ANEXOS

1. **Descripción del funcionamiento de la aplicación**
2. **Funciones y procedimientos desarrollados.**

LISTA DE IMÁGENES.

Imagen A1.1.Enfoque de la cámara. 76
Imagen A1.2.Instalador de la aplicación Eye Tracking. 77
Imagen A1.3. Ventana de inicio de la instalación de la aplicación. 77
Imagen A1.4. Ventana de especificación de carpeta de destino. 78
Imagen A1.5. Ventana final de la instalación. 78
Imagen A1.6. Acceso directo a la aplicación Eye Tracking. 79
Imagen A1.7. Ventana de inicio de la aplicación. 79
Imagen A1.8. Indicador de búsqueda de las pupilas. 80
Imagen A1.9. Botón de salida de la aplicación. 80
Imagen A1.10. Ventana de la aplicación indicando la detección de las pupilas. 80

LISTA DE ALGORITMOS.

Algoritmo 2.1. Función Principal 82
Algoritmo 2.2. Crear Estructura 84
Algoritmo 2.3. Generar datos 84
Algoritmo 2.4. Crear Histograma 85
Algoritmo 2.5. Cargar Modelo 85
Algoritmo 2.6. Calcular Histograma 86
Algoritmo 2.7. Calcular Tamaño 86
Algoritmo 2.8. Establecer Restricción 86
Algoritmo 2.9. Calcular Peso 87
Algoritmo 2.10. Ordenar 87
Algoritmo 2.11. Referencia de Búsqueda 88
Algoritmo 2.12. Forma Circular 89
Algoritmo 2.13. Búsqueda Dispersa 90
Algoritmo 2.14. K-Medias 91
Algoritmo 2.15. Crear Ruleta 93
Algoritmo 2.16. Tirar Ruleta y mover 94
Algoritmo 2.17. Posicionamiento 95
Algoritmo 2.18. Ángulo 97

1. DESCRIPCIÓN DEL FUNCIONAMIENTO DE LA APLICACIÓN

En este apéndice se presenta una especificación del funcionamiento de la aplicación desarrollada, y las especificaciones técnicas de los elementos utilizados.

1.1 **Instalación de los componentes hardware.**

La cámara utilizada es una webcam Omega 3623k3 8.0 M con sensor Cmos, presenta una resolución de video más alta de 1600 x 1200@15fps o 640 x 480@30fps, 24 bits de color verdadero. Los requerimientos mínimos del sistema para el uso de esta cámara son; procesador Pentium 1G o superior, 500 MB de espacio disponible en disco, 256 MB RAM o superior, Puerto USB. El driver del dispositivo viene con el producto o es posible adquirirlo en línea. La cámara se conecta en el ordenador a utilizar y se instala el controlador del dispositivo, una vez instalado, la aplicación hace uso del sensor para ejecutar el procesamiento de las imágenes. Esta cámara cuenta con un lente modelo L0550M de tipo vari-focal de 1/3", presenta un milimetraje de 5 – 50 mm con iris manual. La posición de la cámara es vital para el buen desempeño de la aplicación, esta debe ubicarse en la parte central inferior de la pantalla y enfocar el rostro del usuario. El enfoque debe realizarse como lo muestra la Imagen A1.1, para ello es posible configurarlo manualmente.

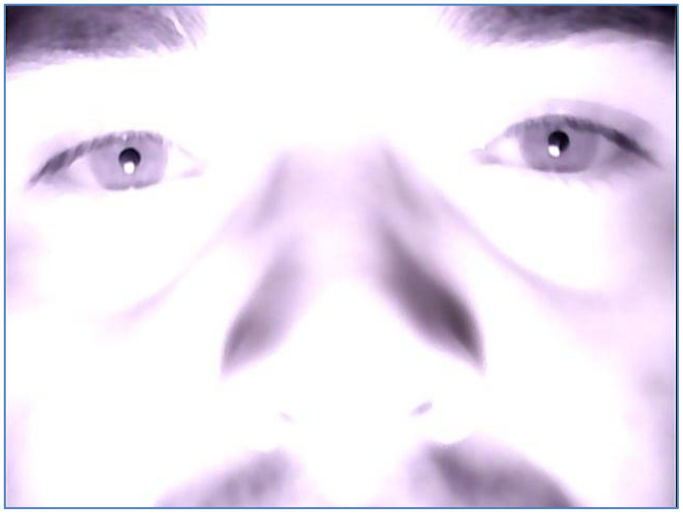

Imagen A1.1.Enfoque de la cámara.

La fuente emisora de luz infrarroja es un dispositivo de 21 LEDs de 5mm configurados en forma circular, con una longitud de onda 840 nm, se alimenta de una fuente de 12 V AC a 1 A. Esta fuente emisora se ubica en la misma línea de visión que la cámara pero sobre ella a una distancia aproximadamente de 20 cm, los haces de luz deben iluminar completamente el rostro del usuario resaltando completamente las pupilas.

1.2 **Limitaciones de movimiento.**

La aplicación implementada presenta una serie de limitaciones de movimiento que el usuario debe cumplir si desea un buen funcionamiento. El más importante de ellos restringe la movilidad de la cabeza, una vez iniciada la aplicación, los movimientos de la cabeza deben ser casi nulos, debido a que un leve desplazamiento en el rostro es posible que se vea reflejado en el posicionamiento del mouse, aunque es complicado mantener estática la cabeza, además de ser algo no natural, es posible realizar lapsos de descanso sacando el rostro de foco y luego volviendo a él, la aplicación sigue funcionando así no encuentre los objetivos. Otra importante limitación de movimiento se presenta en el movimiento de las pupilas, es decir, el usuario debe dirigir la mirada solamente dentro de la pantalla para obtener buenos resultados, de lo contrario es muy probable que el movimiento del cursor no sea el adecuado. Por último, la distancia entre la cámara y el rostro del usuario debe ser de aproximadamente 60 cm y en lo posible mantenerse así durante toda la ejecución de la aplicación.

