TECNOLÓGICO DE MONTERREY

Campus Ciudad de México

Escuela de Graduados en Ingeniería y Arquitectura

Tesis

Detección de objetos para el entretenimiento de grupos de personas

Para la obtención de grado de

Maestro en Ciencias de la Computación

Autor:

César Villanueva Ramírez

Asesor de tesis: Dr. Víctor de la Cueva Hernández
Sinodales: Dr. Alfredo Víctor Mantilla Caeiros
Dr. Raúl Crespo Saucedo

Abril 2014

Dedicatoria

A mi esposa que aguanto conmigo hasta la meta y a mis padres por haberme guiado por un buen camino.

Agradecimientos

A mis profesores por haber sido parte de mi formación profesional y personal.

A mi asesor de tesis Víctor de la Cueva por su apoyo durante todo el proceso.

A mis sinodales Alfredo Víctor Mantilla Caeiros y Raúl Crespo Saucedo

por el tiempo invertido en este trabajo.

Índice

Capítulo 1. Introducción

Una de las diversas necesidades del ser humano es el entretenimiento. El hombre ha creado a través de los años diversas formas para lograr el objetivo de entretener. El cine, teatro, conciertos, opera, deportes, programas de televisión, parques de diversiones son algunos de los ejemplos del tipo de entretenimiento que el hombre ha inventado.

En la época en la que la computadora todavía no se inventaba, los juegos fueron diseñados y jugados en el mundo físico con el uso de propiedades reales. Hoy en día los videojuegos se han convertido en una dominante forma de entretenimiento debido al alto grado de atracción hacia los jugadores [5].

Es en el área del entretenimiento con videojuegos en el que se enfoca este trabajo, particularmente en una variante poco desarrollada, orientada a grupos de personas y que se podría conocer como videojuegos manejados por audiencias o masivos. Estos juegos, a diferencia de los juegos que utilizan una consola especial o la computadora y en donde por lo general participan de 1 a 4 jugadores, están orientados a grupos de personas, y aunque los videojuegos en línea también son masivos por el número de jugadores que pueden participar, los videojuegos masivos hacen partícipes a los jugadores en un mismo espacio físico. Los sistemas que habilitan a audiencias grandes a interactuar ofrecen numerosas posibilidades de entretenimiento, educación y trabajo en equipo [1].

Sería posible por ejemplo, que mientras las personas en una sala de cine esperan que comience la película, incluir un tipo de videojuego que las mantenga entretenidas y concentradas de lo que pasa en pantalla, o en un parque de diversiones ofrecer, como parte de la infraestructura de entretenimiento, una sala de videojuegos masivos en donde las personas a través de movimientos laterales puedan manejar un carro, o utilizando paletas de colores jugar Pong. Este tipo de

juegos generalmente utilizan cámaras que detectan movimientos o patrones de colores combinados con técnicas y algoritmos para la interpretación de estos patrones.

Esta tesis documenta la propuesta, diseño e implementación de un videojuego interactivo orientado a entretener a grupos de personas. El objetivo de este sistema es ejecutar acciones en un videojuego por medio del análisis de imágenes capturadas por una cámara digital y en donde la interacción videojuego - personas es realizada a través de paletas de colores (en forma de círculos con colores azul y rojo) las cuales mueven las barras de un juego de Pong hacia un lado u otro dependiendo de los colores identificados.

Como un punto adicional en este trabajo se detalla un tipo de arquitectura que puede ser utilizada para este tipo de aplicaciones.

1.1 Trabajos Previos

La motivación principal para realizar este trabajo fue una idea presentada en el SIGGRAPH de 1991. Loren y Rachel Carpenter revelaron un sistema de entretenimiento interactivo que permitía a los miembros de una gran audiencia controlar un videojuego usando unas paletas reflejantes en donde cada paleta tenía un lado color rojo y el otro verde al cual llamaron Cinematrix [1]. El videojuego era el popular Pong, en donde cada mitad de la audiencia controlaba una paleta del juego proyectado en una pantalla enfrente de la multitud, y adicionalmente se probó un sistema de votación. En el sistema de votación se contaba el número de paletas con el color verde contra las de color rojo (Figura 1.1).

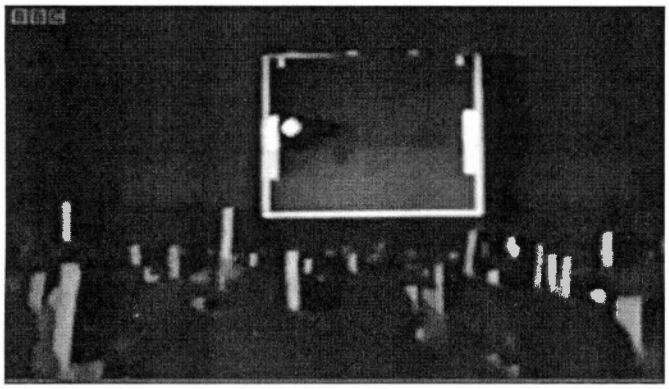


Figura 1.1 Imagen captada del sistema interactivo de los Carpenter.

Posteriormente en el 2002, en la conferencia internacional de interfaces multimodales de la IEEE, Pausch, Maynes-Aminzade y Seitz, citando el trabajo de los Carpenter en 1991, presentaron tres técnicas para implementar una interacción entre grupos de personas de una manera competitiva o cooperativa usando una combinación de procesamiento de imágenes en tiempo real, visión computacional y gráficos en 2D y 3D. La primera de estas técnicas consistiendo en capturar el movimiento hacia la izquierda o la derecha de cientos de personas sentadas y traducido igualmente en movimientos de objetos dirigiéndose hacia ambos lados (Figura 1.2).

La segunda consistía en el seguimiento de la sombra de una bola de playa reflejada en una pantalla de una sala de cine para controlar la mira de un misil, y la tercera técnica pintaba la secuencia de los rayos láser que traían consigo las personas que se encontraban en una sala de cine y que apuntaban a la pantalla de la sala [1].

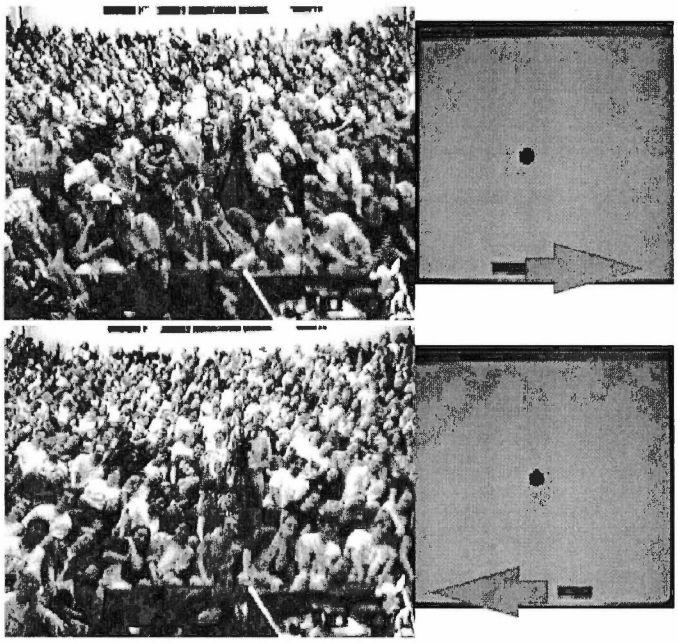


Figura 1.2 Ejemplo de captura de movimiento de personas

En 2004 en un evento del SIGGRAPH se probó un juego con alrededor de 4000 personas de la audiencia, al cual llamaron *Squidball* (Figura 1.3). El objetivo era entretener a las personas que se encontraban dentro del auditorio controlando gráficos y audio en tiempo real mediante el rebote de múltiples globos llenos de helio a lo largo del espacio de *todo* el auditorio [2].

Figura 1.3 Squidball

Sieber, McCallum y Wybill presentaron en una conferencia de gráficos computacionales y técnicas de interacción en Australia una proyección espejo enfrente de las personas presentes en el auditorio. A esta proyección le agregaban balones virtuales, los cuales se podían ver a través de la pantalla y la gente podía "virtualmente golpearlos" para generar el choque y movimiento de estos [3].

Se han presentado también otras formas de interacción con elementos de una pantalla de juego. Con la descripción de "palanca humana", presentaron los juegos de pole-position y Pong en donde a los asistentes en un auditorio se les dio la instrucción de moverse a sus costados para mover a un carro o una barra de Pong hacia la izquierda o derecha. Tal demostración de interacción se ha llevado a cabo comercialmente o publicitariamente también. Algunos de los eventos en donde se han presentado son [4]:

- Conferencia Cisco Live 2007.- Demostración de un juego de Pong.
- Como una actividad previa a la presentación de la película Spiderman 3 en varias ciudades de Estados Unidos
- En 2008 en un programa de la BBC Londres.

Nuevas formas de interacción entre grupos de personas y una pantalla han surgido como es la denominada Realidad Aumentada (Augmented Reality). La realidad aumentada es una variación de la realidad virtual en la que se dibujan objetos virtuales en un ambiente del mundo real. Los usuarios ven su panorama aumentado con objetos en 3D como si realmente existieran en el espacio [5]. La realidad aumentada también ha sido utilizada en cines, en forma de burbujas de agua proyectas en la pantalla las cuales pueden ser manipuladas por las manos de los usuarios.

Estos tipos de juego han sido denominados de diferentes formas, como video juegos guiados por audiencias (Audience Driven Video Games) o Juegos

Interactivos (Interactive Games). Según Alanna Hardy, articulista de temas relacionados con videojuegos, a pesar de que los juegos manejados por audiencias están todavía en su infancia, los beneficios son ya aparentes. Mientras que las innovaciones se abran a un campo de más posibilidades, y más plataformas se vuelvan disponibles para más gente, seremos capaces de ver más colaboración en grupo [6].

1.2 Planteamiento del problema

Dado que aún no se ven muchos escenarios en donde se implemente este tipo de videojuegos, por lo menos en México, lo que se plantea con este trabajo es proponer, más que una solución a un problema, una arquitectura y método simple para su implementación al igual que otros trabajos lo han propuesto a nivel mundial. Se trata de ser un generador de ideas relacionadas con el desarrollo de estos proyectos y que puedan ser llevados a esquemas de experimentación en situaciones y lugares en donde pueda ser necesaria la atención de las personas no solo como entretenimiento si no como una forma de integrar a grupos de personas en la espera de eventos multitudinarios como puede ser la espera del comienzo de una película, un concierto, una obra de teatro, un lugar en un restaurante. Por lo tanto las aplicaciones pueden multiplicarse dependiendo de la creatividad, la necesidad, la utilidad y las herramientas disponibles para aplicar estos modelos de entretenimiento en nuestro país.

1.3Objetivo

Implementar un videojuego controlado por audiencia (2 o más personas) utilizando las características del videojuego Pong, utilizando paletas bicolores, las cuales controlarán la dirección de la barra arriba-abajo de acuerdo al color de la paleta. El sistema deberá poder jugarse a partir de 2 participantes o más, en donde cada grupo o individuo controlará una barra.

1.3.1 Objetivos específicos

- Implementar la interface de hardware (cámara digital) necesaria para la toma de imágenes y la transferencia de estas imágenes al sistema.

- Implementar un método de detección de objetos en imágenes digitales de entrada utilizando algoritmos de reconocimiento de imágenes. El alumno se centrará en el reconocimiento de los colores de las paletas para asociar el color con una acción en el sistema de juego.

- Desarrollar un videojuego que implemente las características del juego Pong y responda a las interpretaciones de las imágenes de entrada con acciones de las barras de juego.

1.4 Justificación

El desarrollo de ideas para el entretenimiento usando la ciencia y la tecnología ha desembocado en la creación de productos que además de innovadores son exitosos. Una categoría de estos productos son los videojuegos, que desde hace tiempo han demostrado ser una gran industria. El producto que se desarrollará en este trabajo se encuentra dentro de una categoría poco explotada, sin embargo es una opción viable para crear nuevos esquemas de diversión orientados a eventos masivos o lugares en donde se reúnan grandes cantidades de personas.

¿Es importante el entretenimiento?

De acuerdo con Shell [2005], de la universidad Carnegie Mellon, el entretenimiento es considerado frívolo y esa es la razón por la que ha permanecido sin examinar por mucho tiempo [7]. Según Shell, proporciona 4 puntos por los cuales es importante entender el entretenimiento entre los cuales se encuentran:

1. El entretenimiento define la cultura de una sociedad
2. El entretenimiento es la piedra angular de la comunicación
3. El entretenimiento es la piedra angular del descubrimiento
4. El ser humano está más vivo cuando esta entretenido.

1.5 Limitaciones y alcances

Las limitaciones son:

- Se necesita un ambiente poco controlado en donde los colores de la ropa de las personas que juegan no coincidan con aquellos de las paletas.
- En esta versión de la aplicación, se utilizará una cámara web, en donde el ángulo de visión es limitado
- La aplicación esta optimizada para grupos de personas posicionadas a 1.8 o 3 metros de distancia de la cámara.

Los alcances son:

- Diseño e implementación de la infraestructura necesaria para el funcionamiento de un juego de video denominado Pong4Groups con capacidad de hasta 6 jugadores y posibilidad de escalarlo a más jugadores agregando cámaras de video adicionales.
- Capacidad de reconocimiento de paletas físicas circulares con 2 colores diferentes (azul y rojo) que serán los controladores del juego

Capítulo 2. Antecedentes

Son los fundamentos teóricos que componen esta tesis los que se presentan en este capítulo. El trabajo presentado se divide en tres partes principales:

1. El juego de video.
2. La captación de imágenes y video.
3. El procesamiento de imágenes.

Se describen las técnicas utilizadas, así como las herramientas requeridas para su funcionamiento.

Las computadoras mirando a través de una cámara hacia las personas, es una técnica potencialmente poderosa para facilitar la interacción humano-computadora. La computadora puede interpretar los movimientos de usuario, gestos y miradas. Los algoritmos visuales fundamentales incluyen seguimiento, reconocimiento de formas y análisis de movimientos [13]. Los elementos principales en esta interacción humano computadora se componen entonces en general de técnicas de visión que pueden ser implementadas a través de una cámara digital, el reconocimiento de estas imágenes capturadas y su implementación dentro del videojuego.

2.1 Procesamiento de imágenes

A lo largo de los últimos 30 años, la visión computacional ha evolucionado en una área madura, abarcando diferentes tópicos y aplicaciones: estos varían desde robots que permiten el ensamblaje automático de autos hasta el manejo automático con guía, de interpretaciones automáticas de documentos hasta la verificación de firmas y desde en análisis de imágenes de objetos remotamente captados hasta la revisión de huellas digitales y células de sangre humana [41].

La visión permite a los humanos percibir y entender el mundo que está a su alrededor y la visión computacional trata de duplicar, de manera electrónica, el efecto de la visión humana analizando e interpretando imágenes digitales. En un sentido general el procesamiento de imágenes en los humanos comienza por capturar la imagen con su sentido de la vista (ojos), después el cerebro se encarga de procesar la información que le es transferida y decide, en base a esa imagen, llevar a cabo acciones como mover ciertas partes del cuerpo, hablar, mover la cabeza, cerrar los ojos, etc. El procesamiento digital de imágenes trata de realizar las mismas funciones utilizando dispositivos de hardware, software y recursos teóricos como lo veremos más adelante.

Una imagen puede ser definida como una función bidimensional $f(x, y)$, donde x y y son coordenadas espaciales y la amplitud de f en cualquier coordenada de (x, y) es llamado intensidad o nivel de gris de la imagen en ese punto. Cuando los valores de x, y y la amplitud (f) son todos finitos, cantidades discretas, llamamos a la imagen una imagen digital, que está compuesta por un numero finito de elementos, en el que cada uno tiene una posición y valor particular. Estos elementos son llamados pixeles [14].

Aunque la distinción entre procesamiento digital de imágenes y análisis digital de imágenes no es inmediatamente obvio, es extremadamente importante. El procesamiento de imagen puede ser descrito como una transformación que toma una imagen y devuelve una imagen. Por otra parte el análisis de imágenes digitales es una transformación de la imagen en algo diferente a una imagen, por ejemplo, una descripción o decisión. Sin embargo, las técnicas de análisis de imágenes digitales son usualmente aplicadas a imágenes previamente procesadas [15].

El tratamiento digital de imágenes comprende una serie de pasos y técnicas generalmente adoptadas por los libros de texto y textos científicos aunque en algunos casos pueden agregarse o quitarse algunas etapas. En general un

esquema de pasos para el procesamiento de imágenes digitales es el que se muestra en la figura 2.1. [14]

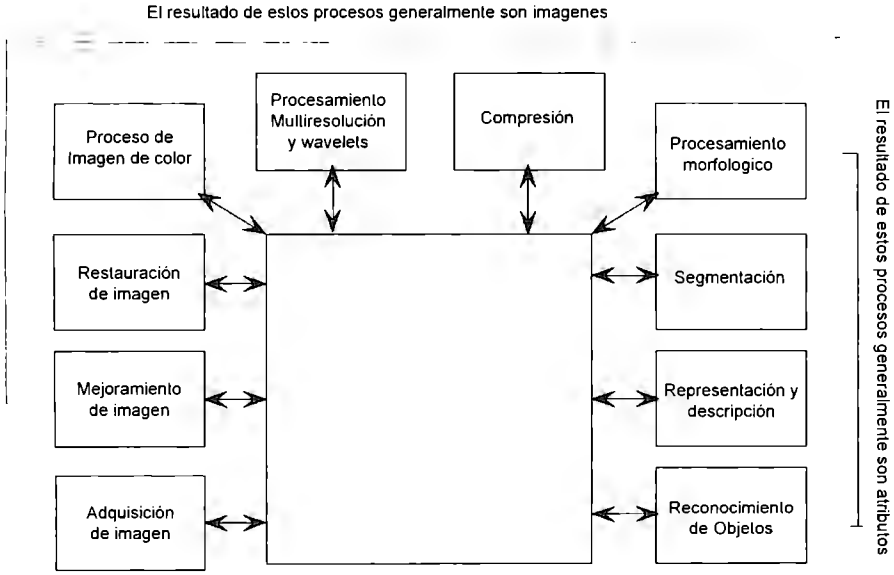


Figura 2.1 Pasos generales en el procesamiento de imágenes

Las etapas que se utilizan para el procesamiento de imágenes en esta tesis incluyen sólo las etapas de adquisición de imagen, proceso de imagen de color, segmentación, representación y descripción y reconocimiento de objetos.

2.1.1 Tipos de imágenes digitales

Por lo general se consideran 4 tipos de imágenes digitales [16]:

Binaria.- Cada pixel es negro o blanco. Puesto que solo hay dos posibles valores para cada pixel, solo es necesario un bit por pixel. Este tipo de imágenes son apropiadas para incluir texto, huellas digitales, planos arquitectónicos. En este trabajo se utiliza la el proceso de conversión a imagen binaria para comenzar un reconocimiento de objetos.

