Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto Tecnológico

Detección y clasificación de choques entre dos automóviles a través de video

Reporte Final

Presenta:
Jorge Molina Loera
209300611

Trimestre 2014 Primavera

Asesora: Silvia Beatriz González Brambila

25 de agosto de 2014

Declaratoria

Yo, Silvia Beatriz González Brambila, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma

Yo, Jorge Molina Loera, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma

Agradecimientos

A mi asesora Silvia González Brambila por darme la oportunidad de trabajar con ella, así también por el apoyo, confianza y amistad brindados a lo largo del desarrollo del proyecto.

A mis padres que a lo largo de mi trayectoria académica me han brindado su confianza y apoyo en el ámbito moral y económico.

A mis amigos que siempre han creído en mí y en lo que soy capaz de lograr.

Resumen

El presente proyecto tiene como objetivo detectar y clasificar choques producidos por dos automóviles que se desplazan de manera aleatoria, y está integrado por dos secciones:

Prototipos de automóvil: Primeramente se construyeron y programaron dos robots LEGO formando y simulando automóviles. Se realizaron pruebas de recorridos aleatorios y choques, debido a las colisiones del prototipo con objetos del entorno o bien con el otro prototipo varias veces fue reubicado el sensor de distancia y los ladrillos que forman la carcasa del automóvil. Para programarlos se utilizó el software LEGO MINDSTORMS NXT 2.0.

Aplicación de detección y clasificación de choques: Utilizando el lenguaje de programación java y las bibliotecas JMF y OpenCV se crearon:

- *Módulo de grabación de videos*: Encargado de realizar la grabación del comportamiento de los dos prototipos en un archivo AVI.
  Utilizando la biblioteca JMF se programó el módulo de grabación de videos, se ajustó la cámara web y se realizaron pruebas.
  El principal reto enfrentado fue que dos procesos no se pueden ejecutar al mismo tiempo, esto era muy indispensable para la grabación de videos, la solución fue utilizar hilos y clonar variables dentro del programa.
- *Módulo de detección y clasificación de choques:* Encargado de leer el archivo AVI y detectar y clasificar los choques entre los dos prototipos automóvil.
  Utilizando la biblioteca OpenCV se programó la lectura del archivo AVI almacenado por el módulo de grabación y se procedió a implementar los algoritmos necesarios para realizar la detección y clasificación de choques.
  Dentro de los retos enfrentados estuvieron que la efectividad de la detección y clasificación de choques variaba dependiendo el tipo de luz bajo el cual fue grabado el video, sombras de objetos en el video, color de piso, entre otras, para solucionarlo, se probaron varios ambientes escogiendo el que mostrara mayor efectividad.

El presente trabajo se elaboró pensando en simular el recorrido de automóviles en calles y avenidas de la Ciudad de México, aprovechando las cámaras de vigilancia instaladas recientemente. Cuando en una cámara se detecte que ha ocurrido un choque, se pueden desplegar servicios como ambulancias, patrullas, compañías de seguros etc. Por razones de complejidad en ese ámbito, para la simulación sólo se utilizaron los dos prototipos automóvil y una cámara web de una computadora portátil.
Trabajar con este tema de visión por computadora mostró que aún es un campo en desarrollo, ya que una efectividad de cien por ciento, no es lograda de manera sencilla.

Contenido

Índice de Figuras

Introducción

En ésta sección se presenta una breve descripción del proyecto desarrollado, incluyendo objetivos, justificación y antecedentes.

1 Introducción

La detección de choques por computadora es un tema de actualidad, no sólo es aplicable en videos grabados utilizados para realizar procedimientos de inteligencia artificial, también es útil en la implementación de juegos de video.

A pesar de que existen diversos algoritmos que pueden detectar colisiones entre dos o más objetos, es difícil seleccionar un método óptimo, ya que en ocasiones se tiene que sacrificar efectividad por tiempo de respuesta.

Aunque pareciera que trabajar colisiones en video grabado o en programación de video juegos tiene la misma dificultad, la experiencia del presente proyecto desmiente ese argumento, ya que al trabajar con cámaras de video es importante tener en cuenta las siguientes consideraciones:

- De Cámara: La resolución y calidad de la imagen y el pixelaje de la cámara.
- De ambiente: Iluminación del entorno, sombra de objetos y color de fondo y alrededores.
- De objetos: Color de los objetos que colisionarán, movimiento y velocidad de objetos en video (éstos factores pueden influir generando una imagen difusa).

Tomando en consideración todo lo que se ha redactado en ésta página, el proyecto de detección y clasificación de choques que se presenta, se construyó de la manera más adecuada posible para mostrar un tiempo de respuesta eficaz.

El funcionamiento de éste proyecto se puede visualizar en el video incluido en el CD o bien en el enlace http://youtu.be/4ynkCMoBfhw.

Especificación técnica

Para la realización y funcionamiento de este proyecto de integración se utilizó:

- Dos robots de LEGO Mindstorms. Los cuales sirvieron para la construcción y programación de dos prototipos de automóvil que realizan recorridos aleatorios.
- NetBeans 7.0 con paquete de desarrollo Java Platform (JDK) 7u21: Utilizado para realizar la programación de los módulos de grabación de videos y de detección y clasificación de choques en lenguaje Java.
- Biblioteca JMF (Java Media Framework) 2.1.1. Facilitó la captura y almacenamiento de video en formato AVI 640*480 pixeles.

Se decidió utilizar para evitar los problemas de codecs de video que suceden con otras bibliotecas como OpenCV.

- Biblioteca de OpenCV 2.4.8. Se empleó para realizar la lectura de videos almacenados, así como del análisis de colores de los prototipos automóvil y entorno. Se decidió usar OpenCV por la facilidad que tiene en el manejo de funciones, así mismo, por la gran capacidad que brinda en el manejo de imágenes. La versión 2.4.8 de manera personal sirvió muy bien en combinación con lenguaje Java, ya que las versiones 2.4.6 y 2.4.4 generaban errores.

Recursos utilizados

- Computadora portátil con cámara web (proporcionado por el alumno).
- Dos kits de robótica LEGO Mindstorms NXT (proporcionados por la asesora).

Entregables

Los productos que acompañan al reporte final en el CD son:

- Código fuente: 13 archivos de extensión java dentro de la carpeta detección_clasificacion o bien se puede utilizar un proyecto listo para abrir de NetBeans. Nota: Es necesario el uso de las librerías JMF y OpenCV.
- Video en formato AVI de desplazamiento de prototipos automóvil (necesario para realizar un análisis). El video fue grabado con el Módulo de grabación de videos del proyecto.
- Video que muestra el funcionamiento y manejo del software de detección y clasificación de choques.

A continuación se desarrolla el documento del proyecto donde primeramente se muestran los objetivos y justificación del trabajo, luego un marco teórico de conceptos necesarios, posteriormente la construcción del proyecto y finalmente los resultados y conclusiones obtenidos del proyecto.

Objetivos

Objetivo general

Diseñar e implementar una aplicación capaz de detectar y clasificar choques entre dos prototipos de automóvil a partir de un video.

Objetivos específicos

- Diseñar y programar a través de robots LEGO trayectorias aleatorias de dos automóviles.
  ¿Cómo se cumplió? Con ayuda de los bloques de plástico incluidos en el kit se dio forma de automóvil al robot, y a través del software LEGO MINDSTORMS NXT 2.0, se programaron los recorridos aleatorios.
- Construir un ambiente para el desplazamiento de los automóviles.
  ¿Cómo se cumplió? Se utilizó una lona blanca que sirvió de piso y se construyeron unas barreras utilizando tablas para evitar que los automóviles sobrepasaran un área permitida. El área del ambiente era de 1.10 m * 1.20 m.
- Capturar en video el desplazamiento de los automóviles.
  ¿Cómo se cumplió? A través de la biblioteca JMF y el lenguaje de programación java se programó el módulo de grabación de video que realiza esa función.
- Detectar si existe choque entre los dos automóviles involucrados en el video, en caso afirmativo, clasificarlo según la postura de los vehículos al momento del impacto.
  ¿Cómo se cumplió? Utilizando una función de detección de movimientos en una imagen, se programaron algunas condiciones que fueron útiles en la detección de choques. Para clasificar los choques se utilizaron los colores blanco y negro generados por la imagen del choque (negro para los vehículos y blanco para los objetos del resto), con esto y conociendo las medidas de los automóviles se programaron las condiciones más acercadas para realizar la clasificación.
- Diseñar una interfaz gráfica intuitiva para el usuario que dirija el análisis de detección de choques, gestione los videos y muestre los resultados obtenidos.
  ¿Cómo se cumplió? Utilizando el lenguaje de programación java y el IDE NetBeans 7.0 se desarrolló de manera sencilla la interfaz del sistema.

Justificación

La detección de choques de prototipos de automóvil en video es un proyecto de relevancia, ya que a futuro puede ser de utilidad para la realización de pruebas de calidad en automóviles de juguete automatizados utilizando video. Si dos automóviles de juguete chocan durante las pruebas, se devolverán para realizar las reparaciones correspondientes.
Otro ejemplo de implementación a futuro es en los videos de las cámaras de seguridad de la Ciudad de México [1], si en algún video se detecta el choque entre dos automóviles, se obtiene la ventaja de realizar un despliegue de servicios como policía, bomberos, ambulancias y compañías de seguros a la zona afectada, así como transmitir avisos en las estaciones de radio, para reducir la carga vehicular, entre otros.
Durante los años 2011 y 2012 la Ciudad de México ocupó el sexto lugar en accidentes automovilísticos ocurriendo16, 466 y 17,120 correspondientemente, siempre por debajo de Estados como Nuevo León, Jalisco, Chihuahua y Guanajuato [2]. Ver Figuras 1 y 2.
Detectar los choques automovilísticos a tiempo no disminuirá la cantidad de accidentes anuales, sin embargo, evitará que otros automovilistas circulen en los alrededores evitando estrés, incluso otros choques al momento de presenciar el accidente.

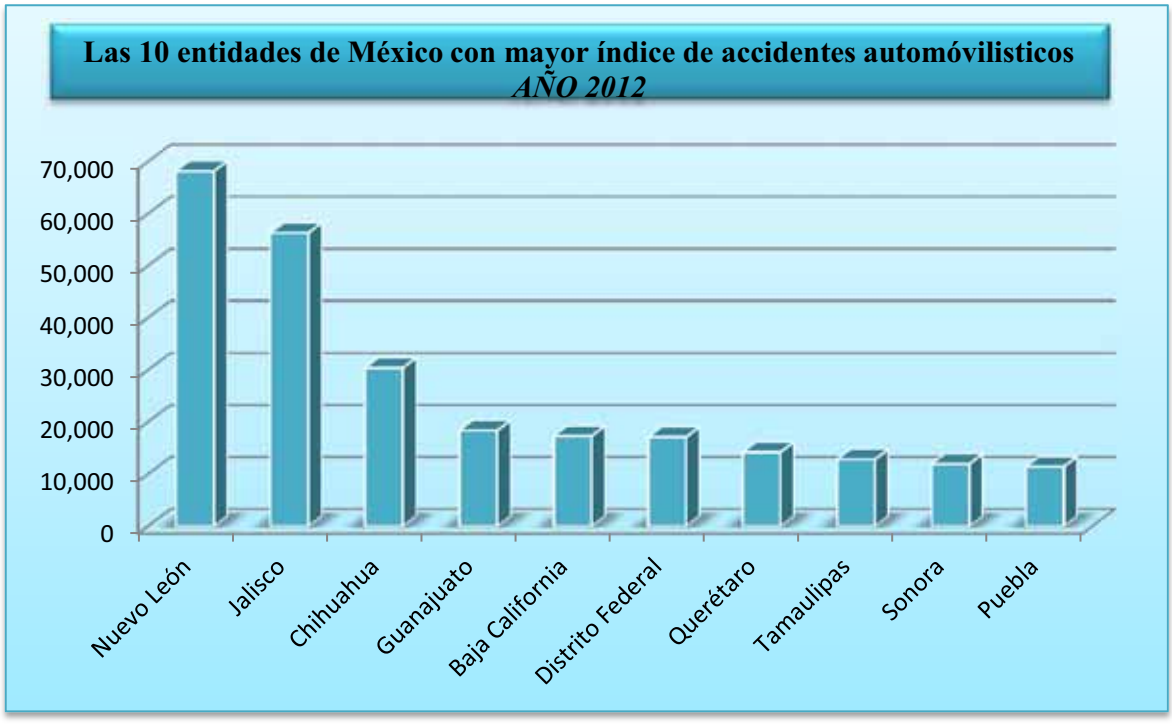

Figura 1. Distrito Federal sexto lugar en accidentes, año 2012 [2]

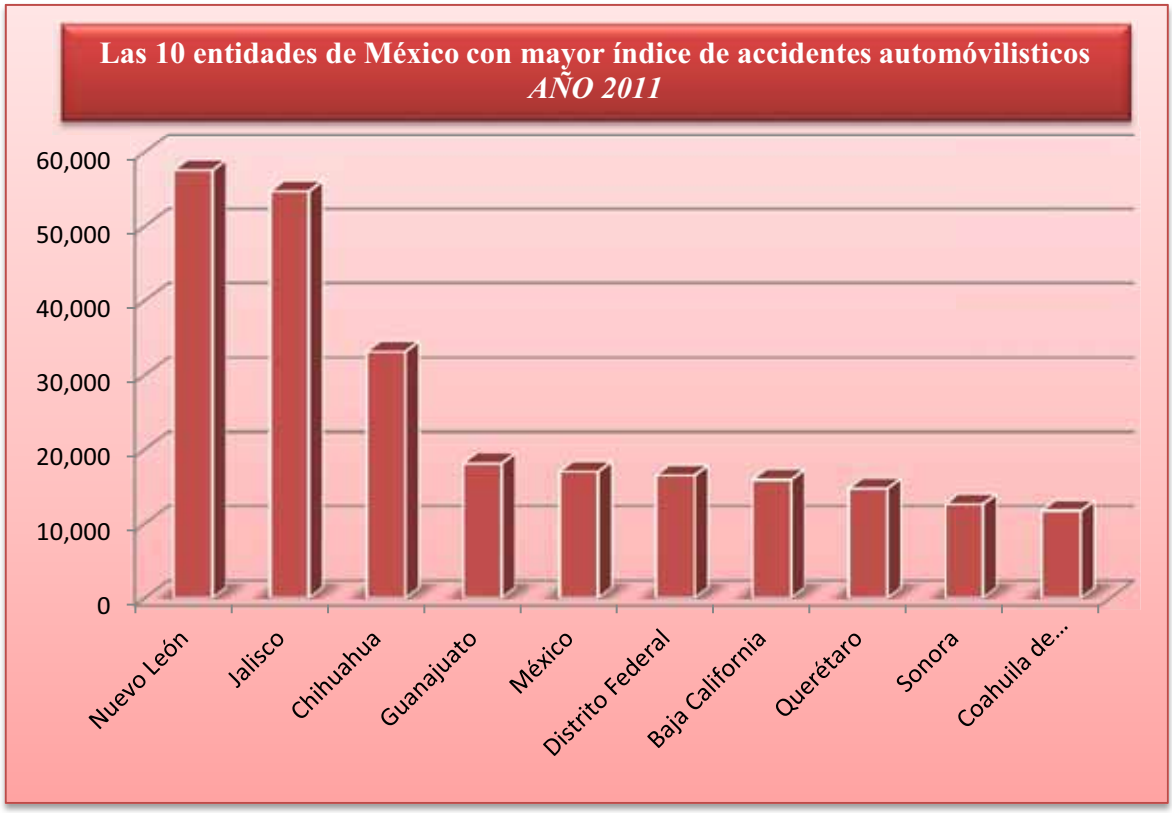

Figura 2. Distrito Federal sexto lugar en accidentes, año 2011 [2]

Analizando las gráficas de la Figura 1 y 2 Nuevo León es la entidad con más choques automovilísticos en ambos años, si decidiéramos implementar el programa en toda la República Mexicana sería la entidad con más prioridad. Mientras tanto, las entidades como México y Coahuila únicamente aparecen en el año 2011 y Puebla y Tamaulipas únicamente aparecen en el año 2012 por lo tanto podrían ser las entidades con menor prioridad.

Antecedentes

Trabajos Externos

Tesis de maestría

1. *Detección de movimiento a través de secuencias de video* [3]. En esta tesis de maestría se realiza la detección en video de objetos y personas en movimiento usando vectores y dos algoritmos propuestos, posteriormente se analizan sus resultados y condiciones de éxito. A pesar de las características anteriores, en ninguna sección trata acerca de choques entre objetos como el proyecto construido.

Artículos de investigación

2. *Shape-based Recognition and Classification for Common Objects - An Application in Video Scene Analysis* [4]. En este artículo se realiza el reconocimiento y la clasificación de cuatro clases de objetos en escenas de video. Para la clasificación, se extraen de la entrada imágenes de objetos en forma de silueta binaria. En el proyecto terminado sólo se necesitó la silueta para conocer las medidas del prototipo e identificar el tipo de choque en una región de interés.

3. *Fast Multiple Object Tracking via a Hierarchical Particle Filter* [5]. En este artículo se expone una manera eficiente de realizar un seguimiento de objetos a través de partículas de colores, analizando las diferencias de la imagen actual con respecto a la imagen anterior, detectando los colores pertenecientes a objetos que no estaban en la imagen anterior bajo una estricta iluminación. El proyecto concluido utiliza un algoritmo de eliminación de fondos estáticos para detectar el movimiento de los prototipos, a comparación del artículo que utiliza el movimiento de colores, además, en la investigación se empleaba una estricta iluminación y en el caso del proyecto la luz empleada variaba en algunas regiones del ambiente.

4. *Real-Time Object Tracking for Soccer-Robots without Color Information* [6]. En este trabajo se realiza el seguimiento de un balón de soccer utilizando los valores RGB de los colores pertenecientes a éste, además se utilizan varias condiciones de iluminación, se realiza un entrenamiento previo para disminuir la cantidad de falsos positivos y negativos. El proyecto concluido tiene relación con el artículo, ya que para desarrollar la detección y clasificación de choques también se combinaron los temas de detección de movimiento y valores RGB.

Software

5. *Acer Crystal Eye WebCam*. Es una aplicación para equipos de cómputo Acer, que tiene la capacidad de grabar video en formato AVI, muy similar al módulo de grabación de videos de éste proyecto [7].

Marco teórico

En ésta sección se presentan los conceptos y conocimientos teóricos necesarios en la elaboración del proyecto.

2 Marco teórico

Este capítulo introduce brevemente sobre qué es LEGO Mindstorms y las partes que lo componen.

Posteriormente, se describirán las bibliotecas JMF (utilizada para captura almacenamiento de videos) y OpenCV (utilizada para realizar la detección y clasificación de choques).

2.1 Lego Mindstorms

Lego Mindstorms es un kit de robótica utilizado para construir un modelo de sistema integrado con partes electromecánicas controladas por computador. Prácticamente todo puede ser representado con las piezas tal como en la vida real, como un elevador o robots industriales.

2.1.1 Componentes del Lego NXT

A continuación se describe de manera breve la función de cada uno de los componentes básicos de un robot LEGO.

