



INSTITUTO TECNOLÓGICO
DE CD. MADERO



DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN



Tesis:

“Detección de la Mano en Ambientes Complejos”

Para Obtener el Grado de:

Maestro en Ciencias en Ciencias de la Computación

Presenta:

I.S.C. Gamaliel Sánchez Orellana

Director de Tesis

M.C. José Apolinar Ramírez Saldívar

Co-Director de Tesis

Dr. Juan Javier González Barbosa

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO



SECRETARÍA DE
EDUCACIÓN PÚBLICA

SEP

Cd. Madero, Tamps; a 17 de Agosto de 2012

OFICIO No.: U5.165/12
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN
DE TESIS

**C. ING. GAMALIEL SÁNCHEZ ORELLANA
PRESENTE**

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestría en Ciencias en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"DETECCIÓN DE LA MANO EN AMBIENTES COMPLEJOS"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

ATENTAMENTE
"Por mi patria y por mi bien"

M. P. María Yolanda Chávez Cinco
M. P. MARÍA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

c.c.p.- Archivo
Minuta



S.E.P.
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
I.T.C.M.

MYCHC *[Signature]*



Ave. 1ª de Mayo y Sor Juana I. de la Cruz, Col. Los Mangos, C.P. 89440 Cd. Madero, Tam.
Tels. (833) 3 57 48 20, Fax: (833) 357 48 20, Ext. 1002, email: itcm@itcm.edu.mx
www.itcm.edu.mx



DECLARACIÓN DE ORIGINALIDAD

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones.

Además, en caso de infracción de los derechos de terceros derivados de este documento de tesis, acepto la responsabilidad de la infracción y relevo de ésta a mi director y codirectores de tesis, así como al Instituto Tecnológico de Ciudad Madero y sus autoridades.

12 de Junio del 2012, Cd. Madero, Tamps.

Ing. Gamaliel Sánchez Orellana

Tabla de Contenidos

Capítulo 1 Introducción.....	6
1.1 Hipótesis.....	7
1.2 Justificación y Beneficios.....	7
1.3 Objetivo.....	8
1.3.1 Objetivo General.....	8
1.3.2 Objetivos Específicos.....	8
1.4 Alcances y Limitaciones.....	8
Capítulo 2 Marco Conceptual.....	9
2.1 Diferencia entre Detección y Reconocimiento.....	10
2.2 Tipos de Ambientes para la Detección de la Mano.....	12
2.2.1 Ambiente Dinámico.....	12
2.2.2 Ambiente Complejo.....	13
2.3 Aprendizaje por refuerzo.....	13
2.3.1 Ambiente.....	15
2.3.2 Estado.....	16
2.3.3 Acción.....	16
2.3.4 Recompensa.....	16
2.3.5 Agente.....	16
2.3.6 Política.....	16
2.3.7 Traza de Elegibilidad.....	17
2.3.8 Complejidad.....	17
2.3.9 Discretización.....	17
2.3.10 Episodio.....	18
2.3.11 Aprendizaje Supervisado.....	18
Capítulo 3 Estado del Arte.....	19
3.2 Detección Robusta de la Mano.....	19
3.3 Detección de objetos usando contorno	19
3.4 Detección de objetos usando textura	20
3.5 Detección Robusta de Objetos en Tiempo Real.....	21
3.6 Camshift.....	22
3.7 Chamfer Matching.....	24
Capítulo 4 Método.....	25
4.1 The Reinforcement Learning Toolbox	25
4.2 Algoritmos de Aprendizaje.....	26
4.3 Conjunto de Estados.....	27
4.3.1 Modelo de Componentes HSV	27
4.4 Acciones	31
4.5 Recompensas	31
4.6 Política	32
4.7 Autómatas finitos deterministas y no-deterministas	34
4.8 Ejemplo determinístico:.....	36
4.9 Propiedad de Markov	37
4.10 Medición de cambios en las variables de Estado	38
CAPITULO 5 IMPLEMENTACION.....	39

5.1 Hardware.....	39
5.2 Software	39
5.2.1 Detector 1: Viola-Jones.....	39
5.2.1 Detector 2: CamShift.....	44
5.2.2 Detector 3: Chamfer Matching.....	45
5.3 Hilos.....	51
5.4 Diseño de la aplicación con RL Toolbox.....	51
5.4 Diseño de la aplicación RL'Toolbox.....	54
5.5 Interfaz de Usuario.....	56
5.6 Qt4.....	56
CAPITULO 6 EVALUACION EXPERIMENTAL.....	59
6.1 Metodo de Evaluación.....	59
6.2 Proceso.....	59
6.3 Conclusión.....	59
6.4 Posibles Mejoras.....	60

Capítulo 1 Introducción

La detección de objetos empleando una cámara conectada a una computadora, es un problema relevante en diferentes campos de la computación, como la visión por computadora, la interfaz humano-máquina y la robótica. En particular la detección de la mano presenta problemáticas propias debido a los cambios en el ambiente, como son la detección en ambientes dinámicos (cambios del escenario) y la detección en ambientes complejos (cambios de iluminación, cambios de coloración en el fondo, la escalabilidad, la posición, orientación o rotación de la mano) , por lo que continúa siendo un problema abierto.

Entre las técnicas más utilizadas para abordar este problema, se encuentran las que tratan con la coloración del objeto y las que emplean métodos estadísticos. Otros métodos se basan en el reconocimiento de patrones, como pueden ser la textura, la forma o el movimiento del objeto.

Se ha demostrado que los métodos de reconocimiento de patrones operan eficientemente en ambientes dinámicos, siempre y cuando no haya cambios importantes en el ambiente, y cuando esto ocurre se requiere volver a calibrar el detector para mejorar su eficiencia.

En este proyecto se propone desarrollar una estrategia de selección en tiempo real entre varios detectores de objetos que emplean diferentes técnicas para su detección, de aquel que resulte mas apropiado para las condiciones medidas del ambiente.

Esta investigación se centra en el problema de detección de la mano humana a partir de imágenes, tomando en cuenta que el ambiente de detección de la misma puede ser complejo. Esta investigación se ubica en el área de Ciencias de la Computación, dentro del campo de la *Visión por Computadora*.

1.1 Hipótesis.

Es posible emplear selectivamente varias técnicas de detección de objetos (reconocimiento de patrones, redes neuronales, etc.) que utilicen diferentes características del objeto a detectar en un ambiente complejo y con un comportamiento grupal eficiente. Es decir, seleccionar la técnica apropiada al estado del ambiente (cambios de iluminación, cambios en el fondo, cambios por movimiento, etc.) que opere de forma eficiente en la tarea de detección de objetos.

Se propone emplear aprendizaje por refuerzo en el entrenamiento de diferentes tipos de detectores para la selección del más apropiado de acuerdo al estado complejo del ambiente.

1.2 Justificación y Beneficios

La problemática que presenta el problema de detección de la mano, ha sido abordada por muchas investigaciones, lo cual ha resultado en el desarrollo de diversas técnicas que permiten la detección de la mano. Sin embargo, se ha demostrado que por las condiciones dinámicas y complejas del ambiente, si se utiliza un solo método de detección, este tiende a fallar. El aumento del poder de cómputo de las computadoras personales actuales, permite el uso simultáneo de varias técnicas de detección con eficiencia similar al uso individual de cada detector en la fecha de su creación

Esta investigación se realiza con el propósito de desarrollar un método basado en aprendizaje por refuerzo que permita el uso de varias técnicas de detección y dependiendo del estado del ambiente en que la mano se encuentra, seleccionar la técnica de detección que resulte apropiada.

La realización de este proyecto pretende mejorar la eficiencia en la detección de la mano, lo que beneficiará en un futuro si se propone utilizarlo como parte de algún proyecto que implique el reconocimiento de la mano; por ejemplo, el uso de la mano para señalar a un robot la tarea a realizar, o el uso del lenguaje de signos para la comunicación humano-computadora.

1.3 Objetivo

1.3.1 Objetivo General

El objetivo general de esta investigación es lograr la detección de la mano tomando en cuenta que el ambiente en el que se encuentra puede ser complejo.

1.3.2 Objetivos Específicos

- 1) Implementación y prueba de diferentes tipos de detectores.
- 2) Aplicación de una estrategia de selección del detector mediante aprendizaje por refuerzo.

1.4 Alcances y Limitaciones

Alcances que tendrá mi proyecto:

- 1.- Se realizará la detección de la mano en ambientes complejos.
- 2.- El detector podrá identificar la mano sin ser afectado por los cambios en la iluminación o el escenario donde se encuentre el objeto.

Limitaciones que tendrá mi proyecto:

- 1.- Se detectará únicamente la mano, no posiciones específicas que se realicen con esta.
- 2.- La mano a detectar siempre se encontrará dentro del rango de visión de la cámara.

3.- La detección siempre se realizará en áreas de interiores.

Capítulo 2 Marco Conceptual

Durante el desarrollo de la civilización humana, la comunicación ha sido la única forma de transmitir ideas, sentimientos o emociones entre los seres humanos, la cual ha sido propiciada gracias al entendimiento y raciocinio del cual es capaz su cerebro. La comunicación humana se puede dar de múltiples maneras además del habla, por ejemplo, mediante gestos con la mano, la postura del cuerpo, la posición de la cabeza, etc.

A partir de la segunda mitad del siglo XX, durante el desarrollo de las computadoras, el ser humano se vio en la necesidad de contar con una interfaz humano-computadora que le permitiera comunicarse de alguna manera con estas máquinas, dando lugar al surgimiento de dispositivos como el teclado y el ratón, entre otros.

En años recientes, se ha incrementado el uso de cámaras como un dispositivo importante para las interfaces humano-computadora, lo que ha planteado diferentes problemas, entre estos el de la detección de objetos, entendiendo por detección la obtención de su posición en la imagen.

Dependiendo de la aplicación, los sistemas de visión computacional deben ser capaces de:

- (1) Manejar inconsistencias en las apariencias y oclusiones del objeto.
- (2) Reiniciar el rastreo cuando el objeto haya salido del área de visión de la cámara.
- (3) Adaptarse a movimientos no predecibles del objeto.
- (4) No verse afectado por las condiciones de iluminación del ambiente.
- (5) Rastrear en tiempo real.
- (6) No necesitar de un conocimiento muy grande del objeto que se quiere rastrear.

Con el fin de obtener mejores resultados en la detección, se ha propuesto el uso de una o varias marcas visuales sobre los objetos [Wang & Popovic 10]. Sin embargo, esto obliga a que cada objeto cuente con su propia marca visual, lo que en muchos casos limita la

eficiencia del detector, ya que si en el ambiente se encuentra algún otro objeto que tenga similitud con la marca del objeto que queremos detectar, el sistema de detección puede detectar otras cosas que no sean nuestro objeto.

Los métodos de detección de objetos normalmente hacen uso de una o varias señales para estimar la posición del objeto. Por ejemplo si el objeto a detectar es el cuerpo humano se emplean como señales partes del mismo como la cabeza o la mano. Estos métodos pueden ser clasificados en 2 grupos:

- (i) Métodos que detectan características del objeto como segmentos de líneas, contornos o color.
- (ii) Métodos que dependen directamente de la intensidad luminosa de la imagen.

Los primeros son sensitivos a la detección de características en toda la imagen y por lo tanto no conviene su uso donde las características que se buscan no son distintivas del objeto.

Los segundos se basan en el movimiento, la deformación o los parámetros de iluminación.

2.1 Diferencia entre Detección y Reconocimiento.

Esta investigación, como su nombre lo indica, se enfocará en la detección de la mano, lo cual requiere una diferenciación entre los términos *detección* y *reconocimiento*, ya que tienden a ser confundidos:

Detección: Consiste en determinar, dada una imagen, si un objeto de cierta clase se encuentra presente en esa imagen, y si se encuentra presente, obtener su localización. [Chen 09], como se muestra en la Figura 1.



Figura 1 . Detección de personas . Las personas son señaladas mediante rectángulos en rojo en la imagen

Reconocimiento: Consiste en la identificación de un objeto en una imagen. Una vez que el objeto especificado fue localizado, este es comparado en una base de datos para obtener una identificación o nombre de dicho objeto [Wikipedia 10], como se muestra en la Figura 2.