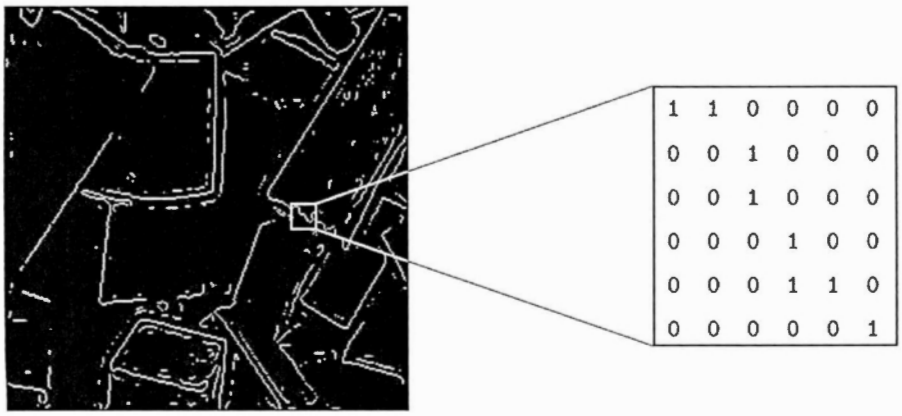


Figura 2.2 Ejemplo de imagen binaria

Escala de grises.- Cada pixel tiene un nivel de gris, normalmente del rango de 0 (negro) a 255 (blanco). Este rango significa que cada pixel puede ser representado por ocho bits, o un byte. Este es un rango muy común para el manejo de imágenes de este tipo (Figura 2.3).

Color Verdadero o RGB.- Cada pixel tiene un color particular; este color es formado con la combinación de rojo, verde y azul. Si cada uno de estos componentes tiene un rango de 0-255, esto da un total de 255^3 = 16,777,216 posibles colores en la imagen. Puesto que el número total de bits requerido por cada pixel es de 24 estas imágenes son también llamadas imágenes de color de 24-bits.

Este tipo de imagen puede ser considerada como una pila de tres matrices, cada una representado los valores de rojo, verde y azul para cada pixel. Esto significa que a cada pixel le corresponden 3 valores (Figura 2.4)

230	229	232	234	235	232	148
237	236	236	234	233	234	152
255	255	255	251	230	236	161
99	90	67	37	94	247	130
222	152	255	129	129	246	132
154	199	255	150	189	241	147
216	132	162	163	170	239	122

Figura 2.3 Imagen en escala de grises.

Rojo					
49	55	56	57	52	53
58	60	60	58	55	57
58	58	54	53	55	56
83	78	72	69	68	69
88	91	91	84	83	82
69	76	83	78	76	75
61	69	73	78	76	76

Verde					
64	76	82	79	78	78
93	93	91	91	86	86
88	82	88	90	88	89
125	119	113	108	111	110
137	136	132	128	126	120
105	108	114	114	118	113
96	103	112	108	111	107

Azul					
66	80	77	80	87	77
81	93	96	99	86	85
83	83	91	94	92	88
135	128	126	112	107	106
141	129	129	117	115	101
95	99	109	108	112	109
84	93	107	101	105	102

Figura 2.4 Imagen RGB

Imagen Indexada.- La mayoría de las imágenes solo tiene un pequeño grupo de los más de dieciséis millones de colores posibles. Para facilitar el factor de guardado y tamaño de archivo, este tipo de imagen tiene asociado un mapa de color o paleta de colores, que es simplemente una lista de todos los colores usados en la imagen. Cada pixel tiene un valor el cual no es un color, sino el valor de un índice en el mapa de colores.

Es conveniente si una imagen tiene 256 colores o menos porque así solo se requerirá de un byte para cada uno. Algunos formatos de imágenes permiten solo 256 colores o menos en cada imagen, precisamente por esta razón.

Figura 2.5 Imagen de color indexada.

2.1.2 Reconocimiento de Objetos en Imágenes

Un sistema de reconocimiento de objetos encuentra objetos del mundo real en una imagen del mundo real usando modelos de objetos que son previamente conocidos. Esta tarea es sorprendentemente difícil. Los humanos realizan este reconocimiento sin esfuerzo e instantáneamente [21].

El Dr. Lior Wolf define los siguientes términos clave en el reconocimiento visual de objetos [22]:

- Reconocimiento de Objetos: Detección de objetos + Identificación de objetos.
- Detección de Objetos: El reconocimiento de una clase de objeto aprendida. Ejemplo: Encontrar carros en una escena de calle.
- Identificación de Objetos: Una instancia individual de un objeto que es reconocida. Ejemplo: La identificación de la cara de una persona.
- Clasificación de imagen: Detección de imagen aplicada a toda la imagen. Ejemplo: ¿Es esta la imagen de un tigre?

2.1.3 Detección de objetos

El problema general de la detección de objetos en imágenes estáticas reside en querer distinguir una clase particular de objetos de todos los demás. Un sistema robusto de este tipo debería ser capaz de detectar los objetos en iluminaciones irregulares, objetos que están rotados en el plano de la imagen, y objetos que están parcialmente ocultos o cuyas partes se mezclan con el fondo. Bajo todas estas condiciones, el contorno de un objeto es usualmente alterado y su forma normal puede no ser perceptible [23].

Los sistemas de detección de objetos que han sido desarrollados hasta ahora (2001) caen en una de tres categorías generales. La primera categoría consiste en sistemas que están basados en modelos, p. ej., un modelo es definido para el objeto de interés y el sistema intenta encontrar este modelo en diferentes partes de la imagen. El segundo tipo son métodos de invariancia de imagen los cuales basan la comparación de objetos en un grupo de relaciones de patrones (p. ej., niveles de brillo) de una imagen que, idealmente, únicamente determinara los objetos que se buscan. El tercer grupo de sistemas de detección de objetos está caracterizado por sus algoritmos de aprendizaje basados en ejemplos. Estos sistemas aprenden de las características de una clase tomando grupos etiquetados como muestras positivas y negativas. Las técnicas basadas en ejemplos han sido exitosamente utilizadas en otras áreas de visión computacional, incluyendo el reconocimiento de objetos [23].

2.2 Introducción a MATLAB

MATLAB es una herramienta de análisis y visualización de datos que ha sido diseñada para el manejo de matrices y las operaciones que tiene que ver con estas. A esto hay que agregar también que MATLAB tiene grandes capacidades para gráficas y su propio lenguaje de programación. Unas de las razones por las que MATLAB se ha convertido en una herramienta tan importante es por los

grupos de herramientas disponibles para tareas en particular. Estos grupos de programas son llamados "toolboxes", y uno de los toolboxes que se utilizan en este trabajo es el de procesamiento de imágenes.

2.2.1 MATLAB Image Acquisition Toolbox (IAT)

Es un módulo de MATLAB que permite adquirir imágenes y video por medio de cámaras digitales para poder manipularlo en MATLAB. Se puede detectar el hardware automáticamente y configurar sus modos de operación. Con el soporte para estándares de la industria y múltiples comerciantes de hardware, se pueden utilizar dispositivos de imagen que van desde una cámara Web hasta dispositivos científicos e industriales que requieren de requerimientos específicos [17].

2.2.2 MATLAB Image Processing Toolbox (IPT)

IPT provee de un amplio grupo de algoritmos estándar, funciones y aplicaciones para el desarrollo de algoritmos, procesamiento, análisis y visualización de imágenes. Se pueden realizar operaciones como mejoramiento de imagen, enfoque de imagen, detección de características, reducción de ruido, segmentación de imagen, transformaciones geométricas y registro de imágenes. Muchas de las funciones son multihilo para tomar ventaja de computadoras con esta tecnología [18].

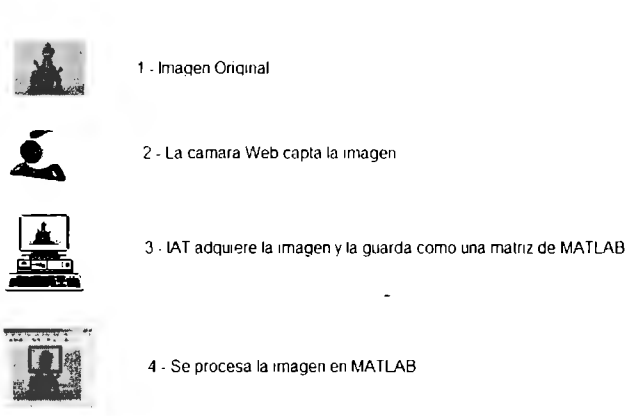


Figura 2.6 Flujo de IAT

Las funcionalidades de los módulos IAT y IPT se complementan para brindar la funcionalidad necesaria para realizar la captación, el procesamiento y la entrega del resultado del reconocimiento de objetos en la imagen en este proyecto.

2.2.3 MATLAB Builder JA

MATLAB Builder JA es una herramienta que permite crear clases Java a partir de programas desarrollados en el entorno de MATLAB. Estas clases Java pueden ser integradas dentro de programas Java y ser ejecutados en servidores o computadoras desktop que no tienen MATLAB instalado, con solo usar el Compilador en tiempo de ejecución MATLAB (MCR) que viene como parte de la suite de productos MATLAB[19].

Cuando se usa el MCR, se crean componentes que hacen posible el acceso a operaciones, visualizaciones e interfaces de usuario de MATLAB a los usuarios finales de Java.

2.3 Transformada de Hough

La transformada de Hough es una técnica que puede ser utilizada para aislar características de una forma particular dentro de una imagen. Debido a que requiere que las características sean especificadas a través de forma paramétrica es utilizada comúnmente para la detección de curvas regulares como líneas, círculos, elipses etc. Hay una forma general de la transformada de Hough que se puede utilizar en casos en donde no hay una descripción analítica de las características [8].

La transformada de Hough fue inventada por Paul Hough en 1962 y patentada por IBM. Se ha convertido en una herramienta estándar en el dominio de la visión computacional. Es particularmente robusta cuando se tienen datos con ruido o parciales.[29]

Su funcionamiento se basa en un procedimiento de votación donde cada punto en la imagen vota para todas las líneas posibles que pasan por este. El punto con más votos corresponde a la que podría ser la línea.

Supongamos que estamos buscando líneas rectas en una imagen. Si tomamos un punto (x',b') en la imagen, todas las líneas que pasan por este pixel tienen la forma:

$$y' = mx' + c \tag{2.1}$$

al variar los valores de *m* y *c*. Como se muestra en la siguiente figura.

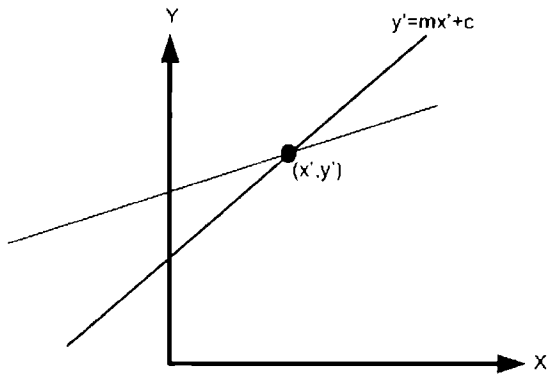


Figura 1.4 Punto en el plano cartesiano por el que pasan dos rectas

Sin embargo la ecuación 2.1 también puede ser escrita como:

$$c = -x'm + y \tag{2.2}$$

Con la ecuación 2.2 ahora estamos considerando a x' y y' como constantes y a m y c como variables. En la siguiente figura se muestra una línea recta en un plano (m, c).

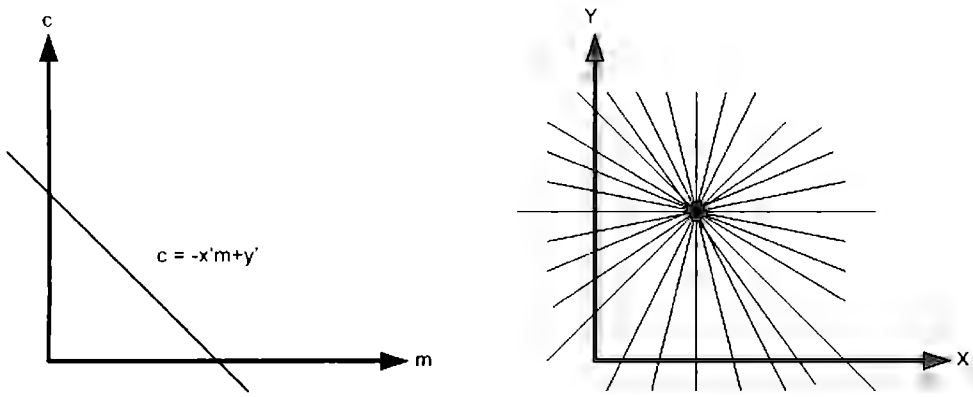


Figura 2.2 Hough Space (derecha), líneas en espacio (x, y) (izquierda)

Cada línea que pasa a través del punto (x', y') corresponde a uno de los puntos de la línea en el espacio (m, c) denominado espacio de hough.

Tomando en cuenta lo anterior podemos decir que:

- Todos los pixeles que se encuentran en la misma línea en el plano (x, y) son representados por líneas las cuales pasan a través de un punto en común en el espacio (m, c).

- El punto en común por el cual todas las líneas pasan proporciona los valores de *m* y *c* en la ecuación 2.1

Sin embargo la ecuación 2.1 no es capaz de representar líneas verticales debido a que m se vuelve infinita. Para evitar este problema se usa la representación polar para las líneas, la cual se convierte en la siguiente ecuación [9].

$$\mathrm{x}\cos\theta + \mathrm{y}\,\mathrm{sen}\,\theta = \mathrm{p} \tag{2.3}$$

Todas las líneas pueden representarse en esta forma cuando $\theta\varepsilon[0{,}360]$ y $\mathrm{p} >= 0$. Por lo tanto el espacio de Hough tiene estas dos dimensiones (θ, p) y una línea es representada por un solo punto (Figura 2.3), correspondientes a un único grupo de parámetros $(\theta 0, \mathrm{p}0)$.

Un concepto importante de la transformada de Hough es la conversión de puntos. La idea es que un punto es mapeado a todas las líneas que pueden pasar por ese punto. Esto resulta en una curva tipo seno en el espacio de Hough, como se muestra en la figura 2.4

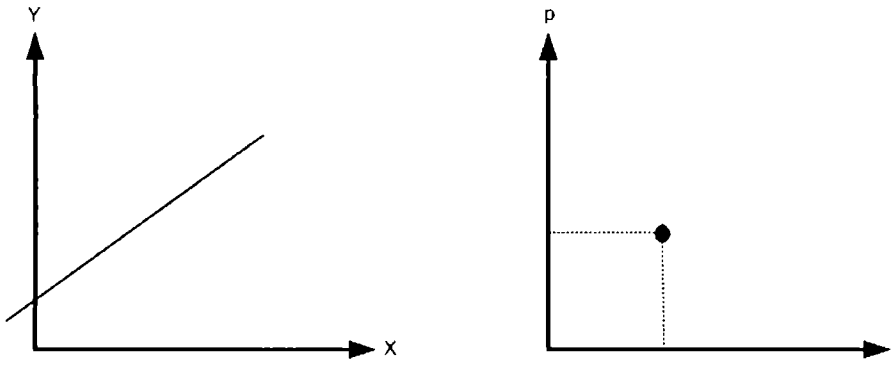


Figura 2.3 Mapeo de una línea a un punto en coordenadas polares

2.3.1 Transformación al Espacio de Hough

La transformada de Hough toma un mapa binario de bordes detectados como entrada e intenta localizar bordes puestos como líneas rectas. La idea es, que cada punto en el mapa de bordes sea transformado a todas las posibles líneas que puedan pasar a través de ese punto. Esto resulta en una curva seno el espacio de Hough (Figura 2.4). La intersección de las curvas indica la línea que pasa por p1 y p2.

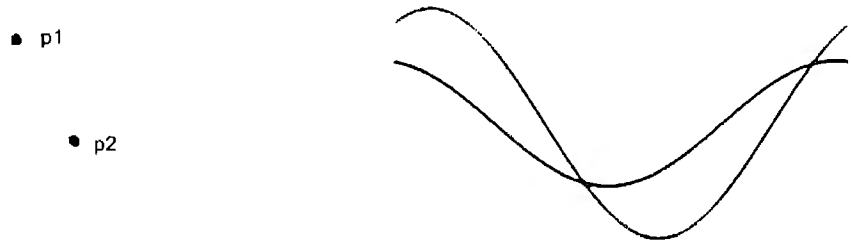


Figura 2.4 Transformación de dos puntos a dos curvas en el espacio de Hough

Una imagen con filtrado de bordes incluye muchos puntos, pero el principio para la detección de líneas es el mismo ilustrado en la figura 2.4. Cada punto de los bordes es transformado a una línea en el espacio de Hough y las áreas en donde más líneas se intersectan en ese espacio es interpretado como una línea verdadera en la imagen de bordes [10].

2.3.2 El Acumulador

Para determinar las áreas donde más líneas se intersectan, un arreglo llamado "acumulador" que cubre todo el espacio de Hough es usado. Cuando un punto de los bordes es transformado, un contador de una celda del acumulador es incrementado para todas las líneas que puedan pasar por ese punto. La resolución del acumulador determina la precisión con la cual las líneas puedan ser detectadas.

En general, el número de dimensiones del acumulador corresponde al número de parámetros en la ecuación utilizada. Por lo tanto, por ejemplo, para una elipse un espacio de 5 dimensiones es requerido (las coordenadas de su centro, la longitud de sus ejes mayor y menor y su ángulo). Para las líneas dos dimensiones son suficientes (r y θ).

Algoritmo

El algoritmo para la detección de líneas rectas puede ser establecido en los siguientes pasos:

1. Realizar la detección de bordes con algún método conocido para este fin, como puede ser la detección de bordes Canny.

2. Convertir los puntos de los bordes detectados en líneas en el espacio de Hough y guardarlos en el acumulador.

3. Encontrar las máximas en el acumulador.

2.3.3 La transformada de Hough para detección de círculos

La transformada de Hough puede ser usada para determinar los parámetros de un circulo cuando un número de puntos que caen dentro del perímetro son conocidos [11]. La detección de círculos en una imagen digital es muy importante en el reconocimiento de formas. La transformada de Hough es el método más conocido para la detección de círculos [12]. El círculo puede ser expresado como:

$$(x-a)^2 + (y-b)^2 = r^2 \tag{2.4}$$

Como puede verse el circulo tiene tres parámetros, r, a y b. Donde a y b son el centro del circulo en la dirección x y y respectivamente y donde r es el radio. Con estos parámetros el círculo puede ser descrito con las siguientes ecuaciones paramétricas:

$$x = a + r\cos(\theta) \tag{2.5}$$

$$y = b + r\,sen(\theta) \tag{2.6}$$

Cuando el ángulo θ barre a través de un rango completo de 360 grados, los puntos (x, y) trazan el perímetro de un círculo.

Si una imagen contiene muchos puntos, algunos de los cuales caen en perímetros de círculos, entonces el objetivo es buscar tripletas de parámetros (a, b, r) para describir cada círculo. El hecho de que el espacio de parámetros sea de tres, hace que la implementación de la técnica de Hough sea más demandante en tiempo de procesamiento y memoria.

2.3.3.1 Búsqueda con radio conocido

Si el radio de los círculos en una imagen es conocido, entonces la búsqueda puede ser reducida a dos parámetros. El objetivo sería encontrar las coordenadas (a, b) de los centros.

Cada punto en el espacio cartesiano de un círculo genera un círculo en el espacio de parámetros. El punto del centro real será común a todos los círculos generados, y puede ser encontrado por medio del acumulador.