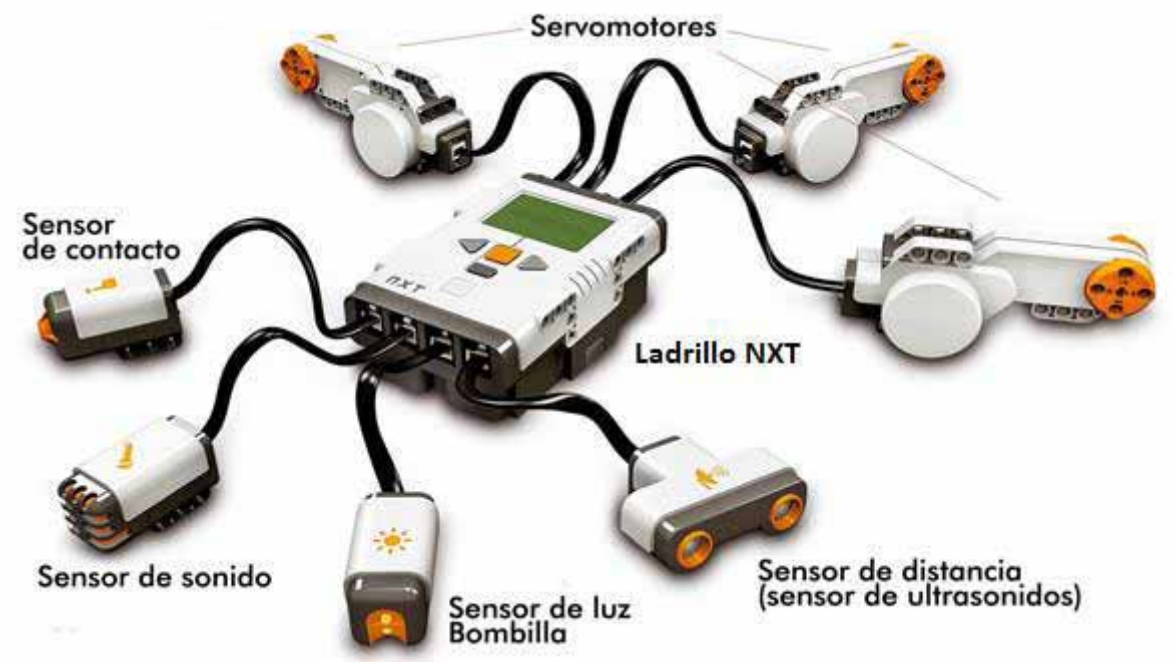

Figura 3. Componentes del robot LEGO conectados al NXT

- *Ladrillo NXT*

El ladrillo NXT es el cerebro de cualquiera de los proyectos que se pueden realizar con este kit. En su interior tiene un microcontrolador programable, dentro del cual se encuentra el programa necesario para que los motores se muevan cuando se tienen que mover, o cualquier otro fin que se nos ocurra (Ver Figura 3).

- *Servomotores NXT*

Los servomotores tienen una doble función. En primer lugar, podemos hacer que se muevan a una determinada velocidad o con determinados ángulos, pero además, también pueden actuar como sensores de rotación. Es decir, podríamos, por ejemplo, conectarlo a un volante, y además de poder girarlo con el motor, si alguien moviera el volante, también nos permitiría saber cuánto lo ha girado (Ver Figura 3).

- *Sensor de sonido NXT*

El sensor de sonido incorpora un pequeño micrófono, permitiendo así que se pueda programar un robot que al escuchar una palmada se quede quieto, o que al escuchar dos palmadas apague la luz (Ver Figura 3).

- *Sensor de contacto*

El sensor de contacto incorpora un pequeño pulsador, con lo que, por ejemplo, se podría crear un pequeño parachoques que detectara si el robot ha impactado con algo que tiene delante (Ver Figura 3).

- *Sensor de distancia*

El sensor de distancia es capaz de medir distancias de 10 centímetros a 1.80 metros, aportando a nuestro robot la capacidad de visión. Funciona al igual que la visión de los murciélagos, a través de ultrasonidos (Ver Figura 3).

- *Sensor de color*

El sensor de color es capaz de distinguir tres niveles de colores (RGB), es decir, rojo, verde, y azul. Está pensado para distinguir el color de las pelotitas de colores que incluye Lego Mindstorms NXT [8] (Ver Figura 3).

2.2 JMF

Java Media Framework (JMF) es un API (*Application Programming Interface*), para la incorporación de medios basados en el tiempo (*time-based medias*) en aplicaciones Java y Applets. Los medios basados en el tiempo son medios tales como el audio, video, MIDI y animaciones que cambian con respecto al tiempo.

Inicialmente, el JMF 1.0 API, habilitaba a los programadores para desarrollar software de tipo Java que presentaban estos tipos de medios. Actualmente, el JMF 2.0 API, extiende el área de trabajo, para proveer soporte para captura y almacenaje de medios (basados en el tiempo), controlando el tipo de procesamiento que es efectuado durante la reproducción y para la personalización del procesamiento sobre flujos de medios.

Los objetivos principales del diseño de JMF 2.0 API son:

- Ser más fácil para programar.
- Tener soporte para la captura de medios.
- Habilitar el desarrollo de aplicaciones para flujos de medios (*media streaming*) y conferencias en Java.
- Habilitar tecnología y desarrollo avanzado que permita implementar soluciones personalizadas basadas en APIs ya existentes y nuevas características fácilmente integrables.
- Proveer el acceso a datos *Raw Media*.
- Habilitar el desarrollo de demultiplexores, codecs, procesadores de efectos, multiplexores, y *renderers* personalizables y descargables (JMF *plug-ins*).
- Mantener la compatibilidad con JMF 1.0 API [9].

2.2.1 Conceptos para facilitar el uso de la webcam con JMF

Clases
A continuación se presentan las clases principales en el manejo de JMF.

DataSource (Fuente de Datos): Encargada de capturar todo lo que entre por la cámara web. El *DataSource* también puede ser la entrada de un procesador (*Processor*) o bien de un reproductor (P*layer*).
Player (Reproductor): Es la encargada de reproducir los medios de audio y video. Para nuestro caso, será la clase encargada de reproducir todo lo que llegue del interlocutor.
Processor (Procesador). Esta clase es un procesador que tendrá como misión tratar el video, convirtiéndolo en algún formato.
DataSink: Sirve para dirigir la salida de un *Processor* o un *Player* hacia la red o al disco duro, o bien a otro *Processor* u otro *Player* [10].

Managers.
El JMF API consiste principalmente en interfaces que definen el comportamiento e interacción de los objetos usados para capturar, procesar y presentar los medios basados en el tiempo. Las implementaciones de estas interfaces funcionan dentro de la estructura del JMF. Usando los objetos intermedios llamados *managers*, JMF hace fácil integrar nuevas implementaciones de interfaces claves que se pueden utilizar junto con las clases existentes.

JMF utiliza cuatro managers:

i. **Manager:** maneja la construcción de *Players, Processors, DataSources y DataSinks*. Este nivel de indirección permite que las nuevas implementaciones sean perfectamente integradas con JMF.
ii. **PackageManager:** mantiene un registro de paquetes que contiene clases de JMF, tales como *Players, Processors, DataSources y DataSinks*.
iii. **CaptureDeviceManager:** mantiene un registro de dispositivos de captura disponibles.
iv. **PlugInManager:** mantiene un registro de componentes de procesamiento de JMF plug-in disponibles, tales como *Multiplexers, Demultiplexers, Codecs, Effects y Renderers*.

Data Model
Los reproductores de medios de JMF utilizan generalmente *DataSources* para manejar la transferencia del contenido de medios. Un *DataSource* encapsula tanto la localización de los medios y el protocolo como el software usado para entregarlo. Una vez obtenido, el *DataSource* no se puede reutilizar para entregar otros medios. Un *DataSource* es identificado por un JMF *MediaLocator* o un URL.

Un *DataSource* maneja un conjunto de objetos de *SourceStream*. Una fuente de datos estándar utiliza un arreglo de bytes como unidad de la transferencia. Una fuente de datos de *buffer* utiliza un objeto de clase *Buffer* como su unidad de la transferencia.
El formato exacto de un objeto de media es representado por un objeto de la clase *Format*. El formato en sí mismo no lleva ningún parámetro específico de codificación o información de sincronización, describe el nombre de la codificación del formato y el tipo de datos que el formato requiere.
JMF extiende *Format* para definir formatos de audio y video específicos [9].

Analogía

Imaginando, una videocámara puede actuar como un *CaptureDevice*, es decir como una fuente de entrega de medios, así mismo una cinta de video puede actuar como un *DataSource* al llevar la información proveniente de la videocámara, luego entonces un video-reproductor puede simular ser un *Player* al reproducir el contenido de un *DataSource* (Ver Figura 4) [10].

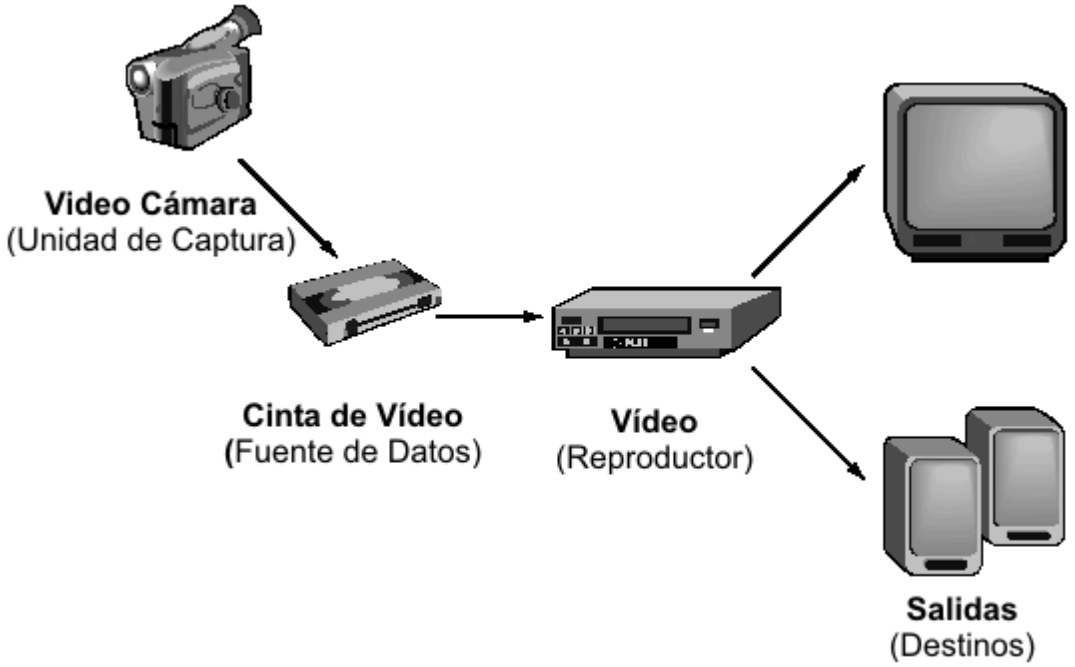

Figura 4. Analogía de un sistema doméstico para entender el manejo de la webcam con JMF.

2.3 Hilos (Threads)

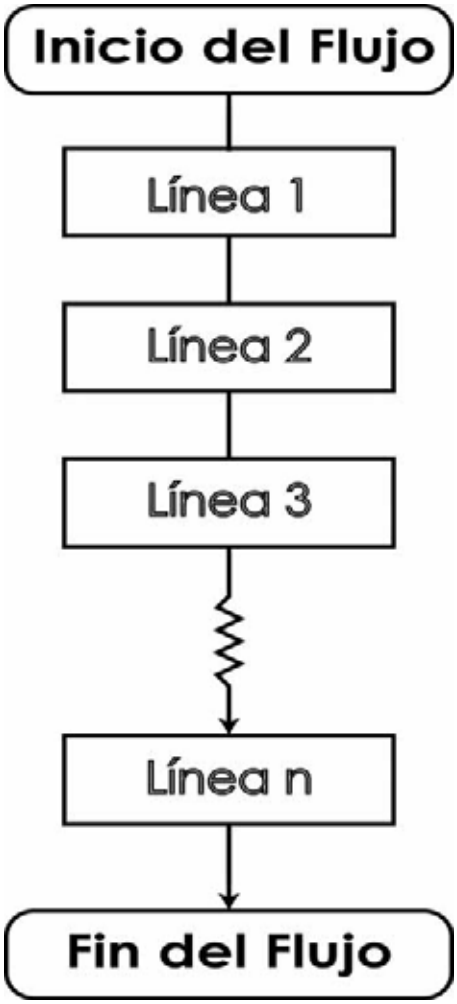

Figura 5. Diagrama secuencial, la principal estructura de ejecución.

En Java, así como en cualquier lenguaje de programación, la principal estructura de ejecución de instrucciones es la estructura secuencial, en la que cada comando, cada línea, cada instrucción se ejecuta una después de otra. El código se ejecuta de arriba hacia abajo y cada línea es ejecutada según el orden en que haya sido escrita por el programador (Ver Figura 5).

En ocasiones puede que el usuario necesite llevar a cabo dos o más procesos a la vez. Supongamos que queremos realizar tres procesos al mismo tiempo (Ver Figura 6).

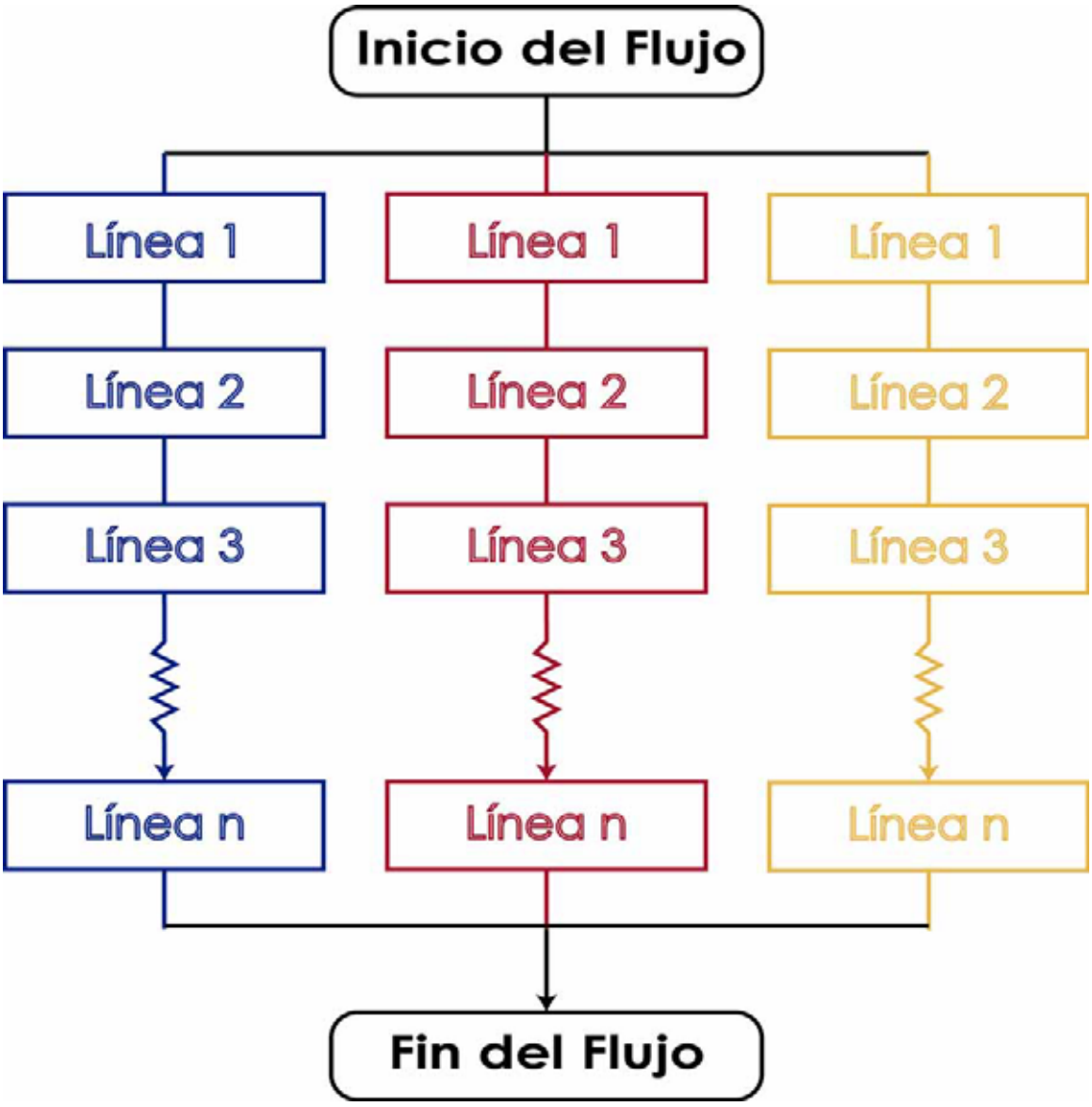

Figura 6. Diagrama Secuencial de tres hilos

Esto es completamente imposible ya que el procesador de una computadora no es capaz de realizar estas funciones. Sin embargo, existen alternativas, como los Hilos (*Threads* en inglés).

Si vemos la Figura 6, podemos identificar tres hilos: Proceso 1, Proceso 2 y Proceso 3. Sin embargo, no es posible ejecutarlo de la forma como lo hemos presentado en el diagrama de flujo.

Si intentamos utilizar una estructura secuencial para ejecutar tres procesos, obtendremos el resultado de la Figura 7. Los inconvenientes de ejecutar los procesos de esa manera son: el *Proceso 1* sólo se ejecutará cuando haya terminado el *Proceso 2*. El *Proceso 3* se empezará a ejecutar cuando haya terminado el *Proceso 1 y 2*.

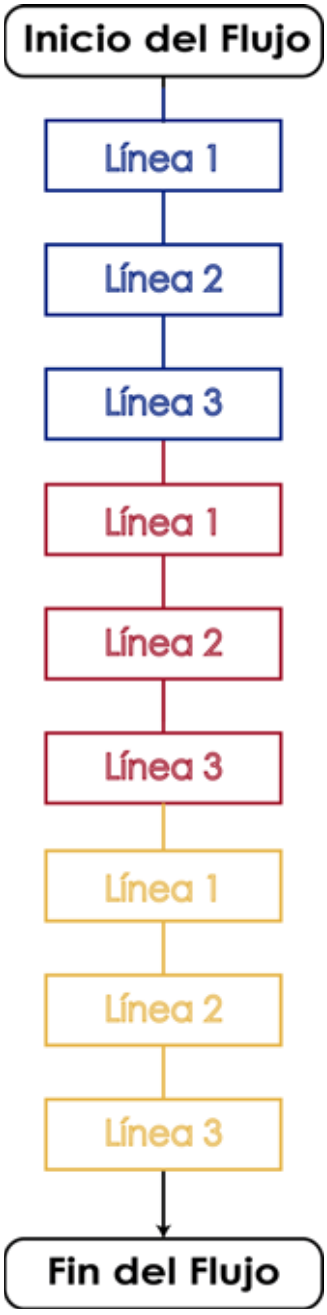

Figura 7. Diagrama secuencial tres procesos

Utilizando hilos o *threads*, podemos obtener una ejecución muy cercana al modelo de la Figura 6. Los *threads* son estructuras secuenciales que combinan las líneas de dos o más procesos en un solo proceso. Expresándolo con un diagrama de flujo se apreciaría como la Figura 8.

Las instrucciones en color azul pertenecen al *Proceso 1*, las instrucciones en rojo son del *Proceso 2* y las instrucciones en marrón del *Proceso 3*.

La forma como se ejecutan los procesos es completamente aleatoria y no siguen un orden específico, esto permite que los tres vayan ejecutándose al mismo tiempo [11].

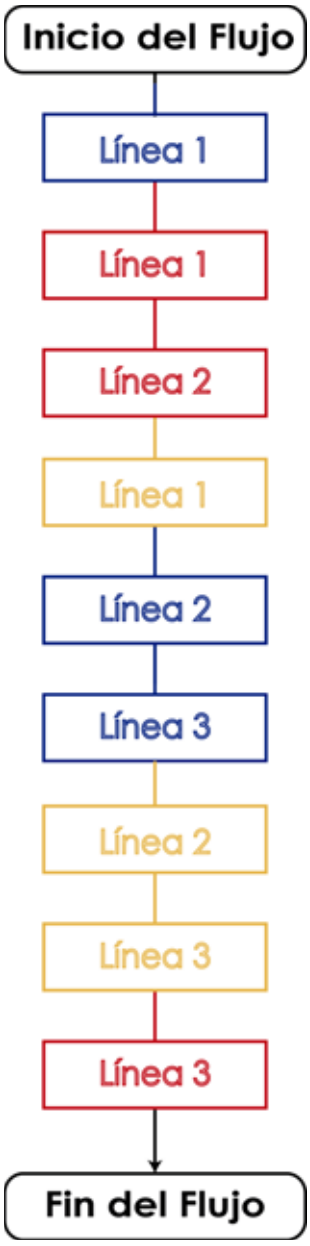

Figura 8. Diagrama secuencial ejecutando tres hilos

2.4 OpenCV

OpenCV (*Open source Computer Vision library*) es una librería abierta desarrollada por Intel. Esta librería proporciona un alto nivel funciones para el procesado de imágenes. Estas librerías permiten a los programadores crear aplicaciones poderosas en el dominio de la visión digital. OpenCV ofrece muchos tipos de datos de alto-nivel como juegos, árboles, gráficos, matrices, etc. OpenCV es *OpenSource*[1] para poder funcionar en muchas plataformas.