1.3 **Funcionamiento de la aplicación.**

La aplicación implementada se desarrolló en lenguaje de programación C++ utilizando el entorno de desarrollo integrado (IDE por sus siglas en inglés) Dev-C++, esta aplicación fue desarrollada para ejecutarse en el sistema operativo Windows 7 en su plataforma de 64 bits, mediante la instalación de su archivo ejecutable.

Para instalar el programa de seguimiento ocular *Eye Tracking* en un ordenador, se debe realizar el siguiente procedimiento:

1. Verificar que el equipo cuenta con los requisitos mínimos de funcionamiento que se mencionaron en el apartado 1.1 de esta sección. Dependiendo de las prestaciones de la máquina, la aplicación se verá afectada en la ejecución.

2. Contar con el archivo instalador de la aplicación ***EyeTrackingSetup.exe*** en el quipo, y dar doble clic, ver Imagen A1.2.

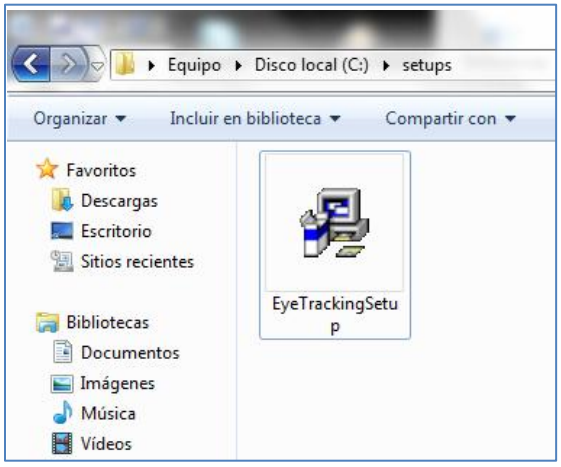

Imagen A1.2.Instalador de la aplicación Eye Tracking.

3. Al dar doble clic sobre el archivo, aparecerá la ventana inicial de instalación del programa (ver Imagen A1.3), mostrando información sobre el procedimiento que se va a realizar. Para continuar con la instalación, se debe dar clic en el botón ***Siguiente>.***

Imagen A1.3. Ventana de inicio de la instalación de la aplicación.

4. Posteriormente debe indicarse la carpeta de destino donde se instalará la aplicación, por defecto la ruta establecida es en el disco local C, archivos de programa y se crea

una carpeta llamada Eyetracking, ***C:\Program Files\Eyetracking***, como lo muestra la Imagen A1.4.

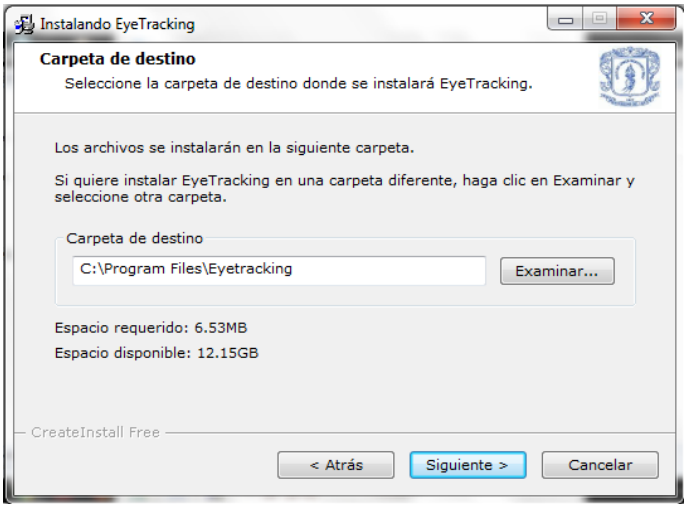

Imagen A1.4. Ventana de especificación de carpeta de destino.

Para cambiar la ruta de instalación debe oprimirse el botón ***Examinar*** e indicar la carpeta de destino donde se desea instalar la aplicación. Para continuar con la instalación debe oprimirse el botón ***Siguiente***>.

5. Una vez oprimido el botón ***Siguiente***>, la aplicación se instalará en la ruta fijada y aparecerá una ventana final confirmado que la instalación se realizó de forma correcta, como lo muestra la Imagen A1.5.

Imagen A1.5. Ventana final de la instalación.

Para terminar el proceso de instalación, debe oprimirse el botón ***Finalizar***.

Una vez instalado el programa, en el escritorio se creará un ícono de acceso directo a la aplicación Eye Tracking que presenta un diseño como se muestra en la Imagen A1.6.

Imagen A1.6. Acceso directo a la aplicación Eye Tracking.

Al dar doble clic sobre este ícono, se ejecuta la aplicación de seguimiento visual, apareciendo en la pantalla una ventana (Imagen A.17) que muestra las imágenes capturadas por la cámara del sistema y un menú básico de funcionamiento.