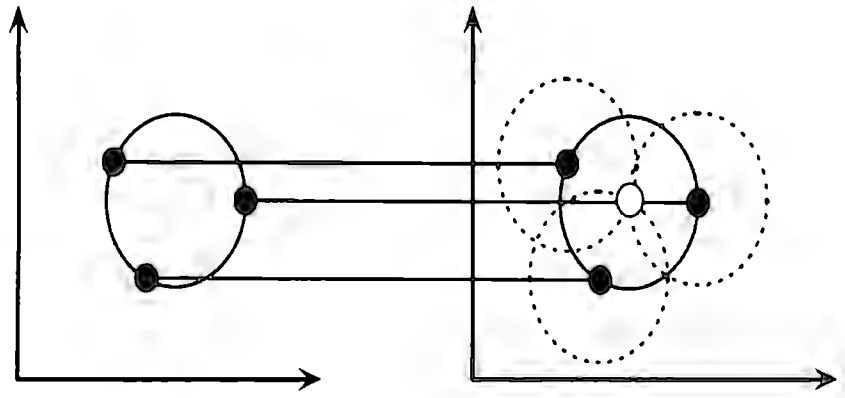


Figura 2.5 Transformada de Hough circular de un espacio (x, y) a un espacio de parámetros para un radio constante.

Múltiples círculos con el mismo radio pueden ser encontrados de la misma manera. En la figura 2.6 los centros están representados como con cuadros rojos en el espacio de parámetros. La sobre posición de los círculos puede causar que círculos falsos sean encontrados, como el del centro mostrado con el cuadro azul. Estos círculos pueden ser descartados realizando una correspondencia con los círculos de la imagen original.

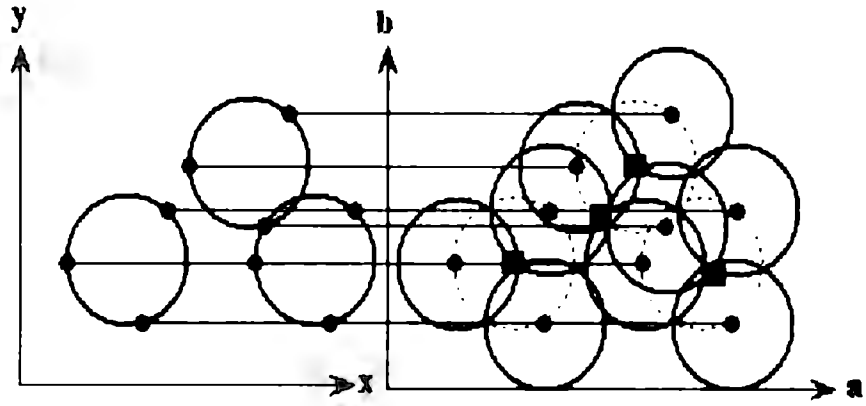


Figura 2.6 Detección de varios círculos usando HT

2.3.3.2 Búsqueda con radio no conocido

Si el radio no es conocido entonces la posición de los puntos en el espacio de parámetros caerá dentro de la superficie de un cono. Cada punto (x, y) en el perímetro de un círculo producirá una superficie cónica en el espacio de parámetros. La tripleta (a, b, r) corresponderá a la celda del acumulador donde el máximo número de superficies cónicas se intersecten.

El algoritmo para obtener los círculos por medio de HT es.

1. Detectar los bordes en la imagen y generar una imagen binaria.
2. Para cada pixel de borde, generar un circulo en el espacio (a, b)
3. Para cada punto en el círculo del espacio (a, b), captura los votos en las celdas del acumulador.
4. Las celdas con el mayor número de votos son los centros.

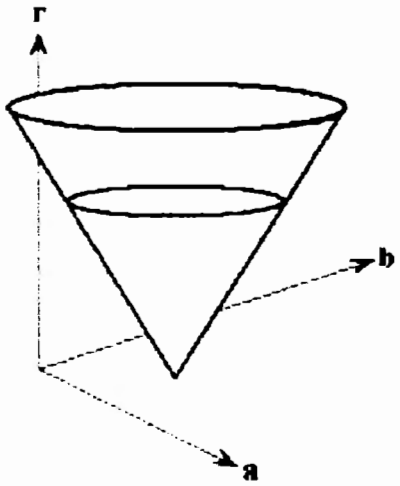


Figura 2.7. Generación de una superficie cónica en el espacio de parámetros para un punto (x, y). Un círculo con diferente radio será construido en cada nivel.

Capítulo 3.- Diseño del videojuego masivo

Como se menciona en la Introducción, la idea de esta tesis nace de un proyecto presentado en los años noventa. Pong4Groups es la denominación que se le da a la implementación del videojuego del presente trabajo, el cual consiste en, dado un espacio físico y un grupo de personas (simbólico), este grupo es dividido en un lado izquierdo y derecho, compiten en el videojuego denominado Pong4Groups, controlando cada barra por medio de paletas bicolores que tienen un lado rojo y otro azul. La figura 3.1 muestra la idea general del diseño del juego.

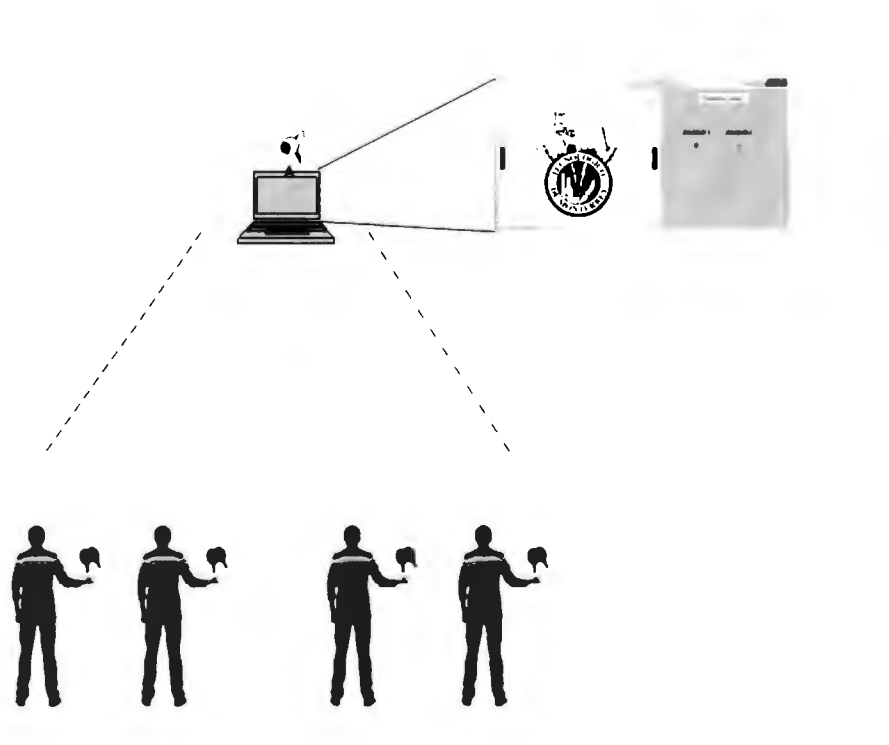


Figura 3.1 Idea general del videojuego Pong4Groups

Tres son los principales componentes que interactúan en este proyecto y que de manera abstracta pueden ser divididos en las siguientes partes:

1. El Videojuego
2. El módulo de adquisición de imágenes
3. El módulo de procesamiento de imágenes

3.1 Videojuego

El diseñador de videojuegos Jesse Schell considera que hay cuatro elementos principales en un videojuego [20]. Aunque puede ser que se refiera al tipo de videojuegos más tradicionales, se encuentra una relación interesante y similar con el diseño del videojuego en este proyecto. Los elementos son:

1. Mecánica: Estos son los procedimientos y reglas del juego. La mecánica describe la meta del juego, como los jugadores pueden o no intentar alcanzarla y que pasa si tratan. Si se compara los juegos a otras experiencias de entretenimiento más lineales (libros de texto, cine) se podrá notar que mientras estas últimas envuelven tecnología, historia y elementos estéticos no incluyen mecánica, por lo que la mecánica es lo que hace a un juego[20].

   La mecánica del videojuego de este trabajo es la misma del juego Pong, subir y bajar una barra sólida, como una pared en donde rebotará una pelota hacía el lado opuesto haciendo que el adversario mueva su barra hacia el lado a donde se dirige la pelota y rebotarla de regreso, en caso contrario el contrincante ganará un punto.

2. Historia: Es la secuencia de eventos que se desencadenan en el juego. Cuando se tiene una historia que se quiere relatar a través del juego, se tiene escoger la mecánica que hará que emerja y se fortalezca la historia [20].

En realidad no hay una historia en el juego de Pong, la meta del juego es rebotar la pelota tanto como se pueda hasta que alguno de los dos extremos deje pasarla.

3. Estética: Esto se traduce en como se ve, suena, huele, sabe y se siente el juego. La estética es un aspecto increíblemente importante del diseño del juego puesto que este tiene la relación más directa con la experiencia que tiene el jugador [20].

   La estética de este trabajo se podría traducir como la interface gráfica. Dos barras sobre un plano y un área para mostrar el marcador es lo que hace la parte grafica del juego.

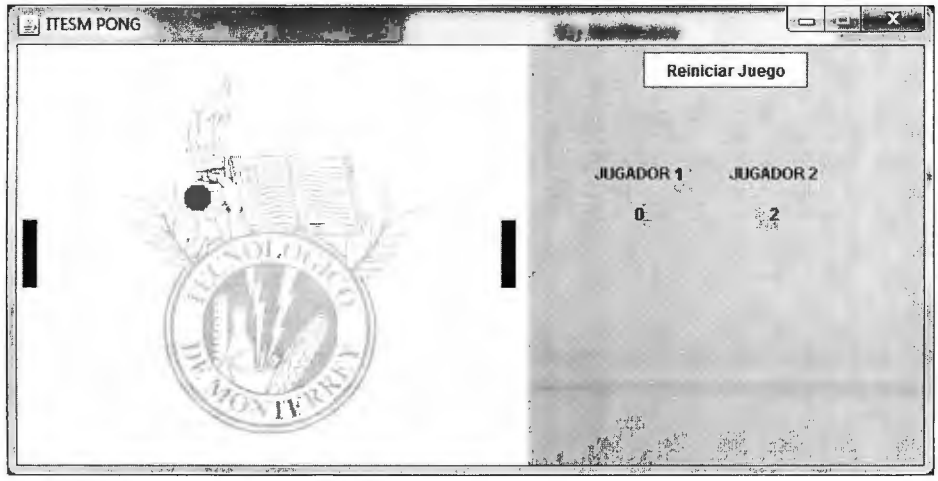


Figura 3.2. Interface del juego

4. Tecnología: No se refiere precisamente a "alta tecnología", sino a cualquier material e interacción que hacen el juego posible tal como papel y lápiz, fichas de plástico o laser de alto poder. La tecnología que se escoge para el juego permite hacer ciertas cosas y prohíbe realizar otras. La tecnología es

esencialmente el medio por el cual la apariencia del juego se muestra, la mecánica ocurre y la historia se cuenta.

La tecnología de este trabajo consiste en varios elementos, tomando en cuenta el párrafo anterior pueden contarse, las paletas bicolores (Figura 3.3) para subir y bajar la barra, la cámara web para obtener la imagen del ambiente y la computadora para procesar y mostrar las acciones realizadas.

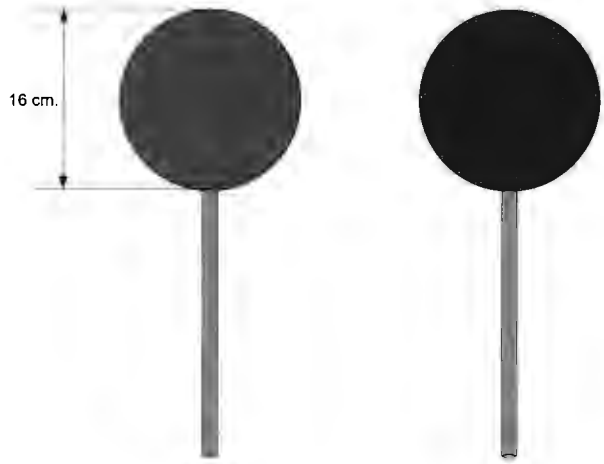


Figura 3.3. Paleta física utilizada en el juego

3.2 Adquisición de imágenes

Para poder mover las barras del juego es necesario convertir acciones en instrucciones por medio de un mecanismo de visión computacional capaz de captar las imágenes en el medio ambiente deseado. Para esta adquisición se considera una cámara web común con conexión USB, la cual tiene la resolución

suficiente para esta versión del juego, aunque el tipo de cámara no está limitado por la aplicación dando por entendido que la cámara debe ser digital.

Para procesar los cuadros que la cámara web capta se utiliza la herramienta llamada *Image Acquisition Tool* que es parte de la suite de MATLAB®. La resolución de la imagen capturada se establece de tamaño de 640x480 pixeles.

3.3 Procesamiento de imágenes

El procesamiento de las imágenes captadas por la cámara y el software de *Image Acquisition Tool* es realizado por el conjunto de utilerías en el *Image Processing Toolbox* que también es parte de la suite de MATLAB®. La distancia que se toma de la cámara hacía las paletas del juego está configurada para 1.8 y 3 metros. Entre más alejados estén los objetos que se tienen que detectar, más pequeña será el área de imagen procesada y más rápido será el desempeño en la traducción del color de la paleta y el movimiento de las barras del juego.

3.3.1 Determinación del método de detección de objetos

Una de las partes más complicadas fue la de determinar el método de detección de las paletas debido al reto que siempre ha ofrecido esta área y la variedad de técnicas utilizadas para este fin. Adicionalmente se agregaba una variable más; el color de los objetos.

Dado que este trabajo está orientado a la implementación de sistema para el entretenimiento de grupos no explotada en nuestro país, usando como motivación una idea anteriormente implementada, y no enfocada a la creación o mejoramiento de alguna técnica reconocimiento de objetos en imágenes, se determinó encontrar y usar un método plenamente identificado e implementado y que pudiera permitir en un futuro su mejora o suplantación por algún otro método.

El primer paso sería determinar las características de los objetos a detectar. Estas características proporcionan información importante para la especificación del método a utilizar. Tomando en cuenta que los objetos a detectar son paletas de colores las propiedades que nos interesan se presentan a continuación:

- Objetos en forma de circulo
- Objetos con forma de un circulo incompleto
- Círculos de color Rojo
- Círculos de color Azul
- Círculos incompletos de color rojo
- Círculos incompletos de color azul

La situación ideal para la detección de las paletas de este trabajo sería asegurar ciertos factores en el ambiente destinado para jugar Pong4Groups, sin embargo, esto requeriría un acondicionamiento del ambiente, iluminación y distancias. Por lo que adicionalmente a las propiedades, se incluyen las variables inherentes en la detección de objetos las cuales serían:

- Punto de visión y escala
- iluminación
- Oclusiones parciales
- Múltiples instancias
- Radio de los círculos

Con respecto al color, la mayoría de las aproximaciones modernas para la detección de objetos en color dependen en características basadas en intensidad e ignoran la información de color de la imagen. Esta exclusión de color es usualmente debido a las largas variaciones en color causadas por cambios en la iluminación, compresión, sombras y reflejos. Estas variaciones hacen que la tarea de describir el color sea especialmente difícil [24].

3.3.2 Investigación de métodos de detección de objetos

La investigación para la selección de un método de detección de objetos, estaba inclinada, aunque no limitada, hacia aquellos que estaban optimizados para la detección de círculos o figuras geométricas. A continuación se describen los métodos considerados para esta tarea.

3.3.2.1 Detección de Objetos usando Características Haar - Like

La detección de objetos utilizando este método se basa en clasificadores. Cada clasificador usa una área rectangular k (características de Harp) para hacer la decisión si la región de la imagen parece igual a una imagen predefinida o no. Primero, un clasificador es entrenado con algunos cientos de vistas de muestra de un objeto en particular (p.ej. una cara o un carro), llamados muestras positivas de un determinado tamaño, y unas muestras negativas del mismo tamaño.

Después de que el clasificador es entrenado, puede ser aplicado a una región de interés (del mismo tamaño al usado durante el entrenamiento) como una imagen de entrada. El clasificador determina verdadero si la región se parece y falso de otro modo [25].

3.3.2.2 Template Matching

La técnica de Template Matching es una técnica usada para clasificar un objeto comparando porciones de imágenes con otra imagen. Es ampliamente usada para procesar imágenes y fotografías. La elección de la coincidencia depende de la naturaleza de la imagen y del problema a ser resuelto. En general las clasificaciones de la técnica de Template Matching son: Aproximaciones con base en área y aproximaciones con base a características [26].

3.3.2.3 Transformada de Hough Circular

La transformada de Hough circular es un tipo de transformada de Hough que puede extraer objetos circulares de una imagen. La transformada de Hough fue presentada por primera vez por Paul Hough en 1962, la transformada consiste en una descripción paramétrica de una característica en cualquier localidad del espacio de imagen. La transformada de Hough esencialmente consiste de dos etapas. En la primera se calcula el mapa de bordes de la imagen, después cada punto de borde contribuye un círculo para un acumulador. En la segunda etapa, el acumulador tiene un tope donde los círculos obtenidos se sobreponen al centro del círculo original y entonces se definen las coordenadas del círculo [27].

3.3.2.4 RANSAC (RANdom Sampling and Consensus)

El concepto del algoritmo RANSAC es muy directo. Dado un set de datos, por ejemplo un contorno, aleatoriamente selecciona un subgrupo de características de muestra, por ejemplo puntos, trata de formar un modelo de figura geométrica, por ejemplo un círculo. El modelo es entonces comparado contra el grupo completo de características. Si un número suficiente de características del set de datos satisface el modelo con una cierta tolerancia, entonces se determina que la característica existe y el proceso RANSAC se termina. Por supuesto hay una cantidad de intentos para intentar emparejar un modelo en un determinado grupo de datos [39].

Se determinó que, debido a que el tipo de forma a detectar puede describirse paramétricamente (Círculos) y el crear muestras positivas y negativas para entrenar calificadores o realizar coincidencias podría ser trabajo innecesario, el método a utilizar estaría entre la Transformada de Hough Circular y el algoritmo RANSAC, sin embargo en la investigación que se realizó se encontró una gran variedad de literatura de la transformada de Hough, además de que esta técnica ha sido ampliamente utilizada desde hace décadas. [28]

3.4 Lógica de programación

El código del juego Pong4Groups está codificado en dos lenguajes de programación. La interface y lógica del juego está desarrollado en Java 1.6 utilizando las librerías de AWT (*Abstract Window Toolkit*). Para el procesamiento de imágenes se utiliza el lenguaje de programación propietario de MATLAB®.

El flujo general que tiene que seguir el juego y en el que se basa la implementación de este se muestra en la figura 3.4.