OpenCV permite:

- Operaciones básicas.
- Procesado de imágenes y análisis.
- Análisis estructural.
- Análisis de movimiento.
- Reconocimiento del modelo.
- Reconstrucción 3d y calibración de la cámara.
- Interfaz gráfica y adquisición.

Rasgos.

OpenCV implementa una gran variedad de herramientas para la interpretación de la imagen. Es compatible con *Intel Image Processing Library* (IPL) que implementa algunas operaciones en imágenes digitales.

OpenCV es principalmente una biblioteca que implementa algoritmos para las técnicas de la calibración (Calibración de la Cámara), detección de rasgos, para rastrear (Flujo Óptico), análisis de la forma (Geometría, Contorno que Procesa), análisis del movimiento (Plantillas del Movimiento, Estimadores), reconstrucción 3D (Transformación de vistas), segmentación de objetos y reconocimiento (Histograma, etc.).

El rasgo esencial de la librería junto con funcionalidad y la calidad es su desempeño. Los algoritmos están basados en estructuras de datos muy flexibles, acoplados con estructuras IPL; más de la mitad de las funciones h[1]an sido optimizadas aprovechándose de la Arquitectura de Intel.

OpenCV usa la estructura *Iplimage* para crear y manejar imágenes. Esta estructura tiene gran cantidad de campos, algunos de ellos son más importantes que otros. Por ejemplo el *width* es la anchura del *Iplimage, height* es la altura, *depth* es la profundidad en bits y

[1] *OpenSource:* Conocido como código abierto, es el software que se puede utilizar libremente, cambiado, y se puede compartir (en forma modificada o sin modificar) por cualquier persona [12].

nChannels el número de canales (uno por cada nivel de gris de las imágenes y tres para las imágenes coloridas).

OpenCV en cuanto a análisis de movimiento y seguimiento de objetos, ofrece una funcionalidad interesante. Incorpora funciones básicas para modelar el fondo para su posterior sustracción, generar imágenes de movimiento MHI (*Motion History Images*) para determinar dónde hubo movimiento y en qué dirección, algoritmos de flujo óptico, etc.

OpenCV viene con una interface gráfica llamada *highGUI*. Esta interfaz gráfica es muy importante porque se necesita bajo OpenCV para visualizar imágenes.

Algunas convenciones que se utilizan

El software de OpenCV tiene las siguientes convenciones:

- Los identificadores constantes se escriben en mayúsculas, por ejemplo: CV_SEQ_KIND_GRAPH.
- Todos los nombres de las funciones usadas para el procesado de imagen tienen el prefijo del cv, por ejemplo: "cvCreateImage", "cvSobel", "cvAdd".
- Todas las funciones externas de OpenCV empiezan con el prefijo cv, y todas las estructuras con prefijo Cv.
- Cada nueva parte de una función empieza con un carácter en mayúscula, por ejemplo: cvContourTree.

Inconvenientes de OpenCV

Dadas las grandes posibilidades que ofrece OpenCV para el tratamiento de imágenes, calibración de cámaras, y otras aplicaciones, es difícil encontrar inconvenientes muy grandes en la biblioteca. A continuación se presentan los pequeños detalles encontrados.

- Para el tema de seguimiento de objetos, OpenCV no ofrece un producto completo, ya que sólo algunas piezas sirven como base para crear un objeto final (Objeto a seguir).
- Otro inconveniente es tener la necesidad de utilizar la librería IPL para tener acceso a funciones de bajo nivel.

Comparando las enormes capacidades de las funciones de OpenCV con respecto a los inconvenientes mencionados, logran mostrar esas desventajas como insignificantes, ya que desde su lanzamiento en el año 2000, se han realizado más de 500 mil descargas de código, siendo utilizado desde aplicación en juguetes hasta la fabricación industrial, exponiendo la utilidad e interés brindados a las personas del mundo actual [13].

Diseño

En ésta sección se presentan los diagramas útiles en la implementación del proyecto.

3 Diseño

Funcionamiento de los prototipos automóvil

Las actividades realizadas para que los automóviles realicen recorridos aleatorios son sencillas y se describen en la Figura 9.

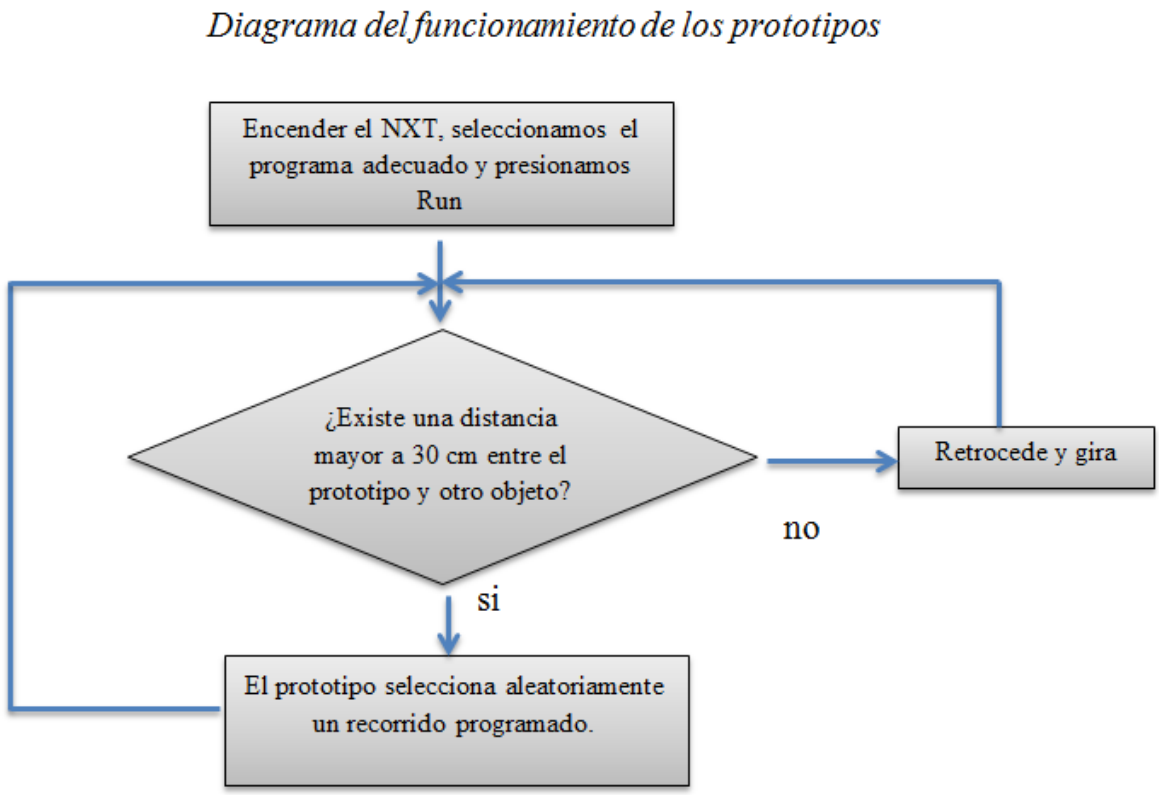

Figura 9. Funcionamiento de los prototipos automóvil.

El módulo de captura de video graba en un archivo AVI el comportamiento de los prototipos automóvil.
El módulo detección y clasificación de choques, lee el archivo AVI proporcionado por el módulo de captura y lo muestra en pantalla para que se proceda a analizar y mostrar los resultados correspondientes, al final se muestran estadísticas de los choques ocurridos.

Diagramas de la aplicación de detección y clasificación de choques

Menú principal

Al ejecutarse el programa se inicia un menú encargado de brindar al usuario ayuda, así como las opciones necesarias para ejecutar las funciones del sistema, ver Figura 10.

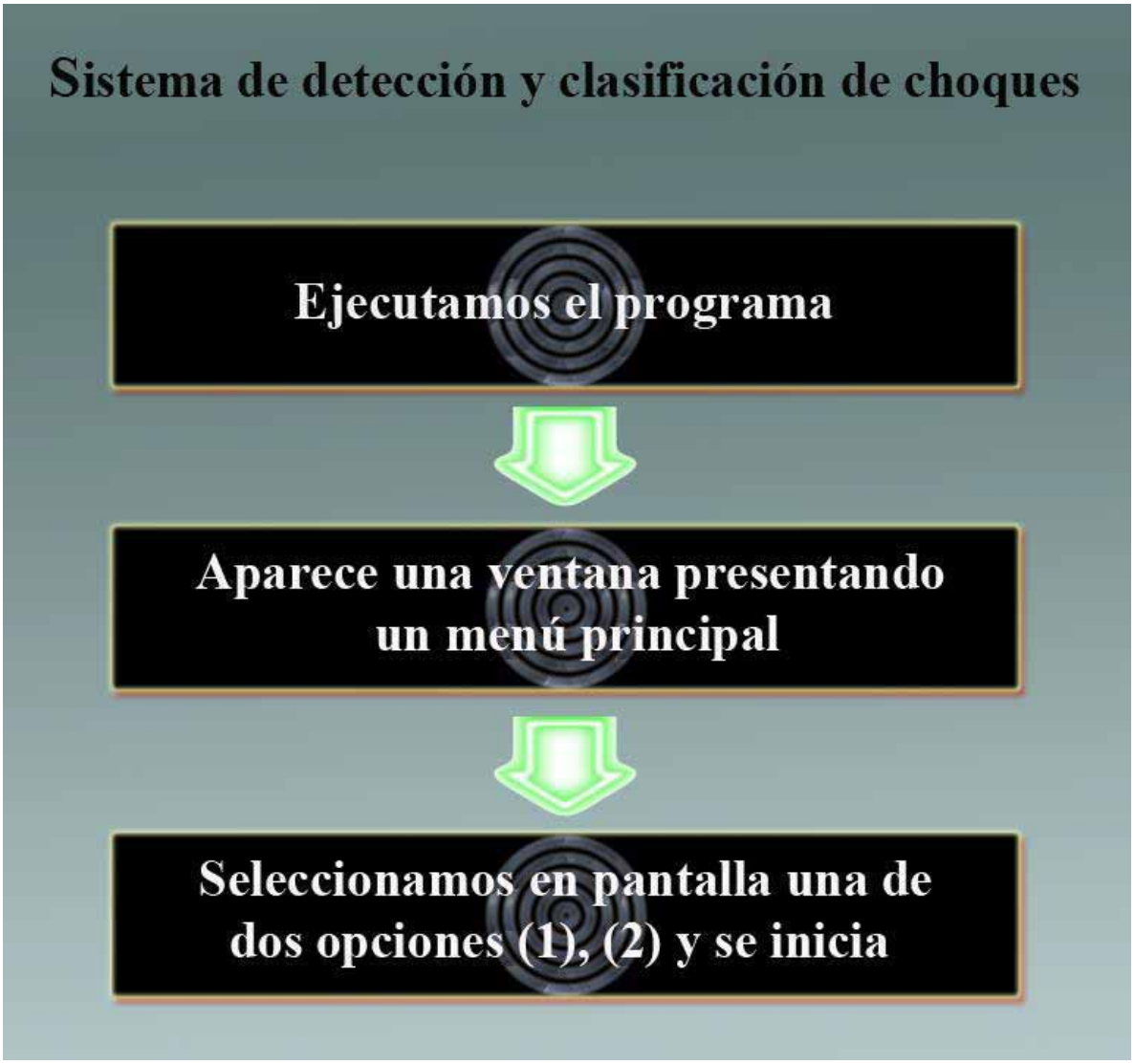

Figura 10. Diagrama de ejecución del menú de detección y clasificación de choques.

(1) Módulo grabación de video

Encargado de guardar en un archivo AVI el comportamiento de los prototipos automóvil, el proceso de ejecución es mostrado en la Figura 11.

Figura 11. Diagrama de ejecución del módulo de grabación de video.

(2) Módulo de detección y clasificación de choques

Encargado abrir el archivo AVI producido por el módulo de grabación de video y en seguida analizar, detectar y clasificar un choque entre los dos automóviles, por último en una ventana nueva se despliegan los resultados de los choques ocurridos en el análisis del video. Ver Figura 12.

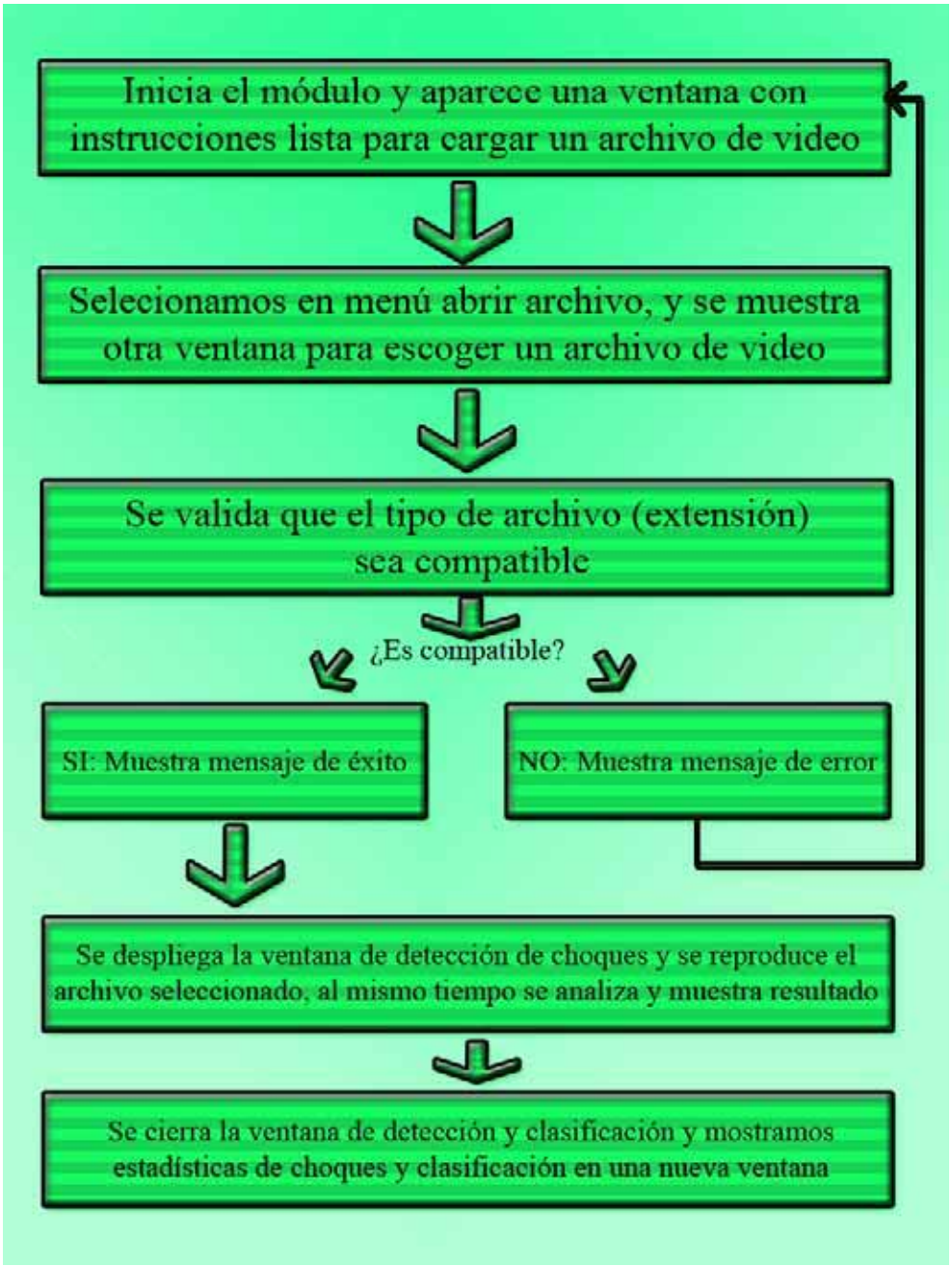

Figura 12. Diagrama de ejecución del módulo de detección y clasificación de choques.

Desarrollo del proyecto

En ésta sección se presenta la implementación del proyecto, no incluye código fuente.

NOTA: El código fuente se encuentra anexo en el directorio correspondiente en el CD de este reporte.

4 Descripción técnica

En esta sección se mostrará la construcción del proyecto. Para hacer más sencilla la explicación se dividió en tres módulos: Construcción y programación de prototipos automóvil, Módulo de grabación de video y Módulo de detección y clasificación de choques.

i. Construcción y programación de prototipos de automóvil

Se construyeron los dos prototipos de automóvil con los bloques y piezas ensamblables incluidos en el kit de tal manera que sean resistentes a choques, ver Figura 13.

Para lograrlo se ensambló la base junto con el NXT y las ruedas que permitirán al automóvil desplazarse, posteriormente se programó.

Los prototipos automóvil se construyeron lo más semejantes posible, sin embargo, debido a ausencia de algunas piezas, pocas partes de los prototipos muestran diferencias. Este detalle no fue obstáculo para el buen funcionamiento de los automóviles, ni para desarrollar los módulos posteriores.

Figura 13. Armado de los prototipos de automóvil.

Armado del robot NXT básico

A continuación se proporciona una breve descripción acerca del ensamblado del robot para construir los automóviles.

El armado se compone de 5 partes principales (Ver Figura 14).

1. Ruedas con motor.
2. Chasis.
3. Chasis parte trasera.
4. Soporte controlador.
5. Rueda trasera.

Figura 14. Señalamiento de las 5 partes principales del robot.

Paso 1: Armado de las ruedas con motor

Primeramente es importante identificar la cantidad de piezas que se muestran en la Figura 15.

Posteriormente ensamblar de manera ordenada como se muestra la Figura 16.

Figura 15. Piezas necesarias para el armado de las ruedas con motores.

Figura 16. Armado de motores con sus correspondientes ruedas.

Paso 2: Armado del chasis.

Verificar la disponibilidad de las piezas que se muestran en la Figura 17.

Realizar el ensamble del chasis como se muestra en la Figura 18.

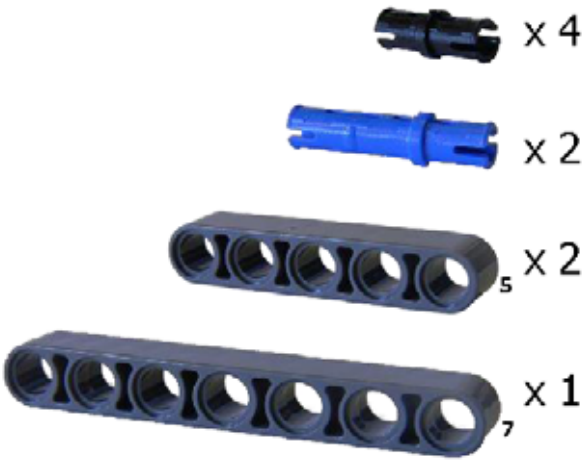

Figura 17. Piezas necesarias para el armado del chasis.

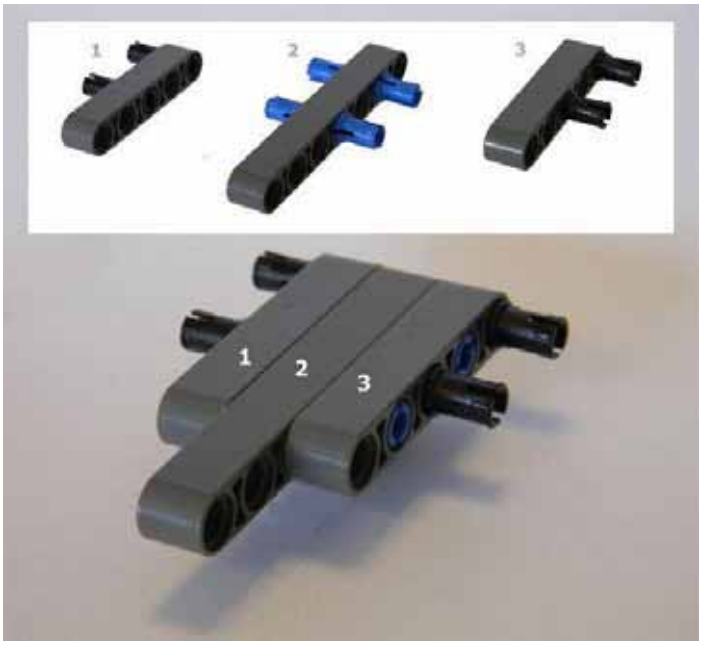

Figura 18. Armado detallado del chasis

Paso 3: Unión de motores

Realizar de manera ordenada y cuidadosa el ensamble de las ruedas del automóvil, tal como se muestra en la Figura 19.