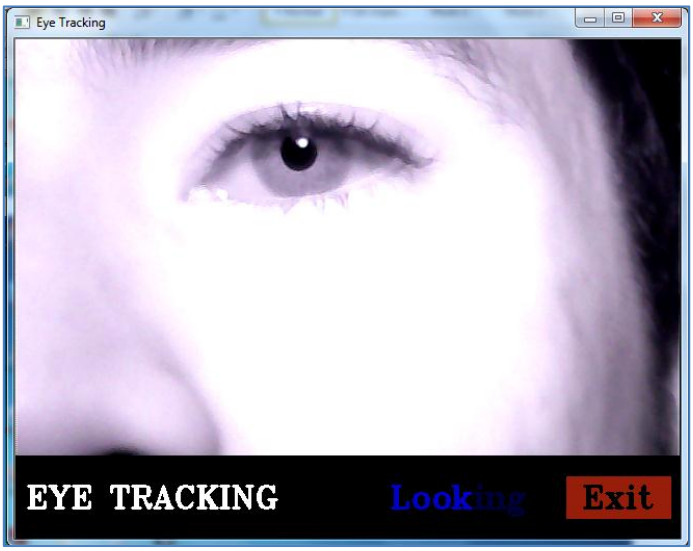

Imagen A1.7. Ventana de inicio de la aplicación.

La ventana cumple el propósito de mostrar al usuario la escena censada por la cámara, dejando ver si el posicionamiento de la fuente de luz infrarroja es el correcto y si el enfoque de la cámara es el adecuado, permitiendo ubicar el rostro de la mejor manera para realizar el seguimiento. En la parte inferior de la ventana se ha colocado un indicador de

búsqueda con la palabra ***Looking*** en letras titilantes azules (Ver Imagen A1.8), el cual se mantiene activo mientras la aplicación encuentra las pupilas.

Imagen A1.8. Indicador de búsqueda de las pupilas.

También dentro de la ventana de la aplicación, se ubica en la parte inferior el botón rojo de salida, ***Exit***, (ver Imagen A1.9), el cual al ser oprimido termina la ejecución y cierra la ventana liberando el mouse del ordenador.

Imagen A1.9. Botón de salida de la aplicación.

Una vez enfocado el rostro del usuario, como se sugiere en la Imagen A1.1, las pupilas se resaltarán con dos círculos, uno de color rojo para la pupila derecha y otro de color azul para la pupila izquierda como los muestra la Imagen A1.10.

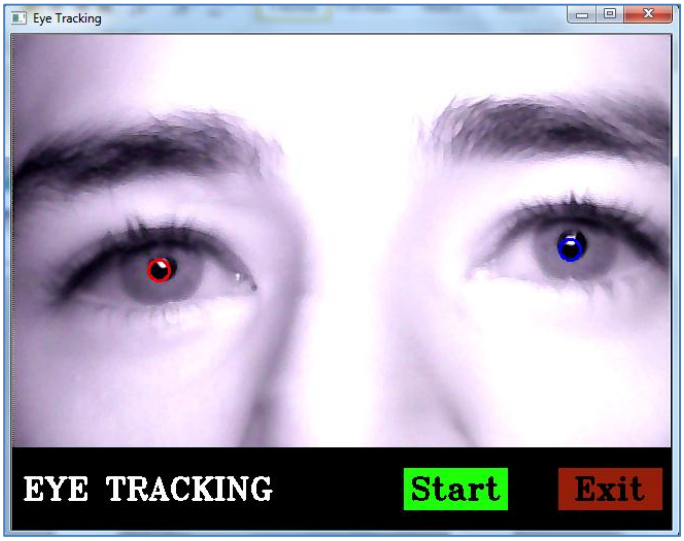

Imagen A1.10. Ventana de la aplicación indicando la detección de las pupilas.

Una vez encontradas las pupilas del usuario, se activa el botón de inicio de la aplicación en la parte inferior resaltado con color verde. Al oprimir este botón, se cierra inmediatamente

la ventada de inicio y el movimiento del cursor empieza a observarse, este movimiento se realiza como normalmente funciona un mouse, pero el programa se diseñó para realizar solo una acción, dar clic, esta acción se ejecuta cuando el usuario fija la mirada en el punto de interés por más de 3 segundos. Una vez el programa realiza un clic, el cursor cesa sus movimientos y espera a que el usuario cierre los ojos por más de un segundo, indicando que está listo para realizar una nueva acción. Para terminar con la aplicación, el usuario debe salir del foco de la cámara, momento en el cual aparece de nuevo la ventana de inicio, y con el ratón del ordenador dirigirse al botón ***Exit*** y oprimirlo.

Para desinstalar el programa, es necesario dirigirse a la carpeta de instalación de la aplicación ***Eyetracking***, en la cual se encuentra el archivo ***uninstall.exe***, al ejecutar este archivo se elimina todo el contenido de la carpeta y el acceso directo.

2. Funciones y procedimientos desarrollados.

La implementación del filtro de partículas se ha dividido en funciones que representan las etapas principales del mismo, así como también procedimientos auxiliares que permiten el funcionamiento del algoritmo, a continuación se presenta el pseudocódigo de las funciones y procedimientos implementados, en primer lugar la función principal, para después detallar cada uno de las funciones y procedimientos contenidos en ella.

Función Principal

En la función principal (Algoritmo 2.1) se presentan todas las funciones y procedimientos implementados para ejecutar el filtro de partículas híbrido, la secuencia de etapas que se ilustró en la figura 3.5 del capítulo 3 se refleja en el siguiente algoritmo. Dentro del diseño del filtro se estableció ejecutar el algoritmo inteligente cada 5 iteraciones, con el fin de optimizar el tiempo de ejecución sin alterar los resultados de precisión, también se estableció una búsqueda ciega de 5 iteraciones, es decir, si durante 5 iteraciones consecutivas no se encuentran partículas de referencia o si no se encuentran formas circulares, se reinicia el filtro, generando partículas por todo el espacio de posibles estados.