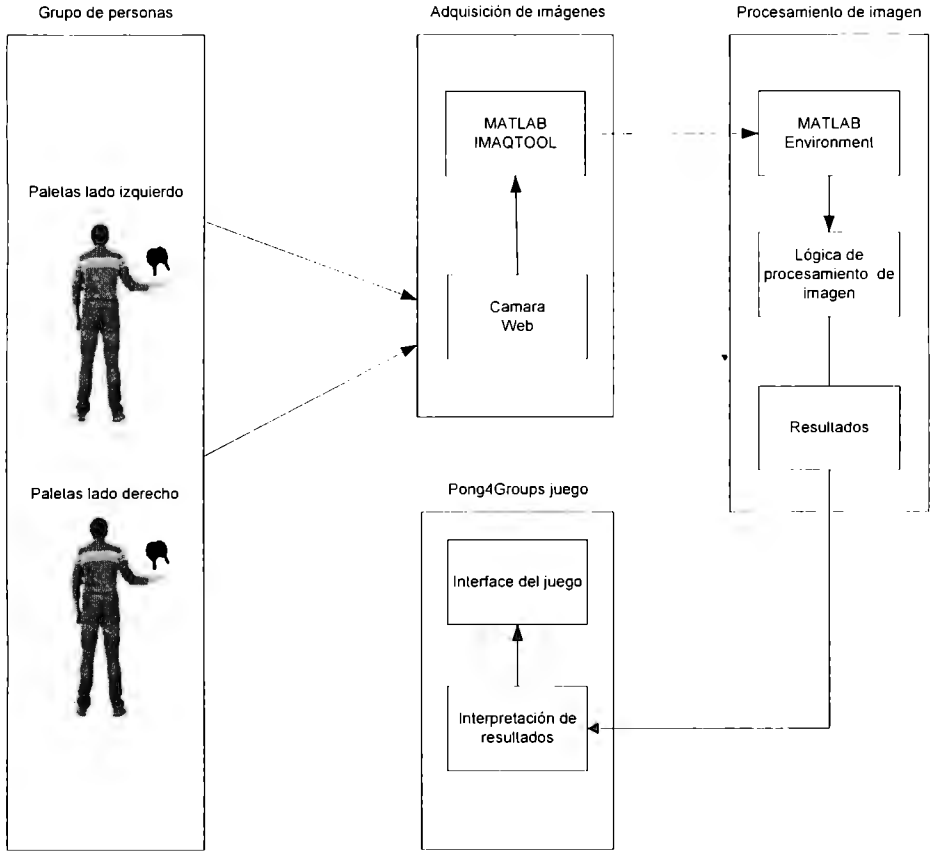


Figura 3.4. Componentes y flujo de procesamiento del juego Pong4Groups

Capítulo 4.- Implementación del juego Pong4Groups

A partir del diseño general se comienzan a desarrollar los diferentes módulos que conforman el juego. Todos los módulos se implementaron por separado. Una vez que cada uno de los módulos cumplía su función se comenzó la integración. La primera implementación correspondió al videojuego, en una primera etapa controlado por medio del teclado de la PC.

4.1 Videojuego

Para implementar el videojuego se consideraron las entidades participantes de este las cuales se definieron como sigue:

Figura 4.1 Diagrama de los componentes del juego

Como primera versión se implementó el control de las barras por medio del teclado del ordenador para después acoplar la parte de la interpretación de las imágenes.

Tomando como base el paradigma de programación orientada a objetos se realizó la implementación de los componentes del juego, comenzando por la clase Pong y esta a su vez conteniendo a todos los elementos propios de la interfaz del videojuego. En la figura 4.2 se muestra el diagrama de relaciones de la clase Pong con las demás clases. Dentro de esta clase se realiza el llamado al pintado de los objetos que la componen.

El objeto Paleta (Figura 4.3) contiene todos los métodos para realizar los movimientos, cambios de direcciones y cambios de velocidad a partir de la entrada recibida.

El objeto Ball (Figura 4.4) contiene los métodos para realizar los cálculos de velocidad y dirección, cambios de ángulo y detección de colisiones.

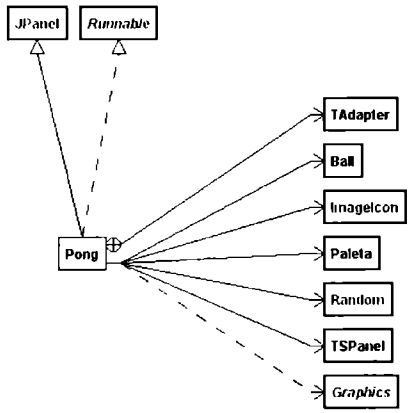


Figura 4.2 Diagrama de relaciones de la clase Pong

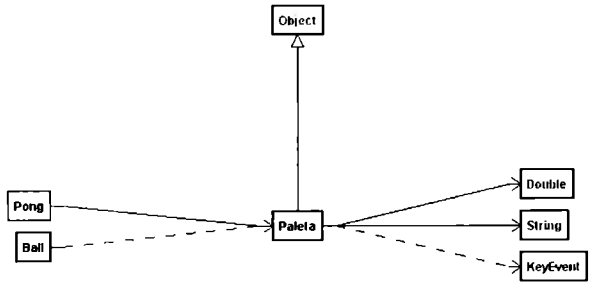


Figura 4.3 Diagrama de relaciones de la clase Paleta

Esta implementación se realizó utilizando la herramienta de desarrollo NetBeans 7.2.1 y el SDK 1.6 de Java.

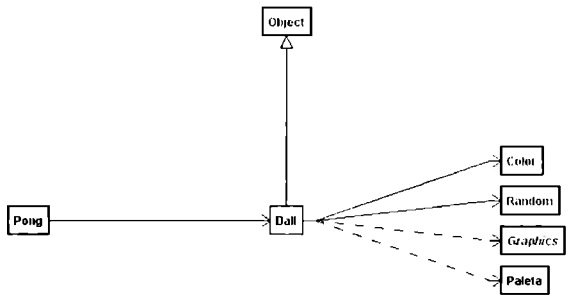


Figura 4.4 Diagrama de relaciones de la clase Ball

4.2 Adquisición de imágenes

La adquisición de imágenes se realiza a través de una cámara web Logitech c270. Los datos de formato de imagen son los siguientes:

- Resolución: 640x480
- Espacio de color de regreso: RGB24

De acuerdo al sitio del fabricante de la cámara web, el FOV (Campo de visión) es de 60°[30]. Para realizar el cálculo físico del campo de visión y saber el ancho que podría cubrir la cámara se realizaron las mediciones correspondientes tomando en cuenta la distancia de la posición de la cámara hacía dos puntos situados a 3 y 1.8 metros. Esto para conocer los límites izquierdo y derecho en donde la cámara tendría la visión de los jugadores involucrados [31].

Tomando en cuenta la figura 4.5 se obtuvieron los siguientes valores

Estableciendo x=300 cm, y=240 cm.

Obtenemos:

α=22.6

Estableciendo x=180 cm, y=152 cm.

Obtenemos:

α=22.9

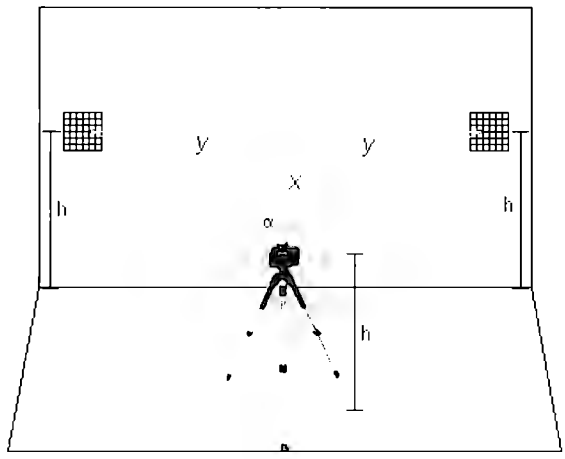


Figura 4.5 Mediciones para obtener el FOV

Por lo que en realidad el FOV que nos entrega la cámara utilizada es de 45.5°. Esto nos da la amplitud del área que podemos utilizar para que las paletas sean detectadas.

La cámara conectada a través del puerto USB es detectada por la herramienta de MATLAB IAT para posteriormente realizar la captura de cuadros de imagen. Se tomaron muestras de imágenes con las paletas situadas enfrente de la cámara a las distancias anteriormente mencionadas para obtener el rango de radio aproximado en pixeles de los círculos que de captaban en la imagen.

4.3 Procesamiento de la imagen

La implementación de la detección de círculos se compone de los siguientes procesos.

- Fase 1.- Tratamiento de la imagen recibida
- Fase 2.- Procedimiento de detección de círculos
- Fase 3.- Manejo del resultado
- Fase 4.- Integración

4.3.1 Tratamiento de la imagen recibida

Para poder determinar los círculos detectados en cada lado y el color de los mismos es necesario iniciar con una fase de tratamiento de imagen y prepararla para una segunda fase que es el procedimiento de detección de círculos. La necesidad de saber el método e implementación de detección círculos a implementar, proporciono los requerimientos de entrada realizados en la fase 1 y las especificaciones del resultado obtenido para la fase 3.

Se probaron diferentes implementaciones de la transformada de Hough en MATLAB para la detección de círculos. Entre estas se encuentra una función ya implementada dentro de la suite de MATLAB, sin embargo solo es para la detección de líneas. Después de una evaluación de cuatro implementaciones de la transformada de Hough para la detección de círculos, se seleccionó una rutina implementada por el Dr. Yuan-Liang Tan de la Universidad Tecnológica de Chaoyang en Taiwan. [32][33]

Dado que esta implementación acepta una imagen binaria como entrada para determinar el número de círculos en una imagen se realiza el siguiente procedimiento para preparar la imagen.

1. Se captura un cuadro de la imagen con el formato previamente especificado.

2. Se obtiene la imagen en escala de grises a partir de la original.

3. Usando el espacio de color RGB, la dimensión del color azul y rojo, por separado, se extraen de la imagen original.

4. Para cada color se realiza una operación de resta de matrices de la capa de color de la imagen con la matriz de escala de grises. Esto nos da los objetos de color rojo o azul de la imagen en intensidades de gris.

5. Aplicamos un filtro a la imagen para quitar ruido de tipo "sal y pimienta"[34].

6. Convertimos la imagen en binaria aplicando un umbral para proveer el correcto aislamiento de los objetos que cumplen con el color.

7. Quitamos objetos que tengan una área menor a la esperada por los círculos a 1.8 y 3 mts.

8. Partimos a la mitad las matrices correspondientes a cada color. Haciendo esto nos queda un lado derecho y un lado izquierdo para cada color, lo que se traduce en 4 imágenes binarias distintas.

9. Se envían las imágenes a la evaluación para detectar círculos.

En la figura 4.6 se muestra el flujo para preparar la imagen de entrada al procedimiento de detección de círculos mostrado en la figura 4.7.

4.3.2 Procedimiento de detección de círculos

Una vez que se tiene la imagen en formato binario, esta es enviada al proceso de la transformada de Hough, el cual entrega un resultado a través de un arreglo nx4 de círculos donde las primeras dos posiciones representan las coordenadas del centro del círculo, la tercera el radio y la cuarta el porcentaje del perímetro del circulo detectado.

Dado que la circunferencia de cada círculo (paleta) varía con la distancia a la que se encuentre alejada de la cámara, se determinaron dos rangos de posibles valores de radios en pixeles de las paletas para 1.8 y 3 metros. Esto mejora la velocidad de procesamiento para detectar los círculos en las imágenes ya que entre más amplio el rango se podrían tener más círculos candidatos. Adicionalmente, otro factor que influye en la velocidad de detección de círculos es el diámetro del círculo detectado el cual es determinado por el tamaño físico de la paleta. Se consideró también el número de Lúmenes en los ambientes probados solo como dato informativo y que puede ser de utilidad para establecer un ambiente controlado en luminosidad. Este dato se obtuvo a través de la aplicación de iPhone llamada LuxMeter situando la posición de la cámara a las mismas distancias probadas [35].

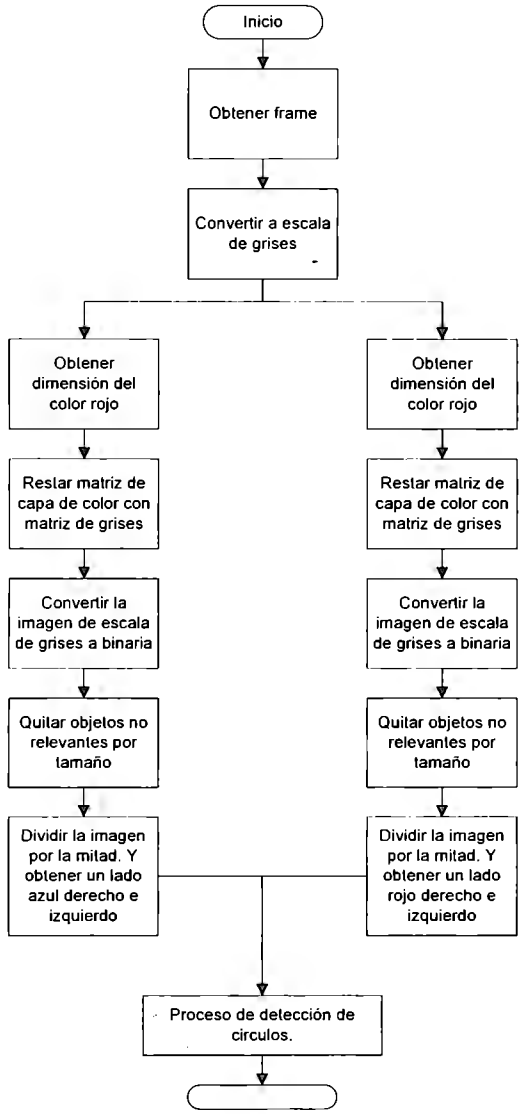


Figura 4.6 Proceso de detección de círculos.

Los datos de las características de los círculos detectados en correspondencia a su distancia se detallan en la tabla 4.1

Característica	Distancia (mts.) 1.8	3
Promedio de diámetro en pixeles	80	48
Área en pixeles	4300	1500
Luminancia en luz natural (lúmenes)	235	542
Luminancia en luz artificial (lúmenes)	34	58

Tabla 4.1. Características y valores de los ambientes probados

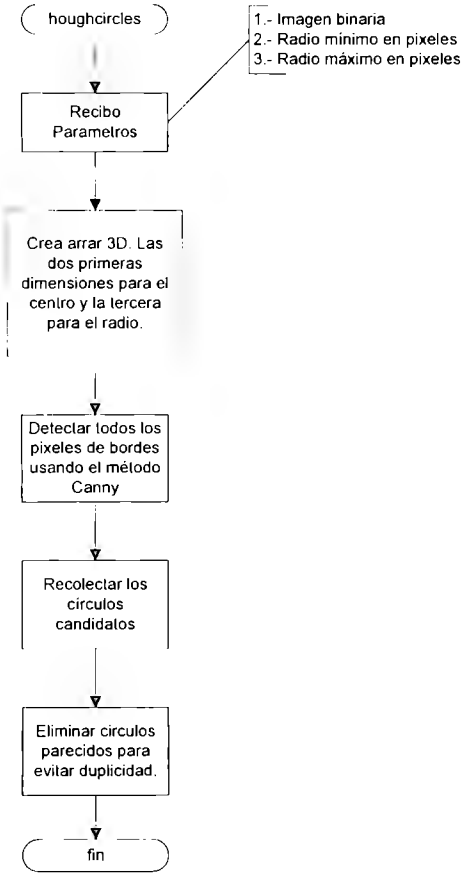


Figura 4.7. Proceso de detección de círculos

4.3.3 Manejo del resultado

Como se mencionó anteriormente, el resultado de la detección de círculos es entregado a través de un arreglo *nx4* con información de los círculos detectados. Cada arreglo significa un lado y un color, por lo que en total se obtienen cuatro arreglos del análisis de la imagen completa. El tamaño total de cada arreglo es el total de círculos detectados por imagen. En la figura 4.8 se muestra la representación del resultado.

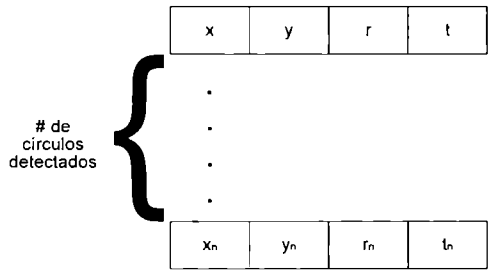


Figura 4.8. Representación de un arreglo de resultado de la detección de círculos

Para simplificar estos datos se obtiene el tamaño de cada arreglo y el valor se deposita en un vector de resultados de 4 posiciones, cada posición contiene el total de círculos detectados por lado por color.

TAI = Totales azul izquierdo
TRI = Totales rojo izquierdo
TAD = Totales azul derecho
TRD = Totales rojo derecho

Figura 4.9. Vector de resultados

4.3.4 Integración

Una vez que los tres principales módulos estuvieron listos se procedió a realizar la integración de los mismos. Teniendo en cuenta que se utilizaron dos plataformas de desarrollo para el funcionamiento de los módulos el siguiente paso constaba en la integración de estos.

Debido a que juego está desarrollado en Java y el procesamiento de imágenes en MATLAB, se requería que el video juego pudiera tener acceso al resultado de la evaluación de las imágenes captadas (Figura 4.10).

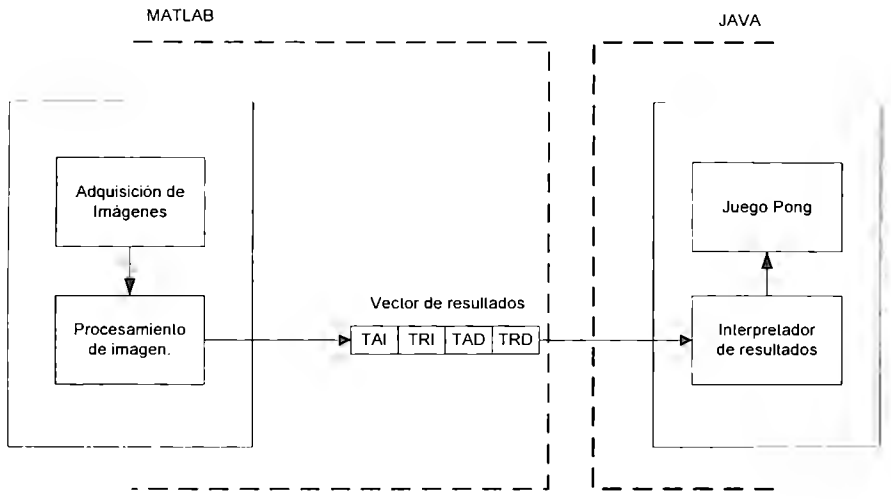


Figura 4.10. Esquema de integración inicial.

El punto más importante es que el videojuego pueda acceder al vector de resultados producto del análisis de la imagen. El poder hacerlo requería convertir el vector de resultados en un tipo de dato que la máquina virtual de Java pudiera contener. Para realizar esto fue necesario convertir la parte de código en MATLAB

a Java utilizando el producto *MATLAB Builder JA*, el cual crea clases Java a partir de un código escrito en MATLAB. Estas librerías resultantes pueden ser integradas a un proyecto en Java e interpretadas a través del compilador en tiempo de ejecución *MATLAB Compiler Runtime*[36]. Las clases resultantes se incluyeron en un archivo llamado Paddles.jar, el cual se incluyó dentro del proyecto de Java del videojuego y así tener la funcionalidad de la parte programada en MATLAB disponible como clases en Java.

Una vez integradas las librerías de Procesamiento de imágenes al proyecto en Java era necesario establecer una relación de servicios, en donde el videojuego toma el rol de consumidor de servicios del procesamiento de imágenes con el objetivo de saber cuáles serán sus acciones.