Figura 19. Unión de los motores con el chasis.

Paso 4: Chasis parte trasera

Verificar que se poseen las piezas que se muestran en la Figura 20.

Ahora seguir el procedimiento mostrado en las Figuras 21 y 22.

Figura 20. Piezas necesarias para el armado del chasis trasero.

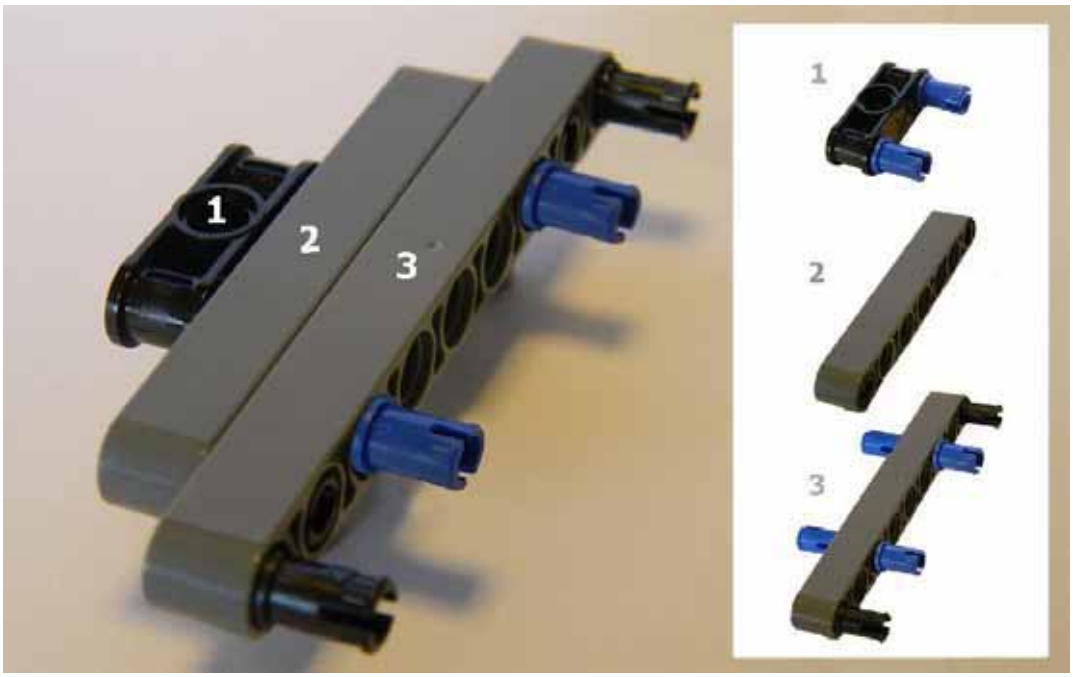

Figura 21. Armado de chasis parte trasera.

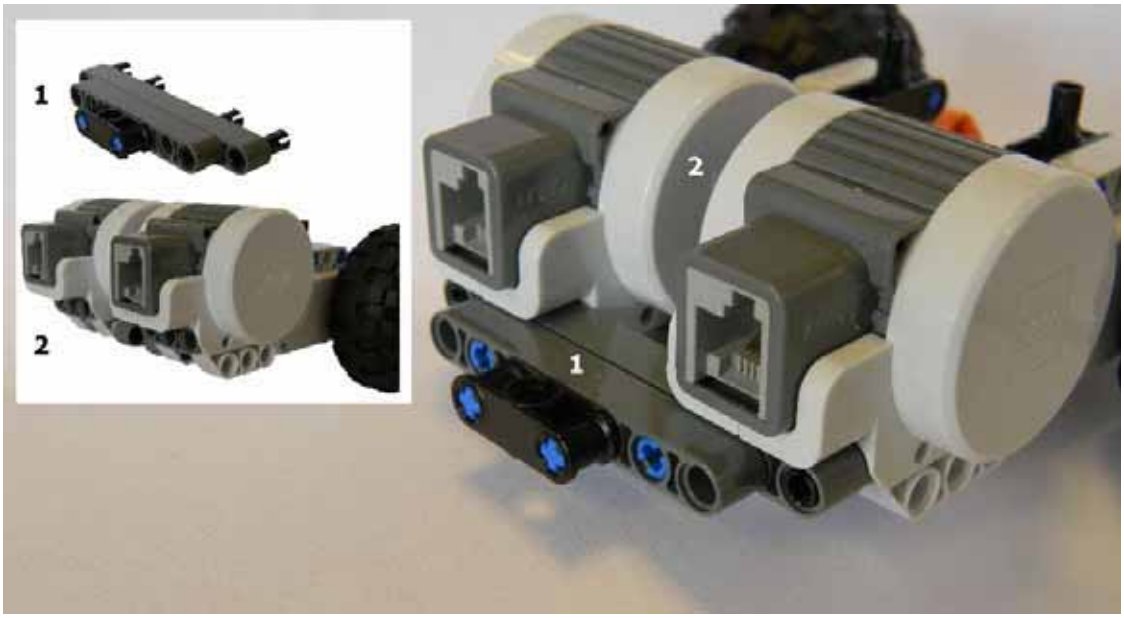

Figura 22. Unión de motores a través de chasis.

Paso 5: Soporte del controlador

Verificar tener disponibles las piezas que se muestran en la Figura 23.

Realizar el procedimiento mostrado en la Figura 23.

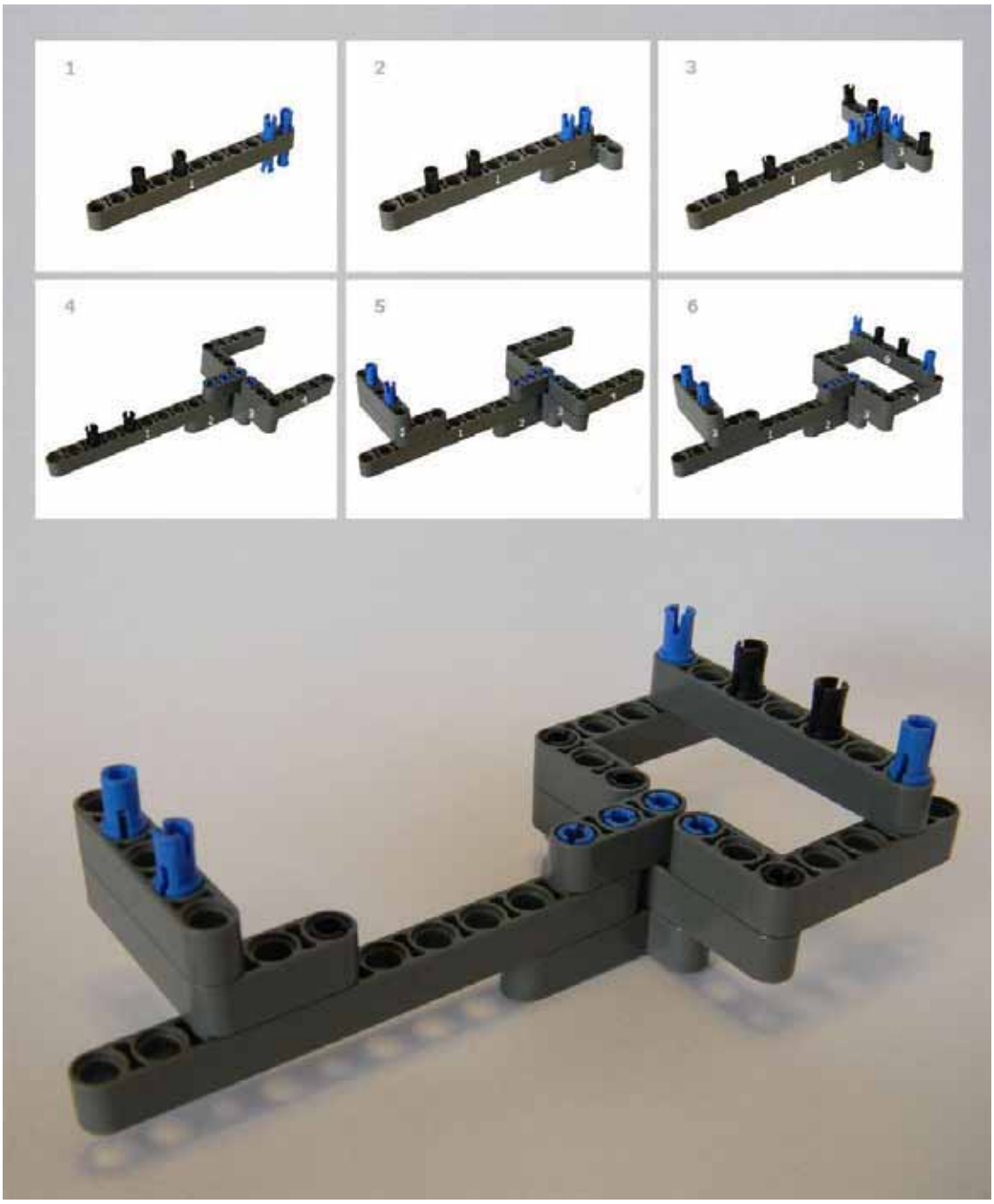

Figura 23. Armado del soporte controlador.

Paso 6: Rueda trasera

Verificar tener en disponibilidad las piezas que se muestran en la Figura 24.

Realizar el procedimiento de construcción mostrado en la Figura 25.

Figura 24. Piezas necesarias para la rueda.

Figura 25. Construcción de la rueda delantera.

Paso 12: Unión de la Rueda delantera con el robot

Verifica que se construyeron correctamente las partes en los pasos anteriores.

Realizar el procedimiento de ensamblado mostrado en la Figura 26.

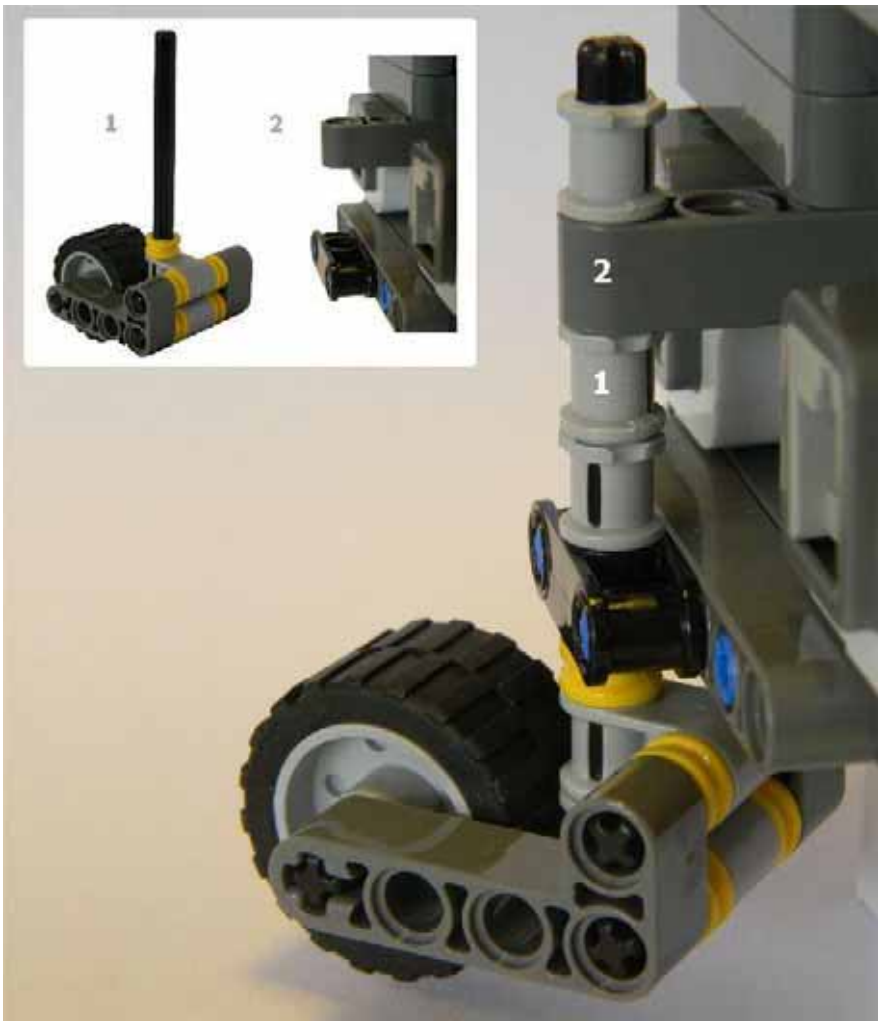

Figura 26. Unión de la rueda con el robot.

Paso 13: Bloque controlador y cables

Colocar el bloque controlador en la parte superior del soporte, de forma que el display quede cerca de la rueda delantera. Ver Figura 27.

Utilizar dos cables para conectar los motores. Conectar un cable en el puerto B del controlador y otro en el C. Ver Figura 28.

Figura 27. NXT ensamblado.

Figura 28. Cables del NXT.

Posteriormente cruzar los cables por el centro del robot (espacio que se encuentra entre el controlador y el soporte) [14]. Ver Figura 29.

Finalmente conectar los cables a los motores de las ruedas como lo muestra la Figura 30.

El resultado final es muy similar a la Figura 31.

Figura 29. Espacio entre controlador y soporte.

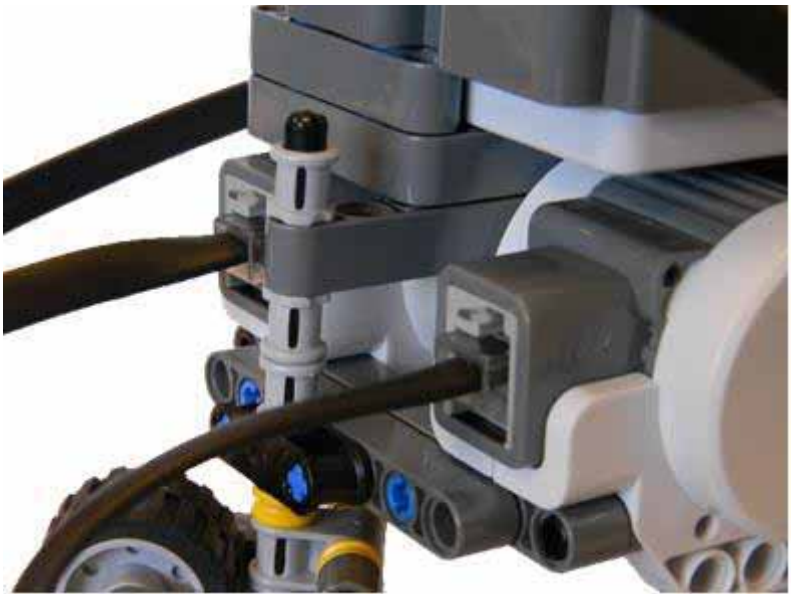

Figura 30. Conexión de cables a los motores de las ruedas.

Figura 31. Producto parcialmente terminado.

Paso 14: Formando un automóvil.

Utilizando los bloques de plástico contenidos en el kit de robótica, se realizó la construcción de la carcasa del automóvil, siempre teniendo en cuenta la norma de colores en los contornos. Debido a que los automóviles serán expuestos a choques se realizaron pruebas de dureza y se corrigieron piezas que no ensamblaban correctamente.

El producto final finalizó como se muestra en la Figura 32.

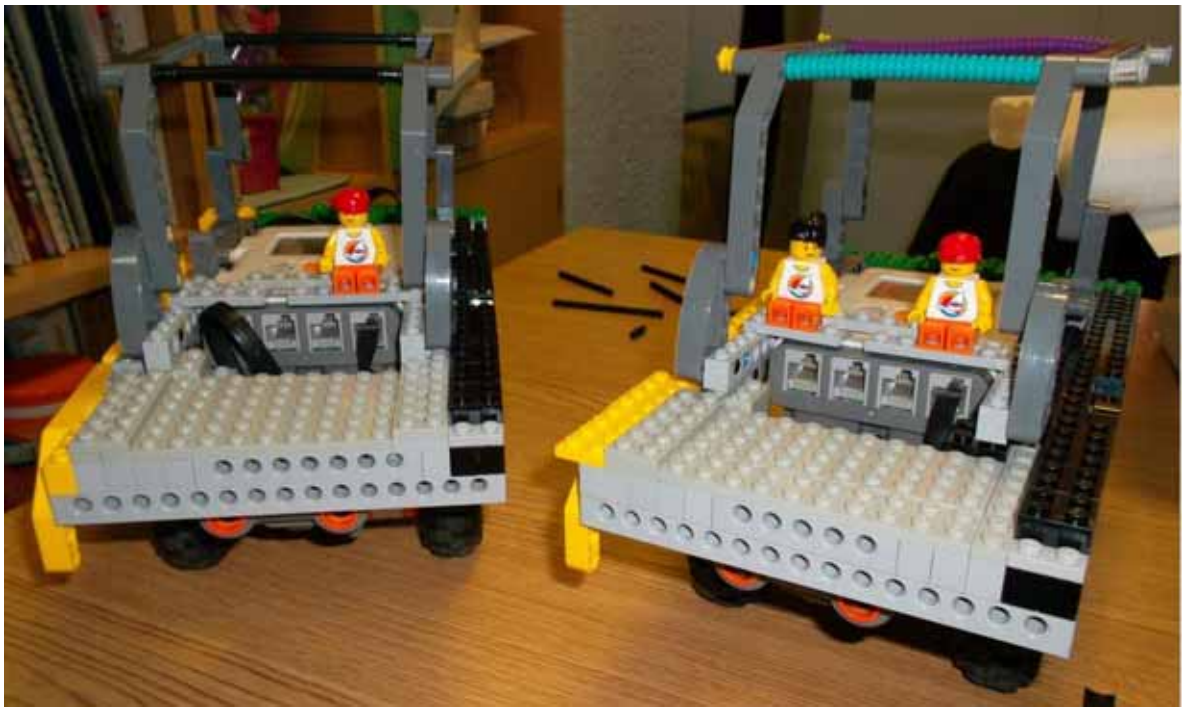

Figura 32. Prototipo automóvil terminado físicamente.

Paso 15: Programando un automóvil (Programación del NXT).

El proceso para la programación fue el siguiente:

- Se descargó e instaló en la computadora el software LEGO MINDSTORMS NXT 2.0 para programar el NXT (el software se obtiene de manera gratuita en la página de LEGO).
- Se procedió a conectar el NXT a la computadora, cuando se comunicó por primera vez fue necesario actualizar el firmware del NXT para tener compatibilidad con la versión del software que se está manejando (Pestaña Tools y luego Update NXT firmware).
- Ya realizado lo anterior la interfaz gráfica del software es muy similar a la Figura 33.
- Se procedió a realizar la programación de los recorridos aleatorios para los prototipos automóvil. Después de familiarizarse con la programación en este entorno, realizar pruebas y detectar fallas, el programa quedó como en la Figura 34.
- Para bajar el programa al NXT, simplemente conectamos y seleccionamos la opción download.