Algoritmo 2.1. Función Principal

```
Función MAIN()

//V_in ← Imagen capturada por la cámara
//N_Part ← Número de partículas
//band ← Variable booleana para indicar si hay seguimiento
//Bandera  ← Variable booleana para indicar si hay dos referencias correspondientes a
//los objetivos
//niter ← Variable que indica cada cuanto se ejecuta el algoritmo híbrido
//Total ← Numero de Círculos encontrados en CIRCULO()

Crear_Estructura
(ancho, alto) ← TAMA()
h1 ← CREATEHIST()
MODELO(h1)
niter = 5

V_in ← Captura_imagen_cámara
band = falso
Mientras verdadero Hacer
  iter ++
  bandera = falso
```

```
  Si band = falso Entonces
    N_Part = 1000
  Fin Si
  Si band = verdadero Entonces
    N_Part = 600
  Fin Si
  band = verdadero ← GENERAR(V_in, ancho, alto)
  ORDENAR()
  Bandera, contador ← REFE(N_part)
  Si Bandera = verdadero y (iter = 1 o iter = Múltiplo de niter) Entonces
    Total ← CIRCULO()
    Si Total = 2 Entonces
      BDIS(V_in)
    Sino Entonces
      Cont ++
    Fin Si

  Fin Si
  Si contador > 5 o Cont > 5 Entonces
    Bandera = falso
  Sino Entonces
  RULETA()
  Fin Si
  POS()
 Fin Mientras
 Fin Función
```

Crear estructura.

En primer lugar se define el número de partículas con el fin de determinar un espacio en memoria que contenga la información que las caracteriza; `N_Part=1000` representa el número de partículas en el primer instante de tiempo, con lo que ahora es posible asignarle un espacio en memoria a cada una de ellas mediante la creación de una estructura (Algoritmo 2.2), esta estructura se ha dividido en tres partes, una parte de 1000 posiciones para almacenar las partículas de búsqueda y seguimiento, una segunda parte de 400 posiciones para almacenar las partículas que se seleccionan para la ruleta, y una tercera parte de 20 posiciones para almacenar las soluciones generadas por la búsqueda dispersa. También se crea una estructura para guardar las posiciones que sirven de referencia para la ejecución del K-Medias y la segmentación por forma, además se guardan las posiciones de predicción y las variables que sirven para el posicionamiento del mouse.

Algoritmo 2.2. Crear Estructura

```
struct PART{
        int X,Y
        float PESO,TIPO
       }
PART Par[1001]
PART Rul[401]
PART Bdi[20]

struct POS{
            Int POSX,POSY
           }
POS Ref[2]
POS Cir[2]
POS Pre[2]
POS Vdir[1]
POS Mouse[4]
```

En la estructura PART se definen las cualidades que cada partícula tendrá, una coordenada o estado (X, Y), un peso (PESO) y un tipo (TIPO) que determina el grupo al que pertenece, pupila derecha o pupila izquierda.

Generación de datos.

Con los espacios de memoria listos para recibir la información, es hora de generar las posiciones de cada partícula mediante la función `GENERAR()` (Algoritmo 2.3), en este primer instante estas posiciones son aleatorias en toda la imagen, se genera un número aleatorio entre el ancho y alto de la imagen, es decir entre 1 y 640 para X, y entre 1 y 480 para Y. En este instante de la iteración el peso de las partículas es el mismo y está definido por uno sobre el número (1/N_Part), después es calculado mediante la función `CALC_PROMEDIO()`(Algoritmo 2.9). El procedimiento `RESTRIC()`(Algoritmo 2.8) se encarga de verificar que las partículas generadas no se posicionen por fuera de la imagen, debido a que el número generado define el centro de la ventana y teniendo en cuenta el tamaño de la partícula, si el centro se ubica sobre algún borde de la imagen una parte de la ventana quedaría por fuera.

Algoritmo 2.3. Generar datos

```
Función GENERAR (V_in, ancho,alto, N_part)
// Comentario: Este algoritmo genera las partículas del filtro
// V_in indica la imagen de entrada.
v_out = Convertir_en_Gris (V_in)
```

```
Para i ← 1 Hasta N_Part Hacer
  x, y = ALEAT(alto, ancho)
  RESTRIC(x, y)
  v_out = Recortar(x,y,v_out)
  Peso←CALC_PROMEDIO (v_out, h1)
  Par(i).X = x
  Par(i).Y = y
  Par(i).PESO = Peso
Fin Para
Fin Función
```

Dentro de la etapa de generación de datos se hace necesario establecer el modelo del objetivo a seguir como ya se ha mencionado anteriormente, este modelo es una imagen del objeto del cual se le extraen las características de color mediante su histograma en escala de grises. En primer lugar se definen los parámetros del histograma mediante la función (Algoritmo 2.4).

Algoritmo 2.4. Crear Histograma

```
Función CREATEHIST ()
// Comentario: Esta función crea la variable para guardar un histograma con los
//parámetros establecidos
rangoscanal ← Rango del histograma
h_size ← Tamaño del histograma
h ← Variable para almacenar datos

h = CvCreateHist(h_size,rangoscanal)
Devuelve h
Fin Función
```

Ahora es necesario adquirir los datos de la imagen, para ello se crea la función `MODELO()` (Algoritmo 2.5) donde se carga la imagen modelo en escala de grises desde la base de datos en memoria del computador y se crea el histograma el cual es utilizado en etapas posteriores para realizar una comparación.