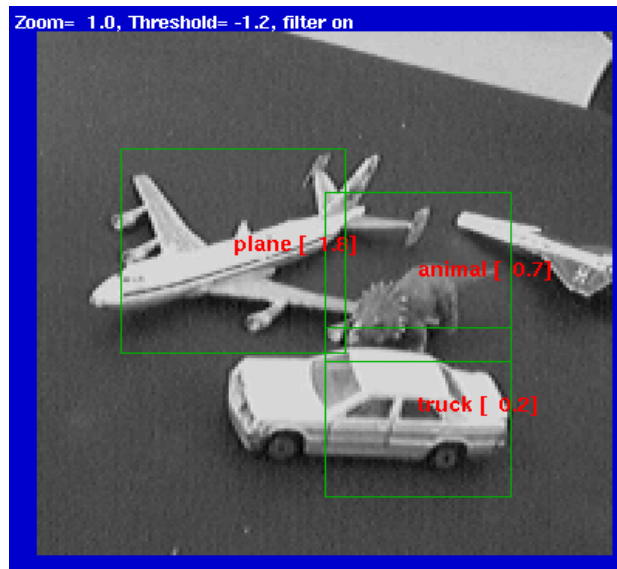


Figura 2. Reconocimiento de objetos. Cada objeto se identifica con la etiqueta de la clase a la que pertenecen. El reconocimiento requiere de la detección como uno de los pasos previos. Los objetos son comparados contra una base de datos que contiene las clases de objeto de interés.

2.2 Tipos de Ambientes para la Detección de la Mano.

Para la detección de la mano, se debe tomar en cuenta el ambiente, es decir, el entorno en el que la mano se encuentra en el momento preciso de analizar la imagen. El ambiente es el principal factor de afectación para la detección de objetos y a continuación se describen los dos tipos de ambientes en los que se puede encontrar un objeto.

2.2.1 Ambiente Dinámico.

El ambiente dinámico es un entorno indefinido, en el sentido de que sus componentes pueden variar en forma, color o tamaño, lo cual complica la detección de un objeto [Hossain, Ahn, Lee, & Chae 05]. (Ver Figura 3)



Figura 3. Mano en un ambiente dinámico, el entorno muestra variación, tanto de color como de objetos que forman parte del ambiente. En esta imagen, la mano puede ser detectada con un fondo más oscuro pero en un fondo más claro se vuelve indetectable para un detector basado en el color del objeto.

2.2.2 Ambiente Complejo.

Un ambiente complejo es un entorno que además de las variantes del ambiente dinámico, considera también cambios de iluminación, lo que dificulta aún más la detección del objeto [Tian, Feris & Hampapur 06]. (Ver Figura 4)



Figura 4. Mano en ambiente complejo, la imagen 1 (izquierda) muestra una mano un ambiente de detección con poca iluminación, mientras que en la imagen 2 (derecha) se muestra la misma mano pero con un aumento en la intensidad de la iluminación, lo que muestra que el ambiente se puede ver modificado notablemente por los cambios en la iluminación que pueden existir en el momento de la detección del objeto.

2.3 Aprendizaje por refuerzo.

Aprendizaje por refuerzo es un método en el que se busca que un agente realice una o varias acciones, las cuales están definidas en función de *pagos* o *recompensas*. Este método de aprendizaje surge de una rama de estudios de psicología experimental, que pueden remontarse a las experiencias de Pavlov con el refuerzo condicionado, y por otro lado es heredero de los métodos de control óptimo que se originan a partir de los trabajos de Bellman [Bellman 57].

El objetivo del sistema es maximizar las recompensas que se reciben durante un periodo de tiempo. Por lo tanto, recompensas altas son otorgadas a comportamientos deseados y recompensas bajas a comportamientos no-deseados.

El sistema no está restringido en su secuencia de acciones, conocida como *politica*, la cual maximiza las recompensas obtenidas.

Un ejemplo clásico de aprendizaje por refuerzo es el de un robot que debe guiarse a sí mismo (de forma autónoma) desde un punto inicial a una meta localizada en otro punto. El método debe otorgar recompensas mas altas por llegar a la meta en el menor tiempo que por topa con un obstáculo. La secuencia de acciones que el robot debe realizar están determinadas por el sistema, el cual las realizará con el objetivo de que se maximice la cantidad de recompensas obtenidas en la realización de la tarea.

Las técnicas de aprendizaje por refuerzo, se basan en la teoría de procesos de decisión de Markov (MDP), que se establece como un proceso estocástico caracterizado por:

- 1) Un espacio de estados finito $X \cup R^n$, donde un estado en el instante t se representa como $x_t \in X$.
- 2) Un conjunto discreto de acciones $A = \cup(A_i)$, donde A_i es el conjunto de acciones que pueden tomarse en el estado x_i
- 3) Una función de decisión para elegir la acción a aplicar en un estado dado, $P(a)$
- 4) Recompensas, establecida como una función $r: X \times A \rightarrow R$ en donde $r_t = r(x_t, a)$ es la recompensa esperada al ejecutar la acción a en el estado x_t .

En la Figura 5 se muestra un esquema básico del aprendizaje por refuerzo.

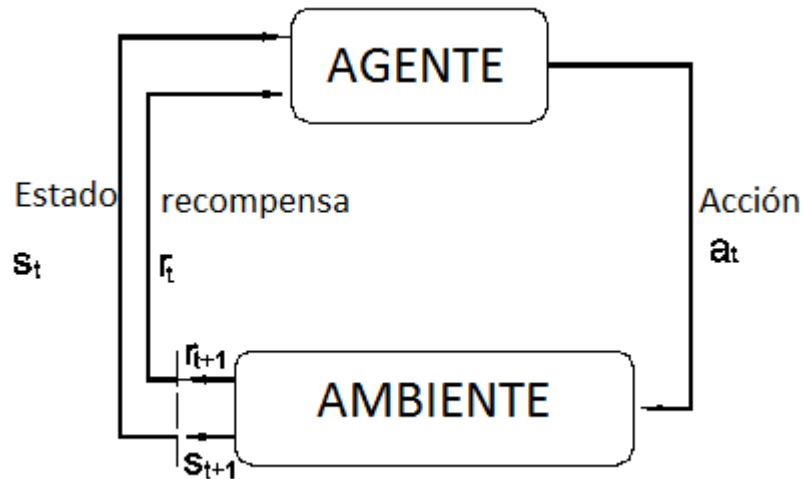


Figura 5. Esquema básico de aprendizaje por refuerzo. El agente realiza una acción en base al ambiente en el que se encuentra; dependiendo del resultado obtenido, se le otorga una recompensa para el estado en el que se ejecutó la acción.

2.3.1 Ambiente

Es la representación del mundo en el que el agente interactúa. Para explicar con facilidad este concepto, se muestra el siguiente ejemplo (Figuras 6 y 7). El ambiente consiste en el mundo que rodea a Andy – una mascota canina –, en lo que puede ver, escuchar y olfatear.



Figura 6. Estado: Andy está sentado. Tiene hambre. Ve a Mary y la comida enfrente de él. Escucha a Mary decir "Go Ahead (Adelante)".



Figura 7 Acción: Andy come su comida
Recompensa: Andy ya no tiene hambre.

El ambiente se representa mediante variables discretas o continuas y puede ser un solo valor o un conjunto de estos.

Normalmente, se emplea la notación siguiente:

S_t , que representa el estado del ambiente en el instante t

2.3.2 Estado

Es el vector S_t de descripción del estado que representa la situación específica en la que se encuentra el ambiente en el instante t . Por lo regular, el estado compone el ambiente, e incluye todas las variables que sean necesarias para poder decidir la siguiente acción que realizará el agente.

2.3.3 Acción

Una acción es lo que el agente decide realizar. Es la manera en la que el agente modifica el estado actual en el que se encuentra. Para Andy, una acción es sentarse, comer o ladrar (Figuras 6 y 7). Realizar estas acciones cambian por lo regular el estado actual. El

objetivo de un agente es realizar una acción que dé como resultado la recompensa mas alta.

Notación:

a_n representa la acción realizada en el estado x_n .

2.3.4 Recompensa

Son los medios con los cuales se enseña al agente qué acciones son consideradas buenas o malas dependiendo del estado en el que se encuentre. Para Andy, una recompensa positiva sería recibir un hueso, una recompensa mala sería un regaño.

La política de recompensas define el problema a resolver. En caso de no dar una buena recompensa a una acción considerada como apropiada no se obtendrán los resultados esperados en el sistema.

2.3.5 Agente

El agente es una entidad que estando en cierto estado, realiza una acción. El perro Andy es el agente en el ejemplo dado en las figuras 6 y 7.

2.3.6 Política

En los sistemas de aprendizaje por refuerzo simples, la acción apropiada es la que proporcione la recompensa más alta. A esto se le llama “politica greedy”, la cual busca escoger siempre la mejor acción posible. La desventaja de la politica greedy es que el agente realiza muy poca exploración en el espacio de estados. Otras políticas, como softmax o epsilon greedy son estocásticas al momento de escoger una acción que sin ser la mejor permite la exploración del espacio de estados. Por ejemplo, para Andy, un ladrido significa que obtendrá como recompensa un hueso, pero si en vez de ladrar se le ocurre hacer algún comportamiento por el cual no haya recibido recompensa aún, puede obtener una recompensa diferente; por lo tanto Andy realizó una exploración en el estado en el que se encontraba al realizar una acción diferente a la establecida.

2.3.7 Traza de Elegibilidad

En el caso general, la recompensa acumulada es la que se busca maximizar, que se determina mediante el valor de retorno $E\{\sum \gamma^t r_t\}$, donde la constante $0 < \gamma < 1$ se denomina factor de descuento, y que asegura que las recompensas en el corto plazo tengan mayor peso que en aquellas acciones tomadas en el largo plazo.

Para acelerar el aprendizaje, el historial de los estados y acciones se almacena hasta que se otorga una recompensa, entonces esa recompensa es distribuida hacia todo el camino que el agente eligió. Esto significa que el agente no solo aprenderá que la última acción fue buena o mala dependiendo del estado, sino que aprenderá que los estados y acciones anteriores fueron parte de la recompensa otorgada. Esto es importante ya que no se puede saber de todas las acciones que se tomaron cual fue crucial para obtener la recompensa. La recompensa distribuida por la traza de elegibilidad es reducida por un factor de descuento tomando en cuenta el tiempo y distancia que transcurrió entre la acción y la recompensa.

2.3.8 Complejidad

La complejidad de un sistema de aprendizaje normalmente se determina con el tamaño del espacio de estados multiplicado por el tamaño del conjunto de acciones. Si la complejidad resulta alta, es posible entender muchas situaciones diferentes y aprender diferentes comportamientos, pero es lento y la generalización entre los estados es baja. Sin embargo, si la complejidad es baja, el aprendizaje será más rápido y más generalizado. En ciertos problemas el espacio de estados llega a ser muy grande debido sobre todo al uso de variables de estado continuas, lo que lleva al problema de la discretización del espacio de estados.

2.3.9 Discretización

Si el espacio de estados incluye variables continuas la complejidad se vuelve muy alta de manera muy rápida. Por lo tanto, es importante contar con una buena representación del espacio de estados. Esto se logra discretizando el espacio de estados, reduciéndolo a un tamaño adecuado a las capacidades de cómputo empleadas. La discretización se puede hacer de diferentes maneras. Una manera simple es dividir una variable de estado en un número específico de partes. Otra manera es analizar la información de una variable de estado específica, encontrando grupos de interés y dividirlos de manera uniforme.

Por ejemplo, la iluminación es una variable de estado del ambiente que se mide en un rango de 0 a 255 niveles enteros; en nuestro caso, podemos considerar cambios significativos de la iluminación cada 10 niveles, con lo que la discretización de esta permitiría medirla en un rango de {0,10,20,30,.....,250}, reduciendo el tamaño del espacio de estados para esta variable a solo 25 valores.

2.3.10 Episodio

Es una descripción de una sesión completa realizada por el agente en el ambiente. Un episodio puede resultar exitoso (con una recompensa acumulada positiva) o fallido (con una recompensa negativa). Es común separar sesiones de aprendizaje diferentes, en las cuales la traza de elegibilidad no afecta a cada episodio.

2.3.11 Aprendizaje Supervisado

Es un método que también es llamado “aprendiendo con un maestro”. En este tipo de aprendizaje, el sistema recibe explicaciones específicas sobre como resolver el problema. Esto significa que obtiene un número de ejemplos sobre como resolver el problema en diferentes situaciones y a partir de esos ejemplos tratar de aprender como resolver ese problema por sí mismo.

En este trabajo, se entrenará el sistema a través de aprendizaje supervisado, dándole ejemplos de cómo actuar en el ambiente para que así pueda elegir el método de detección de la mano más apropiado.

Capítulo 3 Estado del Arte

A continuación se mencionan artículos relacionados con el problema de detección de la mano así como una descripción del método que utilizan sus autores para resolver el problema de detección.

3.1 Detección de objetos usando contorno

Este detector[Kasprzak , Skrzynski 06] realiza la detección de la mano ubicando el contorno de esta por medio de una binarización de la imagen que reduce el ruido en el fondo mediante filtros, se utiliza otra función para extraer las líneas de la imagen y obtener los contornos de la mano. La figura 8 muestra la ejecución de este detector.



figura 8. Se puede observar en la parte derecha de la imagen el contorno obtenido de la región donde se determina que se encuentra la mano, la parte izquierda muestra la detección de la mano hecha por el programa.