La primera versión de la implementación represento un problema de sincronización de los módulos, debido a que el Videojuego contenía un solo hilo (*thread*) el cual tiene la tarea de refrescar el panel de juego. Dentro de este refrescamiento se encuentra la lógica del movimiento de la pelota, las barras y colisiones. La integración del módulo de procesamiento de imagen al hilo principal del Videojuego representaba instanciar una clase Paddle (Clase principal de la rutina de procesamiento de imágenes) en cada ciclo del hilo principal. Esto significó una sobrecarga de objetos Paddle debido a que la inicialización de esta clase tomaba un tiempo considerable, solo obtenía un *frame* cuando otro objeto se estaba iniciando tratando de utilizar la cámara pero encontrando que estaba siendo utilizada por el objeto anterior, porque lo que eventualmente esto generaba un error y a pesar que el vector de resultados se obtenía como una variable más en el ambiente de ejecución, no era lo suficientemente rápido para efectuar el movimiento de las barras de Pong (Figura 4.12).

A pesar de que el módulo de procesamiento de imágenes se pudo integrar a la misma plataforma del juego, esta forma de implementación resultó ser inservible. El requerimiento era obtener los *frames* de una manera continua sin tener que

inicializar el módulo de procesamiento de imágenes en cada ciclo de pintado del tablero de juego.

La solución consistió en crear un ciclo infinito en el procesamiento de imagen que obtuviera cuadros de imagen en tanto estuviera activo. El envío del vector de resultados, sin embargo, debería ser por un método distinto. Por lo que se agregó una capa adicional. La solución consiste en utilizar *Sockets*. Un *socket* es el punto final en una cadena de comunicación entre dos programas ejecutándose en una red informática. Las clases de *Sockets* son usadas para representar la conexión entre un programa Cliente y un programa Servidor [37]. En el Anexo A se incluye más información sobre el tema de los *Sockets*.

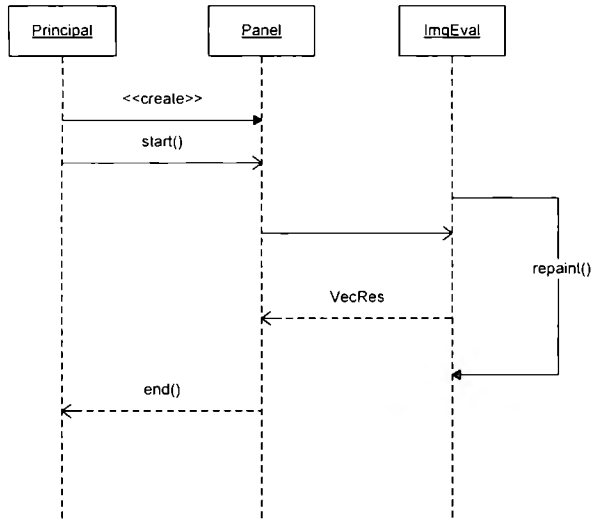


Figura 4.12 Primera implementación de la integración de módulos.

Se desarrolló un servidor en Java que fuera capaz de recibir peticiones TCP/IP en un puerto específico y un cliente en MATLAB que conectara a este servidor para

enviar el vector de resultados. En este esquema, el envío de resultados depende de la velocidad de procesamiento de las imágenes. Por lo tanto se agregaron al videojuego dos procesos (hilos) adicionales que en paralelo se ejecutarían con el proceso principal del juego. Estos dos procesos adicionales corresponden al módulo de procesamiento de imágenes, en donde el Vector de Resultados ya no es regresado directamente como variable en el ambiente de Java, sino, enviado a través de un cliente al segundo proceso adicional que es un Servidor el cual recibe los datos y los guarda en una variable que posteriormente es interpretada para determinar los movimientos correspondientes a las barras de Pong. La secuencia de inicio de los procesos del juego se muestra en la figura 4.13.

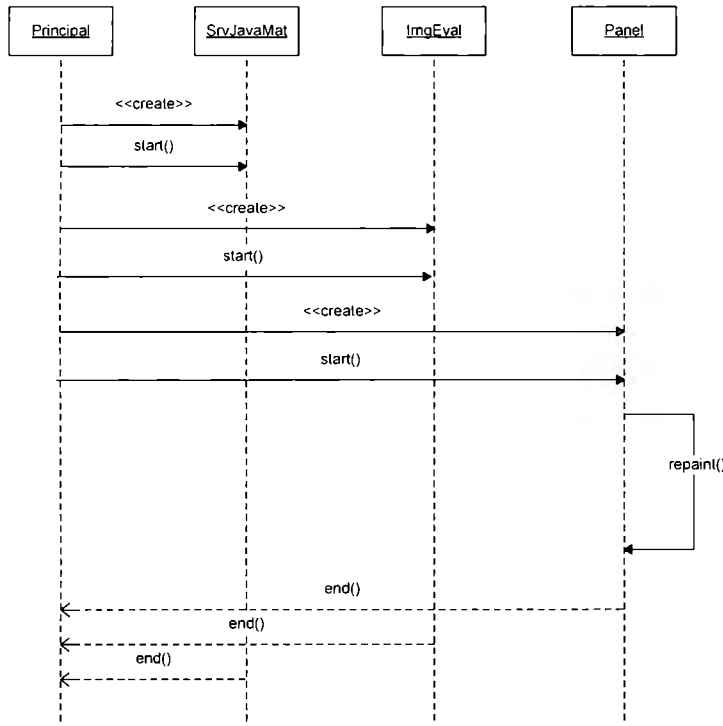


Figura 4.13. Esquema de procesos (hilos) del juego.

4.4 Parámetros de inicio del juego

Se consideraron tres variables como parámetros de entrada al inicio del juego (Figura 4.14). Estos valores son utilizados tanto en el procesamiento de imágenes como en la lógica del juego. Los tres valores a los que se hace mención son los listados a continuación:

1. Número de Jugadores: Una variación del juego se puede dar cuando de un lado hay más jugadores que del otro. La velocidad de cada barra estará determinada por el número total de paletas registradas. Por lo tanto la velocidad será calculada de acuerdo a la siguiente fórmula:

$$\text{Vel. Paleta} = \text{Vel. Default} * \frac{\#\text{ paletas configuradas}}{\#\text{ paletas detectadas}}$$

2. Distancia: La distancia a la que se va a jugar. Este valor permite seleccionar un rango específico en el tamaño de la circunferencia de los círculos a detectar. Entre más distancia, más jugadores, menos el tamaño de los círculos y más rápido el proceso de detección de círculos.

3. Tipo de Luz.- Se configuraron diferentes filtros para la conversión de imagen de escala de grises a binaria de acuerdo a valores arrojados con Luz Natural y Luz Artificial en ambientes de prueba. Aunque estos valores pueden variar de acuerdo a la cantidad y el tipo de luz en el ambiente en donde se juega.

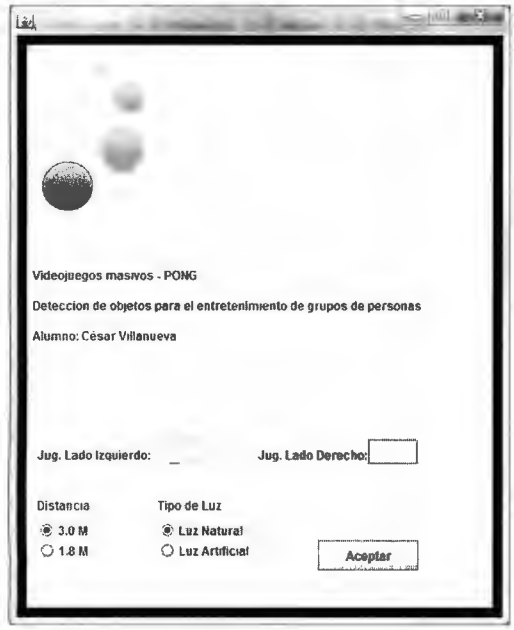


Figura 4.14 Pantalla de inicio del Juego.

4.5 Herramientas y movimientos del Juego

La mecánica del juego como ya se mencionó en la sección anterior, consiste detectar paletas bicolores (Figura 4.15) a partir de una imagen dividida en dos partes. Las paletas están hechas de material de cartulina en color rojo y azul. Los colores accionan la barra de Pong hacía arriba o abajo. Cuando se tiene la posición correcta de la barra y no se quiere mover a ningún extremo es necesario quitar el color expuesto de la paleta ya sea bajando la paleta o situándola en una posición donde no se exponga el circulo de color.

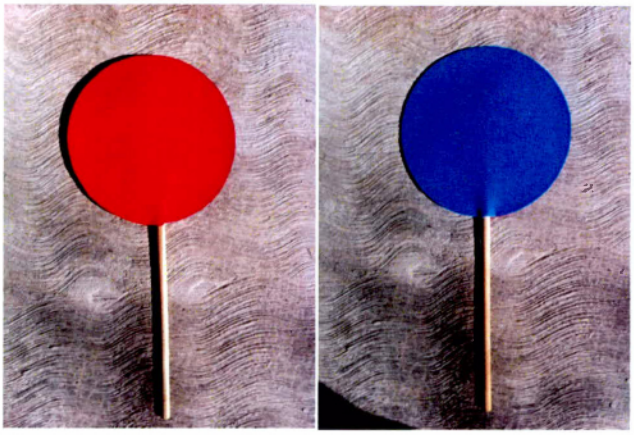


Figura 4.15. Vista física de la paleta

Color	Dirección de la Barra
	Abajo
	Arriba
Nulo o figura no reconocida	Inmóvil

Figura 4.16 Movimientos de la paleta

Capítulo 5. Resultados

Como se esperaba, varios factores del medio ambiente inciden directamente en el análisis de la imagen para detectar las paletas. El más importante siendo la cantidad de luz y la dirección de está afectando directamente el contraste y color capturado por la cámara y segundo la dirección de la luz aplicada sobre las paletas. Estos factores se describen en el curso de este capítulo, así como la parte social y tecnológica.

5.1 Generalidades

El juego inicia solicitando los parámetros necesarios para su optimización para posteriormente iniciar los procesos correspondientes a la interface gráfica, el servidor de recepción de resultados y el módulo de procesamiento de imágenes.

La detección de objetos es tan efectiva en proporción a la óptima ambientación en que se desarrolla el juego. Mientras se tenga un ambiente óptimo la detección de las paletas será más precisa. De la misma manera, debido a la libertad de movimiento sin restricciones de las paletas de juego, la detección de estas también está sujeta a la posición en la que se encuentran. Se registraron las variaciones en los movimientos de las paletas detectadas.

5.2 Angulo y posición de la paleta

Suponiendo un ambiente de iluminación óptimo los círculos de las paletas tienen algunas restricciones para poder ser detectados, con base a su ángulo vertical y horizontal. Las paletas son detectadas hasta con 45° de giro horizontal (Sobre el eje Y). Cuando el giro es de más de 45°, la paleta seguirá siendo detectada como objeto potencialmente circular sin embargo el procedimiento de detección de círculos discriminará el objeto a analizar debido que el valor de su radio sale del rango especificado. También es de notar que el algoritmo tiene la capacidad de

discriminar objetos adicionales que cumplen con el patrón de color pero no con la forma del objeto deseado, en este caso, circular. Las variaciones en el ángulo de inclinación hacia adelante o hacía atrás son detectadas en general mientras se tenga la cantidad de iluminación suficiente para resaltar el color.

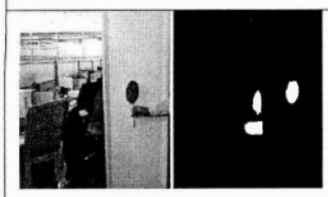
Imagen/Objetos potenciales	Positivo/#	Observaciones
	Si/1	Posición Frontal. Óptima.
	Si/1	Giro de 45°
	No	Giro de 75°
	Si/1	Inclinación hacia atrás
	Si/1	Inclinación hacia adelante

Tabla 5.1 Diferentes ángulos captados de la paleta con optima iluminación

5.3 Diferentes tipos de círculos (bien formados, malformados), figuras y ruido.

Dentro del ambiente de juego, en donde se pueden extraer objetos que emparejan el color de las paletas utilizadas. Adicionalmente se puede obtener un círculo mal formado, con forma parcial o con ruido. Estos factores crean elementos generadores de fallas en la precisión de la detección de los círculos. A pesar de que la Transformada de Hough es un método conocido por ser capaz de detectar círculos en ambientes con ruido es propenso a errores atribuibles a la calidad de la imagen. En la tabla 5.2 se muestran algunas imágenes que ejemplifican este caso. En la primera imagen se puede apreciar que existen varios objetos de diferentes tonalidades de color azul dado como resultado una detección inexacta al 25% para este caso, afectando las acciones del juego que dependen de este proceso. En el caso de las paletas color rojo con una detección del 100% de precisión. En la figura 5.3 se muestran círculos que a pesar de que están mal formados o incompletos el algoritmo detecta el número correcto de objetos.

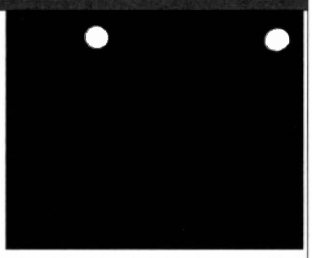
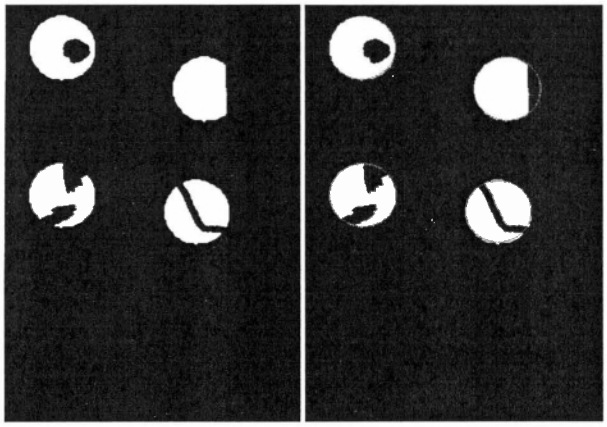
Imagen original	Imagen analizada para paletas azules	Imagen analizada para paletas rojas
		
Resultado:	4 círculos azules en el lado izquierdo, 1 en el lado derecho	1 círculo rojo por cada lado

Figura 5.2 Imagen con objetos significativos y no significativos

Se asigna un valor importante al factor de iluminación para la detección de objetos cuando no hay una restricción en el ambiente. La intensidad de la luz, el tipo de material, la posición, la reflexión, son solo algunas de las variables que entran en juego cuando existen variaciones de luminosidad. El no poder controlar este factor representa una gran desventaja para este tipo de aplicaciones.



a b

Figura 5.3 Imagen con círculos mal formados (a) y su detección (b).

Adicionalmente de realizaron pruebas con paletas con otro tipo de figura, que coincidieran en el factor de área total pero no en la forma, dando como resultado el comportamiento esperado. Aunque estos objetos son candidatos para la detección de círculos, ya que cumplen con los filtros de color y el área, el algoritmo no los reconoce como tal generando una salida de cero paletas detectadas.

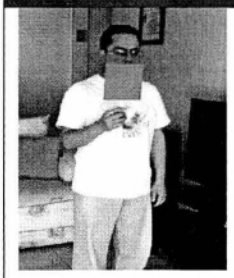
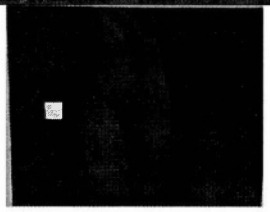
Imagen original	Imagen candidata para analizar
	
Resultado:	Ninguna paleta detectada

Figura 5.4 Paleta con figura de cuadro y mismos colores

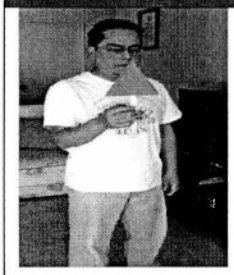
Imagen original	Imagen candidata para analizar
	
Resultado:	Ninguna paleta detectada

Figura 5.5 Paleta con figura de triangulo y mismos colores

El factor de ruido es un aspecto que podría controlarse, sin embargo en el ambiente para las pruebas en Pong4Groups se considero que ruido ocasional y no controlado podría intervenir en la detección de las figuras. Para los casos en los que se observo este tipo de ruido, incidió directamente en la detección de los objetos debido a que puede tomar cualquier forma y pasar de un cuadro de

imagen en donde la forma no corresponda a la figura deseada, a un cuadro de imagen donde podría parecer un circulo mal formado por lo que el sistema puede interpretarlo como un objeto valido. El ruido, en el caso del ambiente probado, puede dar como resultado aleatoriamente un falso positivo. Para este sistema, el ruido se origina cuando existen objetos, en el ambiente de juego, con un color parecido al de los objetos significativos (paletas). Las acciones en contra de la detección un falso positivo por motivo de ruido se dan a través de la discriminación de objetos por área total y la no coincidencia de la figura.

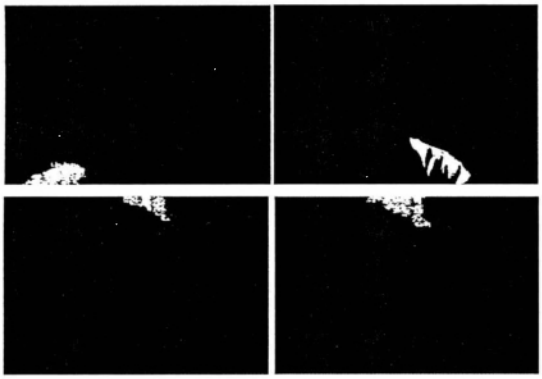


Figura 5.6 Diferentes imágenes con ruido

5.4 Velocidad de detección de algoritmos

Todos los componentes del videojuego se ejecutaron en una PC con las siguientes características:

Característica	Valor
Procesador	Intel Core I7 - 3610QM 2.30 Ghz
Memoria RAM	6 GB
S.O.	Windows 7 Professional

Disco Duro	750 GB
Cámara Web	Logitech c270

Se probó la velocidad de detección con dos imágenes muestra que contenían 4 y 8 círculos, correspondientes a 4 y 8 jugadores. La detección corresponde solo al proceso de la transformada de Hough el cual es el más significativo en cuanto a consumo de procesamiento. Los resultados fueron los siguientes.

Muestra	Tiempo promedio (ms.) (1.8 mts.)	Tiempo promedio (ms.) (3.0 mts.)
Imagen con 4 círculos	357	315
Imagen con 8 círculos	69	468
Imagen con 12 círculos	N.A.	649

Entre los factores con más importancia que afectan el rendimiento de la detección se tienen los siguientes:

Factor	Observación
Tamaño del objeto	Mayor el tamaño del objeto, mayor el tiempo de procesamiento por objeto.
Distancia de la cámara	Mayor la distancia, menor el tiempo de procesamiento por objeto.
Rango de radios de círculos	Entre más pequeño sea el rango de radios de círculos potenciales menor será el tiempo de procesamiento Total.
Ruido	Entre más ruido contenga la imagen, mayor será el tiempo de procesamiento Total.
Número de objetos	Mayor el número de objetos, mayor el tiempo de procesamiento por cuadro.

5.5. Prueba productiva.

Se realizó una demostración del funcionamiento del sistema con 2 grupos de tres personas, en total 6 personas. Para esta prueba se tomo como referencia una distancia de 3 metros, que es la distancia máxima configurada del sistema y en donde el ángulo de visión de la cámara permite colocar 6 personas de lado a lado de una manera cómoda y se calibro de acuerdo a los niveles de luminosidad que se presentaban en ese momento. Previamente también se ofreció una breve explicación de en qué consistía el juego y su funcionamiento.