Algoritmo 2.5. Cargar Modelo

```
Función MODELO (h)
// Comentario: Esta función crea el histograma de la imagen que se utiliza como modelo
//del objetivo
i_in ← Variable para guardar la imagen.
h ← CREATEHIST()
h1 ← h Variable para almacenar el histograma del modelo
i_in = Cargar_Imagen(Imagen_Modelo) // Comentario: La Imagen_Modelo se encuentra
//almacenada en memoria del computador.
h1 = HISTOGRAMA (i_in)
```

```
Fin Función
```

La creación del histograma como tal, implica el desarrollo de una función que realice esta tarea, la función `HISTOGRAMA()` (Algoritmo 2.6) se encarga de tomar los datos de cada pixel de la imagen ya cargada y contabilizar las veces que se repite el mismo nivel de intensidad luminosa. Esto permite guardar los datos en la variable del histograma creada anteriormente, pero un requisito para realizar la comparación mediante la distancia de Bhattacharyya es que el histograma debe estar normalizado, para ello se utiliza `cvNormalizeHist()` que arroja como resultado un histograma con valores menores que la unidad.

Algoritmo 2.6. Calcular Histograma

```
Función HISTOGRAMA (i_in)
// Comentario: Esta función crea la variable para guardar los datos de un histograma.
hr11 ← Variable donde se almacena un histograma.
hr11 = Calcular_Histograma (i_in)
hr11 = Normalizar_Histograma(hr11)
Fin Función
```

A continuación se presentan los procedimientos auxiliares, en primer lugar el procedimiento encargado de determinar la resolución de la pantalla que utilizada para el posicionamiento del mouse (Algoritmo 2.7).

Algoritmo 2.7. Calcular Tamaño

```
Procedimiento TAMA ()
// Comentario: Este procedimiento Calcula el tamaño de la pantalla
  ancho = Ancho de la pantalla // Función de la librería Windows.h
  alto = Alto de la pantalla // Función de la librería Windows.h
Devuelve alto, ancho
Fin Procedimiento
```

Y finalmente el procedimiento encargado de asegurarse que las partículas generadas se ubiquen dentro de la imagen (Algoritmo 2.8).

Algoritmo 2.8. Establecer Restricción

```
Procedimiento RESTRIC (V_in, Tamaño)
// Comentario: Este procedimiento evita que las partículas se tomen posiciones fuera de //
la pantalla.
x ← Coordenada x de la partícula
y ← Coordenada y de la partícula
Tamaño ← Tamaño de las partículas
Si x < Tamaño Entonces
  x = x+1
Fin Si
Si y < Tamaño Entonces
```

```
  x = y+1
Fin Si
Si x > ancho – Tamaño Entonces
  x = ancho – (Tamaño +1)
Fin Si
Si y > alto – Tamaño Entonces
  y = alto – (Tamaño +1)
Fin Si
Fin Procedimiento
```

Calcular Pesos

El cálculo de los pesos de cada partícula se realiza mediante la distancia de Bhattacharyya, que determina el grado de similitud entre dos histogramas, para ello se ha implementado la función `CAL_PROMEDIO()` (Algoritmo 2.9) que recibe como parámetros el histograma modelo y la imagen (ventana de 7 x 7 píxeles) de cada partícula y retorna el valor del peso.

Algoritmo 2.9. Calcular Peso

```
Función CAL_PROMEDIO (Im_out, h1)
// Comentario: Esta función compara dos histogramas mediante Bhattacharyya
Im_out ← Imagen que representa cada partícula
h1 ← Histograma del modelo del objeto
peso ← Grado de similitud entre histogramas
hr11 = HISTOGRAMA (Im_out)
peso = Comparar_Histogramas(h1,hr11, COMPARAR_CON_BHATTACHARYYA)
Devuelve peso
Fin Función
```

Una vez todas las partículas tienen su respectivo peso que es proporcional a la probabilidad de encontrarse sobre el objetivo, se realiza una etapa auxiliar de ordenamiento, ubicando en las primeras posiciones de la estructura aquellas partículas con mayor peso. Para esto, se crea el procedimiento `ORDENAR()`(Algoritmo 2.10), que recibe como parámetro el número de partículas a ordenar, el proceso de ordenamiento se realiza con el método de la burbuja, explicado en el capítulo 3.

Algoritmo 2.10. Ordenar

```
Procedimiento ORDENAR(num)
// Comentario: Ordena los elementos de mayor a menor por el método de la Burbuja
num ← Numero de elementos a ordenar
Para i ←1 Hasta num Hacer
  Para j ← 1 Hasta num Hacer
    Si Par(j).PESO < Par(i).PESO Entonces
      aux =  Par(j)
      Par(j) = Par(i)
```

```
      Par(i) = aux
    Fin Si
  Fin Para
Fin Para
Fin Procedimiento
```

Segmentación por forma.

La segmentación por forma se encarga de encontrar los contornos circulares dentro de la imagen. Para lograr una eficiencia computacional, la función implementada llamada `CIRCULO()` (Algoritmo 2.12), utiliza las partículas de mayor peso como puntos de referencia para realizar un recorte de la imagen donde muy probablemente se encuentre la pupila del ojo para no tener que procesar la imagen completa. En este sentido se desarrolló la función `REFE()` (Algoritmo 2.11) que encuentra las dos partículas de mayor peso que representan las pupilas y se almacenan en la estructura POS Ref.