3.2 Detección de objetos usando textura

Se define como primera instancia el término Texel (Texture Element o Texture Pixel), que es la unidad mínima de una textura aplicada a una superficie. De la misma forma que una

imagen digital se representa mediante una matriz de píxeles, una textura se puede representar mediante un matriz de Texels.

Se revisa el método propuesto por [Shotton,Blake, Cipolla 05], el cual realiza una fusión del método de detección de contornos con un método de detección de texturas, la Figura 9 muestra el proceso realizado por este detector.

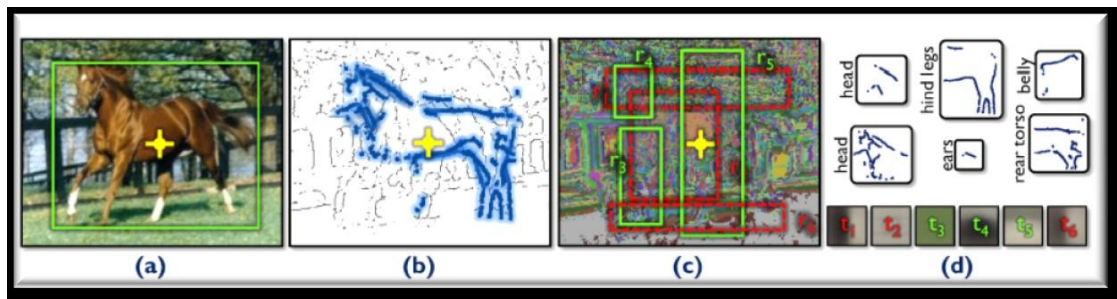


Figura 9. (a) El caballo es correctamente detectado usando contorno y textura. (b) Contornos de la imagen original, resaltando los que coinciden con la silueta del caballo. (c) Filtros de textura aplicados a la imagen, los cuadros verdes indican la presencia de texturas reconocidas por el programa, los cuadros rojos indican la ausencia de texturas en esa región. (d) contornos y texturas obtenidas del proceso de segmentación de la imagen.

3.3 Detección Robusta de Objetos en Tiempo Real.

Presentado por [Viola & Jones 01] “*Robust Real-time Object Detection*” El método consiste en determinar una serie de características basadas en las sumas y restas de los niveles de intensidad en la imagen. Para ello se utilizan filtros de Haar de un cierto tamaño y calculados para las posiciones concretas de la sub-imagen que se quiere clasificar, como lo muestra la Figura 10.



Figura 10 Filtros aplicados a una imagen, las operaciones se realizan en los niveles de intensidad de los píxeles abarcados por cada filtro, en el caso de la imagen del centro, a las regiones de los ojos y las mejillas, a la imagen de lado izquierdo, a las regiones de los ojos y la nariz.

Se utiliza un algoritmo para construir una cascada de clasificadores la cual logra niveles altos de detección en un tiempo muy corto (0.7 segundos para una imagen de 384 x 288 en una PC Pentium III a 700Mhz). Las fases en la cascada se construyen entrenando clasificadores con el algoritmo AdaBoost, como se muestra en la Figura 11.



Figura 11. Descripción esquemática de una cascada de detectores. Una serie de clasificadores son aplicados a cada imagen, el clasificador inicial elimina un gran número de sub-ventanas negativas de la imagen con muy poco procesamiento. Los clasificadores siguientes eliminan sub-ventanas negativas adicionales de la imagen pero requieren cálculos más complicados. Después de varias etapas de clasificación el número de sub-ventanas es reducido hasta obtener un área positiva de la imagen que es donde se encuentra el objeto que se busca.

3.4 Detección Robusta de la Mano.

En esta investigación presentada por [Kölsch & Turk 04], se analiza el método de reconocimiento de objetos propuesto por Viola y Jones, enfocándolo al reconocimiento de la mano en posturas específicas.

Se propone un método para estimar de manera rápida el potencial de clasificación, basado solamente en pocas imágenes de entrenamiento para cada postura. Se analizaron ocho posturas, vistas desde un punto fijo, las cuales fueron seleccionadas por su facilidad de realización y por la diferencia entre cada una de ellas. Para cada postura se entrenó un detector en “cascada”, obteniendo como resultado un detector de mano con muy pocos falsos positivos. En resumen, esta investigación demuestra como el detector de objetos de Viola-Jones puede lograr rangos de detección muy aceptables cuando se utiliza para la detección de la mano.

3.5 Camshift

[Allen, Xu, Jin 06] Método que realiza la detección de un objeto basándose en el color que este tiene, el color es definido por el usuario, lo que crea un histograma el cual es usado por el algoritmo para reconstruir la imagen, y desechar cualquier cosa de esa imagen que no tenga el color exacto del objeto, logrando así una detección rápida. Las Figuras 12 y 13 muestran la ejecución del algoritmo Camshift.

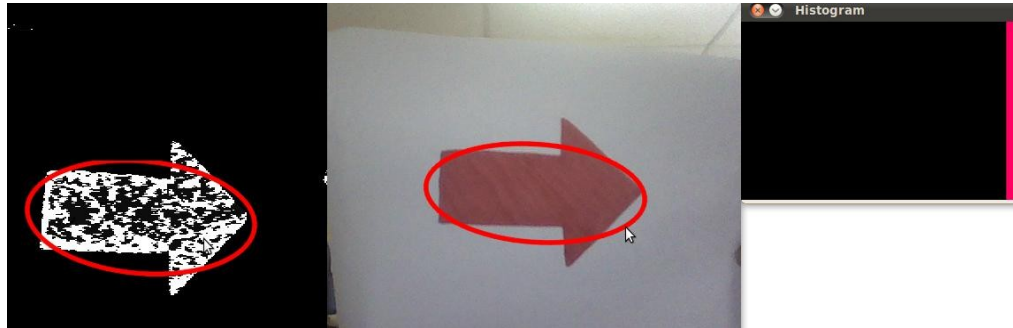


Figura 12. Ejecución de CamShift. La imagen del centro muestra el objeto que se quiere detectar, en este caso una flecha de color rojo, el usuario le indica la región de la imagen donde se encuentra el objeto, posteriormente se crea un histograma del color con el que está compuesto ese objeto, el cual se muestra en la parte derecha de la figura, el algoritmo reconstruye la imagen como se puede observar en la imagen de la izquierda ignorando cualquier pixel que no coincida con el color almacenado en el histograma del objeto.

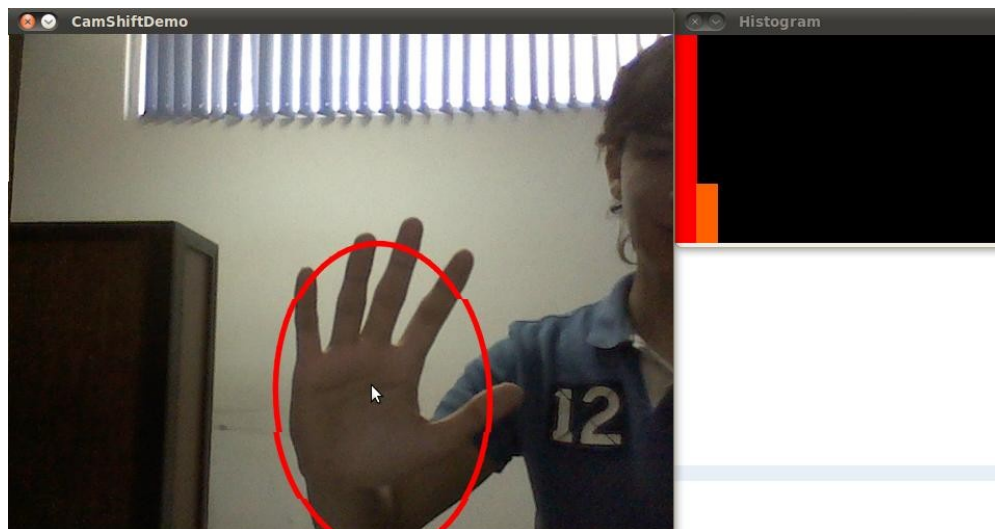


Figura 13. Detector CAMSHIFT en tiempo real. Se muestra en la imagen la ejecución del método CAMSHIFT, en la parte derecha se encuentra el histograma de color generado para la región de la mano dada.

Las ventajas de este detector son que es invariante a la rotación del objeto, no requiere de ningún entrenamiento previo para poder la realizar la detección del objeto.

3.6 Chamfer Matching

[Thayananthan , Stenger , Torr, Cipolla 03] Método que realiza la detección de un objeto por medio de comparaciones de contornos. El usuario proporciona al detector una imagen “muestra”, la cual contiene unicamente el objeto que quiere ser detectado, esta imagen es tratada con el algoritmo Canny, cuyo propósito es convertir esta imagen a una imagen binarizada, la cual contendrá unicamente los contornos de los objetos presentes en ella. Canny también es aplicado al ambiente observado por la cámara. El detector realiza una comparación de contornos de lo que contiene la imagen muestra y de lo que se encuentra en el ambiente en ese momento.

Para este trabajo se seleccionaron 3 detectores: Viola-Jones, CamShift y chamfer, los criterios utilizados para la selección de los mismos fueron su eficiencia y bajo costo computacional, se realizó una prueba con una base de datos de 200 imagenes para medir la eficiencia de cada uno de los detectores (Ver Tabla 1).

Tabla 1. Eficiencia de detectores. Se consideraron como factores de medición el numero de falsos positivos y negativos que se obtuvieron de los detectores al probarlo con cada imagen de la base de Datos.

Detector	Falsos Positivos	Falsos Negativos	% de Eficiencia
Viola-Jones	156	44	78.00%
CamShift	145	55	72.50%
Chamfer	161	39	80.50%

Capítulo 4 Método

4.1 The Reinforcement Learning Toolbox

Se decidió importar herramientas y métodos existentes para la implementación de algoritmos de aprendizaje. Se realizó una búsqueda de la mejor implementación de aprendizaje por refuerzo cuyo código fuera abierto. Después de diversas evaluaciones, el método escogido fue “The Reinforcement Learning Toolbox”. Este sistema contiene los algoritmos más usados de aprendizaje por refuerzo, con la cualidad de que permite una fácil modificación y adaptabilidad de nuevos algoritmos.

La estructura básica para una tarea de aprendizaje en RL-Toolbox (RLT) está compuesta de dos bloques principales: El *agente* y el *ambiente*. El agente tiene representaciones S_t del estado actual del ambiente. Basado en esta información el agente realiza una acción A_t , la cual ejecuta la función de transición del ambiente. La función de transición cambia el estado interno del ambiente, que es la información exacta del estado actual. El agente es informado del nuevo estado al recibir S_{t+1} . La Ilustración 10 muestra los bloques principales presentados por Gerhard Neumann.

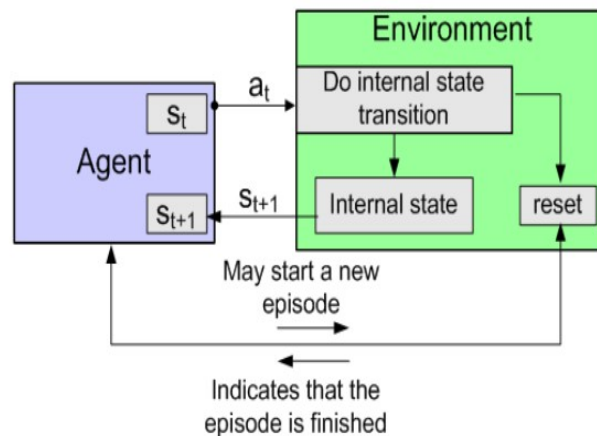


Ilustración 10. Bloques principales del modelo de aprendizaje por refuerzo en la Herramienta RLT, se observan los bloques del Agente y del Ambiente.

La función de recompensa en RLT fue diseñada para ser flexible. En lugar de tener una función global de recompensa como parte del ambiente, RLT introduce el concepto de *Listeners*, que son mantenidos por el agente. El agente puede tener varios *listeners*, cada listener recibe como entrada $[S_t, A_t, S_{t+1}]$. Los listeners de mayor relevancia son los llamados *reward listeners*, La recompensa es evaluada en cada paso, y el listener puede usar esta información adicional para actualizar su información interna (como la función de valor). Los listeners pueden entonces llegar a influenciar la selección de acción del agente, como se muestra en la Ilustración 11 presentada por Gerhard Neumann.