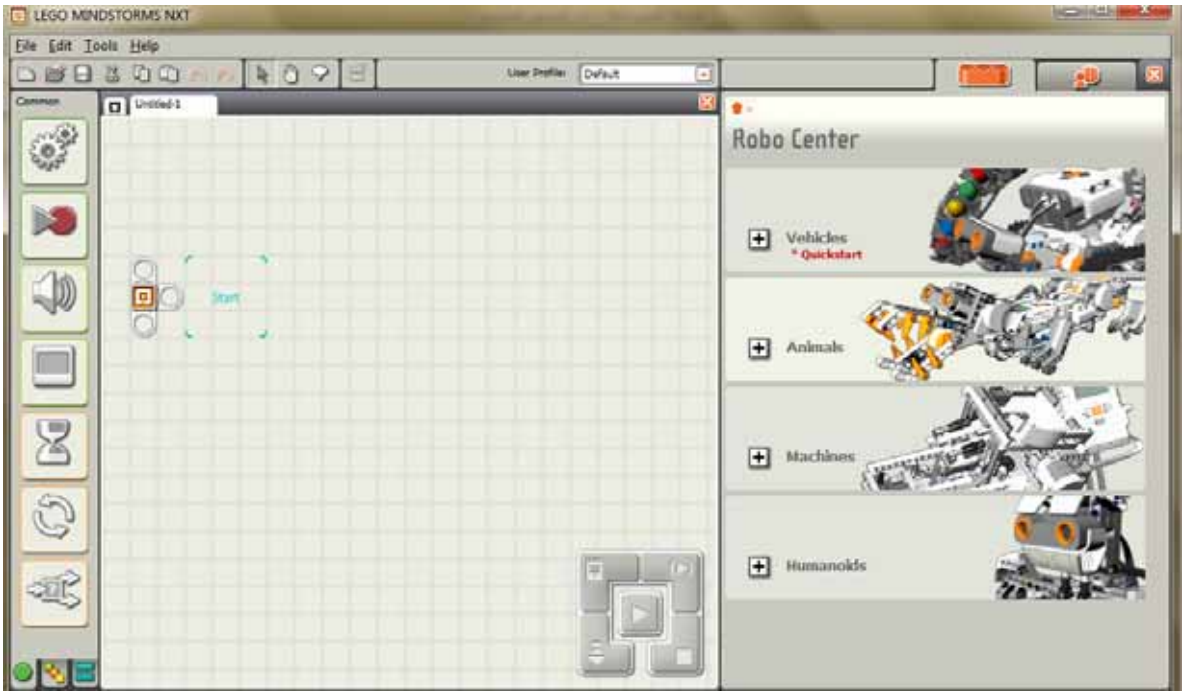

Figura 33. Interfaz gráfica del software LEGO MINDSTORMS NXT 2.0 para programar el NXT.

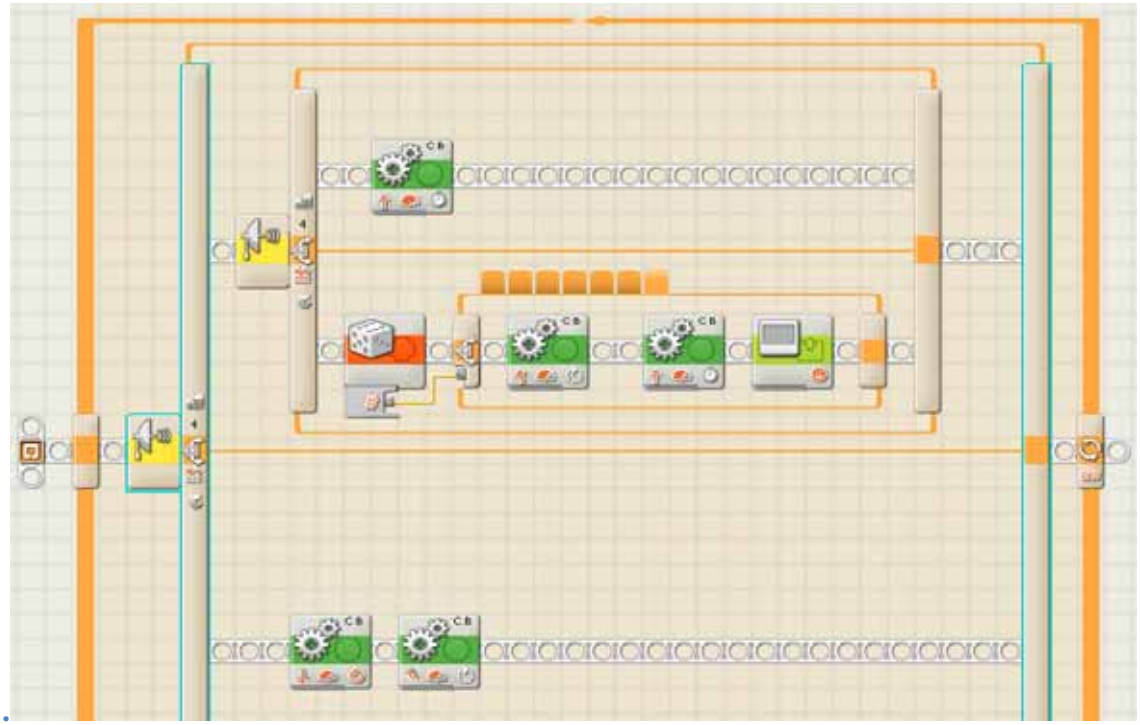

Figura 34. Programación de recorridos aleatorios para el NXT.

A continuación, muestro el algoritmo del programa (Figura 34) que utilizan los prototipos automóvil para realizar sus recorridos aleatorios.
Dónde la variable *SD* significa sensor de distancia del NXT y la variable *EL* significa Espacio Libre para que los vehículos se desplacen

Inicio
Si el SD detecta EL mayor a 30 cm

Entonces
Si el SD detecta EL mayor a 80 cm
Entonces
Los servomotores correrán hacia adelante a toda velocidad durante dos segundos

En caso contrario
Seleccionar aleatoriamente un recorrido programado
Fin del Si
En caso contrario
Girar los servomotores media vuelta hacia atrás y girar 360°.

Fin del Si
Fin del Algoritmo.

ii. Módulo de grabación de video

Figura 35. Prueba de la biblioteca JMF.

El módulo de captura de videos se realizó en dos etapas (Instalación de la biblioteca JMF y construcción del módulo de captura de videos)

Instalación de la biblioteca JMF.

- Descargué e instalé la biblioteca JMF 2.11 desde la página oficial.
- Al terminar la instalación se creó en el escritorio un icono llamado JMStudio que servía para identificar la webcam y verificar que se instaló correctamente (ver Figura 35).

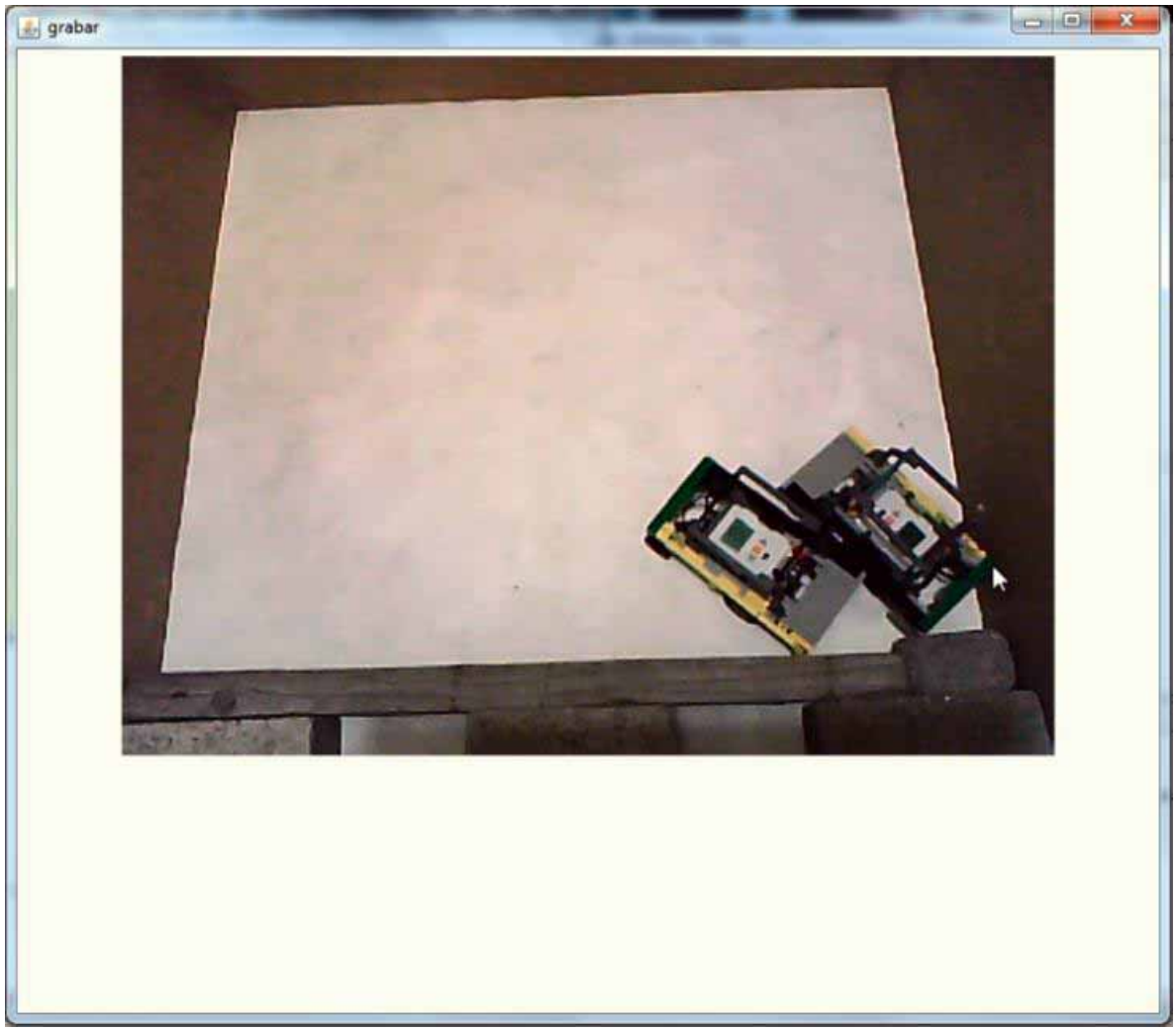

Figura 36. Interfaz gráfica de captura de video.

Construcción del módulo de captura de videos.

Utilizando lenguaje java en el entorno de desarrollo NetBeans 7.0, los conceptos de la introducción, así como ejemplos se construyó el módulo de captura de video. Cuenta con una interfaz gráfica para observar lo que se está grabando con la cámara web muy similar a lo mostrado en la Figura 36.

Dificultades y soluciones al implementar módulo de captura de videos.

Debido a que el proceso de captura y guardado de archivo se debe realizar al mismo tiempo se necesitó el manejo de hilos para simularlo lo más cercano posible [15].

Otra dificultad fue que una cámara web sólo puede ser manejada por una clase de JMF, por lo tanto o salvamos el archivo AVI o bien mostramos en pantalla las imágenes tomadas por

la cámara web, para ello se usó un concepto llamado clonación de Fuente de Datos (*DataSource*) [16].
Al tener dos Fuentes de Datos una se usó para guardar el archivo AVI y el otro para mostrar en pantalla la información obtenida de la cámara web.

ii. Módulo de detección y clasificación de choques

Para comenzar se descargó la biblioteca OpenCV en su versión 2.48 y la interfaz para java "JavaCV".

Posteriormente se agregaron los archivos opencv248, javacpp y javacv a un nuevo proyecto en NetBeans IDE 7.0.

Utilizando ejemplos, documentación y un libro [17], se creó el módulo de detección y clasificación de choques que utiliza únicamente dos pantallas: la pantalla de detección y clasificación de choques y la pantalla de región de interés (ROI).

- Pantalla de detección y clasificación de choques: Reproduce un video en formato AVI grabado por el módulo de grabación de videos, al mismo tiempo analiza si existe o no un choque entre los prototipos automóvil clasificándolo por posición de los automóviles al momento del choque. Ver Figura 38.
- Pantalla ROI (Región de Interés): Utilizada para apreciar el choque de manera más específica, también es utilizada por el algoritmo para decidir qué tipo de choque se formó. Ver Figura 37.

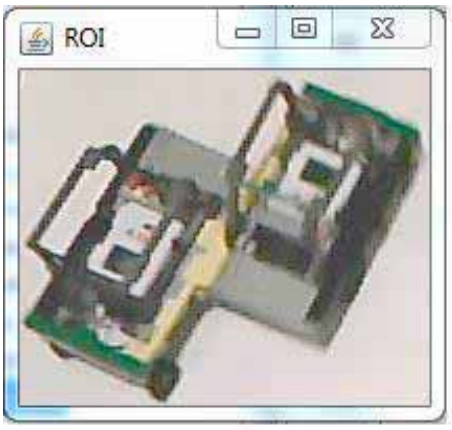

Figura 37. Ventana de Región de Interés

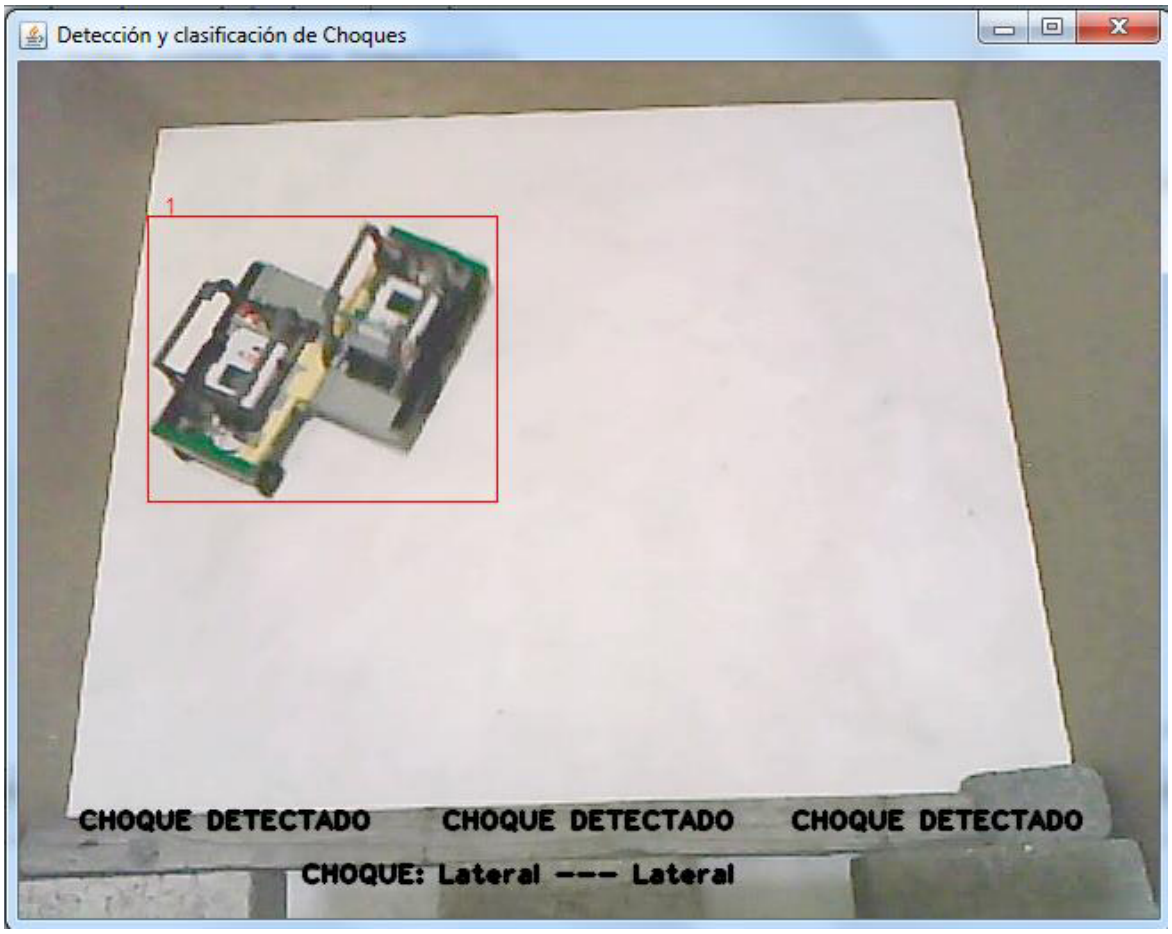

Figura 38. Pantalla de detección y clasificación de choques.

El módulo de detección y clasificación de choques no siempre se apreció como en las Figuras 37 y 38, para serlo posible pasó por varias pantallas de transformación, algunas se ocultaron para ahorrar memoria y otras simplemente evolucionaron. A continuación presentamos esas pantallas útiles en el desarrollo y su función.

- Pantalla de reproducción de video: Ésta fue la primera pantalla en la etapa de construcción del módulo de detección y clasificación de choques, su única función era reproducir el video en formato AVI creado por el módulo de grabación de videos. Ver Figura 39.

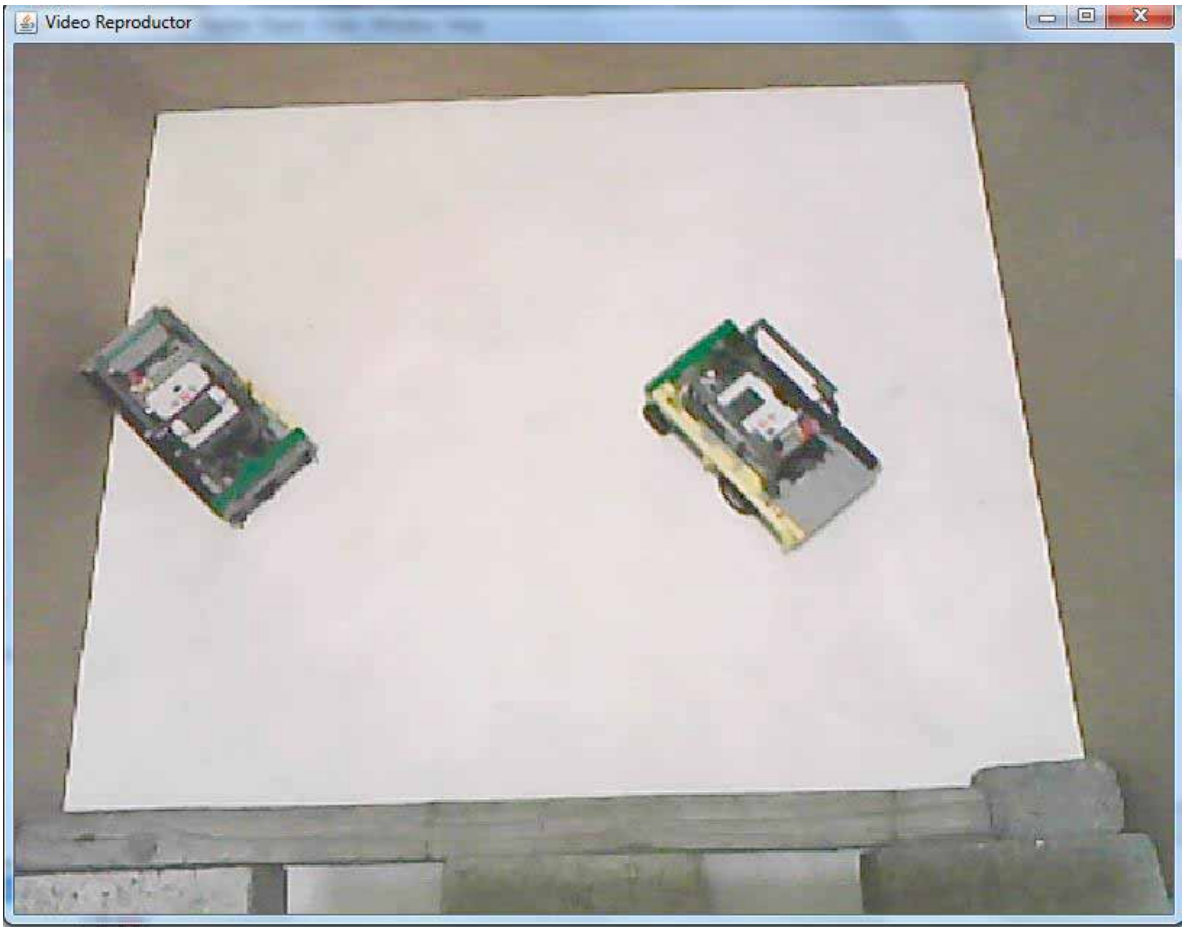

Figura 39. Pantalla de reproducción de video.