Algoritmo 2.11. Referencia de Búsqueda

```
Función REFE (N_Part)
// Comentario: Busca los dos puntos de referencia cuando se encuentran los círculos de //
la búsqueda circular
disrefex ← Parámetro de diseño distancia de referencia para x
disrefey ← Parámetro de diseño distancia de referencia para y
k ← Contador
Mientras verdadero Hacer
  k ++
  Si Par(k).PESO = 0  o k < N_Part Entonces
    Salir
  Fin Si
  distx = distancia entre (Par(1).X y Par(k).X)
  disty = distancia entre (Par(1).Y y Par(k).Y)
  Si distx > disrefx y disty < disrefey Entonces
    Ref(1) = Par(1) // Partícula 1 de mayor peso
    Ref(2) = Par(k) // Partícula 2 de mayor peso
    Salir
  Fin Si
Fin Mientras
```

Ahora es posible implementar la función `CIRCULO()` que recibe como parámetro de entrada la imagen adquirida por el sensor y las posiciones de las partículas de mayor peso, con estas posiciones se establece un recuadro alrededor de ellas, en este caso se toma un recuadro de 100 pixeles con la función `cvSetImageROI()`, estos recuadros se procesan para encontrar contornos circulares, en primer lugar se aplica un filtro de Canny `cvCanny()` que encuentra los bordes existentes de los objetos contenidos ahí y

posteriormente mediante la función `cvHoughCircles()` de la librería utilizada, se ejecuta la búsqueda de formas circulares y se calibra para que encuentre dos círculos, valor que retorna la función:

Algoritmo 2.12. Forma Circular

```
Función CIRCULO (V_in)
// Comentario: Esta función encuentra formas circulares en las dos zonas más con mayor
// probabilidad de encontrarse los objetivos.
gris ← Variable para almacenar el recorte de la imagen
reco ← Dimensión para recortar la imagen

gris = Convertir_en_gris(V_in)
Mientras verdadero Hacer
  i++
  Si i > 2 Entonces
  Salir
  Fin Si
  Si i = 1 Entonces
    x = Ref(1).POSX – reco
    y = Ref(1).POSY – reco
  Fin Si
  Si i = 2 Entonces
    x = Ref(2).POSX – reco
    y = Ref(2).POSY – reco
  Fin Si
  RESTRIC(x,y)
  gris = Recortar(x,y)
  gris = Canny(gris)
  results = Encontrar_Circulos(gris)
  Si results = 1 Hacer
    Si i = 1 Entonces
    Cir(1).POSX = Posición X del circulo 1
    Cir(1).POSY = Posición Y del circulo 1
    Fin Si
    Si i = 2 Entonces
    Cir(1).POSX = Posición X del circulo2
    Cir(1).POSX = Posición Y del circulo 2
    Fin Si
  Total ++
  Fin Si
  Si Total = 2 Hacer
    Salir
  Fin Si
Fin Mientras
Fin Función
```

Búsqueda dispersa

La Búsqueda Dispersa se encarga de optimizar las soluciones encontradas por el filtro de partículas, para ello se ha implementado la función `BDIS()`(Algoritmo 2.13), que se encarga de realizar este proceso. La función recibe como parámetros la imagen de entrada y el histograma del modelo del objeto. En primer lugar se define el área de búsqueda con la ayuda de los centros de las pupilas brindados por la segmentación por forma, la función recurre a la estructura *POS Cir* y toma estas posiciones, luego se crea un recuadro de 30 píxeles alrededor estas posiciones, para generar aleatoriamente 10 posibles soluciones en esta región.

Estas partículas se evalúan comparando sus histogramas con el modelo inicial del objeto y se calculan los pesos en la función `CAL_PROMEDIO()` (Algoritmo 2.9), estas partículas son almacenadas en la estructura *PART Bdi*. Posteriormente se ordenan de mayor a menor peso con la función `ORDENAR()` (Algoritmo 2.10) y se procede a realizar la etapa combinatoria de la Búsqueda Dispersa. Cómo se expuso en la sección 3.2.1 de este trabajo, se realiza una combinación de las mejores soluciones con las de menos calidad, para ello se promedian las distancias en X e Y de cada pareja y se establece en ese punto una nueva solución, a la cual se le aplica el método de búsqueda local para mejorar aún más su calidad. Al final del proceso se obtienen 10 partículas con altos pesos, las cuales se añaden a las generadas por el filtro para ser replicadas en la ruleta, estas reemplazan a las de menor calidad en las 400 escogidas para sobrevivir.

Algoritmo 2.13. Búsqueda Dispersa

```
Función BDIS (V_in, h1)
Mientras verdadero Hacer
  Conta++
  Si Par(conta).PESO = 0 Entonces
    Salir
  Fin Si
Fin Mientras

  Para m ← 1 Hasta 2 Hacer
    Si m = 1 Entonces
      Partx = Cir(1).POSX
      Party = Cir(1).POSY
    Fin Si
    Si m = 2 Entonces
      Partx = Cir(2).POSX
      Party = Cir(2).POSY
    Fin Si
    Para i ←1 Hasta 10 Hacer
      rx = aleatorio (de -15 a 15)
      ry = aleatorio(de -15 a 15)
```

```
    x = Partx + rx
    y = Party + ry
    RESTRIC (V_in, (x,y))
    v_out = Recortar (V_in,x,y)
    HISTOGRAMA(v_out)
    Peso ← CALC_PROMEDIO(v_out,h1)
    Bdi(i).X = x
   Bdi(i).Y = y
   Bdi(i).PESO = Peso
  Fin Para
    ORDENAR()
   Combinar Soluciones
   Búsqueda Local
  ORDENAR()
  Si m = 1 Entonces
    Almacenar en Par(conta)
  Fin Si
  Si m = 2 Entonces
    Almacenar en Par(conta+10)
  Fin Si

 Fin Para
Fin Función
```

K-medias

El algoritmo de las K-Medias es utilizado para mantener un número mínimo de partículas en los dos máximos de la función de densidad de probabilidad generada por el filtro, estos máximos son la representación estadística de las pupilas de los ojos. Para ello se ha implementado la función `K_MEDIAS()` (Algoritmo 2.14), que realiza el proceso de agrupamiento de las partículas en dos clases; pertenecientes a la pupila derecha y pertenecientes la pupila izquierda. En primer lugar se definen los dos puntos de referencia para realizar el agrupamiento, estos pueden ser los encontrados por la función `CIRCULO()` (Algoritmo 2.12), si esta función ha encontrado los dos círculos, si no es así los puntos los define la función `REFE()` (Algoritmo 2.11). También se define un radio `rd` donde cada partícula de referencia es su centro, y se denomina radio de búsqueda, y tiene la función de determinar la distancia máxima a la cual cada partícula puede encontrarse para pertenecer al grupo.