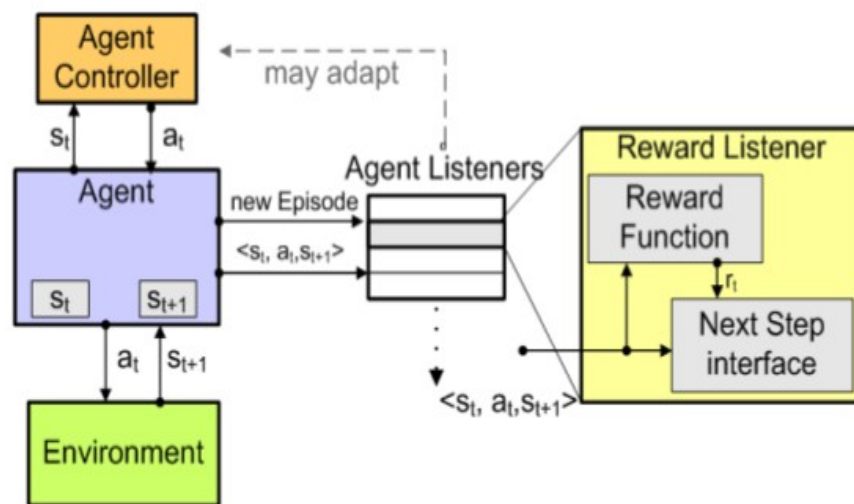


Ilustración 11. *Agent Listeners* como interface para algoritmos de aprendizaje.

4.2 Algoritmos de Aprendizaje

El aprendizaje puede ser online u offline. Online se refiere a que el aprendizaje se hace en vivo, por lo tanto las decisiones hechas por el algoritmo de aprendizaje indican lo que el agente hará en ese momento. Durante el aprendizaje Offline, las decisiones se basan en las transiciones estado-acción grabadas de una sesión anterior.

Los algoritmos de aprendizaje pueden o no hacer uso de la elegibilidad. Los algoritmos con una traza de elegibilidad usan la extensión (λ) en su nombre, siendo λ el factor de

descuento de las trazas de elegibilidad. Los algoritmos de aprendizaje escogidos para este trabajo son $Q(\lambda)$ y $SARSA(\lambda)$.

$SARSA(\lambda)$ toma en cuenta una política de exploración. Esto significa que si existe alguna recompensa negativa significativa, le dará prioridad a no elegir esa acción. Sin embargo $Q(\lambda)$ fue escogido para las sesiones de entrenamiento online y $SARSA(\lambda)$ para las sesiones offline. La razón para escoger $Q(\lambda)$ es por que permite al usuario interferir durante el entrenamiento, pudiendo en cualquier momento darle al agente una acción específica a realizar. Esto no sería posible con $SARSA(\lambda)$, ya que requiere de antemano conocer la acción que precede a la siguiente.

4.3 Conjunto de Estados

El conjunto de estados es representado por variables obtenidas de imagenes procesadas del ambiente en tiempo real por la cámara, siendo estas Iluminación y el Color. Para obtener cada uno de estos valores se realiza una conversión del modelo de colores RGB al modelo HSV, del cual se separan estos dos valores, obteniendo un valor general de cada variable.

4.3.1 Modelo de Componentes HSV

El modelo HSV (del inglés Hue, Saturation, Value – Matiz, Saturación, Valor), también llamado HSB (Hue, Saturation, Brightness – Matiz, Saturación, Brillo), define un modelo de color en términos de sus componentes:

Matiz: el tipo de color (como rojo, azul o amarillo). Se representa como un grado de ángulo cuyos valores posibles van de 0 a 360° (aunque para algunas aplicaciones se normalizan del 0 al 100%). Cada valor corresponde a un color. Ejemplos: 0 es rojo, 60 es amarillo y 120 es verde.

Saturación: Se representa como la distancia al eje de brillo negro-blanco. Los valores posibles van del 0 al 100%. Cuanto menor sea la saturación de un color, mayor tonalidad grisácea habrá y más decolorado estará.

Brillo: Representa la altura en el eje blanco-negro. Los valores posibles van del 0 al 100%. 0 siempre es negro. Dependiendo de la saturación, 100 podría ser blanco o un color más o menos saturado. La Ilustración 12 muestra el espacio de color del modelo HSV.

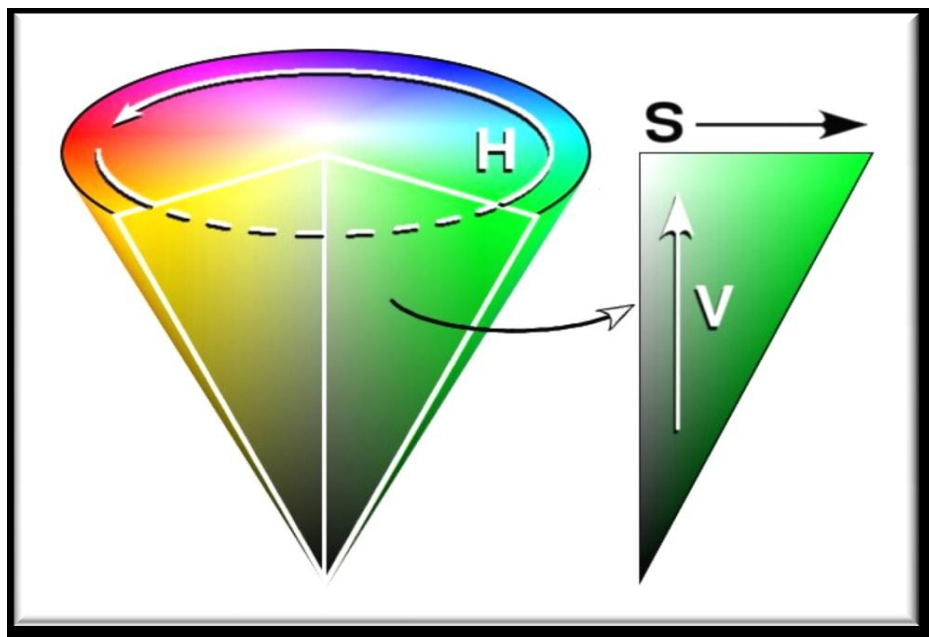


Ilustración 12. espacio de color del modelo HSB representado de manera cónica.

Brillo

Del modelo de componentes HSB (Hue, Saturation, Brightness – Matiz, Saturación, Brillo) se medirá el valor de Brillo en una imagen. Debido a que medir este valor usando todos los píxeles de la imagen consumiría muchos recursos en la computadora, se obtendrá un promedio del Brillo en la imagen, tomando valores de Brillo de los píxeles de ciertas regiones de la imagen.

Matiz

Del modelo de componentes HSB (Hue, Saturation, Brightness – Matiz, Saturación, Brillo) se medirá el valor de Matiz en una imagen. De la misma manera que se obtiene el brillo también es posible obtener el valor de Matiz en una imagen, la Ilustración 13 muestra la conversión de los colores de una imagen RGB al modelo HSB.



Ilustración 13. Conversión de colores RGB a HSB. Se pueden apreciar los diferentes valores del modelo HSB, Hue (Matiz), Saturation (Saturación), Brightness (Brillo).

4.4 Acciones

Las acciones del algoritmo de aprendizaje pueden hacer al agente realizar la selección de uno de los 3 detectores, es decir, las acciones son 3: Detector 1, Detector 2 y Detector 3.

4.5 Recompensas

Las recompensas son dadas en base a la indicación del usuario de que el Detector seleccionado por el agente fue el correcto o no.

Lo que hace este proyecto diferente de otros es que la recompensa no se sabe de antemano, son obtenidas en base a retroalimentación dada por el usuario.

4.6 Política

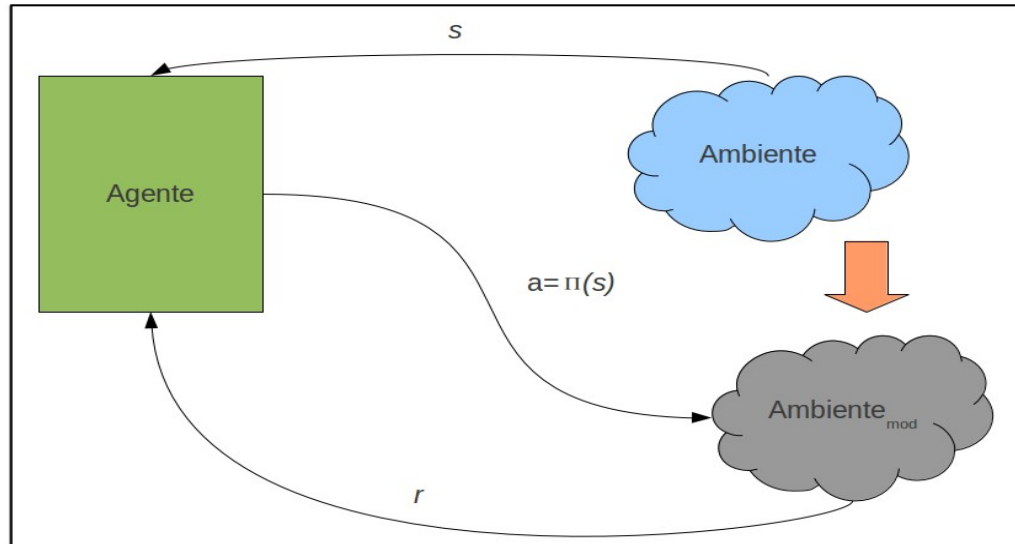
La formula softmax [Figura] será usada como política de exploración. El parametro β determina que tan aguda es la distinción entre una acción buena o mala. Para $\beta = 0$ la distribución será uniforme y para $\beta \rightarrow \infty$ la política será similar a la greedy, siempre escogerá la acción que tenga el valor mas alto. $Q(s,a)$ es la recompensa para la acción a en el estado s . La idea básica de la politica softmax es que si una acción da como resultado una recompensa alta, esa acción será elegida la mayoría de la veces por el agente, pero si varias acciones dan como resultado recompensas similares la política sera distribuida de manera mas equitativa entre esas acciones.

Por lo tanto, la exploración es alta si existe alta incertidumbre, y sera baja si es obvio cual accion es la mas favorable. De acuerdo a [referencia] no se sabe si la política epsilon-greedy es mejor que la polític softmax, y no se han encontrado estudio comparativos eficientes de estas reglas de acción-selección. Por lo tanto, de acuerdo a [referencia] la política tradicional de aprendizaje por refuerzo no funciona bien con el tipo de problemas que se afronta en este trabajo, por lo tanto fue escogida la política softmax para el mismo.

Ilustración 14

$$P(a_i) = \frac{e^{\beta * Q(s, a_i)}}{\sum_j e^{\beta * Q(s, a_j)}}$$

Del modelo de aprendizaje por refuerzo:



Se considera entonces:

- Un conjunto de estados S
- Un conjunto de acciones A
- Un comportamiento a aprender $\Pi: S \rightarrow A$

El conjunto de Estados estará compuesto por dos variables:

1.- Iluminación: Del modelo de color HSV, esta se maneja con un rango de 0-100%, pero debido a que estos valores en una imagen son representados con bytes, OpenCV realiza una transformación que determina un rango de 0 – 255.

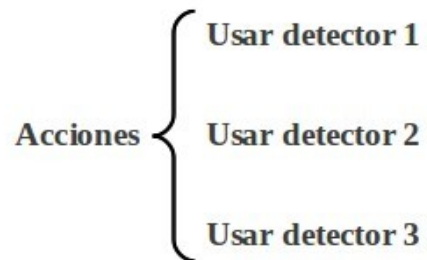
2.- Tinte: Del modelo de color HSV, esta se da en un rango de 0-360, pero debido a que estos valores en una imagen son representados con bytes, que almacenan un valor máximo de 256, OpenCV realiza una transformación que tiene un rango de 0-179 para

poder abarcar todos los colores disponibles.



El conjunto de acciones se modelará de la siguiente forma:

Cada acción del agente consistirá en el uso de un detector.



La función de recompensa otorgará:

(+1) si el agente se mantiene usando el mismo detector. Se busca que exista una continuidad en el uso de detectores, para que el agente se mantenga el mayor tiempo posible seleccionando el mismo detector, un cambio continuo de detectores limitaría la eficiencia de la detección por que se perdería mucho tiempo realizando un cambio de un detector a otro.

(-1) Si el agente pierde el objeto. De esta manera se da énfasis sobre el hecho de que se requiere que el agente trate de conservar la selección de un mismo detector el mayor tiempo posible.

(-0.5) Si selecciona otro detector, para propiciar que el agente elija el mismo detector el

mayor tiempo posible .

4.7 Autómatas finitos deterministas y no-deterministas

Un autómata finito determinista [4] (abreviado AFD) es un autómata finito que además es un sistema determinista; es decir, para cada estado $q \in Q$ en que se encuentre el autómata, y con cualquier símbolo $a \in \Sigma$ del alfabeto leído, existe siempre a lo más una transición posible $\delta(q,a)$.

En un AFD no pueden darse ninguno de estos dos casos:

- Que existan dos transiciones del tipo $\delta(q,a)=q_1$ y $\delta(q,a)=q_2$, siendo $q_1 \neq q_2$;
- Que existan transiciones del tipo $\delta(q,\epsilon)$, salvo que q sea un estado final, sin transiciones hacia otros estados.