Figura 5.7 Imágenes del funcionamiento del juego

De una sesión de 10 minutos se obtuvieron los siguientes datos:

- Se realizaron 1383 procesos de detección de objetos, esto nos arroja la siguiente información:

- 138.3 procesos de detección de objetos por minuto en promedio
- 2.305 procesos de detección de objetos por segundo en promedio
- 433.8 milisegundos en promedio por proceso de detección
- Se obtuvieron aproximadamente 35 resultados con uno o dos objetos adicionales detectados, esto nos genera un porcentaje de error del 2.53% específicamente para el ambiente probado.

Tomando en cuenta que la localidad en donde se realizó la prueba y las condiciones no fueron totalmente controladas, se puede establecer que el sistema tuvo una buena respuesta a pesar de haber diferentes elementos considerados como ruido potencial. Un segundo factor que influyó en la correcta detección de las paletas fue la posición en las que se encontraban, con una inclinación horizontal o vertical, en tales casos algunas veces las paletas no eran detectadas.

5.8 Factores que influyen en la correcta detección de objetos.

En general, a pesar de no contar con un movimiento 100% preciso y exacto, eso no determino que las personas pudieran jugar con una fluidez del juego aceptable y suficiente para lograr envolver a los participantes y proporcionar diversión en la sesión que al final es el objetivo principal.

Capítulo 6. Conclusiones

El sistema se logró implementar tomando en cuenta valores de iluminación disponibles durante su desarrollo, funcionando adecuadamente con una previa calibración. Sin embargo el ambiente ideal siempre será aquel donde se tenga control de los factores que afectan el comportamiento del juego y en esta primera versión del videojuego Pong4Groups no fue el ideal. La luminosidad jugó un factor determinante. La variación de movimientos y posiciones en las que puede captar una paleta es muy amplia y por lo tanto una paleta puede ser descartada por el simple hecho de no estar en el ángulo o inclinación correcto afectando las acciones del juego.

Este tipo de juegos podría convertirse en una alternativa para mantener a ciertas multitudes entretenidas, siempre y cuando se mantenga una mecánica de juego entretenida que envuelva a la persona y la relación de costo/beneficio de implementación no represente un gran gasto tomando en cuenta el objetivo que se busca.

La implementación con una cámara web para la adquisición de imágenes presenta una limitación en calidad de imagen y ángulo de visión. A pesar de esto, la infraestructura que se presentó para la implementación del videojuego Pong4Groups tiene mucha flexibilidad para crecer en cuestiones técnicas. La adquisición de imágenes desde diferentes fuentes (cámaras) crearía un campo de visión mucho más amplio. Con equipos dedicados a procesar las imágenes de cada posición, los resultados de tal procesamiento pueden enviarse de manera continua al módulo controlador del juego distribuyendo la carga en dos capas, la capa de adquisición y procesamiento de imágenes y la capa de interpretación y dinámica del videojuego.

Adicionalmente el método de detección de objetos puede ser mejorado o cambiado e incluso cambiar el lenguaje utilizado con el fin de optimizar el tiempo

de procesamiento y adaptarse al esquema de manejo de resultados sin modificar el módulo del videojuego.

Las paletas usadas en este trabajo se hicieron con materiales comunes y baratos. Pueden mejorarse utilizando cierto tipo de luz que garantice una cierta luminosidad. Este tipo de material ya es utilizado por los videojuegos existentes en el mercado [38], agregando detección de movimiento y profundidad. Esto por supuesto agregando un costo que dependiendo del objetivo de uso puede ser o no justificable.

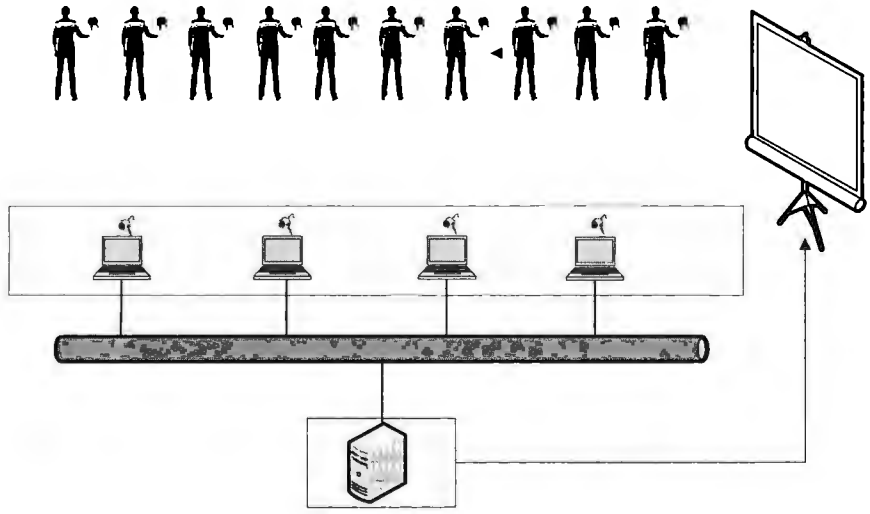


Figura 6.1. Múltiples cámaras sirviendo como adquisición de imágenes

De este sistema pueden también desprenderse varios temas dignos de ser tomados en cuenta como investigación, como lo es la importancia de la incidencia de luz sobre los objetos que se desean detectar o reconocer, el mejoramiento de la detección de ciertos objetos en tiempo real o nuevas tecnologías de entretenimiento.

Será el tiempo el que nos confirme si este tipo de videojuegos puede tener una oportunidad real de entrar en un esquema productivo y popular o permanecer como potenciales proyectos para una sociedad con una gran cartera de opciones de entretenimiento.

Referencias Bibliográficas

[1] Daynes-Amizande D., Pausch R., Seitz S.2002. Techniques for Interactive Audience Participation. Proceedings of the 4th IEEE International Conference on Multimodal Interfaces

[2] Bregler, Castiglia, DeVincezo. 2004. Squidball An Experiment in Large Scale Motion Capture and Game Design.

[3] Sieber, J., McCallum, S., Wyvill G. 2009. Ball Bouncer: Interactive games for theater audiences. In Chi'09 workshop - Crowd -Computer interaction (pp.29-31). Boston.

[4] Daynes-Amizande D. Audience Interaction. [Online]. http://www.monzy.org/audience/

[5]. Magerkurth C., Cheok A., Regan R.,Nilsen T. 2005. Pervasive Games: Bringing Computer Entertainment Back to the Real World. ACM Computers in Entertainment, Vol. 3, No. 3, Julio 2005.Articulo 4A

[6] Hardy, A., 2012. Audience-Driven Video Games For Conferences. Corbin Ball Associates. [Online] http://www.corbinball.com/articles_technology/index.cfm?fuseaction=cor_av&artID=8930

[7] Jesse Schell. 2005. Understanding Entertainment: Story and Gameplay Are One. ACM Computers in Entertainment, Vol. 3, No. 1

[8] Fisher, Perkins, Walker, Wolfart. Hypermedia Image Processing Reference : Wiley, 1997

[9] Indian Institute of Technology Madras. Department of Computer Science and Engineering. Das Sukhend. [Online].
http://www.cse.iitm.ac.in/~sdas/vplab/courses/CV_DIP/PDF/LECT_smooth_HT.pdf

[10] Computer Vision & Media Technology Laboratory (CVMT). Aalborg University. Dinamarca. [Online] 2009. http://www.cvmt.dk/education/teaching/f09/VGIS8/AIP/hough_09gr820.pdf

[11] Rhody, H., 2005. Lecture 10: Hough Circle Transform. Rochester Institute of Technology.[Online]. http://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf

[12] Teh-Chuan Chen, Kuo-Liang Chung. An Efficient Randomized Algorithm for Detecting Circles. and Image Understanding 83, 172–191 (2001)

[13] Roth M., Tanaka K., Weissman C., Yerazunis W. Computer Vision for Interactive Computer Graphics. IEEE Computer Graphics and Applications, May-June, 1998, pp. 42-53

[14] Gonzalez R.C., Woods R.E. 2002. Digital Image Processing, Second Edition. Prentice Hall.

[15] Vernon D. Machine Vision. Pearson Education Limited.

[16] Alasdair McAndrew. An Introduction to Digital Image Processing with MATLAB. Notes for SCM2511 Image Processing 1.Victoria University of Technology.

[17] MATLAB. Image Acquisition Toolbox – MATLAB.[Online]. http://www.mathworks.com/products/imaq/

[18] MATLAB. Image Processing Toolbox – MATLAB.[Online]. http://www.mathworks.com/products/image/

[19] MATLAB.MATLAB BuilderJA.[Online].http://www.mathworks.com/products/javabuilder/

[20] Jesse Schell. 2008. The Art of Game Design. Carnegie Mellon University. Elsevier.

[21] Jain R., Kasturi R.,Schunck B. 1995. Machine Vision. McGraw-Hill.

[22] Wolf Lior. Visual Object Recognition. [Online] . School of Computer Science, Tel-Aviv University. http://www.cs.tau.ac.il/~wolf/OR/#presentations

[23] Mohan A., Papageorgiou C., Poggio T. Example-Based Object Detection in Images by Components. IEEE Transactions on pattern analysis and machine intelligence, vol. 23, no. 4, 349-361, April 2001.

[24] Anwer, R.M. ; van de Weijer, J. ; Bagdanov, A.D. ; Vanrell, M. ; Lopez, A.M. Color attributes for object detection. Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, Junio, 2012, pp. 3306 - 3313

[25] OpenCV. Cascade Classification. [Online] http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html

[26] T. Mahalakshmi, R. Muthaiah and P. Swaminathan. An Overview of Template Matching Technique in Image Processing. Research Journal of Applied Sciences, Engineering and Technology. Abril , 2012

[27] Jain Neelu, Jain Neha. Coin Recognition Using Circular Hough Transform. International Journal of Electronics Communication and Computer Technology. Volume 2 Issue 3, Mayo 2012

[28] Richard O. Duda, Peter E. Hart. Use of the Hough Transformation To Detect Lines and Curves in Pictures. Graphics and Image Processing. Communications of the ACM, Vol. 15, No. 1, Enero 1972.

[29] Nandini N., Srikanta Murthy K., G. Hemantha Kumar, " Estimation of Skew Angle in Binary Document Images Using Hough Transform", World Academy of Science, Engineering and Technology 42, 2008, pp 44-49

[30] Logitech HD Webcam C270 Technical Specifications. [Online]. http://logitech-en-amr.custhelp.com/app/answers/detail/a_id/17556/section/troubleshoot/crid/405/lt_product_id/7079/tabs/1,3,2,4,5/cl/us,en

[31] Measuring Lens Field of View (FOV) and the Entrance Pupil -- PanoHelmp.com, [Online], http://www.panohelp.com/lensfov.html

[32] Yuan-Liang Tang's Home Page. [Online]. http://www.cyut.edu.tw/~yltang/

[33] Detects multiple disks (coins) in an image using Hough Transform- File Exchange - MATLAB Central, [Online], http://www.mathworks.com/matlabcentral/fileexchange/22543-detects-multiple-disks-coins-in-an-image-using-hough-transform

[34] 2-D median filtering - MATLAB medfilt2, [Online], http://www.mathworks.com/help/images/ref/medfilt2.html

[35] LuxMeter on the App Store on iTunes, [Online], https://itunes.apple.com/us/app/luxmeter/id526675593?mt=8

[36]. MATLAB.MATLAB Builder JA, [Online].http://www.mathworks.com/products/compiler/mcr/index.html

[37] Lesson: All About Sockets. The Java Tutorials, [Online],
http://docs.oracle.com/javase/tutorial/networking/sockets/

[38] PlayStation®Move Motion Controller, [Online], us.playstation.com/ps3/playstation-move/

[39] Bishop, C.M., "Neural Networks for Pattern Recognition", Oxford, University Press 1996

[40] Nitasha, Shammi Sharma, Reecha Sharma. Compárison Between Circular Hough Transform And Modified Canny Edge Detection Algorithm For Circle Detection.
International Journal of Engineering Research & Technology. Vol.1 - Issue 3 (Mayo - 2012)

[41] E.R. Davies. 2005 . Computer and Machine Vision. Third Edition. Elsevier.

[42] Rusty Harold Elliote. 2000. Java Netowk Programming, segunda edición. O'Reilly Media

ANEXO 1

Código de Procesamiento de Imagen

Código de adquisición de Imágenes (MATLAB)

```
function getPads(luz, distancia)
% getPads Función que regresa un array con los círculos detectados
% Parámetros de entrada
% luz: Tipo de Luz. Natural o Artificial
% distancia: Detectar círculos a 1.8 mts o 3 mts.

%%
%Evaluación y asignación de parámetros
%
%Valores por defecto para luminosidad
%establecemos el valor de luminosidad que se tomará en cuenta para extraer
%los objetos de cada color de acuerdo con las camaras digitales probadas
tipoLuzAzul=.06;    %.25 para microsoft, .14 para logitech %ultimo 20
tipoLuzRoja=.22;    %.22 para microsoft

%Valores por defecto de tamaño de pixeles a detectar
%Los establecemos para distancia de 1.8 mts
valorPixMin=18;
valorPixMax=35;

%Area de pixeles a discriminar
valorArea=1000;

%Si hay valores de entrada se revisa
if luz==0
    tipoLuzAzul=.28;
    tipoLuzRoja=.22;
    disp('Luz Natural')
end;

if distancia==0
    valorPixMin=10;
    valorPixMax=25;
    valorArea=600;
    disp('3.0 metros')
end;

%%

%Arreglo de resultados
arrtotales=zeros(1,4);

%se captura la información del dispositivo de video
try
   % Caramra Logitech
   vid = videoinput('winvideo', 1, 'RGB24_640X480');
catch
   try
      % Camara Microsoft.
      vid = videoinput('winvideo', 1, 'YUY2_640X480');
   catch
```

```
      errordlg('No webcam available');
   end
end
%vid = videoinput('winvideo', 1, 'YUY2_640X480');

%Se establecen las propiedades del video
vid.FramesPerTrigger =1;
vid.TriggerRepeat = Inf;
vid.ReturnedColorspace = 'rgb';
vid.FrameGrabInterval = 1;

%clc
%Comienza la captura del video
start(vid)
contador = 0;

%Variable que me permitirá mostrar la imagen de detección de círculos como
%una imagen en blanco y negro
imHandleBin=imshow(zeros(480,640));

%%

% Comienzo el loop para la adquisición de los frames
while(true)

     %Contador de ciclos
     contador=contador+1;

     %Obtiene la imagen de la cámara en un array de 3 dimensiones
     data=getdata(vid,1,'uint8');

     %Convertimos la imagen en imagen espejo
     data = flipdim(data,2);

     % Obtenemos el color rojo del frame.
     % Se resta el componente rojo de la imagen en escala de grises para
     % obtener los objetos rojos.
     im_rojo = imsubtract(data(:,:,1), rgb2gray(data));

     % Obtenemos el color azul del frame.
     % Se resta el componente azul de la imagen en escala de grises para
     % obtener los objetos azules.
     im_azul = imsubtract(data(:,:,3), rgb2gray(data));

     %Filtramos el ruido en la imagen que contiene objetos rojos
     im_rojo = medfilt2(im_rojo, [3 3]);
     %Filtramos el ruido en la imagen que contiene objetos azules
     im_azul = medfilt2(im_azul, [3 3]);

     %Convertimos la imagen de objetos rojos y azules de una imagen en
     %escala de grises a una imagen binaria
     im_rojo = im2bw(im_rojo,tipoLuzRoja);   %25 por default
     im_azul = im2bw(im_azul,tipoLuzAzul);  %.28 luz natural |  .15 luz
artificial Para logitech, .25 para microsoft

     % Quita los pobjetos con area menor a 1000 pixeles
     im_rojo = bwareaopen(im_rojo,valorArea);
     im_azul = bwareaopen(im_azul, valorArea);

     %Muestra los objetos de color y azul detectados y que se analizaran
```

```
    %para detectar circulos
    set(imHandleBin, 'CData',im_rojo+im_azul);

    %Obtengo la mitad izquierda de cada color
    matazulizq=im_azul(:,1:end/2);
    matrojoizq=im_rojo(:,1:end/2);
    %Obtengo la mitad derecha de cada color
    matazulder=im_azul(:, end/2:end);
    matrojoder=im_rojo(:, end/2:end);

    %Analizo cada una de las imagenes en total 4, 2 por cada color
    totazulizq=houghcircles(matazulizq, valorPixMin, valorPixMax, .33, 25);
    totrojoizq=houghcircles(matrojoizq, valorPixMin, valorPixMax, .33, 25);
    totazulder=houghcircles(matazulder, valorPixMin, valorPixMax, .33, 25);
    totrojoder=houghcircles(matrojoder, valorPixMin, valorPixMax, .33, 25);

   %Obtengo el primer resultado paletas azules lado izquierdo
     arrtotales(1)=size(totazulizq,1);
   %Paletas rojas lado izquierdo
     arrtotales(2)=size(totrojoizq,1);
   %Paletas azules lado derecho
     arrtotales(3)=size(totazulder,1);
   %Paletas rojas lado derecho
     arrtotales(4)=size(totrojoder,1);
     %Envío los resultados a través de un socket al servidor del juego
     %Convertimos el vector de resultados a cadena
     client('otherc-PC', 8765, 0, mat2str(arrtotales));
    %Borramos el frame de la memoria
    flushdata(vid);

end
%%
% Desactivamos la cámara
stop(vid);

finishup = onCleanup(vid);
% Limpiamos el video de la memoria
flushdata(vid);
```

