Movimientos de la cabeza disponibles para el modo de comandos

Así, si en la interfaz se captura una inclinación hacia la izquierda o hacia la derecha, la silla se moverá hacia ese lado; en cambio, si el usuario inclina la cabeza hacia adelante o atrás, la imagen de la silla se moverá hacia abajo o hacia arriba ya que en la interfaz se muestra una vista superior del movimiento.

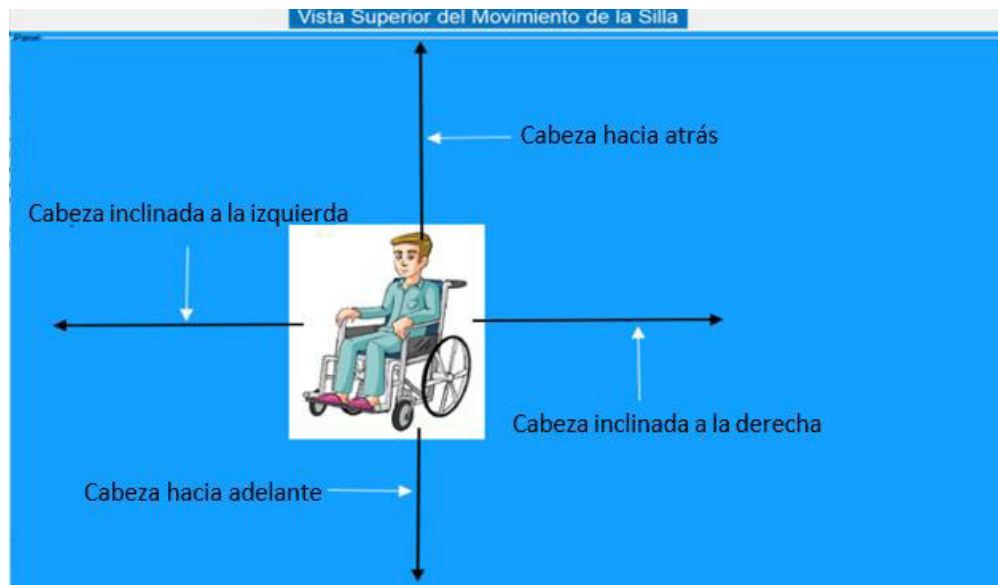


Figura A. 13. Movimientos de la silla en la interfaz gráfica

ORDEN DE EMPASTADO