Un autómata finito no-determinista[4] (abreviado AFND) es aquel que, a diferencia de los autómatas finitos deterministas, posee al menos un estado $q \in Q$, tal que para un símbolo $a \in \Sigma$ del alfabeto, existe más de una transición $\delta(q,a)$ posible.

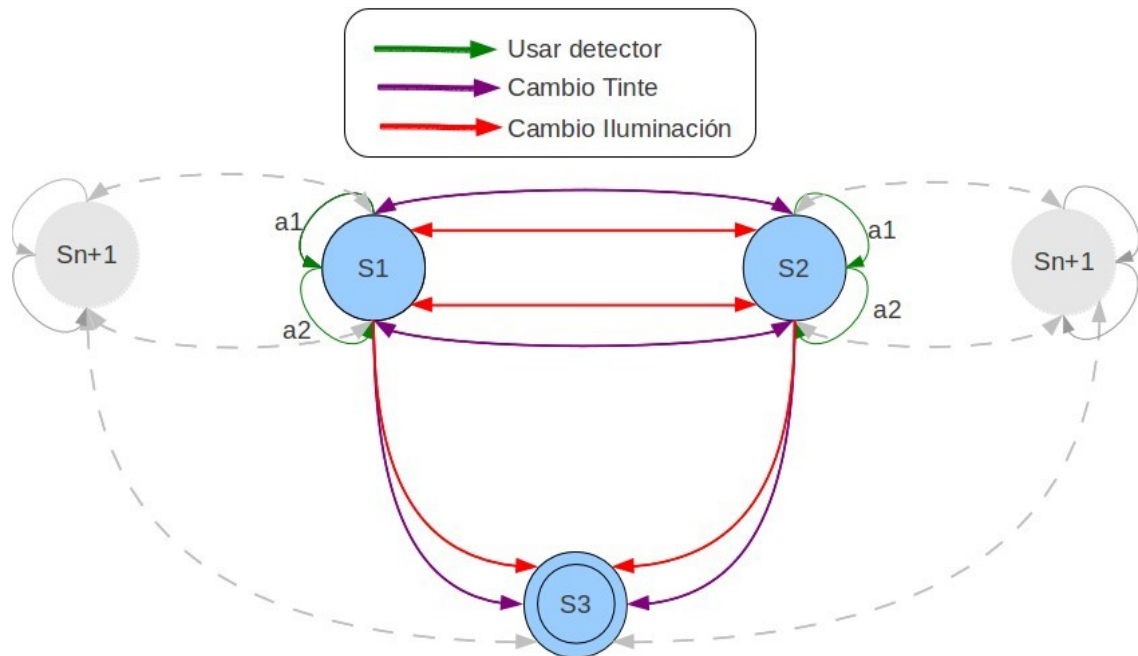
Haciendo la analogía con los AFDs, en un AFND puede darse cualquiera de estos dos casos:

- Que existan transiciones del tipo $\delta(q,a)=q_1$ y $\delta(q,a)=q_2$, siendo $q_1 \neq q_2$;
- Que existan transiciones del tipo $\delta(q,\epsilon)$, siendo q un estado no-final, o bien un estado final pero con transiciones hacia otros estados.

En esta investigación el problema que se presenta es de tipo determinista, debido a que cualquier acción que se presente en el ambiente solo puede llevar a un estado específico del ambiente y no a varios estados posibles.

4.8 Ejemplo determinístico:

El problema que se presenta en esta investigación al ser de tipo determinista se puede representar, de manera simplificada, con el siguiente grafo:



Se presenta un estado S1 el cual está definido por las variables Iluminación y Tinte, para ese estado se escogerá una acción la cual será elegir un detector, mientras no haya un cambio significativo en el ambiente la acción debe seguir siendo la misma. Un cambio en el ambiente está dado por Cambios en Iluminación o Tinte, de manera que cuando se presente un cambio significativo en cualquiera de estas 2 variables, el agente se encontrará en un estado diferente S2 donde seleccionará de nuevo un detector adecuado para ese estado.

Si se da el caso de que ningún detector encuentre la mano aunque esta se encuentre presente en el ambiente, entonces el agente se encontrará en un estado terminal S3 que corresponderá a que no hay ninguna acción disponible para ese estado.

4.9 Propiedad de Markov

Se dice que un proceso cumple la propiedad de Markov[5] si el cálculo de los estados futuros depende solo del estado actual, es decir, dado el estado actual, el estado futuro no depende de los estados anteriores. Un proceso con esta propiedad se denomina Markoviano o Proceso de Markov.

En nuestro caso, el cálculo del estado siguiente no está dado por los estados anteriores ya que estos no influyen de ninguna manera en los cambios que el ambiente presenta, por lo que se determinó que el problema que aborda esta investigación es Markoviano.

4.10 Medición de cambios en las variables de Estado

Debido a que se necesita determinar cuando ha cambiado el ambiente, se realizaron programas que miden el promedio general de Iluminación y el promedio general de Tinte, estos programas se deberán estar ejecutando en todo momento, de esta manera cuando se presente un cambio significativo en el ambiente se le podrá notificar al algoritmo sobre este cambio.

CAPITULO 5 IMPLEMENTACION

Este capítulo describe la implementación de las ideas presentadas en el capítulo anterior, dando a conocer la arquitectura de hardware y software utilizado, así como su diseño.

5.1 Hardware

El Hardware utilizado para este proyecto es Hardware comercial standard, consiste en:

- Toshiba Core 2 Duo T6600 2.2 Ghz, 6GBDDR2 RAM.
- Web Camera integrada.

5.2 Software

El software implementado esta basado en C++, desarrollado para Linux Ubuntu OS con NetBeans IDE. Para el desarrollo de la interfaz del usuario se utilizo Qt4. Los algoritmos de aprendizaje usados son de la librería abierta, The Reinforcement Learning Toolbox [referencia]. Los 3 detectores utilizados [referencia] son de código abierto.

5.2.1 Detector 1: Viola-Jones

Implementado en la librería OpenCv para C++, se hizo una revisión de sus capacidades para detectar diversos objetos en tiempo real.

Para lograr la detección de la mano se requiere que este detector sea entrenado, esto se realizó de la siguiente manera:

a) Preparación de Datos:

-Recolección de imágenes positivas: Una imagen positiva es la que contiene el objeto que se quiere detectar, en este caso, la mano. Lo recomendable para un buen entrenamiento son 1000+ imágenes positivas.

Para la recolección de imágenes positivas es recomendable que los píxeles de fondo no sean mayores a los que tiene el objeto a ser detectado, ya que el detector podría recordar estos como parte de las características del objeto que se quiere detectar, sin embargo, tampoco es recomendable que no se tenga ningún píxel de fondo. Por lo tanto se tiene que tener una pequeña cantidad de píxeles en el fondo.

-Recolección de imágenes negativas: Una imagen negativa es la que No contiene el objeto que se quiere detectar, se recomienda que las imágenes sean variadas, que no contengan un solo objeto si no ambientes con diferentes variedades de objetos.

La proporción de imágenes positivas:negativas = 1:2 , por lo que se recolectaron 1500 imágenes positivas y 3000 negativas.

b) Object Marker:

Una vez que se han recolectado todas las imágenes positivas que se utilizaran en el entrenamiento, es necesario que de cada una sea tratada de manera individual, indicando la región de la imagen donde se encuentra la mano, esto se realiza con una herramienta implementada en OpenCv llamada *ObjectMarker*, la cual crea un archivo de texto en donde guarda la ruta de la imagen donde se encuentra cada una de las imágenes positivas que se están utilizando.

Al ejecutar el programa este abre una por una las imágenes positivas, el usuario entonces tiene la posibilidad de indicar en qué región se encuentra el objeto, el programa guarda las

coordenadas de esa región en el archivo de texto generado, una vez terminado el tratamiento de esa imagen el programa la guarda y abre la siguiente imagen para que pueda ser tratada de la misma manera. Este es un procedimiento tardado ya que cuando se trabaja con bases de datos de más de 100 imágenes, tiende a ser un trabajo muy tedioso y tardado.

La Ilustración 17 muestra la ejecución del Programa ObjectMarker.

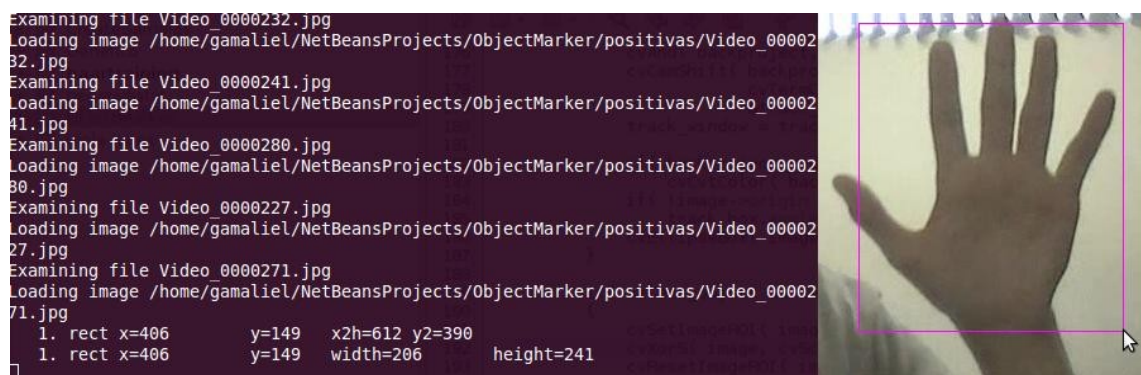


Ilustración 17. Object Marker en ejecución. El programa carga cada una de las imágenes que se encuentran almacenadas como positivas, el usuario le indica por medio de un rectángulo el área de la imagen donde se encuentra el objeto, el programa guarda en un archivo de texto las coordenadas que se obtienen del rectángulo indicado por el usuario.

c) Creación de Muestras:

La herramienta *CreateSamples* permite al usuario crear muestras del objeto que se seleccione en cada imagen con la herramienta *ObjectMarker*, de esta manera durante la generación del archivo de muestras únicamente la región que está indicada en el archivo de texto generado anteriormente es tomada, no se utiliza toda la imagen, únicamente las coordenadas donde se encuentra la mano. Este proceso permite acelerar el proceso de entrenamiento.

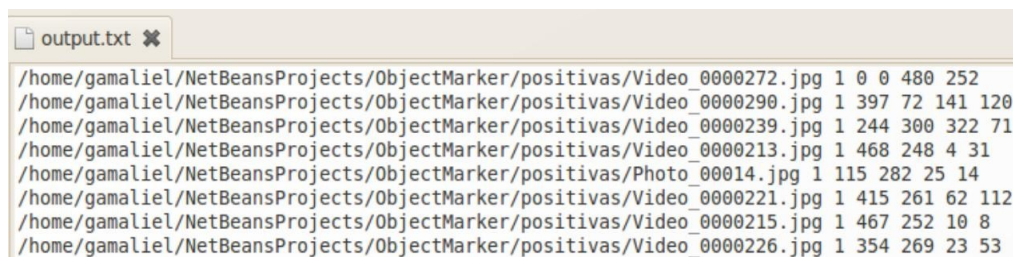
Como resultado, todas las imágenes positivas son convertidas a un mismo formato en un archivo generado con las siguientes características:

Nombre_de_imagen_positiva

numero_de_objetos x y

largo_y_ancho x y

La Ilustración 18 muestra el archivo de texto generado.



```
output.txt ✕
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Video_0000272.jpg 1 0 0 480 252
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Video_0000290.jpg 1 397 72 141 120
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Video_0000239.jpg 1 244 300 322 71
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Video_0000213.jpg 1 468 248 4 31
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Photo_00014.jpg 1 115 282 25 14
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Video_0000221.jpg 1 415 261 62 112
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Video_0000215.jpg 1 467 252 10 8
/home/gamaliel/NetBeansProjects/ObjectMarker/positivas/Video_0000226.jpg 1 354 269 23 53
```

Ilustración 18. Archivo de texto generado en el cual se observa la ruta completa donde se encuentra la imagen positiva, seguida de un espacio, el número de objetos que se seleccionó de cada imagen, y el largo y ancho de ese objeto.

d) Entrenamiento

Cuando el archivo de muestras es generado, se necesita crear otro archivo con la ruta de las imágenes negativas que se recolectaron, este es un archivo de texto que debe contener la ruta exacta de cada una de las imágenes negativas, como se muestra en la Ilustración 19.

```
negativasACT.txt ✕
NetBeansProjects/ObjectMarker/negativas/neg-0002.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0003.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0004.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0005.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0006.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0007.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0009.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0010.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0011.jpg
NetBeansProjects/ObjectMarker/negativas/neg-0012.jpg
```

Ilustración 19. Archivo de Texto que contiene la ruta completa donde se encuentran todas las imágenes negativas que se utilizaran para el entrenamiento del detector.