- Pantalla Movimiento: Detecta el movimiento de los prototipos automóvil, utiliza la función de OpenCV BackgroundSubtractorMOG2 para ese propósito, con ella se evitó la implementación del algoritmo AABB para detectar choques, ya que se conocen las medidas de los automóviles (cada uno mide en pantalla alrededor de 90 * 160 pixeles con una incertidumbre de 15 pixeles por causa de sombras o alrededor de los automóviles). Sabiendo que en el proyecto únicamente participan dos automóviles, cuando se detecta en ésta imagen un único objeto de proporciones similares al doble de la medida en pixeles de un automóvil, ha ocurrido un choque. Justo por el análisis anterior se evitó un desperdicio de memoria, procesador, tiempo de ejecución y realizar programación de más líneas de código para obtener resultados similares. Ver Figura 40.
- Pantalla ROI Falso: Útil para detectar un choque o un falso positivo. En ocasiones, la pantalla de movimiento detecta que ha ocurrido un choque, sin embargo, no es

verdad, antes de avisarnos que ha ocurrido un choque y desplegar la pantalla ROI se analiza que no sea un falso positivo.

Primeramente obtenemos una imagen ROI temporal y la transformamos a blanco y negro, luego dibujamos un rectángulo contorneando las figuras creadas, si se crea más de un rectángulo es un falso positivo, en caso contrario, es un choque real. Ver Figura 41.

Por motivos de variación de luz, distorsión de imagen, entre otros, la clasificación de choques no se pudo realizar utilizando el color de la carcasa de los prototipos, por lo tanto, cuando se detecta que el choque no es un falso positivo, se aprovecha la imagen ROI falso y el conocimiento de las medidas en pixeles de los automóviles para realizar la clasificación de choques.

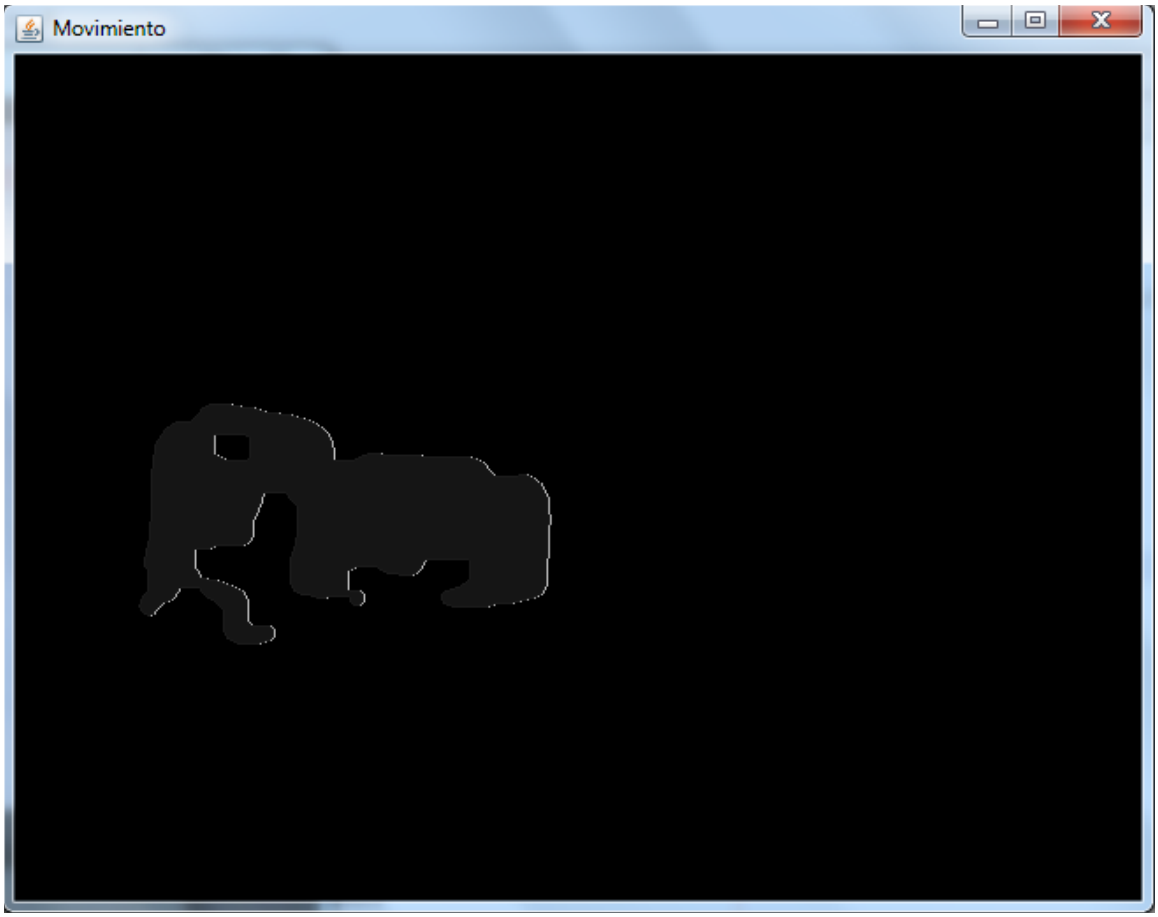

Figura 40. Pantalla de movimiento que muestra los prototipos y detecta los choques entre ellos.

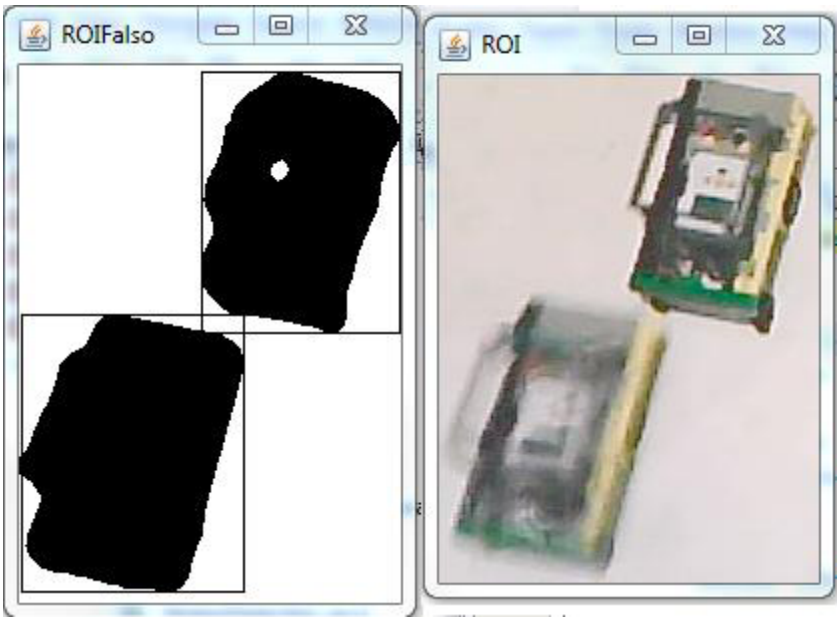

Figura 41. Pantalla de ROI falso y ROI temporal mostrando un falso positivo

Principales funciones de OpenCV utilizadas en el módulo de detección y clasificación de choques

A lo largo de la programación del módulo de detección y clasificación de choques entre los prototipos, se necesitó planificar qué acciones se deberían programar y cuáles eran las funciones de OpenCV útiles para realizar esa acción, a continuación enlisto las principales funciones empleadas:

1.- Abrir y reproducir un video formato AVI

File f = new File(video.avi);

FrameGrabber grabber = new OpenCVFrameGrabber(f);

grabber.start();

2.- Clonar imagen para dar otro uso a la imagen clon

IplImage frame = grabbedImage.clone();

3.- Detectar el movimiento de los prototipos automóvil y crear una imagen en gris y negro únicamente de los prototipos omitiendo el entorno.

BackgroundSubtractorMOG2 mog = new BackgroundSubtractorMOG2(30, 16, true);

mog.apply(grabbedImage, foreground, -1);

4.- Crear una imagen de región de interés.

CvRect r = new CvRect(ex, ey, w, h);

cvSetImageROI(roiimage, r);

IplImage cropped = cvCreateImage(cvGetSize(roiimage), roiimage.depth(), roiimage.nChannels());

cvCopy(roiimage, cropped);

5.- Obtener imagen en blanco y negro del choque de los automóviles para determinar la existencia de un falso positivo y en *caso* de que no exista clasificar un choque.

cvInRangeS(imgThreshold, cvScalar(0,0, 0, 0), cvScalar(30,30, 30, 0), imgThreshold1);

6.- Acceder a un pixel para clasificar un choque a través de un ciclo, explorando las medidas de la sombra generada por el choque.

CvScalar s=cvGet2D(src,j,i);

Algoritmo utilizado para la detección y clasificación de choques

Trabajar con visión por computadora mostró opciones distintas a las que se tenían planteadas en el comienzo del proyecto.

Al inicio se planeaba trabajar con un algoritmo de detección de colisiones para detectar los choques entre los prototipos automóvil y utilizar el color de la carcasa de los vehículos para clasificar los choques.

Al comenzar, me percaté que utilizando únicamente la imagen en gris y negro generada por el movimiento de los automóviles era de gran utilidad para detectar también un choque, ver Figura 40, luego se observó que la imagen de los choques obtenida por el video eran muy difusas y los valores RGB de los colores variaban demasiado.

Por las situaciones anteriores programé lo siguiente.

1) Leer un archivo AVI seleccionado.
2) Obtener la imagen de la escena.
3) A través de un algoritmo de OpenCV eliminamos todos los objetos ajenos a los prototipos automóvil de la imagen.

4) Se elimina el ruido de la imagen.
5) Se genera un rectángulo alrededor de cada vehículo. Cada rectángulo tiene una altura y un ancho en pixeles, en promedio de 90 x 160 pixeles, por lo tanto no podían exceder de un tamaño de 190 pixeles en alto ni en ancho.
6) Justo por las condiciones anteriores únicamente ocurre un choque si se puede crear un solo rectángulo con medidas en ancho o alto superiores a 190 pixeles.
7) Se obtiene una imagen de Región de interés para mostrar el choque en otra ventana.
8) Para eliminar falsos positivos se convierte la imagen de región de interés a blanco y negro.
9) Se generan rectángulos alrededor de todas las manchas (objetos) de la imagen ROI a blanco y negro como se mostró en la Figura 41, si se genera más de un rectángulo ocurrió un falso positivo y no se avisará que ocurrió un choque, si únicamente se genera un rectángulo el programa avisará que un choque se ha efectuado.
10) Si un choque ocurrió, a través de un ciclo se medirá la línea negra más larga generada por la imagen ROI a blanco y negro, esa medida obtenida se comparará con las medidas de los prototipos (90 * 160 pixeles en promedio), por ejemplo un choque Frontal – Frontal ocurre únicamente si los frentes de los automóviles se juntan, por lo tanto existirá una línea con una medida aproximada de 320 pixeles, un choque lateral frontal existirá con una línea de tamaño aproximado de 250 pixeles y uno lateral – lateral ocurrirá con una línea aproximada de 180 pixeles.
11) Si el video no ha terminado volvemos al paso 2.

¿Bajo qué ambiente se desarrolló el proyecto?

Obstáculos, pruebas y soluciones.

Fue complicado encontrar las condiciones favorables para detectar y clasificar los choques, lo común siempre fue una imagen difusa capturada al momento del choque. A continuación se enlistan los ambientes probados:

1. La primera prueba de grabación en video de los choques de los prototipos, se desarrolló en un ambiente similar a la Figura 42. Sin embargo, las barreras eran muy frágiles, el color del piso en momentos de obtener por computadora los valores RGB se llegaba a confundir con los automóviles, se generaban bastantes sombras que al analizar choques generaban bastantes falsos positivos.
2. Por las complicaciones anteriores, se procedió a realizar la segunda prueba, utilizando 5 focos para eliminar sombras (uno arriba y uno en cada uno de los lados del contorno de la pista de desplazamiento). A pesar de esto, aún se generaban sombras muy tenues en los 4 lados del contorno del prototipo, por ello mismo este intento también fracasó.

3. Como tercer intento se construyó una pista bajo sombra con una hora alrededor de las 18:30 horas, a pesar que a simple vista no se veía la sombra de los prototipos, la computadora si la detectaba. Por otro lado, los valores RGB de los colores variaban demasiado, incluso, hasta el color de la lona blanca que coloqué de piso.
4. En el cuarto intento se procedió a utilizar sol como luz, los resultados no fueron favorables ya que el color de los automóviles aun usando calcomanías era demasiado tenue, por otro lado, se generaba ruido en la imagen capturada.
5. En el último intento que fue el mejor de todos se utilizó luz blanca de 100w, la luz se encontraba por un costado de la pista, como piso se hizo uso de una lona blanca de 1.20 m * 1.10 m, como barreras 3 tablas de 60 cm de altura y una de 15 cm de altura para evitar obstruir la visibilidad de la cámara web que se encontraba a una altura aproximada de 1.20 m, cabe mencionar que en las cinco pruebas mencionadas se utilizó esta altura, el motivo fue que la pantalla que mostraba la grabación de video enfocaba todo el ambiente.

Al menos, bajo esas condiciones los valores RGB del color de la lona blanca variaban poco y el color de los automóviles no se confundía con el piso como en los casos anteriores, ver Figura 43.

Figura 42. Primer ambiente de desplazamiento de los prototipos

Figura 43. Ambiente para el desplazamiento de prototipos automóvil utilizado.

Resultados

En ésta sección se comprueba que los resultados del proyecto sean correctos, el funcionamiento del sistema se verificó a través de pruebas.

5 Pruebas y Resultados

Pruebas del módulo de grabación de video.

Como todo archivo, el nombre de un video almacenado en disco debe evitar algunos caracteres. Precisamente, el sistema antes de comenzar a grabar solicita al usuario escribir un nombre adecuado para el video, posteriormente, se verifica que el nombre sea válido.

A continuación se mencionan las reglas empleadas para el uso de nombres de videos.

I. El nombre de video debe estar formado por letras y números. No se puede utilizar asteriscos, signos de puntación entre otros.
II. La longitud del nombre debe tener una longitud mínima de tres caracteres y máxima de 10 caracteres, con ello evitaremos nombres muy cortos, muy largos
III. Siempre debe existir un nombre para el video.
IV. Si el nombre del video cumple las características, comenzará la grabación.

Prueba 1 "Validación de caracteres de un nombre de video"

Prueba	Resultado
Introducir caracteres distintos a un número o letra para nombrar un video (Regla I).	Mostrar un mensaje de error de escritura para volver a escribir un nombre para el video. Figura 44

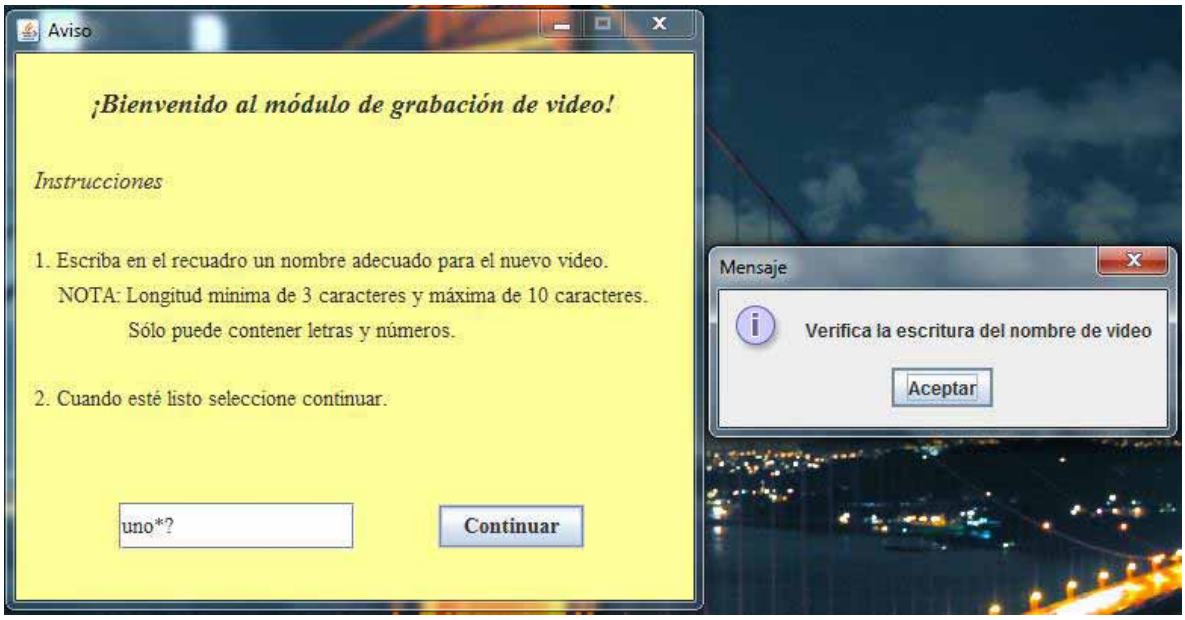

Figura 44. Verificación de caracteres de nombre.

Prueba 2 “Nombre de video con longitud pequeña”

Prueba	Resultado
Introducir un nombre muy corto para el video (Regla II).	Mostrar un mensaje de error de longitud para volver a escribir un nombre para el video. Figura 45

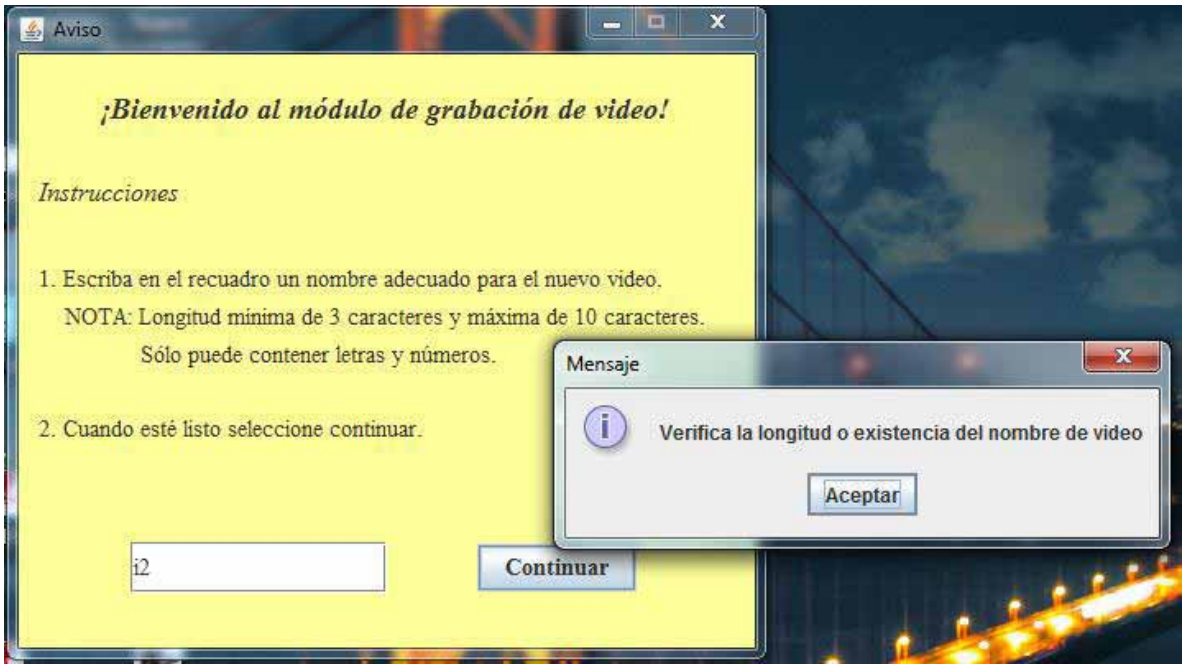

Figura 45. Longitud muy corta para el nombre del video.

Prueba 3 “Nombre de video con longitud extensa”

Prueba	Resultado
Introducir un nombre muy largo para el video (Regla II).	Mostrar un mensaje de error de longitud para volver a escribir un nombre para el video. Figura 46.