Algoritmo 2.14. K-Medias

```
Función K_MEDIAS ()
// Comentario: Este algoritmo realiza el K-Medias para clasificar las partículas en dos
//grupos, pupila derecha y pupila izquierda.
```

```
rd ←Radio de Búsqueda
iter ← Iteraciones del K-Medias
rx1 ← Referencia en x para punto 1
ry1 ← Referencia en y para punto 1
rx2 ← Referencia en x para punto 2
ry2 ← Referencia en y para punto 2
Para i ← 0 Hasta iter Hacer
  Si i = 1 Entonces
    rx1 = Ref(1).POSX o Cir(1).POSX
    ry1 = Ref(1).POSY o Cir(1).POSY
    rx2 = Ref(2).POSX o Cir(2).POSX
    ry2 = Ref(2).POSY o Cir(2).POSY
  Fin Si
    Para j ← 0 Hasta N_part Hacer
      distancia1 = Distancia de Partícula j con respecto al punto1
      distancia2 = Distancia de Partícula j con respecto al punto2
      Si distancia1 < rd Hacer
        Par( j) . TIPO = 1
        suma1++
        Recalcular_pesos.
      Fin Si
      Si distancia2 < rd Hacer
        Par( j) . TIPO = 2
        Suma2++
        Recalcular_pesos
      Fin Si
    Fin Para
    rx1 = Promedio de X en TIPO 1
    ry1 = Promedio de Y en TIPO 1
    rx2 = Promedio de X en TIPO 2
    ry2 = Promedio de Y en TIPO 2
Fin Para
Devuelve suma1, suma2
Fin Función
```

Donde `distancia1` es la variable donde se almacena la distancia en pixeles de la partícula `j` con respecto al centroide de referencia 1, y `distancia2` con respecto al centroide de referencia 2.

Con estas distancias calculadas y el radio definido, ya cada partícula puede ser asignada a un grupo, simplemente escogiendo la menor distancia y que se encuentre dentro del radio de búsqueda, se ha determinado la notación 1 para tipo 1, pupila derecha y 2 para tipo 2, pupila izquierda, con lo que en la variable TIPO de cada partícula en la estructura se guardará este valor.

Una vez se hayan asignado todas las partículas a su respectivo grupo, se hace una actualización de los centroides, calculando el promedio de distancias en X y en Y para cada grupo.

Dentro de esta función se implementa la etapa de recalcular los pesos, es decir, al peso obtenido por similitud de color, se adiciona el peso por cercanía con el objetivo, para ello simplemente se aplica la ecuación 3.9 a cada partícula, conociendo la distancia brindada por K-Medidas.

Crear ruleta.

La creación de la ruleta se realiza con las 400 partículas que se conservan para el seguimiento, de las cuales se ha establecido un número de 200 para cada grupo, es decir 200 partículas se encargan de seguir y estimar la posición para la pupila derecha y 200 para la pupila izquierda. En este sentido, la función `RULETA()` (Algoritmo 2.15) se encarga de normalizar los pesos de las partículas de cada grupo para asignar su respectivo porcentaje de supervivencia. Estas partículas son almacenadas en la estructura *PAR Rul*.

Algoritmo 2.15. Crear Ruleta

```
Función RULETA (N_Part)
suma1 ← Sumatoria de los pesos de TIPO 1
suma2 ← Sumatoria de los pesos de TIPO 2
div ← 0
aux ← 2001
K_MEDIAS() // Comentario: Devuelve suma1 y suma2
ORDENAR(N_Part)
Para i ← 1 Hasta N_Part Hacer
  Si Par(i).TIPO = 1 Entonces
    Porcentaje = Par(i).PESO * (200/suma1)
  Fin Si
  Si Par(i).TIPO = 2 Entonces
    Porcentaje = Par(i).PESO * (200/suma2)
  Fin Si
  Si Par(i).TIPO = 1 o Par(i).TIPO = 2 Entonces
    Para j ← aux Hasta (j+div) Hacer
      Rul(j) = Par(i)
    Fin Para
   aux = aux+div
  Fin Si
Fin Para
Fin Función
```

Estimar

La estimación del sistema se realiza en el programa principal y simplemente consta de promediar las posiciones para cada grupo clasificado, es decir, para las 200 partículas que representan la pupila derecha y para las 200 de la pupila izquierda. Con esto se obtiene las dos posiciones más representativas de los objetivos que brinda el Filtro de Partículas Híbrido, por lo tanto son las posiciones de las pupilas del usuario.

Tirar la ruleta, difusión y modelo de movimiento.

Esta parte del filtro se encarga de seleccionar las partículas que pasarán a la siguiente iteración, para ello se seleccionan posiciones al azar de la ruleta, teniendo en cuenta que las partículas con mayor peso tienen más espacio en la misma, por lo que cuentan con una mayor probabilidad de ser elegidas. La elección se realiza en el código principal mediante la función `ALEAT()` (Algoritmo 2.16), generando números aleatorias en el rango de 1 a 400, posiciones de la estructura donde están almacenadas *PART Rul*. Dentro del diseño del filtro se ha decidido generar 600 partículas a partir de las 400 almacenadas en la ruleta, ya que es un número suficiente para mantener una diversidad de muestras el espacio de estados, que ahora es más limitado porque se conoce las regiones más probables de encontrar el objetivo. Para aplicar la difusión y el modelo de movimiento, a las partículas seleccionadas se les distribuye aleatoriamente dentro de la zona a la que pertenece esa partícula con una velocidad y dirección que es determinado por el vector director del movimiento del punto estimado en el instante anterior.