Implementación de la transformada de Hough

```
function circles = houghcircles(im, minR, maxR, thresh, delta)
%
% HOUGHCIRCLES detects multiple disks (coins) in an image using Hough
% Transform. The image contains separating, touching, or overlapping
% disks whose centers may be in or out of the image.
%
% Syntax
%   houghcircles(im, minR, maxR);
%   houghcircles(im, minR, maxR, thresh);
%   houghcircles(im, minR, maxR, thresh, delta);
%   circles = houghcircles(im, minR, maxR);
%   circles = houghcircles(im, minR, maxR, thresh);
%   circles = houghcircles(im, minR, maxR, thresh, delta);
%
% Inputs:
```

```
%    - im: input image
%    - minR: minimal radius in pixels
%    - maxR: maximal radius in pixels
%    - thresh (optional): the minimal ratio of the number of detected edge
%          pixels to 0.9 times the calculated circle perimeter (0<thresh<=1,
%          default: 0.33)
%    - delta (optional): the maximal difference between two circles for them
%          to be considered as the same one (default: 12); e.g.,
%          c1=(x1 y1 r1), c2=(x2 y2 r2), delta = |x1-x2|+|y1-y2|+|r1-r2|
%
% Output
%    - circles: n-by-4 array of n circles; each circle is represented by
%          (x y r t), where (x y), r, and t are the center coordinate, radius,
%          and ratio of the detected portion to the circle perimeter,
%          respectively. If the output argument is not specified, the original
%          image will be displayed with the detected circles superimposed on it.
%
%
% Copyright (c), Yuan-Liang Tang
% Associate Professor
% Department of Information Management
% Chaoyang University of Technology
% Taichung, Taiwan
% http://www.cyut.edu.tw/~yltang
%
% Permission is hereby granted, free of charge, to any person obtaining
% a copy of this Software without restriction, subject to the following
% conditions:
% The above copyright notice and this permission notice should be included
% in all copies or substantial portions of the Software.
%
% The Software is provided "as is," without warranty of any kind.
%
% Created: May 2, 2007
% Last modified: Jan. 8, 2009
%

% Check input arguments
%disp(nargin);
if nargin==3
  thresh = 0.33;    % One third of the perimeter
  delta = 12;       % Each element in (x y r) may deviate approx. 4 pixels
elseif nargin==4
  delta = 12;
end
%fprintf('minR = %d, maxR = %d,  thresh =%d, delta=%d', minR,maxR,thresh,delta)
if minR<0 || maxR<0 || minR>maxR || thresh<0 || thresh>1 || delta<0
  disp('Input conditions: 0<minR, 0<maxR, minR<=maxR, 0<thresh<=1, 0<delta');
  return;
end

% Turn a color image into gray
% origim = im;
% if length(size(im))>2
%   im = rgb2gray(im);
% end

% Create a 3D Hough array with the first two dimensions specifying the
% coordinates of the circle centers, and the third specifying the radii.
% To accomodate the circles whose centers are out of the image, the first
% two dimensions are extended by 2*maxR.
maxR2 = 2*maxR;
hough = zeros(size(im,1)+maxR2, size(im,2)+maxR2, maxR-minR+1);
```

```
% For an edge pixel (ex ey), the locations of its corresponding, possible
% circle centers are within the region [ex-maxR:ex+maxR, ey-maxR:ey+maxR].
% Thus the grid [0:maxR2, 0:maxR2] is first created, and then the distances
% between the center and all the grid points are computed to form a radius
% map (Rmap), followed by clearing out-of-range radii.
[X Y] = meshgrid(0:maxR2, 0:maxR2);
Rmap = round(sqrt((X-maxR).^2 + (Y-maxR).^2));
Rmap(Rmap<minR | Rmap>maxR) = 0;

% Detect edge pixels using Canny edge detector. Adjust the lower and/or
% upper thresholds to balance between the performance and detection quality.
% For each edge pixel, increment the corresponding elements in the Hough
% array. (Ex Ey) are the coordinates of edge pixels and (Cy Cx R) are the
% centers and radii of the corresponding circles.
edgeim = edge(im, 'canny', [0.15 0.2]);
[Ey Ex] = find(edgeim);
[Cy Cx R] = find(Rmap);
for i = 1:length(Ex);
  Index = sub2ind(size(hough), Cy+Ey(i)-1, Cx+Ex(i)-1, R-minR+1);
  hough(Index) = hough(Index)+1;
end

% Collect candidate circles.
% Due to digitization, the number of detectable edge pixels are about 90%
% of the calculated perimeter.
twoPi = 0.9*2*pi;
circles = zeros(0,4);     % Format: (x y r t)
for radius = minR:maxR    % Loop from minimal to maximal radius
  slice = hough(:,:,radius-minR+1);  % Offset by minR
  twoPiR = twoPi*radius;
  slice(slice<twoPiR*thresh) = 0;  % Clear pixel count < 0.9*2*pi*R*thresh
  [y x count] = find(slice);
  circles = [circles; [x-maxR, y-maxR, radius*ones(length(x),1), count/twoPiR]];
end

% Delete similar circles
circles = sortrows(circles,-4);  % Descending sort according to ratio
i = 1;
while i<size(circles,1)
  j = i+1;
  while j<=size(circles,1)
    if sum(abs(circles(i,1:3)-circles(j,1:3))) <= delta
      circles(j,:) = [];
    else
      j = j+1;
    end
  end
  i = i+1;
end

% if nargout==0   % Draw circles
%   figure, imshow(origim), hold on;
%   for i = 1:size(circles,1)
%     x = circles(i,1)-circles(i,3);
%     y = circles(i,2)-circles(i,3);
%     w = 2*circles(i,3);
%     rectangle('Position', [x y w w], 'EdgeColor', 'red', 'Curvature', [1 1]);
%   end
%   hold off;
% end
```

Código del cliente que envía el resultado al videojuego

```
% CLIENTE TCP que se conecta a un servidor en java
%
% Uso client(host, port, intentos, arrtotalesstr)
function client(host, port, intentos, arrtotalesstr)

    import java.net.*
    import java.io.*

    %Por defaul
    retry       = 0;
    client_socket = [];

    while true

        retry = retry + 1;
        if ((intentos > 0) && (retry > intentos))
            fprintf(1, 'Demasiados intentos\n');
            break;
        end

        try

            % Creo el socket cliente
            client_socket = Socket(host, port);


            %Se crea el stream de salida al server
            outToServer = client_socket.getOutputStream();
            out = DataOutputStream(outToServer);

            fprintf(1, 'Conectado al servidor\n');

            %Envío los resultados
            out.writeBytes(arrtotalesstr);

            % Cerramos el socket
            client_socket.close;
            break;

        catch
            if ~isempty(client_socket)
                client_socket.close;
            end

            % pausa antes de reintentar
            pause(1);
        end
    end
end
```

ANEXO 2

Código del videojuego

```
/***************Clase principal del juego***************/
package com.itesm.pong;

import com.itesm.imagen.ImageEval;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.JWindow;

/*
 * Clase iniciadora del juego
 */

/**
 * clase principal del juego.
 * @author otherc
 */
public class Principal extends JFrame implements ActionListener{
   JFrame frame = new JFrame("ITESM PONG");
   //Clase que recibe los resultados
   private SrvJavMat srvJavMat = new SrvJavMat();
   //Funciones de tratamiento de la imagen
   private ImageEval colorPaddle = new ImageEval();
   JPanel content = (JPanel)getContentPane();
   JButton aceptar = new JButton("Aceptar");
   JRadioButton dLarga  = new JRadioButton("3.0 M"  , true);
   JRadioButton dCorta  = new JRadioButton("1.8 M"  , false);
   JRadioButton lNatural = new JRadioButton("Luz Natural", true);
```

```
JRadioButton lArtificial = new JRadioButton("Luz Artificial", false);
JTextField juglzq = new JTextField();
JTextField jugDer = new JTextField();
stalic boolean ejecuta = false;

public void actionPerformed(ActionEvent e) {
    if (e.getSource()==aceptar) {
        ImageEval.distancia = dLarga.isSelected() ? 0 : 1;
        ImageEval.tipoLuz = lNatural.isSelected() ? 0 : 1;
        Pong.jIzquierda = new Integer(juglzq.getText()).intValue();
        Pong.jDerecha = new Integer(jugDer.getText()).intValue();
        System.out.println("Jugadores Izquierda:"+Pong.jIzquierda+ "Jugadores Derecha:"+Pong.jDerecha);
        ejecuta=true;
    }
}

//Pantalla de entrada a la aplicación
public void showSplash() {

    content.setBackground(Color.white);
    ImageIcon imagen;

    // Definimos los limites de la ventana
    int width = 500;
    int height =615;
    Dimension screen = Toolkit.getDefaultToolkit().getScreenSize();
    int x = (screen.width-width)/2;
    int y = (screen.height-height)/2;
    setBounds(x,y,width,height);

    imagen=new ImageIcon("about.png");

    setLayout(null);

    setFont(new Font("Sans-Serif", Font.BOLD, 12));
    JLabel label = new JLabel(imagen);
    JLabel tesis1 = new JLabel("Videojuegos masivos - PONG");
    JLabel tesis2 = new JLabel("Detección de objetos para el entretenimiento de grupos de personas");
    JLabel alumno = new JLabel("Alumno: César Villanueva");
    JLabel lJuglzq = new JLabel("Jug. Lado Izquierdo:");
    JLabel lJugDer = new JLabel("Jug. Lado Derecho:");
    JLabel jLluz = new JLabel("Tipo de Luz");
    JLabel jLDist = new JLabel("Distancia");


    //JButton aceptar = new JButton("Aceptar");
    aceptar.setBounds(300, 500, 100, 30);
```

```
label.setBounds(10, 10, imagen.getIconWidth(), imagen.getIconHeight());
tesis1.setBounds(15, imagen.getIconHeight()+50, width, 20);
tesis2.setBounds(15, imagen.getIconHeight()+80, width, 20);
alumno.setBounds(15, imagen.getIconHeight()+110, width, 20);
lJuglzq.setBounds(20,400,130,30);
lJugDer.setBounds(240,400,130,30);
jLluz.setBounds(140,450,130,30);
jLDist.setBounds(20,450,130,30);

juglzq.setBounds(140, 400, 50, 25);
jugDer.setBounds(350,400, 50, 25);
dLarga.setBounds(20, 480, 100, 20);
dCorta.setBounds(20, 500, 100, 20);
lNatural.setBounds(140, 480, 100, 20);
lArtificial.setBounds(140, 500, 100, 20);

ButtonGroup dGroup = new ButtonGroup();
      dGroup.add(dLarga);
      dGroup.add(dCorta);

ButtonGroup lGroup = new ButtonGroup();
lGroup.add(lNatural);
lGroup.add(lArtificial);


contenl.add(label, BorderLayout.EAST);
contenl.add(tesis1, 1);
contenl.add(tesis2, 2);
content.add(alumno, 3);
content.add(aceptar);
content.add(juglzq);
content.add(jugDer);
contenl.add(dLarga);
content.add(dCorta);
content.add(lNatural);
contenl.add(lArtificial);
content.add(lJuglzq);
content.add(lJugDer);
content.add(jLluz);
content.add(jLDist);

aceptar.addActionListener(this);

Color ora = new Color(20);
content.setBorder(BorderFactory.createLineBorder(ora, 10));
```

```
    setVisible(true);

   //Tiempo de espera
    while(!ejecuta){
       try {
          Thread.sleep(100);
       } catch (Exception e) {}
    }

    //Esperamos un valor del boton apretado para llamar a los siguientes
    //Iniciamos el servidor Java que escuchara peticiones de matlab

    setVisible(false);
    //Iniciamos el servidor que recibe los valores de las imagenes
    new Thread(srvJavMat).start();
    //Iniciamos el evaluador de imagenes de matlab
    new Thread(colorPaddle).start();

}


public static void main (String[] args) {

    Principal prin= new Principal();
    //Llamamos a la ventana
    prin.showSplash();

       javax.swing.SwingUtilities.invokeLater(new Runnable() {

          public void run() {
          //Frame general que contiene el tablero de marcador
          JFrame frame = new JFrame("ITESM PONG");
          //Panel de informacion
          TSPanel jpanel1 = new TSPanel(Color.LIGHT_GRAY);
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          frame.setSize(680,340);//380, 340
          frame.setLocationRelativeTo(null);
          frame.setTitle("ITESM PONG");
          frame.setResizable(false);
          frame.getContentPane().setLayout(new BorderLayout());
          //Agregamos el panel de juego con la clase Pong
          frame.getContentPane().add(new Pong(380,340, jpanel1), BorderLayout.WEST);
          frame.getContentPane().add(jpanel1, BorderLayout.CENTER);
          frame.setVisible(true);

       }
```

```
        } );
      System.out.println("Adios");
   }

}
```

```
/**********************Clase  PONG controladora de los objetos del videojuego*******************/

package com.itesm.pong;

import com.itesm.imagen.EvalArray;
import com.itesm.imagen.ImageEval;
import java.awt.Color;
import java.awt.Event;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.geom.Rectangle2D;
import java.util.Random;
import javax.swing.ImageIcon;
import javax.swing.JPanel;

/*
 * Clase que controla los objetos del videojuego
 * Llama a los metodos de comportamiento de las paletas y la pelota.
 */

/**
 *
 * @author otherc
 */
public class Pong extends JPanel implements Runnable {

   private int x_ball = 100;
   private int y_ball = 100;
   private int radio = 10;
   private ImageIcon backimage;
   public static int velocidad=6;
   private int width;
   private int height;
   private Paleta pr;
   private Paleta pl;
```

```
private boolean mover_jugador1=false;
private boolean mover_jugador2=false;
private int factor_pelota=1;
private int anguloGrados;
private Ball ball;
private Random rand = new Random();
private TSPanel board;
private boolean pausa=true;
public static int jIzquierda;
public static int jDerecha;
public static int jugador1=0;
public static int jugador2=0;
//private ImageEval colorPaddle; //= new ImageEval();


/**
 * Constructor del juego donde se inicializan los objetos paleta y ball
 * @param width
 * @param heigth
 */
public Pong(int width, int heigth, TSPanel board) {

    addKeyListener(new TAdapter());
    this.board=board;
    this.width=width;
    this.height=heigth;
    x_ball = width - radio;
    y_ball = heigth - radio;
    //Se inicializan los grados de dirección de la pelota
    anguloGrados = Math.round(rand.nextFloat()*100)+130;
    ball = new Ball(x_ball,y_ball,radio,velocidad, anguloGrados);
    setFocusable(true);
    this.setBackground(Color.white);
    this.setSize(width, height);
    this.setMaximumSize(new java.awt.Dimension(width, height));
    this.setMinimumSize(new java.awt.Dimension(width, height));
    this.setPreferredSize(new java.awt.Dimension(width, height));
    //paleta derecha
    pr = new Paleta(width-20, height-(height/2)-40,10,50, width, height, "rigth");
    //paleta izquierda
    pl = new Paleta(4, height-(height/2)-40,10,50, width, height, "left");
    backimage = new ImageIcon("logo_ilesm.gif");  //imagen de fondo


}
```

```
/**
 *
 */
public void ballUpdate() {
  pausa = ball.detectaColision(0,0,width, height, pr, pl);

}

@Override
public void addNotify() {
  super.addNotify();
  Thread th = new Thread(this);
  th.start();

}

public void run() {

  while(true){
    //x_ball+=xDir;
    repaint();

    try{
      Thread.sleep(70);

    }catch (InterruptedException ex){

    }

     if(!pausa)
     {
       ballUpdate();
       repaint();
       board.draw();
     }

    //condiciones para mover a los jugadores
    //Ahora aqui aplica la evaluación de los valores de las paletas
    pr.movAutom();
    pr.move();
    pl.movAutom();
    pl.move();
    repaint();

     if(this.mover_jugador1){
       pr.moveKey();
       repaint();
```

```
        }
        if(this.mover_jugador2){
            pl.moveKey();
            repaint();
        }

    }


}


@Override
public void paint(Graphics g) {

    super.paint(g);
    Graphics2D g2 = (Graphics2D)g;
    //g2.dra

    int yima = backimage.getImage().getHeight(null);
    int xima = backimage.getImage().getWidth(null);

    g2.drawImage(backimage.getImage(), (this.width/2)-(xima/2), (this.height/2)-(yima/2), null);
    //this.setSize(width, height);
    width = getWidth();
    height = getHeight();

    //System.out.println("x,y" + width + ","+ height);
    g2.setColor(Color.red);
    //pintamos la bola, pasamos el contexto grafico
    ball.draw(g);

    g2.setColor(Color.blue);
    //pr.pintaBarra();
    g2.draw(pr.rr);
    g2.fill(pr.rr);


    g2.setColor(Color.blue);
    //pl.pintaBarra();
    g2.draw(pl.rr);
    g2.fill(pl.rr);


}

//Para jugar el juego con el teclado.
```

```
private class TAdapter extends KeyAdapter {

   @Override
   public void keyPressed(KeyEvent e) {

   if (e.getKeyCode() == KeyEvent.VK_UP) {
        pr.keyPressed(e);
        Pong.this.mover_jugador1=true;
       //System.out.println("|");
      }

   if (e.getKeyCode() == KeyEvent.VK_DOWN) {
        pr.keyPressed(e);
        Pong.this.mover_jugador1=true;
      }

    if (e.getKeyCode() == 65) {
       pl.keyPressed(e);
       Pong.this.mover_jugador2=true;
      }

   if (e.getKeyCode() == 90) {
      pl.keyPressed(e);
      Pong.this.mover_jugador2=true;
      }

   if(e.getKeyCode() == KeyEvent.VK_SPACE){
      pausa=false;
   }

   }

    @Override
   public void keyReleased(KeyEvent e) {

   if (e.getKeyCode() == KeyEvent.VK_UP) {
        Pong.this.mover_jugador1=false;
        pr.keyReleased(e);
       //System.out.println("|");
      }

   if (e.getKeyCode() == KeyEvent.VK_DOWN) {
        Pong.this.mover_jugador1=false;
        pr.keyReleased(e);
      }

    if (e.getKeyCode() == 65) {
```

```
            Pong.this.mover_jugador2=false;
            pl.keyReleased(e);
            //System.out.println("|");
           }

        if (e.getKeyCode() == 90) {
            Pong.this.mover_jugador2=false;
            pl.keyReleased(e);
            //System.out.println("|");
           }

           //repaint();
        }

    }

}
```