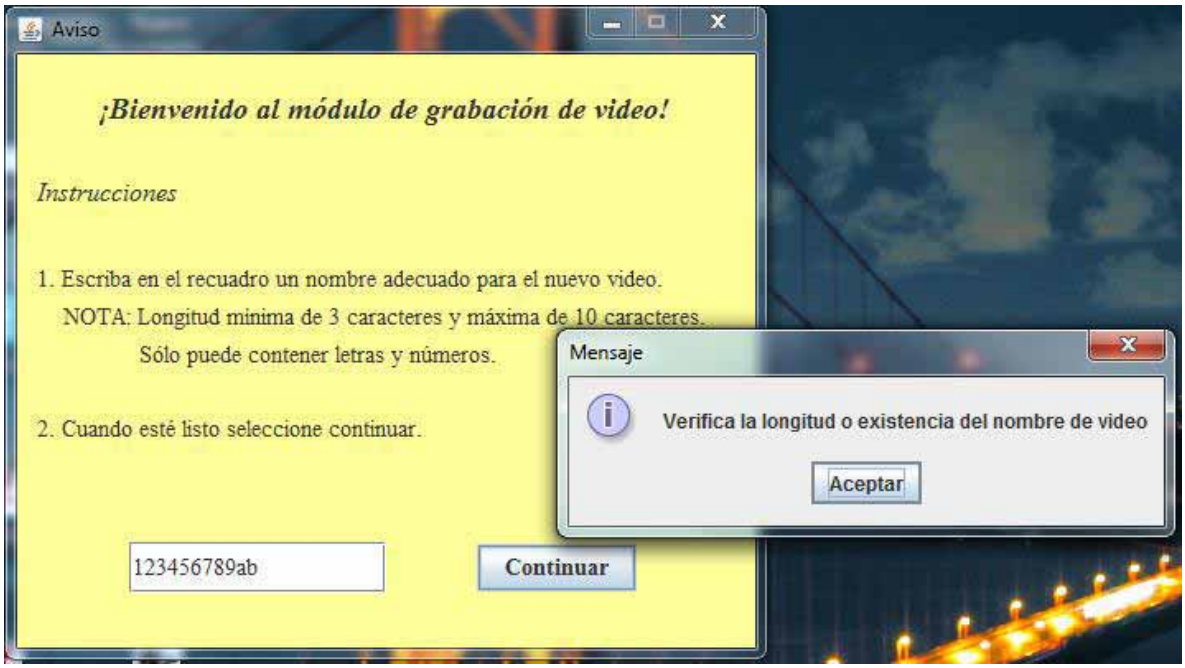

Figura 46. Longitud muy larga para el nombre del video.

Prueba 4 “No asignar un nombre para el video”

Prueba	Resultado
No introducir un nombre para el video (Regla III).	Mostrar un mensaje de error de no existencia de un nombre para el video. Figura 47.

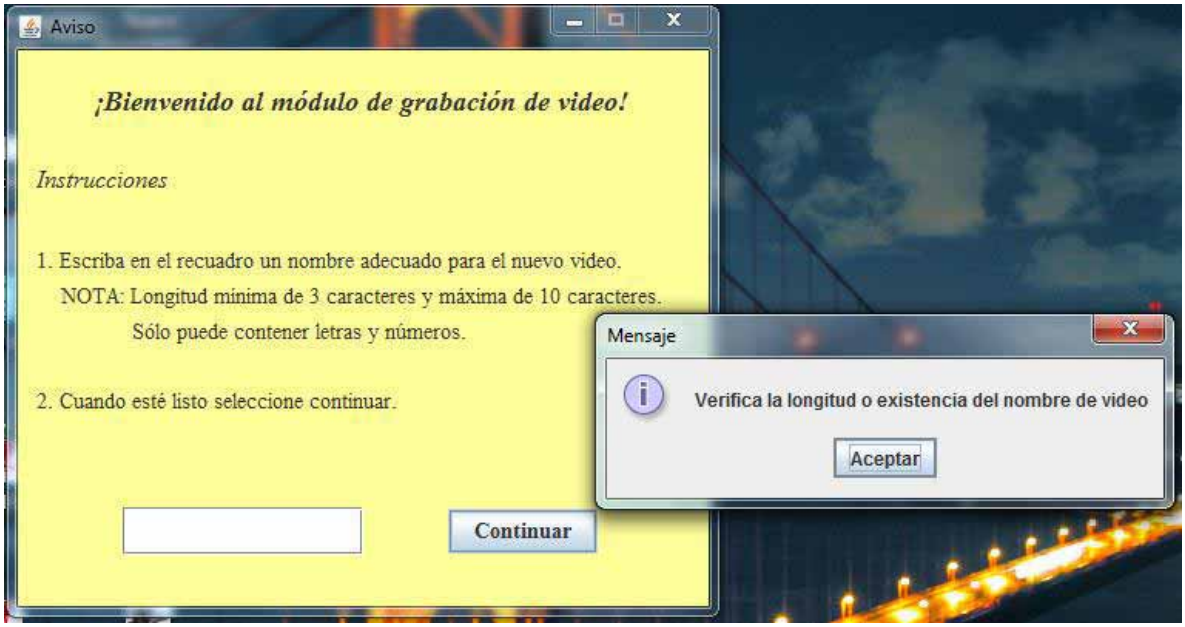

Figura 47. Inexistencia de un nombre para el video.

Prueba 5 “Asignar un nombre correcto para el video”

Prueba	Resultado
Introducir un nombre correcto para el video (Regla IV).	Mostrar un mensaje de éxito y comenzar a grabar en un archivo AVI la información que obtiene la cámara web. Figura 48.

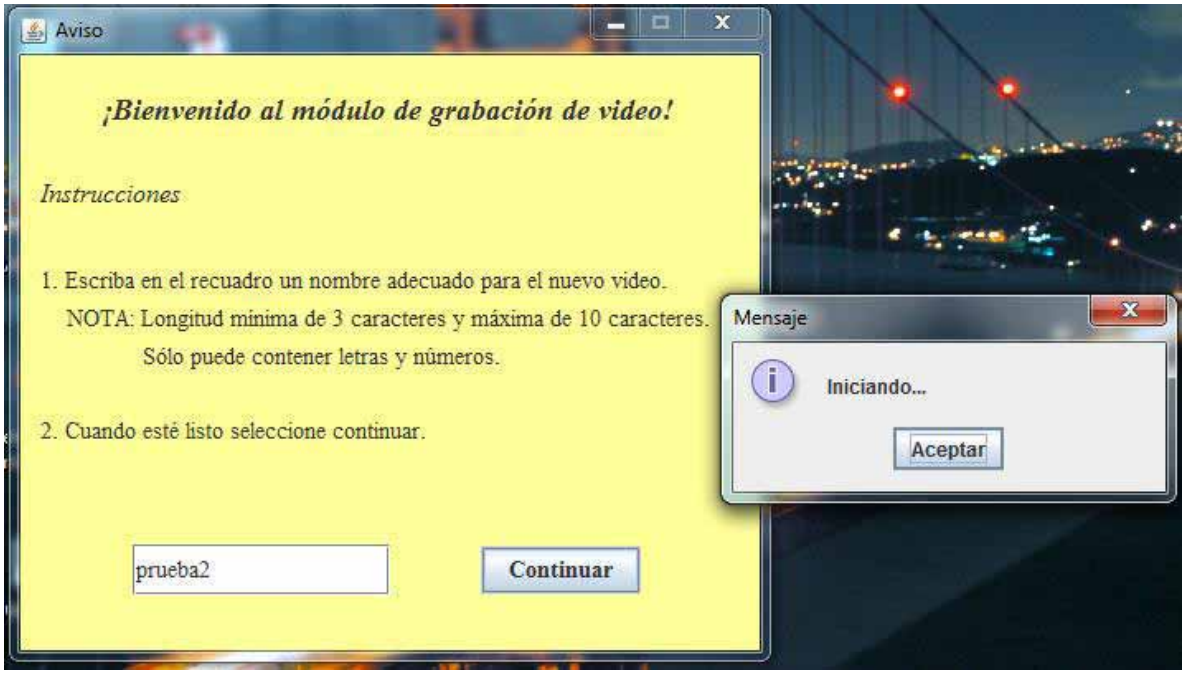

Figura 48. Éxito en el nombre asignado para el video.

Prueba 6 “Grabar y almacenar un video”

Prueba	Resultado
Almacenar un video utilizando un nombre correcto.	Captura y almacena un video en formato AVI (Figura 49), cuando ha terminado muestra un mensaje “Video guardado” (Figura 50).

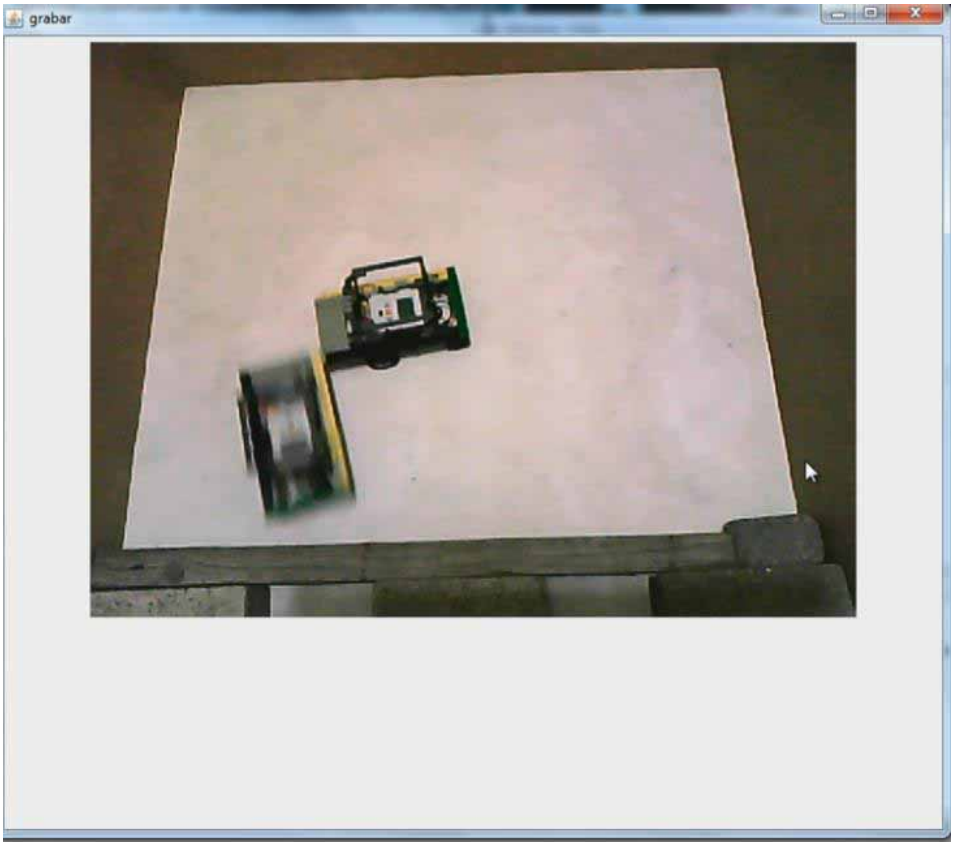

Figura 49. Captura y almacenamiento de un video.

Figura 50. Mensaje de éxito de video guardado en disco.

Pruebas del Módulo de detección y clasificación de choques.

Para realizar la detección y clasificación de choques, primeramente es necesaria la lectura de un video en formato AVI. Desde la pantalla de bienvenida al módulo de detección y clasificación de choques seleccionamos archivo y luego abrir video, ver Figura 51.

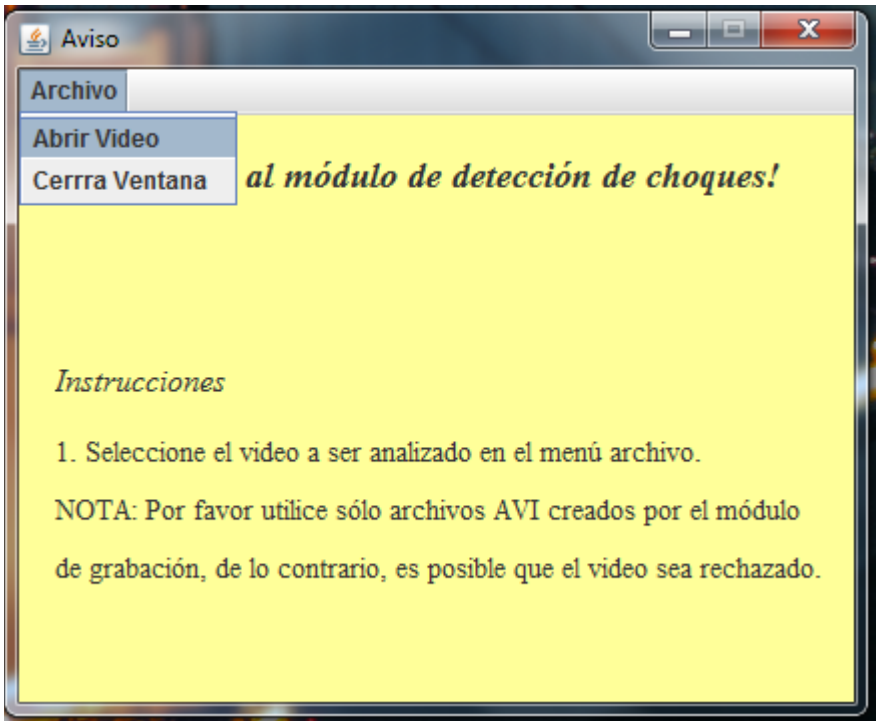

Figura 51. Pantalla de bienvenida al módulo de detección de choques.

Prueba 7 “Validar el formato de archivo”

Prueba	Resultado
Seleccionar un archivo que NO es un video formato AVI.	Mostrar un mensaje de error diciendo que no es un formato válido y volver a intentarlo. Figura 52.

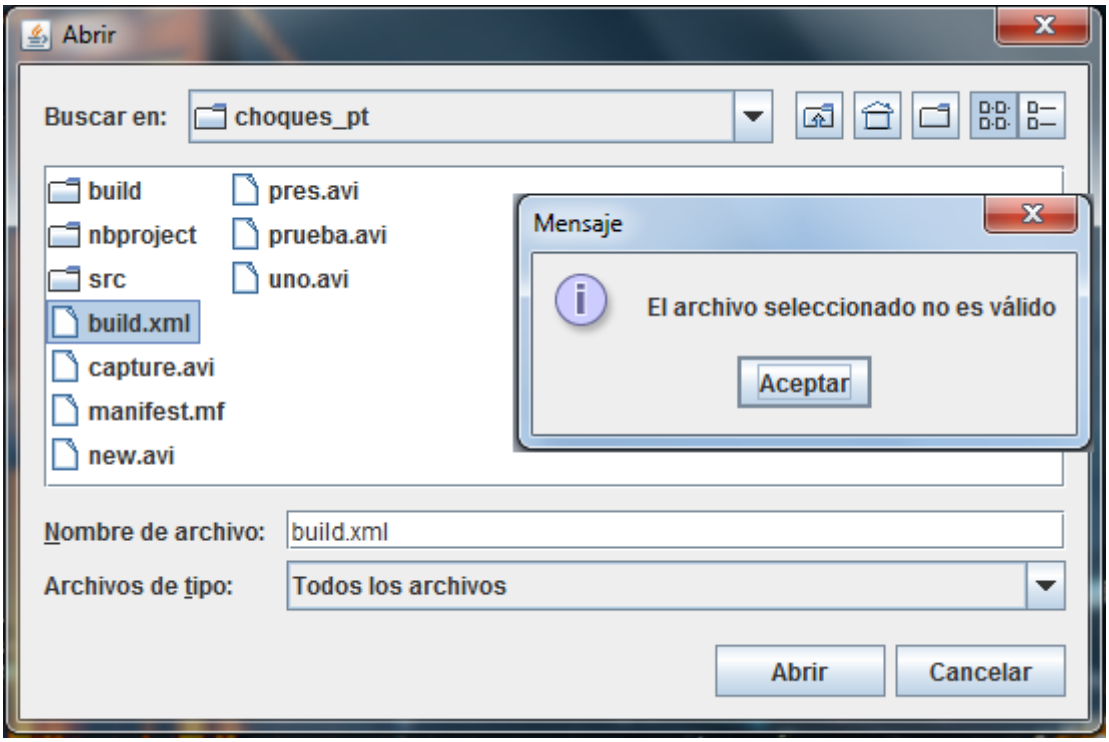

Figura 52. Validación del archivo al intentar abrir.

Prueba 8 “Abrir carpeta predeterminada”

Prueba	Resultado
Seleccionar abrir en el menú de la Figura 51.	De manera inmediata se abre la carpeta en la cual, el módulo de grabación guardó los archivos de video de formato AVI, mostrándolos exclusivamente. Figura 53.

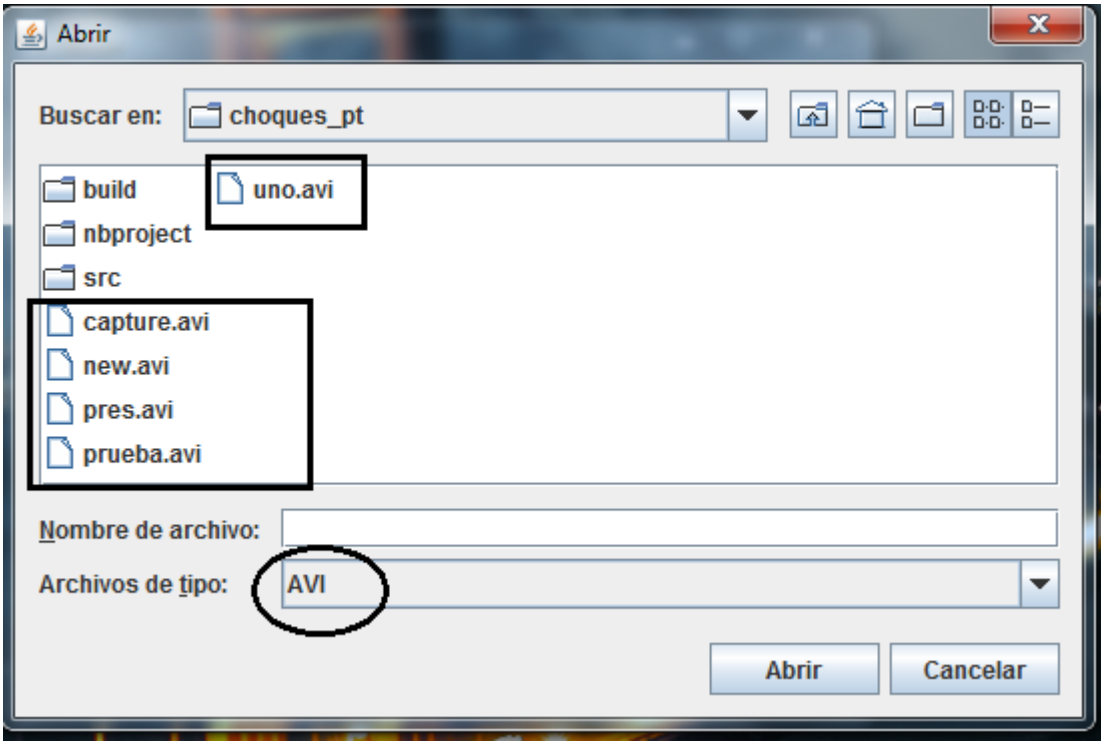

Figura 53. Muestra de archivos AVI de interés.

Si se ha abierto un video en formato AVI de manera exitosa, se procederá con el análisis del video, en donde se detectará y clasificará un choque entre los prototipos automóvil.

Prueba 9 "Detección de choques Frontal/Trasero --- Frontal/Trasero"

Prueba	Resultado
Analizamos el video y observamos que cuando exista un choque Frontal/Trasero --- Frontal/Trasero, en la mayoría de los casos se detecte de esa manera.	En la mayoría de los casos se detecta como choque Frontal/Trasero --- Frontal/Trasero. Figura 54.