El entrenamiento del detector se realiza con la herramienta *HaarTraining*.

La Ilustración 20 muestra una captura del detector de Viola-Jones ejecutándose en tiempo real.



Ilustración 20. Captura del Detector de Viola-Jones. La imagen muestra la detección realizada por el detector de Viola-Jones, enmarcada por el programa con un cuadro color violeta.

5.2.1 Detector 2: CamShift

Método que realiza la detección de un objeto basándose en el color que este tiene, el color es definido por el usuario, lo que crea un histograma el cual es usado por el algoritmo para reconstruir la imagen, y desechar cualquier cosa de esa imagen que no tenga el color exacto del objeto, logrando así una detección rápida. Las Ilustraciones 21 y 22 muestran la ejecución del algoritmo Camshift.

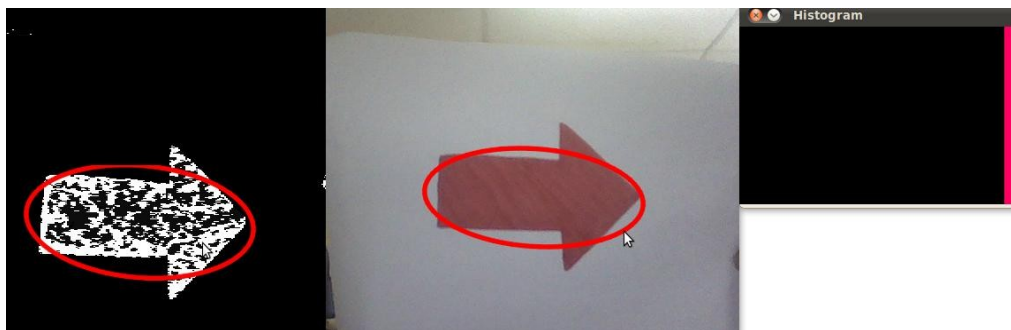


Ilustración 21. Ejecución de CamShift. La imagen del centro muestra el objeto que se quiere detectar, en este caso una flecha de color rojo, el usuario le indica la región de la imagen donde se encuentra el objeto, posteriormente se crea un histograma del color con el que está compuesto ese objeto, el cual se muestra en la parte derecha de la figura, el algoritmo reconstruye la imagen como se puede observar en la imagen de la izquierda ignorando cualquier pixel que no coincida con el color almacenado en el histograma del objeto.

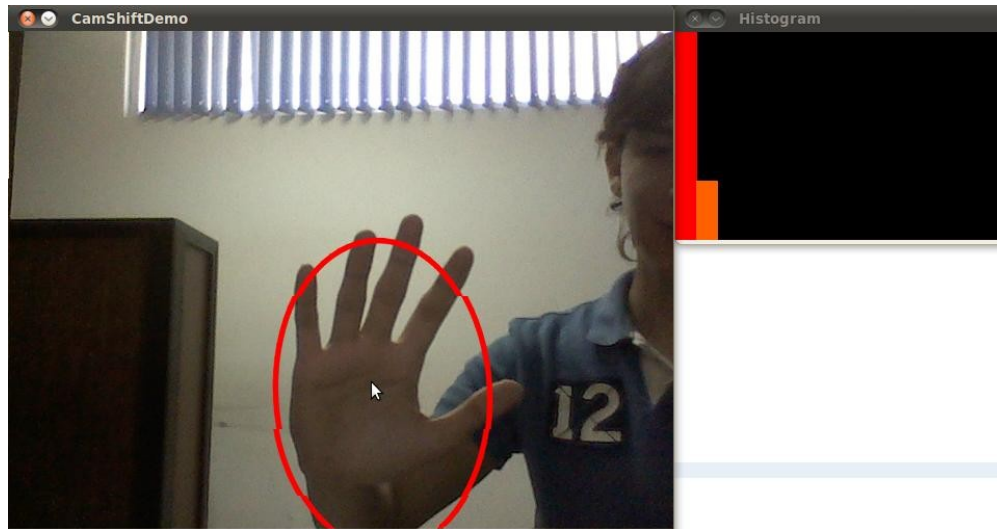


Ilustración 22. Detector CAMSHIFT en tiempo real. Se muestra en la imagen la ejecución del método CAMSHIFT, en la parte derecha se encuentra el histograma de color generado para la región de la mano dada.

Las ventajas de este detector son que el objeto no tiene que estar en una posición específica para que pueda ser detectado, este detector no requiere de ningún entrenamiento previo para poder la realizar la detección del objeto.

5.2.2 Detector 3: Chamfer Matching.

Método que realiza la detección de un objeto por medio de comparaciones de contornos. El usuario proporciona al detector una imagen “muestra”, la cual contiene únicamente el objeto que quiere ser detectado, esta imagen es tratada con el algoritmo Canny, cuyo propósito es convertir esta imagen a una imagen binarizada, la cual contendrá únicamente los contornos de los objetos presentes en ella. Canny también es aplicado al ambiente observado por la cámara. El detector realiza una comparación de contornos de lo que contiene la imagen muestra y de lo que se encuentra en el ambiente en ese momento.

Se realizó una modificación de este detector para que realizará la detección con imágenes en tiempo real y no solo con imágenes estáticas.

Las Ilustraciones 23,24, 25 y 26 muestran el procedimiento realizado por el método Chamfer Matching.



Ilustración 23. La figura corresponde a la imagen que se le proporciona al programa como muestra del objeto que se debe detectar en el ambiente.

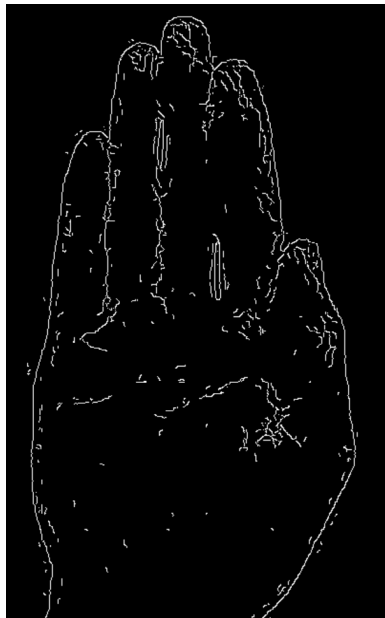


Ilustración 24. En la figura se puede observar el resultado de haber aplicado el algoritmo Canny a la imagen muestra.

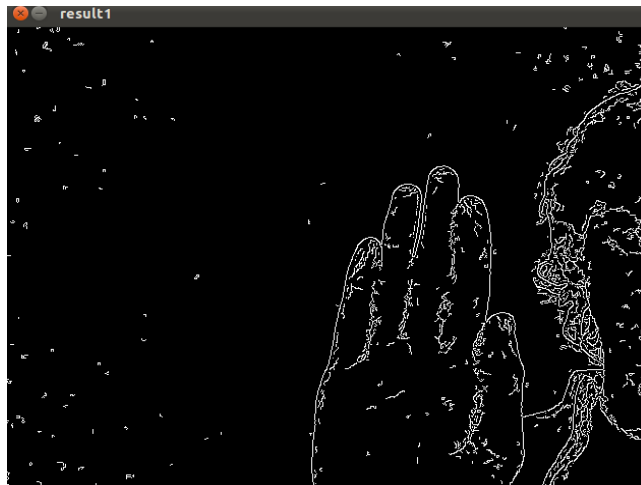


Ilustración 25. Se observa el resultado de haber aplicado el algoritmo Canny a la imagen capturada por la cámara en el momento de la ejecución del programa.

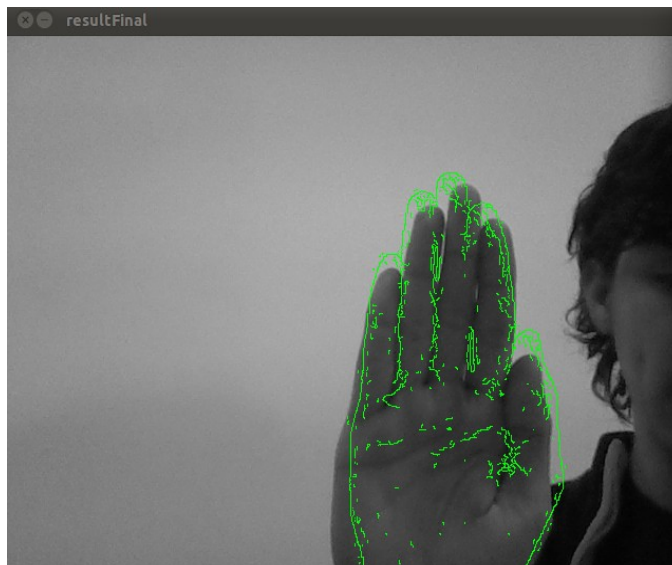


Ilustración 26. Se observa el resultado final del algoritmo, la imagen sobrepuesta (compuesta por líneas verdes) se coloca sobre el área de la imagen que contiene patrones similares a la imagen muestra.

5.3 Hilos de Ejecución

Un hilo de ejecución es una característica que permite a una aplicación realizar varias tareas concurrentemente. Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente, por ejemplo si programamos videojuegos podemos tener un hilo para la Inteligencia Artificial, otro para la física y otro para el renderizado de los gráficos, pero todos ellos se ejecutarán de manera paralela.

Para poder lograr la ejecución de múltiples detectores es necesario utilizar Hilos para cada uno de estos procesos, ya que accesan a recursos similares como la cámara, la cual no puede ser utilizada más de una vez al mismo tiempo. El uso de Hilos nos permite compartir variables para que cada programa pueda acceder a ellas conforme lo van necesitando.

Para este proyecto se requirió obtener conocimientos básicos en el manejo de Hilos en lenguaje C++, se realizaron pruebas con librerías destinadas al manejo de Hilos en C++ obteniendo mejores resultados con la librería Glib.

5.4 Glib

Glib es una biblioteca de propósito general que se usa para implementar muchas funciones no gráficas. Aunque GTK+ lo necesita, Glib puede usarse de forma independiente. Por eso, algunas aplicaciones usan Glib sin usar la biblioteca GTK+.

Uno de los mayores beneficios de usar Glib es que provee una interfaz de plataforma

independiente que permite que el código pueda ser usado en diferentes sistemas operativos. Otro aspecto de Glib es la amplia gama de tipo de datos que deja disponible al desarrollador.

- GLib provee tipos de datos en C que usualmente se incluyen de forma estándar a otros lenguajes, como por ejemplo listas enlazadas. Otros tipos de datos básicos incluidos son colas doblemente enlazadas, árboles binarios autobalanceados, etc.
- Las cadenas de caracteres en GLib son similares a las de C++, porque son buffers de texto que crecen automáticamente cuando se agregan datos. Son fáciles de integrar a la familia de funciones printf().
- Los segmentos de memoria (memory slice) son una forma eficiente de crear secciones de memoria que tienen todos el mismo tamaño. Pueden ser usados para crear vectores aún de elementos de tamaño conocido.
- Las cachés (Hash Tables desde la versión 2.32) son usadas para compartir grandes y complejas estructuras de datos en una sencilla API, lo cual ayuda a ahorrar espacio. Se usan en GTK+ para los estilos y contextos gráficos, ya que estos objetos consumen muchos recursos.

GLib además de proporcionar varios tipos de datos, también dispone de numerosos tipos de funciones. Se encuentran funciones de manipulación de archivos, soporte de internacionalización, cadenas de caracteres, advertencias, banderas de depuración, carga dinámica de módulos, sólo por nombrar algunas.

GLib maneja funciones que pueden ser llamadas cuando el procesador no está haciendo nada en la aplicación. También puede llamar funciones en un intervalo arbitrario de tiempo.

5.4 Diseño de la aplicación con RL Toolbox

5.4.1 Creando el ambiente

Se crea el ambiente donde el agente actuará y en el cual se describen las transiciones de estados. De la clase principal de RL Toolbox (RLT) se deriva la clase `EnvironmentModel`. El usuario debe proporcionar los métodos que realizarán la transición de estados, el reseteo del modelo, y la conversión del modelo de estados al objeto de estados. Estas funcionalidades se realizan con 3 funciones que deben ser re-declaradas.

La función `doNextState(CPrimitiveAction *)` Calcula la transición de estados. Para indicar que el modelo debe ser reseteado después de algún paso, se debe activar la bandera `reset`, lo que indica que el episodio actual resultó fallido.

La función `getState(CState *state)` permite al agente obtener el estado actual. Por lo tanto las variables internas deben estar contenidas en el estado objeto.

La función `doResetModel()` es donde se resetea las variables internas del modelo.