//**********Clase de la pelota del juego**********//

```
package com.itesm.pong;

/*
 * Clase de la Pelota del juego
 */

import com.itesm.imagen.ImageEval;
import java.awt.Color;
import java.awt.Event;
import java.awt.Graphics;
import java.util.Random;


/**
 *
 * @author CVR
 */
public class Ball {
   double x, y;          // coordenadas del centro de la pelota
   double velX, velY; // velocidad
   int radio;        // radio
   private Color color;  // color
   private static final Color DEFAULT_COLOR = Color.RED;
```

```
private Random rand = new Random();
private double anguloGrados;
private double dist;
private double ping=0;
private double pong=2;
//private ImageEval colorPaddle = new ImageEval();


/**
 *
 * @param x
 * @param y
 * @param radio
 * @param vel
 * @param anguloGrados
 * @param color
 */
public Ball(float x, float y, int radio, float dist, float anguloGrados,
    Color color) {
  this.x = x;
  this.y = y;
  // Convertimos la velocidad y el angulo en coordenadas
  this.anguloGrados=anguloGrados;
  System.out.println("anguloGradosIni="+anguloGrados);
  this.dist=dist;
  this.velX = (float)(dist * Math.cos(Math.toRadians(anguloGrados)));
  this.velY = (float)(-dist * (float)Math.sin(Math.toRadians(anguloGrados)));
  this.radio = radio;
  this.color = color;
}
/** Constructor con el color por defaull de la pelota
 * @param x
 * @param y
 * @param radio
 * @param vel
 * @param anguloGrados
 */
public Ball(float x, float y, int radio, float vel, float anguloGrados) {
  this(x, y, radio, vel, anguloGrados, DEFAULT_COLOR);
}

 public void cambiaAngulo (){

  //velX = (float)(dist * (float) Math.cos(Math.toRadians(anguloGrados)));
  //velY = (float)(-dist * (float)Math.sin(Math.toRadians(anguloGrados)));
  velX = (float)(pong * (float) Math.cos(Math.toRadians(anguloGrados)));
  velY = (float)(-pong * (float)Math.sin(Math.toRadians(anguloGrados)));
```

```
}

public void restart (){
   //System.out.println("vel "+ Pong.velocidad);
   pong=2;
   anguloGrados=rand.nextInt(70)+10;
   velX = (float)(pong * (float)Math.cos(Math.toRadians(anguloGrados)));
   velY = (float)(-pong * (float)Math.sin(Math.toRadians(anguloGrados)));
}

/** Pinta la pelota.
 * @param g
 */
public void draw(Graphics g) {
  g.setColor(color);
  g.fillOval((int)(x - radio), (int)(y - radio), (int)(2 * radio), (int)(2 * radio));
}


/**
 *
 * @param xmin
 * @param ymin
 * @param xmax
 * @param ymax
 * @param pr
 * @param pl
 */
public boolean detectaColision(int xmin, int ymin, int xmax, int ymax, Paleta pr, Paleta pl) {
  // Limites
  float ballMinX = xmin + radio;
  float ballMinY = ymin + radio;
  float ballMaxX = xmax - radio;
  float ballMaxY = ymax - radio;
  boolean pausa = false;

  // Clacula la posicion de la pelota
  x += velX;
  y += velY;
  //Condiciones de la velocidad
  //Condicion para los limites de X
  if (x+radio+10 < xmin ) {
    x = ballMaxX/2;
    y = ballMaxY/2;
    //y = rand.nextInt(ymax - radio*2 - 20) +radio + 10;*/
    anguloGrados = 180 - anguloGrados;
```

```
  //cambiaAngulo();
  Pong.jugador1 = Pong.jugador1+1;
  restart();
  pausa=true;
   /*velX = -velX; // Refleja de acuerdo a la normal*/
   //x = ballMinX;
} else if (x > xmax+radio+10) {

  x = ballMaxX/2;
  y = ballMaxY/2;
  //y = rand.nextInt(ymax - radio*2 - 20) +radio + 10;*/
  anguloGrados = 180 - anguloGrados;
  //cambiaAngulo();
  Pong.jugador2 = Pong.jugador2+1;
  restart();
  pausa=true;

}

//Condicion para los bordes de Y
if (y < ballMinY) {
   //velY = -velY;
  anguloGrados = 360 - anguloGrados;
   y = ballMinY;
} else if (y > ballMaxY) {
   //velY = -velY;
   anguloGrados = 360 - anguloGrados;
   y = ballMaxY;
}


//golpe en la paleta derecha
if(x+radio>pr.x && (y>=pr.y && y<=(pr.y+pr.height)) && x<pr.x+(pr.width/2)){
    //new Thread(colorPaddle).start();
    //System.out.println("pr.x="+pr.x+" "+"pr.y="+pr.y);
    //System.out.println("x="+x+" "+"y="+y);
    System.out.println("anguloGradosAntes= "+anguloGrados+" pr.contador= "+pr.contadorD);
    System.out.println("Velocidad antes "+ pong);
    anguloGrados = 180 - anguloGrados - (pr.contadorD*2);
    //cambiaAngulo();
    System.out.println("anguloGradosDespues="+anguloGrados);

    if(ping==3){
       pong++;
       ping=0;
    }else{
       pong=pong+(1*.5f);
```

```
        }
        //System.out.println("VelX="+velX);
        //System.out.println("VelX=" + (velX * (Math.PI/180)));
        //velX = -1*velX;
        System.out.println("Velocidad despues "+ pong);
        x = pr.x-radio; //Asignamos nuevo valor de x en el limite de la paleta
    }



    //golpe en la paleta izquierda
    if(x-radio<pl.x+pl.width && (y>=pl.y && y<=(pl.y+pl.heighl)) && x>pl.x+(pl.width/2)){
        System.out.println("Velocidad antes "+ pong);
        anguloGrados = 180 - anguloGrados - (pl.contadorl*2);
        //cambiaAngulo();
        System.out.println("anguloGradosDespues="+anguloGrados);

        if(ping==3){
           pong++;
           ping=0;
        }else{
            pong=pong+(1*.5f);
        }
        //velX = -1*velX;
        System.out.println("Velocidad despues "+ pong);
         x = pl.x+pl.width+radio;

    }


    cambiaAngulo();

    return pausa;

  }

}
```

/**********CLASE PALETA**********/

```
package com.itesm.pong;


import com.itesm.imagen.EvalArray;
import java.awt.event.KeyEvent;
import java.awt.geom.Rectangle2D;

/*
 * Clase Paleta
 */

/**
 *
 * @author otherc
 */
public class Paleta {


    public int x;
    public int y;
    public int width;
    public int height;
    private int screenw;
    private int screenh;
    public double dx; //desplazamiento en X
    public double dy; //desplazamiento en Y
    private int factor_paleta = 4;
    public double velocidadPaletaD=4;
    public double velocidadPaletaI=4;
    public double acelDUp=0;
    public double acelDDw=0;
    public double acelIUp=0;
    public double acelIDw=0;
    public double contadorI=0;
    public double contadorD=0;
    private String id;
    double movAzulDer; //Guarda el porcentaje de velocidad aplicado a la Derecha
    double movRojoDer; //Guarda el porcentaje de velocidad aplicado a la Derecha
    double movAzulIzq; //Guarda el porcentaje de velocidad aplicado a la Izquierda
    double movRojoIzq; //Guarda el porcentaje de velocidad aplicado a la Izquierda



    public Rectangle2D.Double r;

    /**
     *
```

```
 * @param x
 * @param y
 * @param width
 * @param height
 * @param screenw
 * @param screenh
 * @param id
 */
public Paleta(int x, int y, int width, int height, int screenw, int screenh, String id){

    this.x=x;
    this.y=y;
    this.width=width;
    this.height=height;
    this.id=id;
    this.screenw=screenw;
    this.screenh=screenh;

    rr = new Rectangle2D.Double(x,y,width, height);
    //aqui va el rectangulo
}


/**
 * Aplica condiciones para mover la paleta y establece los nuevos valores cuando es por detección de imagen
 */
public void move() {
    //if((y+dy)=)
    //if(y<0) dy=0;
    //Si tenemos detección de paletas entonces entra
    //if(movAzulDer!=movRojoDer || movAzulDer!=movRojoDer){
    if(EvalArray.azulder > 0 || EvalArray.azulizq > 0 || EvalArray.rojoder > 0 || EvalArray.rojoizq > 0) {
        System.out.println("Entrando a move="+dy);
        if (y+dy<0)
            {
                y=0;
                dy=0;
                rr.setRect(x, y, width, height);
            }

        if ((y+dy+height)>= screenh-28)
            {
                y=screenh-height-28;
                rr.setRect(x, y, width, height);
            }
        if((y+dy)>= 0 && (y+dy+height)< screenh-28){
                x += dx;
```

```
            y += dy;
            rr.setRect(x, y, width, height);
          }
    }
}

 /**
 * Aplica condiciones para mover la paleta y establece los nuevos valores cuando es por teclado
 */
public void moveKey() {

    //Si tenemos detección de paletas entonces entra
     System.out.println("Entrando a movekey="+dy);
      if (y+dy<0)
         {y=0; dy=0; rr.setRect(x, y, width, height);}
      if ((y+dy+heighl)>= screenh-28)
         {y=screenh-height-28;  rr.setRect(x, y, width, height); }
      if((y+dy)>= 0 && (y+dy+heighl)< screenh-28){
                  x += dx;
                  y += dy;
                  rr.setRect(x, y, width, height);
      }   }

/**
 * Metodos que se llaman al presionar las teclas arriba,abajo,a,z
 * @param e
 */
public void keyPressed(KeyEvent e) {

    int key = e.getKeyCode();

    //System.out.println("Char "+(char)key + "int Value"+ key );

    if(id.equals("rigIh")) {
    if (key == KeyEvent.VK_UP) {

       if(y+dy<0) {y=0; dy=0;}
       if(acelDDw>0 && acelDUp==0){
          contadorD=0;
          acelDDw=0;
       }

       if(y+(-1*(velocidadPaletaD+contadorD))>=0) {
       dy = -1*(velocidadPaletaD+(contadorD));

       //aceleramos y aumentamos la velocidad cada
       acelDUp++;
```

```
    if(contadorD<2){
       contadorD=contadorD+2;
    //mover();
       System.out.println("acelDUp "+ acelDUp+","+contadorD);
       }
    }else{
       dy=0;
       y=0;
    }
   }
 if (key == KeyEvent.VK_DOWN) {
 System.out.println("dd y="+ y+" dy="+dy+" heigth="+ height+ " Screens="+ screenh);
    if(y+dy<0)
       {y=0; dy=0;}
    if(acelDUp>0 && acelDDw==0){
       contadorD=0;
       acelDUp=0;
    }

    dy = 1*(velocidadPaletaD+contadorD);
    acelDDw++;
    if(contadorD<2){
       contadorD=contadorD+2;
    System.out.println("acelDDw "+acelDDw+","+contadorD);
    }
  }
 } else {
  if (key == 65 ) {
     System.out.println("iu y="+ y+" dy="+dy+" heigth="+ height+ " Screens="+ screenh);

    if(y+dy<0) {y=0; dy=0;}
    if(acelIDw>0 && acelIUp==0){
       contadorI=0;
       acelIDw=0;
    }

     if(y+(-1*(velocidadPaletaI+contadorI))>=0) {
    dy = -1*(velocidadPaletaI+(contadorI));
    //aceleramos y aumentamos la velocidad cada
    acelIUp++;

    if(contadorI<2){
       contadorI=contadorI+2;
       System.out.println("acelIUp "+ acelIUp+","+contadorI);
       }
    }else{
```

```
            dy=0;
            y=0;
          }


        }

      if (key == 90) {
         System.out.println("id y="+ y+" dy="+dy+" heigth="+ height+ " Screens="+ screenh);
       if((y+dy)>= 0 && (y+dy+height)< screenh-25){
          if(acellUp>0 && acellDw==0){
            contadorI=0;
            acellUp=0;
          }

          dy = 1*(velocidadPaletal+contadorI);

          acellDw++;

          if(contadorI<2){
            contadorI=contadorI+2;
            System.out.println("acellDw "+ acellDw+","+contadorI);
            }
          }
        }
      }
}


public void keyReleased(KeyEvent e) {

    int key = e.getKeyCode();

    if(id.equals("rigth")) {
    if (key == KeyEvent.VK_UP) {

          contadorD=0;
          acelDUp=0;

      }
    if (key == KeyEvent.VK_DOWN) {

          contadorD=0;
          acelDDw=0;

        }
    } else {
```

```
    if (key == 65 ) {

        contadorI=0;
        acelIUp=0;

    }

    if (key == 90) {

        contadorI=0;
        acelIDw=0;

    }
   }
 }

public void movAutom() {

  //Obtengo los procentajes de movimiento de acuerdo a las paletas y colores detectados
  //Lado derecho
  movAzulDer=Math.abs((double)EvalArray.azulder/Pong.jDerecha);
  movRojoDer=Math.abs((double)EvalArray.rojoder/Pong.jDerecha);
  //Lado izquierdo
  movAzulIzq=Math.abs((double)EvalArray.azulizq/Pong.jIzquierda);
  movRojoIzq=Math.abs((double)EvalArray.rojoizq/Pong.jIzquierda);

  if(id.equals("rigth")) {

    //Paleta estatica

    if(movAzulDer>movRojoDer){  //Si hay más porcentaje azul

      if (y + dy < 0) {
        y = 0;
        dy = 0;
      }
      if (acelDDw > 0 && acelDUp == 0) {
        contadorD = 0;
        acelDDw = 0;
      }

      if (y + (-1 * (velocidadPaletaD + contadorD)) >= 0) {
        dy = -1 * ((velocidadPaletaD*movAzulDer) + contadorD);
        //aceleramos y aumentamos la velocidad cada
        acelDUp++;

        if(contadorD<2){
```

```
            contadorD = contadorD + 2;
            //mover();
            System.out.println("acelDUp " + acelDUp + "," + contadorD);
        }
    } else {
        dy = 0;
        y = 0;
    }
} else if (movAzulDer==movRojoDer){  //si no se detecta
    dy = 0;  //Observar CVR
    contadorD = 0;
    acelDUp = 0;
}


if(movRojoDer>movAzulDer){ //si hay mas porcentaje rojo
    if (y + dy < 0) {
        y = 0;
        dy = 0;
    }
    if (acelDUp > 0 && acelDDw == 0) {
        contadorD = 0;
        acelDUp = 0;
    }

    dy = 1 * ((velocidadPaletaD*movRojoDer) + contadorD);
    acelDDw++;

    if(contadorD<2){
        contadorD = contadorD + 2;
        System.out.println("acelDDw " + acelDDw + "," + contadorD);
    }
} else if(movRojoDer==movAzulDer){

    dy = 0;  //Observar CVR
    contadorD = 0;
    acelDDw = 0;

}
} else { //Procesamiento del lado izquierdo

  if(movAzulIzq>movRojoIzq){   //Si hay más porcentaje azul

    if (y + dy < 0) {
        y = 0;
        dy = 0;
    }
```

```
    if (acellDw > 0 && acellUp == 0) {
      contadorI = 0;
      acellDw = 0;
    }

    if (y + (-1 * (velocidadPaletaI + contadorI)) >= 0) {
      dy = -1 * ((velocidadPaletaI*movAzulIzq) + contadorI);
      //aceleramos y aumentamos la velocidad cada
      acellUp++;

      if(contadorI<2){
        contadorI = contadorI + 2;
        System.out.println("acellUp " + acellUp + "," + contadorI);
      }
    } else {
      dy = 0;
      y = 0;
    }

  } else if(movAzulIzq==movRojoIzq) {
    dy = 0;  //Observar CVR
    contadorI = 0;
    acellUp = 0;
  }

   if(movRojoIzq>movAzulIzq){   //Si hay más porcentaje azul
    if ((y + dy) >= 0 && (y + dy + height) < screenh - 25) {
      if (acellUp > 0 && acellDw == 0) {
        contadorI = 0;
        acellUp = 0;
      }

      dy = 1 * ((velocidadPaletaI*movRojoIzq) + contadorI);

      acellDw++;

      if(contadorI<2){
        contadorI = contadorI + 2;
        System.out.println("acellDw " + acellDw + "," + contadorI);
      }
    }
  } else if (movRojoIzq==movAzulIzq) {
    dy = 0;  //Observar CVR
    contadorI = 0;
    acellDw = 0;
  }
```

```
  }
 }

}
```

```
/**********CLASE INICIADORA DE SERVICIOS DE IMAGEN**********/

package com.itesm.imagen;

import Paddles.Paddles;

/**
 * Esta clase inicia el servicio de adquisición y procesamiento de imagenes de MATLAB
 * Llama a la clase principal del código generado por el JABuilder de Matlab
 * Lo inicia como un thread adicional de la aplicación.
 * @author otherc
 */
public class ImageEval implements Runnable
{
   private boolean threadStop = false;
   public static int tipoLuz;
   public static int distancia;


public void stop() {
      threadStop = true;
   }

   public void getImmage () {

      Paddles paddles = null;

               try
               {

                 /*Creamos el objeto matlab */
                 paddles = new Paddles();
                 //Llamamos a la función de Matlab y le enviamos los parámetros de Luz y distancia
                paddles.getPads((new Double(tipoLuz)).doubleValue(),(new Double(distancia)).doubleValue());

               }
               catch (Exception e)
               {
                 System.out.println("Exception: " + e.toString());
```

```
            }

            finally
            {
              /* Liberamos recursos */
              if (paddles != null) {
                paddles.dispose();
          }
            }

  }


//Se inicia el Thread para mantener este proceso corriendo paralelamente con el del juego
  public void run() {

    while(!threadStop){

      getImmage();

    }

  }

}


/**************CLASE SERVIDOR QUE RECIBE LAS PETICIONES************/

package com.itesm.pong;

import com.itesm.imagen.EvalArray;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;

/**
* Clase para ejecutar un servidor con hilo independiente
* para recibir los resultados del analisis de imagen de MATLAB
 */
```

```
public class SrvJavMat implements Runnable{

        private boolean threadStop = false;
        private ServerSocket serverSocket;
        public String str;
        public int port = 8765;

  public void stop() {
    threadStop = true;
  }

  public void run()
  {

   try {
        serverSocket = new ServerSocket(port);
        } catch (IOException e1) {

                e1.printStackTrace();
        }
        while(!threadStop)
   {
     try
     {
       System.out.println("Esperando peticiones en el puerto " +
       serverSocket.getLocalPort() + "...");
       Socket server = serverSocket.accept();
       System.out.println("Conectando con el cliente " + server.getRemoteSocketAddress());


       //Leer los datos del stream de entrada
       BufferedReader br = new BufferedReader(new InputStreamReader(server.getInputStream()));
       while ((str = br.readLine()) != null) {
          EvalArray.setPaletas(str);
       }


      DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
      //Tomamos la hora y fecha actuales
           Date date = new Date();
      System.out.println(dateFormat.format(date));

       server.close();
     }catch(SocketTimeoutException s)
     {
       System.out.println("Socket timed out");
       break;
```

```
        }catch(IOException e)
        {
          e.printStackTrace();
          break;
        }
      }
    }
}

/**********CLASE QUE MANEJA LAS VARIABLES DEL RESULTADO RECIBIDO**********/

package com.itesm.imagen;

/**
 *
 * @author otherc
 */
public class EvalArray {

    public static int azulizq=0; //Valor azul detectado en el lado izquierdo de la imagen. Hacia arriba
    public static int azulder=0; //Valor azul detectado en el lado derecho de la imagen. Hacia arriba
    public static int rojoizq=0; //Valor rojo detectado en el lado izquierdo de la imagen. Hacia abajo
    public static int rojoder=0; //Valor rojo detectado en el lado derecho de la imagen. Hacia abajo


    public static void setPaletas (String array){

        String [] temp;

        System.out.println(array);
        array=array.replace("[","").replace("]","");

        temp = array.split(" ");

        azulizq=Integer.parseInt(temp[0]);
        rojoizq=Integer.parseInt(temp[1]);
        azulder=Integer.parseInt(temp[2]);
        rojoder=Integer.parseInt(temp[3]);

    }

}
```

ANEXO 3

Sockets en Java

Una gran parte de lo que hacen los programas de red es simplemente entrada y salida, moviendo bytes de un sistema a otro. La E/S en Java está basada en *streams* (flujos). Los *streams* de entrada leen datos, los *streams* de salida escriben datos[42].

Los datos son transmitidos a través de la red en paquetes de tamaño finito llamados datagramas. Cada datagrama contiene un *header* y un *payload*. El *header* contiene la dirección y el puerto a dónde va el paquete, la dirección y el puerto de donde viene el paquete y otros datos para asegurar una transmisión confiable. El *payload* contiene los datos. Para la transmisión y el control de estos paquetes existen los *Sockets*. Los *Sockets* son una innovación de Berkeley Unix que permiten al programador tratar una conexión de red como cualquier *stream* en el cual los bytes pueden ser escritos o leídos [42].

Un *Socket* es una conexión entre dos servidores. Pueden realizar siete operaciones básicas:

- Conectarse a una maquina remota.
- Enviar datos.
- Recibir datos.
- Cerrar una conexión.
- Enlace a un puerto.
- Esperar datos de entrada.
- Aceptar conexiones de equipos remotos en el puerto enlazado.

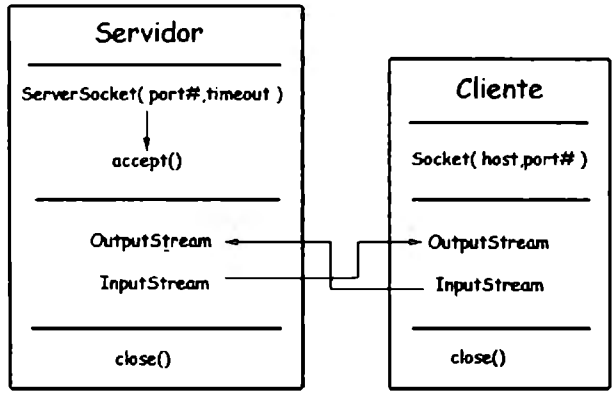


Figura A3. Ejemplo del intercambio de datos de Sockets en Java