Algoritmo 2.16. Tirar Ruleta y mover

```
Función ALEAT (ancho, alto,Bandera)
// Esta función genera números aleatorios
Bandera ← Variable booleana proveniente de GENERAR()
Vdi  ← Vector director proveniente de movimiento

Si Bandera = Falso Hacer
  Ini_randx = ancho
  Ini_randy = alto
  Velx = 0
  Vely= 0
Fin Si
Si Bandera = Verdadero Hacer
  Ini_randx = 40
  Ini_randy = 40
  Velx = aleatorio (de 0 a 1000) / 1000 * Vdi.POSX
  Vely= aleatorio (de 0 a 1000) / 1000* Vdi.POSY
Fin Si
```

```
  x = aleatorio (de 0 a Ini_randx)+Velx
  y = aleatorio (de 0 a Ini_randy)+Vely

Devuelve x,y.
Fin Función
```

Los anteriores algoritmos representan el funcionamiento del filtro de partículas híbrido, que recibe las imágenes del sensor y traduce las posiciones de las pupilas en un punto en la pantalla del usuario. La siguiente etapa del algoritmo final consiste en traducir esa posición en la posición del cursor, para lo que se ha implementado un par de funciones que realizan esta tarea.

Algoritmo 2.17. Posicionamiento

```
Función POS ( ancho, alto, iter)

ref = 3
posx= ( Pre[1].POSX + Pre[2].POSX )/2
posy= ( Pre[1].POSY + Pre[2].POSY )/2

Si iter = 1 Entonces
  posantx = posx
  posanty = posy
  conta = 1
Fin Si

dist = Raiz_cuadrada((posx-posantx)^2 + (posy-posanty)^2)

Vdi.POSX = posx - posantx
Vdi.POSY = posy - posanty

Si dist >2 y dist <4 Entonces
  Vdi.POSX = Vdi.POSX *1.5
 Vdi.POSY = Vdi.POSY *1.5
Fin Si

Si dist > 4 Entonces
  Vdi.POSX = Vdi.POSX *2
  Vdi.POSY = Vdi.POSY *2
Fin Si

Si dist >ref y conta =1 Entonces
  ANGULO(posx, posy)
  conta=2
Fin Si

posantx = posx
```

```
posanty = posy

Si conta =2 Entonces
  contdis=condis+1
  dismouse=dismouse+dist
  if contdis>3 Entonces
    dismouse=dismouse*40
    contdist=0
    conta=3
  fin Si
fin Si

Si conta=3 Entonces
  Si dismouse > dismouseinit Entonces
    dismouseinit = Raiz_cuadrada((Posmx-Posmantx)^2 + (Posmy-Posmanty)^2)
      Si dismouse-dismouseint > 10 Entonces
        vel = 0.1* ( dismouse-dismouseint )
      fin Si
      Si dismouse-dismouseint < 10 Entonces
        vel = 1
      fin Si
  Si no  Entonces
    conta=1
    click=1
    vel=0
  fin Si
Si no  Entonces
Posmantx=Posmx
Posmanty=Posmy
fin Si

Posmx = vel * Dir1+Posmx
Posmy = vel * Dir2+Posmy

SetCursorPos(Posmx,Posmy)

Si click = 1y iter>2 Entonces
  contclick = contclick+1
    Si contclick = 50 Entonces
      mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, GetMessageExtraInfo());
      contclick =0
      click = 0
      mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, GetMessageExtraInfo());
    fin Si
fin Si

Fin Función
```

Para el posicionamiento del mouse en la pantalla una de las principales etapas es encontrar el ángulo, este establece la dirección que deberá tomar el cursor para llegar a la siguiente posición. La función *ANGULO ()* retorna la dirección en las variables *DIR1* y *DIR2*.

Algoritmo 2.18. Ángulo

```
Función ANGULO(posx, posy)
//posx ← Posición actual en x del sistema
//posx ← Posición actual en y del sistema
//posantx ← Posición anterior en x del sistema
//posanty ← Posición anterior en y del sistema

VD1 = (–10, 0)
VD2=(posx – posax , posy – posay)

Ppunto = ((posx – posax)* –10)
Modulo1 = raiz_cuadrada((-10) ^ 2)
Modulo2 = raíz_cuadrada((posx – posax) ^ 2 +( posy – posay) ^ 2)
Pmod = Modulo1 * Modulo2
Valor = Ppunto / Pmod

Angulo = acos(Valor)

Si Angulo < 22.5 o Angulo >337.5 Entonces
  Dir1 = 1
  Dir2 = 0
Fin Si
Si Angulo < 67.5 y Angulo >22.5 Entonces
  Dir1 = 1
  Dir2 = 1
Fin Si
Si Angulo < 112.5 y Angulo >67.5 Entonces
  Dir1 = 0
  Dir2 = 1
Fin Si
Si Angulo < 157.5 y Angulo >112.5 Entonces
  Dir1 = -1
  Dir2 = 1
Fin Si
Si Angulo < 202.5 y Angulo >157.5 Entonces
  Dir1 = -1
  Dir2 = 0
Fin Si
Si Angulo < 247.5 y Angulo >202.5 Entonces
  Dir1 = -1
  Dir2 = -1
Fin Si
```

```
Si Angulo < 292.5 y Angulo >247.5 Entonces
  Dir1 = 0
  Dir2 = -1
Fin Si
Si Angulo < 337.5 y Angulo >292.5 Entonces
  Dir1 = 1
  Dir2 = -1
Fin Si

Fin Función
```