Nuestro método contiene 2 variables discretas y 0 variables continuas, la iluminación y el tinte, que componen las variables del ambiente. La cabecera donde se declaran estas variables se encuentra en el archivo **`cmultipole.h` y `cmultipole.cpp`**.

CENVIRONMENTMODEL (DECLARACION DE VARIABLES)

En el constructor de la clase del modelo se deben establecer las propiedades del modelo del estado. Las propiedades del modelo del estado especifican cuantas variables discretas y continuas tiene nuestro modelo (en nuestro caso solo 2 variables discretas). Adicionalmente se definen los valores mínimos y máximos de las variables discretas. Cuando se utilizan estados discretos, se debe establecer el tamaño de los mismos.

CENVIRONMENTMODEL (limites de las variables discretas)

5.4.2 Reseteo del Modelo

Es la parte mas sencilla del ambiente. Se llama al método *doResetModel* cada vez que el agente ha fallado o cuando el usuario quiera empezar un nuevo episodio (esto se hace con el método del agente *startNewEpisode*), por lo tanto, se necesita re-definir esta función. Nuestra implementación define todas las variables a *cero*.

DoResetModel

5.4.3 Obteniendo el Estado

El agente y el algoritmo de aprendizaje necesitan saber el estado actual, por esa razón se debe proveer el método *getState(CState *state)*. En este método, el estado actual está escrito en el estado objeto. La clase *Cstate* es una interfaz universal para los estados en RL. Un estado objeto puede contener un número arbitrario de estados continuos y discretos. Las propiedades del estado objeto, las cuales son enviadas a la función *getState* ya han sido especificadas en el constructor del modelo (así como también las 2 variables discretas y 0 continuas).

Todo lo que resta hacer es establecer las variables discretas del estado objeto. Esto se hace con el método *setDiscreteState(int dim, double value)*.

5.4.4 Realizando una Acción

Las acciones son ejecutadas en el método *doNextState(CPrimitiveAction *action)*. Pero antes de realizar alguna acción se implementa nuestra propia clase para la acción.

5.4.5 Derivando la clase para la acción

En nuestro método se tienen 3 acciones, 1 por cada detector que se debe utilizar. La acción se guarda en una variable la cual es mandada al método *doNextState(CPrimitiveAction *action)* cada vez que se quiere realizar una acción. Por lo tanto se debe derivar la clase *CprimitiveAction* y agregarle esa variable de acción. Alternativamente se pueden utilizar acciones discretas (clase *CdiscreteAction*), pero por simplicidad utilizamos solo una clase individual para todas las acciones.

5.4.6 Especificando la función de transición: El método *doNextState*.

Siempre que el agente quiera realizar una acción, se llama al método de la clase del ambiente *doNextState*, con la acción como argumento. Esta función debe implementar la transición interna de estados. Si el modelo necesitara ser reseteado en el siguiente turno (se perdió el objeto a detectar), se debe activar la bandera *reset*, si se quiere indicar también que el episodio ha fallado, se debe activar la bandera *failed*. En nuestro caso el modelo es reseteado cada vez que la mano se perdió, en este caso el agente falló al seleccionar un detector adecuado para las condiciones del ambiente que se presentaban en ese momento.

5.4.7 Discretización del Estado

Se utilizará una representación standard de estado discreto. Nuestra representación de estado discreta tiene X estados. La iluminación se divide en X particiones, el tinte en X particiones. Esto nos da un tamaño de estado discreto de $X * X = ?$. La desventaja de este particionado es que el problema pierde su propiedad de Markov en algunos casos, por lo tanto el algoritmo de aprendizaje puede no llegar a

converger.

En RL Toolbox se tienen 2 posibilidades para discretizar nuestro estado. Se pueden utilizar las clases ya incluidas para discretizar estados, las cuales nos ofrecen suficiente funcionalidad para la mayoría de las aplicaciones, o para algunos casos especiales cuando se necesita usar clases de discretización individuales, se debe derivar el discretizador de la clase *AbstractStateDiscretizer* e implementar el método *getDiscreteStateNumber*, el cual debe regresar el índice del estado discreto. Pero en esta investigación se utilizaron las clases ya incluidas en RL Toolbox.

5.4.8 Utilizando las Clases incluidas en RL Toolbox

Con las clases incluidas en RL Toolbox se tiene la posibilidad de discretizar variables continuas del modelo de estado individualmente (con *CsingleStateDiscretizer*). Se debe proveer las particiones que contendrán cada variable de estado continua como un arreglo doble. Estos estados discretos, que discretizan solo una variable de estado, pueden entonces ser combinados por el operador “and” (*CdiscreteStateOperatorAnd*) en una variable discreta global. En nuestra función principal se deben generar las clases de discretización.

La clase para discretizar estados individuales toma el índice de la variable de estado continua, el tamaño del arreglo partición y el arreglo partición como tal como argumentos. Debido a que solo se almacenan los límites de las particiones en el arreglo partición, el tamaño del estado discreto del discretizador de estado individual será el tamaño del arreglo partición más 1. Después de que se generaron los discretizadores de estados individuales, se crea nuestro operador *and* para combinarlos. Cuando la construcción del discretizador ha terminado, es esencial para arquitectura del estado que el discretizador sea agregada a la lista del

modificador del agente. Ningun modificador puede ser usado para el aprendizaje si no ha sido agregado a la lista del modificador del agente.

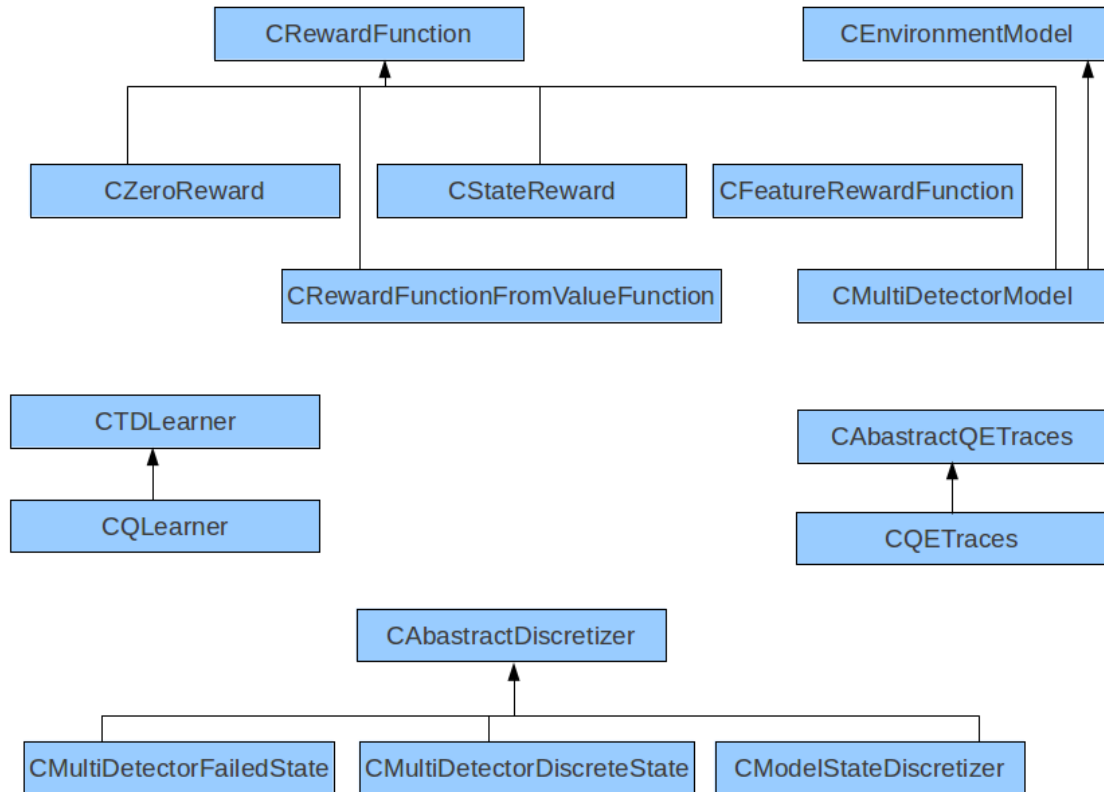
5.4.9 Calculando La Recompensa

En nuestro método se implementa la función de recompensa en la clase del modelo del ambiente, aunque la función de recompensa también puede ser implementada en una clase independiente. En la clase derivada función de recompensa se debe implementar la función *getReward(CStateCollection *oldState, CAction *action, CStateCollection *newState)*.

La función de recompensa recibe el estado anterior (como un objeto *CstateCollection*), la acción actual y el nuevo estado como argumentos. Un objeto *state collection* es una colección de objetos del estado, que contiene el modelo del estado y todos los estados que vienen de los modificadores de estado que han sido agregados a la lista de modificadores del agente. En nuestro caso se quiere obtener el modelo del estado de la colección de estados, así que se utilizará las propiedades de estado del modelo del ambiente como argumento para el método *getState*. Si especificamos un apuntador NULL como argumento para el método *getState*, siempre se obtiene el modelo del estado.

La función de recompensa otorga un 0.5 si al cambiar el ambiente el detector actual continua detectando correctamente la mano. Se otorga una recompensa de 0.0 si al cambiar el ambiente se cambia de detector por que el anterior no fue capaz de encontrar la mano. Se otorga un -1.0 si al cambiar el ambiente ningun detector es capaz de encontrar la mano.

5.4 Diseño de la aplicación RL'Toolbox

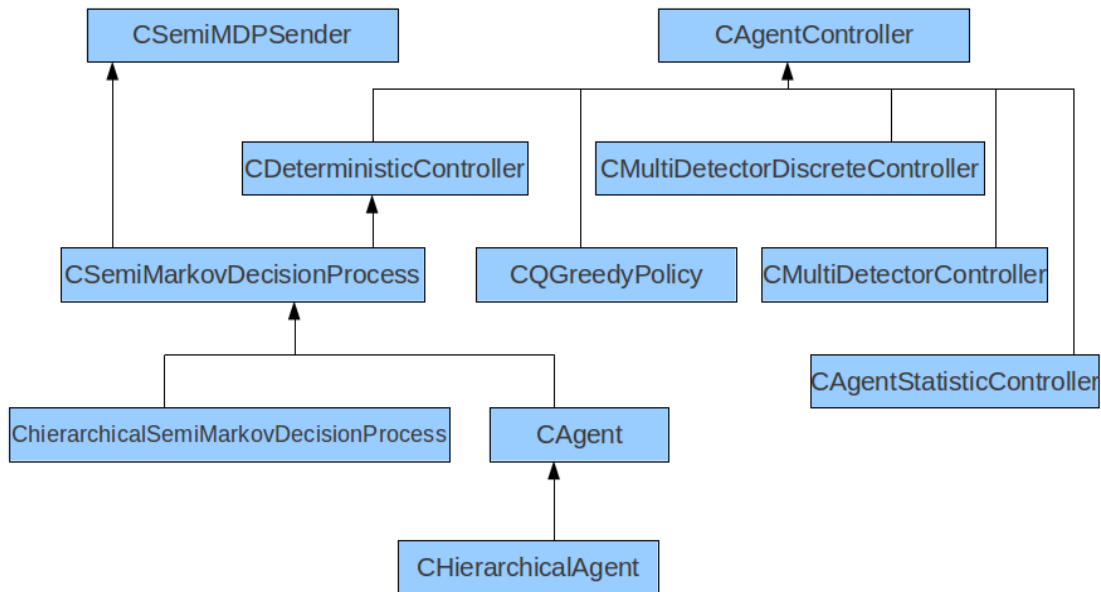


Los siguientes diagrama de clases muestra el diseño de la aplicación:

De este diagrama, la clase más importantes es la que modela el ambiente, llamada **CMultiDetectorModel**, que se describe a continuación:

CmultiDetectorModel:

Aquí se registran las variables del modelo, que son el promedio de Tinte y el promedio de Iluminación, dentro de esta clase se recibe el estado actual del ambiente y el siguiente estado del ambiente, que es determinado por una función que estará sensando en todo momento los cambios que se presenten en el ambiente, determina también la recompensa.



Del diagrama anterior, las clases mas relevantes son:

CAgent: Que indica al controlador del agente la acción que se realizará.

CAgentController: Aplica la función Q y Controla las acciones del agente indicando cual será la siguiente acción a realizarse.

5.5 Interfaz de Usuario

La interfaz de usuario implementada fue desarrollada con Qt4 en C++.

5.6 Qt4

Qt es una biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con una interfaz gráfica de usuario así como también para el desarrollo de programas sin interfaz gráfica como herramientas para la línea de comandos y consolas para servidores.

Qt es utilizada en KDE, un entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros. Qt utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de *bindings*. También es usada en sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos como frigoríficos.

Funciona en todas las principales plataformas, y tiene un amplio apoyo. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales.

Distribuida bajo los términos de GNU Lesser General Public License, Qt es software libre y de código abierto.

Es producido por la división de software Qt de Nokia, que entró en vigor después de la adquisición por parte de Nokia de la empresa noruega Trolltech, el productor original de Qt, el 17 de junio de 2008. Nokia anuncio que va a detener sus desarrollos en Symbian para usar la plataforma de Microsoft para sus telefonos inteligentes en Febrero de 2011. En marzo del mismo año, anunciaron la adquisición de las licencias comerciales de Qt por Digia PLC, aunque Nokia continúa como la fuerza de desarrollo principal detrás de la librería.

Su función es el control general del proceso de aprendizaje, permite monitorear el desempeño en vivo de cada uno de los 3 detectores y los valores actuales de las variables de estado. La Ilustración 28 muestra la interfaz desarrollada con Qt4

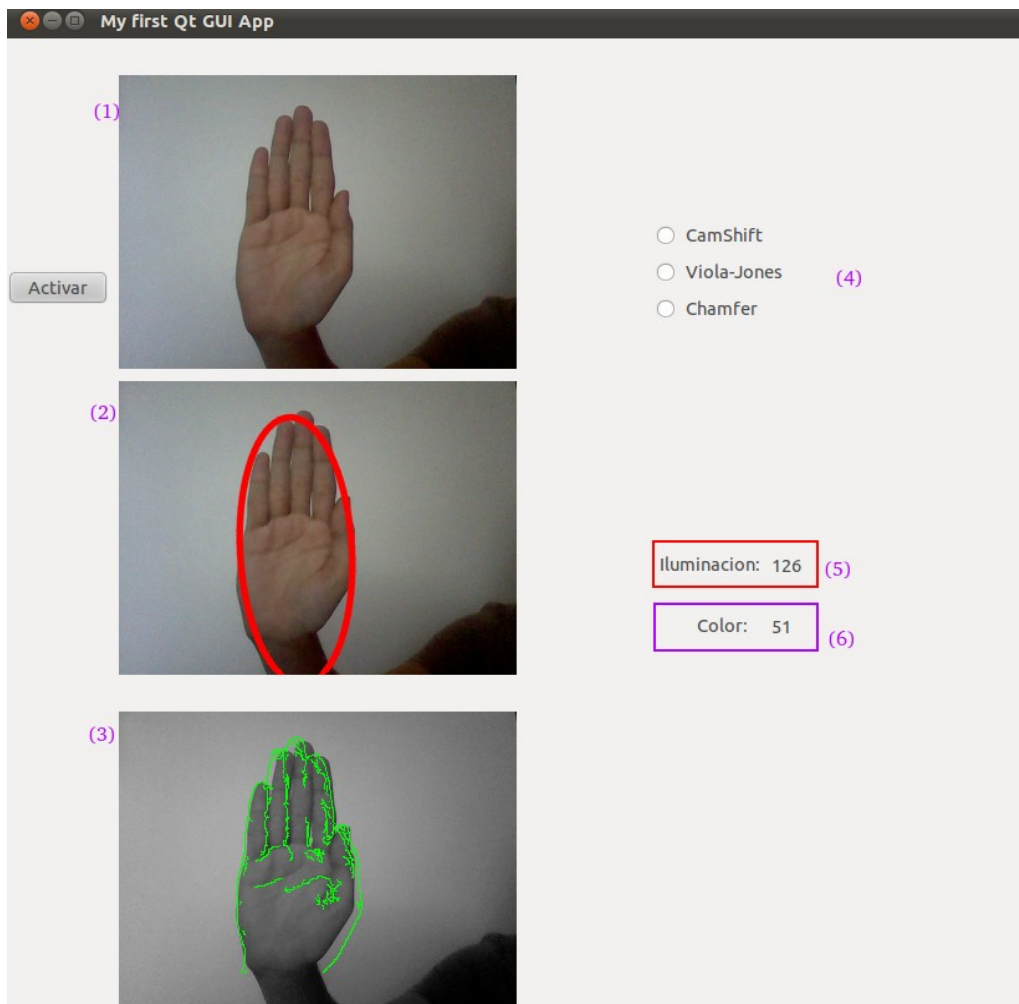


Ilustración 28.(1) Detector Viola-jones; (2) Detector CamShift; (3) Detector CamShift; (4) Botones de Control, sirven para indicarle al sistema que detector es el que funciona bien en el momento de entrenamiento; (5) Medidor de Iluminación, indica el valor general de iluminación del ambiente; (6) Medidor de Tinte, indica el color general de la imagen.

CAPITULO 6 EVALUACION EXPERIMENTAL

Este capítulo abarca todas las pruebas y evaluaciones que se realizaron en el sistema implementado. También se muestran los resultados de las pruebas y se mencionan las posibles mejoras que se podrían hacer al sistema.

6.1 Metodo de Evaluación

Hubo diferentes experimentos realizados en el sistema desarrollado. Dado que el objetivo de este trabajo fue que el sistema pudiera elegir cual de los detectores es el adecuado para el ambiente que se presenta, se realizaron pruebas en diferentes tipos de ambientes.

6.2 Proceso

Para poder evaluar la eficiencia se definió el siguiente proceso de evaluación:

- 1.- Se definió como ambiente para realizar las pruebas del sistema el laboratorio de visión por computadora.
- 2.- La Iluminación en el Ambiente puede ser modificada por el usuario.
- 3.- El Tinte Del ambiente tambien puede ser modificado por el usuario.

Se realizarón las pruebas con diferentes versiones del mismo ambiente, siendo este modificado por el usuario de manera controlada.

6.3 Conclusión

La conclusión de la prueba de aprendizaje es que el sistema implementado parece funcionar. Es muy importante entender algunos problemas que se presentaron al realizar

las pruebas. El problema principal es que el objetivo de los algoritmos de aprendizaje es aprender un comportamiento, lo que significa que se realiza una acción dependiendo del ambiente, lo que lleva a una modificación del ambiente, sin embargo, en este proyecto una acción no lleva a un ambiente específico diferente.

Como es normal, los experimentos realizados en tiempo real con algoritmos de aprendizaje por refuerzo llegan a presentar varias dificultades. El problema principal es que el sistema necesita un entrenamiento offline adecuado para poder presentar resultados al ser ejecutado. Al no ser un problema abordado anteriormente, el proyecto sufrió diversos retrasos como lo fue la adaptación de un algoritmo de aprendizaje por refuerzo a un problema nuevo.

La optimización hecha con la webcam también consumió mucho tiempo, ya que no existían antecedentes de varios procesos que hicieran el uso de la webcam al mismo tiempo para realizar una tarea diferente cada uno, lo más común es utilizar 1 cámara por cada proceso que requiera su uso, pero esto hubiera implicado el uso de 5 cámaras diferentes, lo que no era posible, por lo que se tuvo que encontrar una manera de racionar recursos de la computadora para poder utilizar solo 1 webcam para los 5 procesos.

6.4 Posibles Mejoras

La carga de trabajo de la computadora al ejecutar el sistema es muy pesada, por lo tanto solo es óptima su ejecución con procesadores actuales con más de 1 core, sin embargo, puede ser ejecutada en cualquier procesador aunque este cuente con 1 solo core. Para lograr la mejora del consumo de recursos del sistema es posible optimizar utilizando diferentes métodos de ahorro de recursos de la computadora, como lo es el uso de hilos directamente sobre cada proceso ejecutado en OpenCv.

APENDICE A INTERFAZ DE USUARIO

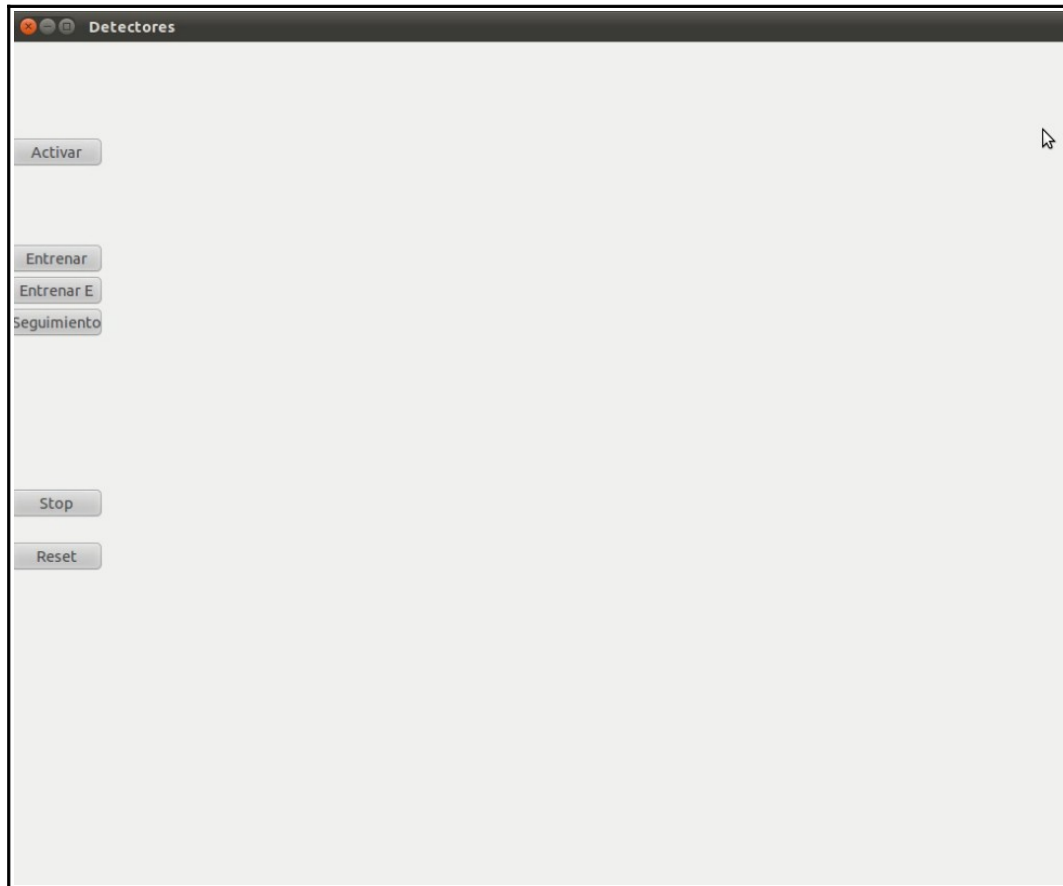


Figura A.1: Pantalla Principal

En la pantalla principal del sistema (Figura A.1), se puede observar los botones que servirán al usuario para activar y controlar el sistema.

LITERATURA CITADA

- [Wang & Popovic 10] “*Real-time hand tracking with a color glove*” Robert Y. Wang Jovan Popovic 2010 .
- [Longbin Chen 09] “*Estimating face pose by facial asymmetry and geometry*”.
Y Hu, L Chen, Y Zhou, H Zhang *Automatic Face and Gesture Recognition, 2004. Proceedings.*
- [Wikipedia 10] “*Outline of Object Recognition*”.
[http://en.wikipedia.org/wiki/Outline_of_object_recognition.](http://en.wikipedia.org/wiki/Outline_of_object_recognition)
- [Hossain,Ahn,Lee, Chae 05] “*Moving Object Detection in Dynamic Environment*”
KES(4) 2005: 359-365.
- [Tian,Feris, Hampapur 06] “*Real-Time Detection of Abandoned and Removed Objects in Complex Environments* ” Ying-li Tian, Rogerio Feris, Arun Hampapur, 2006.
- [Bellman 57] [Bellman57a] Bellman, R. E.. *Dynamic Programming.* Princeton University Press, Princeton, NJ.
[Bellman57b] Bellman, R. E. (1957b). A Markov decision process. *Journal of Mathematical Mech.* , 6:679—684.
- [Kölsch, Turk 04] “*Robust Hand Detection*” Mathias Kölsch and Matthew Turk [2004].

[Włodzimirz, Skrzynski 06] “*Hand image interpretation based on double active contour tracking*”. Włodzimirz Kasprzak and Piotr Skrzynski Warsaw University of Technology, Institute of Control and Computation Warsaw, Poland, 2006.

[Shotton, Blake, Cipolla 05] “*Contour-Based Learning for Object Detection*”. Jamie Shotton , Andrew Blake , Roberto Cipolla . 2005.

[Viola, Jones 01] “*robust real-time object detection*” second international workshop On Statistical and computational theories of Vision modeling, learning, computing, And Sampling Vancouver, Canada, 2001.

[Allen, Xu, Jin 06] “*Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces*” John G. Allen, Richard Y. D. Xu, Jesse S. Jin School of Information Technologies University of Sydney Madsen Building F09, University of Sydney, NSW 2006 .

[Thayananthan , Stenger , Torr, Cipolla 03] “*Shape Context and Chamfer Matching in Cluttered Scenes*”. A. Thayananthan , B. Stenger , P. H. S. Torr , R. Cipolla . 2003.