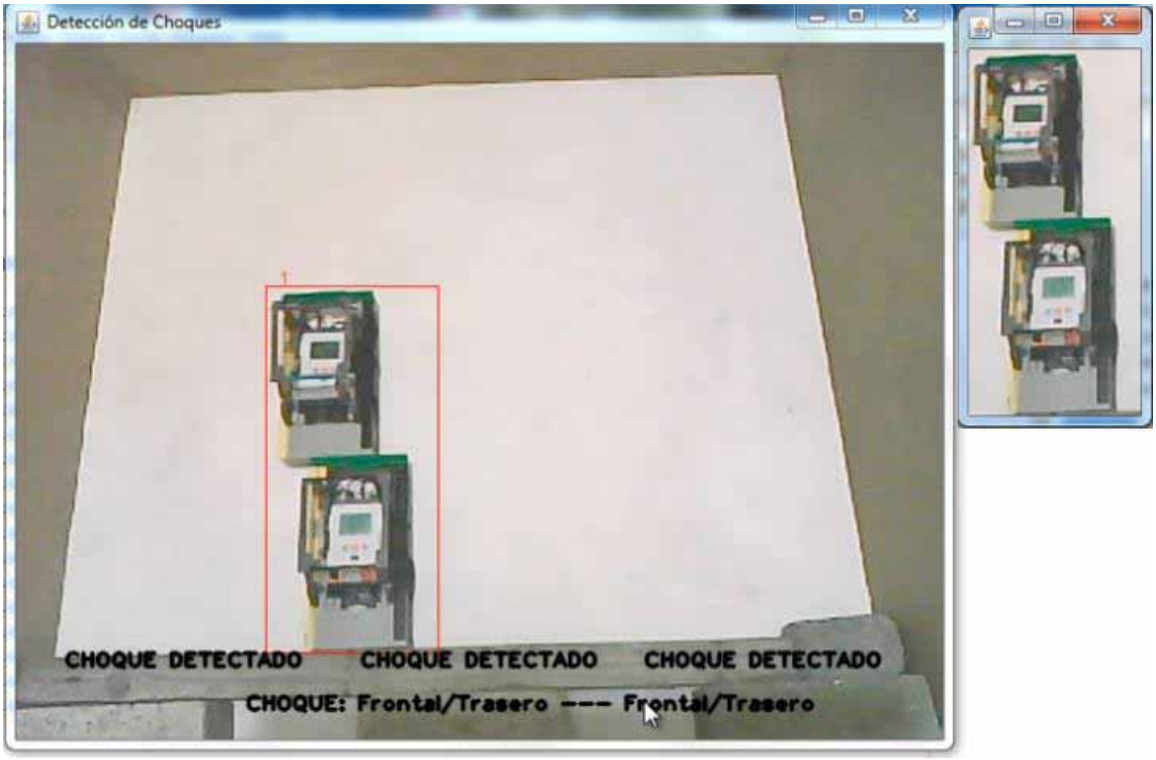

Figura 54. Detección de choque Frontal/Trasero --- Frontal/Trasero

Prueba 10 “Detección de choque Lateral --- Lateral”

Prueba	Resultado
Analizamos el video y observamos que cuando exista un choque Lateral --- Lateral, en la mayoría de los casos se detecte de esa manera.	En la mayoría de los casos se detecta como choque Lateral --- Lateral. Figura 55.

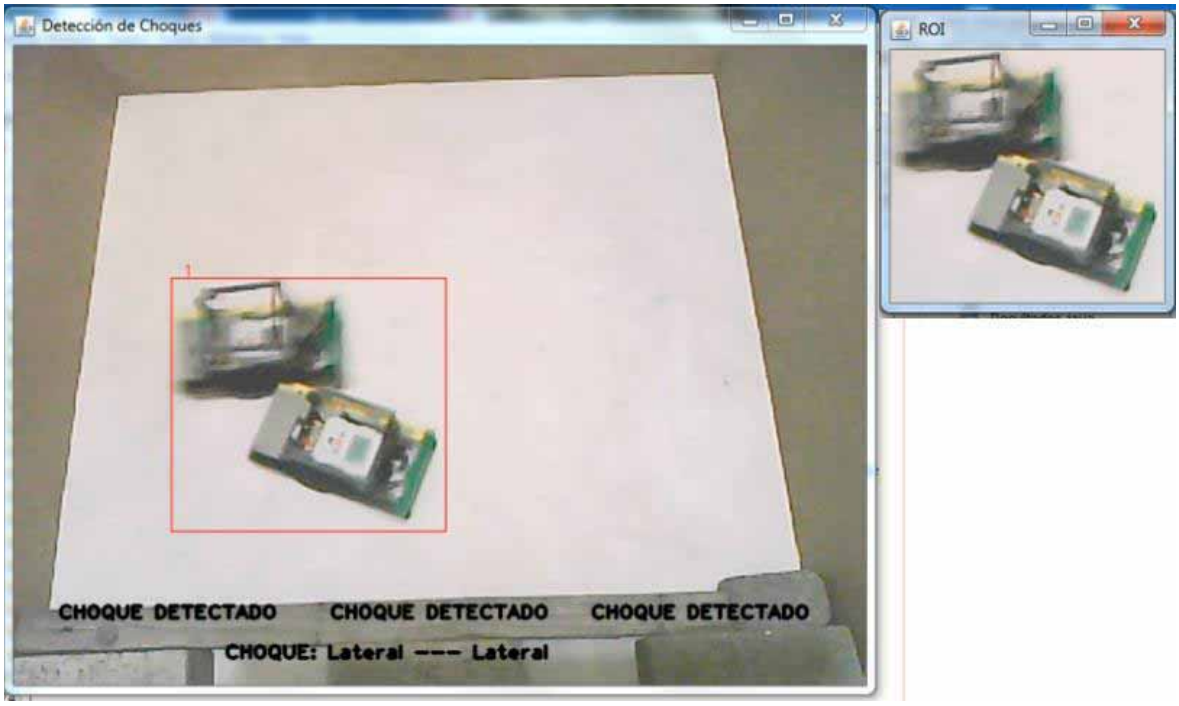

Figura 55. Detección de choque Lateral --- Lateral

Prueba 11 “Detección de choque Lateral --- Frontal/Trasero”

Prueba	Resultado
Analizamos el video y observamos que cuando exista un choque Lateral --- Frontal/Trasero, en la mayoría de los casos se detecte de esa manera.	En la mayoría de los casos se detecta como choque Lateral --- Frontal/Trasero. Figura 56.

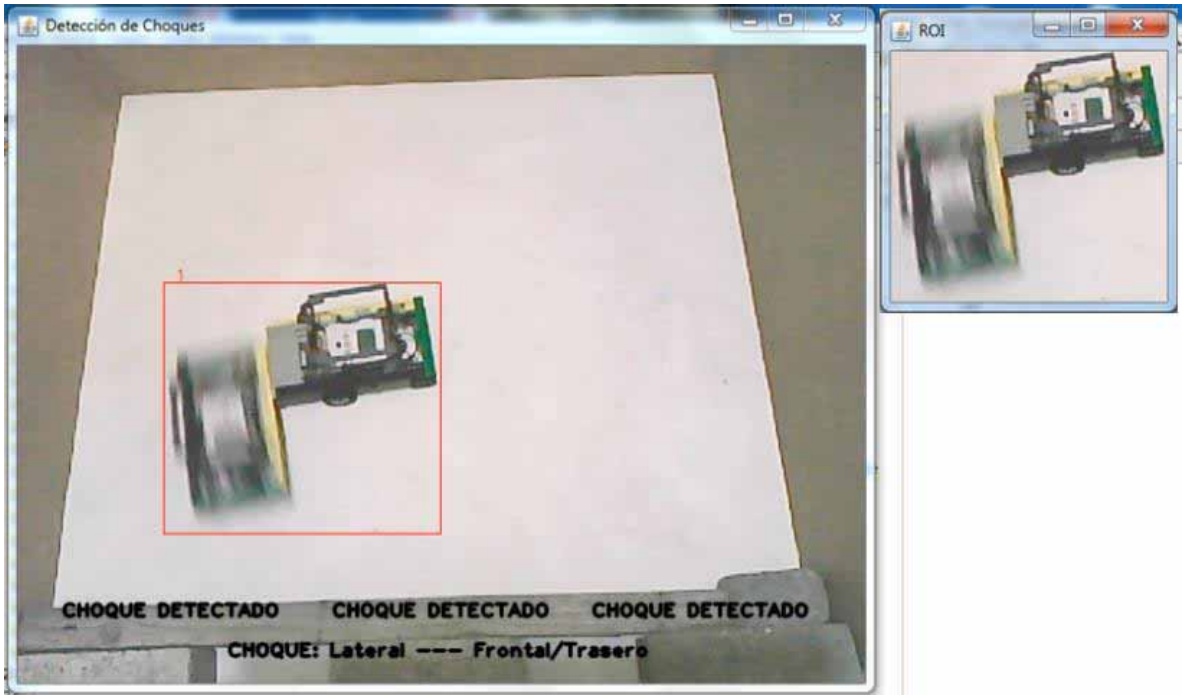

Figura 56. Detección de choque Lateral --- Frontal/Trasero

Prueba 12 “Estadísticas de choques”

Prueba	Resultado
Después del análisis del video, esperamos las estadísticas de la detección y clasificación de choques.	Aparece una ventana con los resultados en porcentaje y tiempo en segundo de los choques ocurridos. Figura 57.

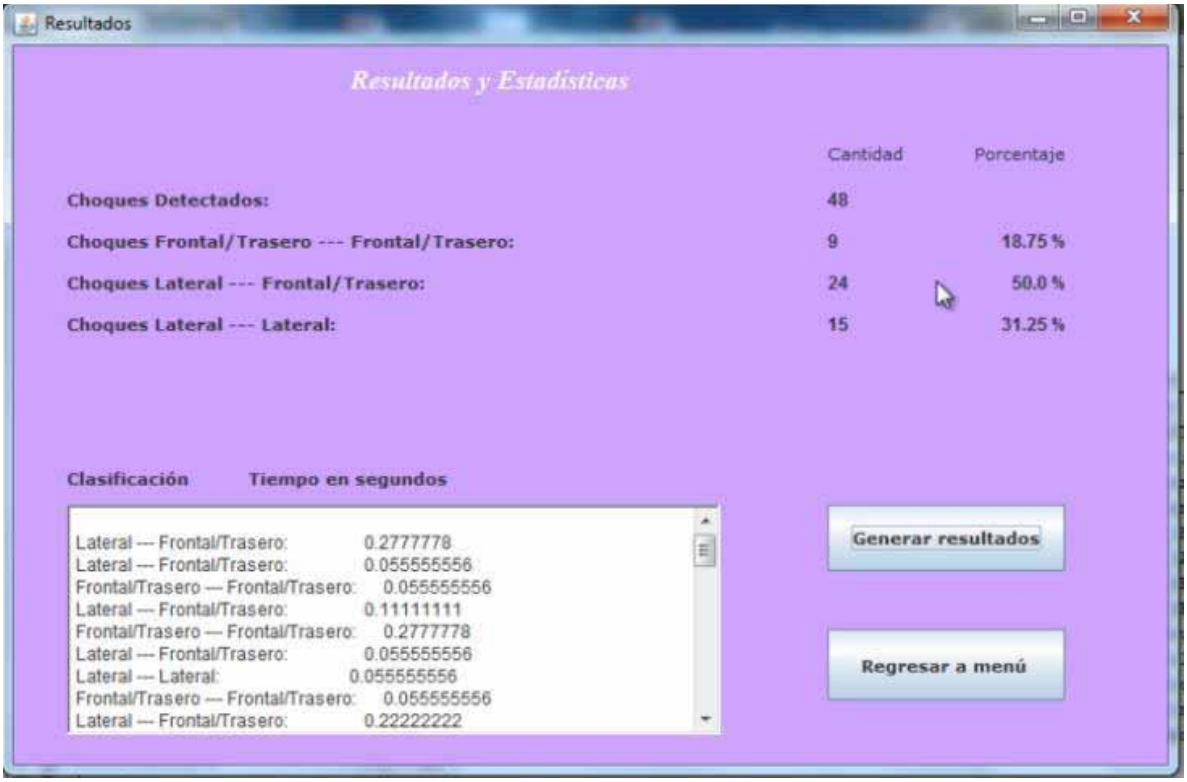

Figura 57. Resultados y estadísticas de la detección y clasificación de choques.

Resultados “Falsos positivos, Falsos negativos y Choques detectados”

Se escogieron tres videos grabados, de los cuales se obtuvo la información de cuántos falsos positivos, falsos negativos y choques detectados ocurrieron.

Como referencia de resultados tomé lo siguiente:

- Falsos positivos: Son aquellos choques que fueron detectados, sin embargo, al ver la imagen de región de interés se aprecia que no hay choque. En la mayoría de los casos fueron producidos por imágenes difusas por el movimiento y distorsión de colores (ver Figura 58).
- Falsos negativos: Son aquellos choques que se aprecia en pantalla que ocurrieron, sin embargo, no se detectaron. La razón de no detectarlo es que al momento de grabación de video la cámara web no se encontraba a una altura adecuada, por lo tanto, el programa desconoce la medida en pixeles de los automóviles de ese video (ver Figura 59).
- Choques detectados: Son los choques que si ocurrieron en un video (ver Figura 60).

Figura 58. Choque falso positivo.

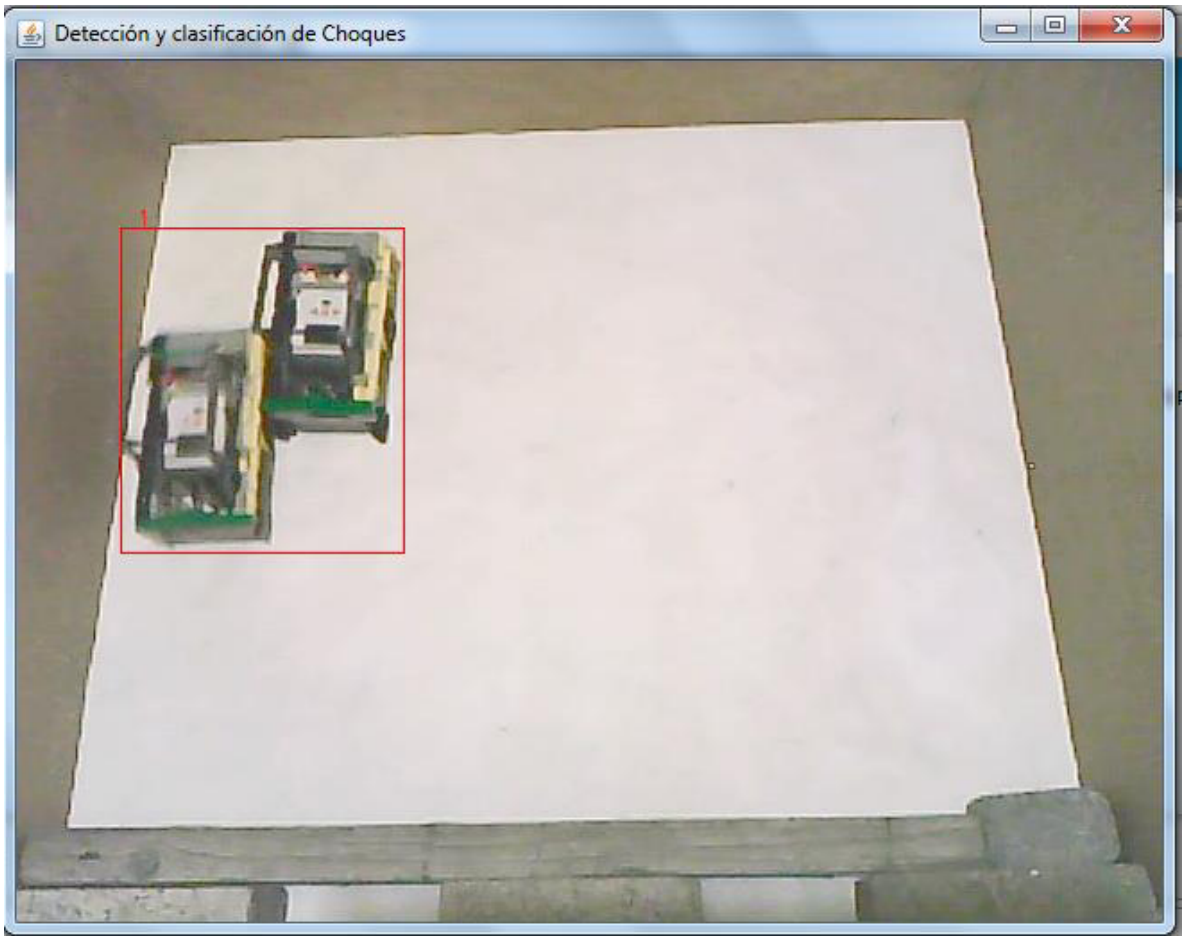

Figura 59. Choque falso negativo.

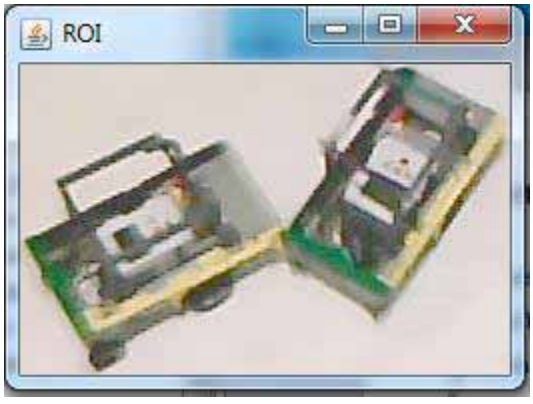

Figura 60. Choque detectado

Retomando la estadística de las pruebas realizadas, en la Figura 61 se muestra la tabla comparativa de los resultados obtenidos.

Al video dia4g.avi se le practicó la prueba, durante la grabación de dos minutos que realizó el módulo de grabación de videos, existieron un total de 48 choques reales ocurridos que representa el 87.27%, mientras tanto ocurrieron 7 falsos positivos que representa el 12.72% y el mejor de los casos no ocurrió algún falso negativo.
La detección de choques en éste archivo fue buena ya que además de tener un alto porcentaje de detección, se observó que cinco de los siete falsos positivos detectados ocurrieron únicamente por que los automóviles estaban a punto de chocar o bien por que ya habían chocado y estaban por retomar su nuevo camino.

La prueba realizada al video dia4j.avi obtuvo 50 choques reales que representa el 78.12%, mientras que ocurrieron 11 falsos positivos representando un 17.18% y tres falsos negativos que representan 4.68%. Los resultados fueron buenos ya que el número de choques detectados era superior al video anterior, aunque es importante mencionar que en esta ocasión si existieron falsos negativos e incrementó el número de falsos positivos.

Finalmente el archivo dia4k.avi mostró una disminución en todos los parámetros, ocurrieron 16 choques reales representando 72.72%, con cuatro falsos positivos que representan 18.18% y dos falsos negativos que representa 9.1%. Los resultados obtenidos son regulares, ya que disminuyó la cantidad de choques reales, aunque es importante mencionar que no es motivo del sistema, si no del video, ya que no contenía suficientes choques ocurridos entre los prototipos automóvil.

De manera general, analizando los resultados obtenidos en tres videos distintos, se puede decir que en promedio ocurren 38 choques reales representando 80.80%, 7 falsos positivos representando 14.89% y dos falsos negativos que representa 4.25%. Los resultados promedio son buenos ya que los choques reales muestran un porcentaje superior al 80%.

Nombre de video	Choques detectados	Falsos positivos	Falsos negativos
dia4g	48	7	0
dia4j	50	11	3
dia4k	16	4	2

Figura 61. Tabla comparativa de choques, falsos positivos y falsos negativos.

Conclusiones

En ésta sección se presenta las conclusiones obtenidas después del desarrollo del proyecto.

6 Conclusiones

Trabajar en éste proyecto de visión por computadora ha sido uno de los retos más grandes e interesantes de mi trayectoria académica, incluso, como jamás había desarrollado algún programa básico de visión artificial llegué a pensar que podía ser imposible.

La construcción y programación de los prototipos automóvil utilizando LEGO, fue una tarea divertida y sencilla, con el material correspondiente, realicé pruebas sencillas de funcionamiento, observando que se pueden construir muchos tipos de robots básicos, siguiendo nuestra imaginación e ingenio.

El diseño de la carcasa y cuerpo de los automóviles utilizando los bloques fue entretenido, siempre pensando, que debía ser resistente a los choques, para evitar que en la grabación de un video pudiera deshacerse o causarse daño a sí mismo, como tropezarse o no poder avanzar por obstaculizarse.

La programación de los recorridos aleatorios se realizó tomando en cuenta varias pruebas, giro de las ruedas, velocidad del vehículo, la distancia a un objeto cercano para evitar golpearse o golpearse demasiado duro y en caso de chocar con alguna barrera o bien con el otro vehículo, ser capaz de librarse así mismo, evitando por completo la interacción del humano en ese proceso.

Para ayudarme en el tema de visión por computadora utilicé la biblioteca especializada OpenCV. Con la experiencia que obtuve en el proyecto, me percaté que manejar ésta biblioteca en combinación con lenguaje de programación Java, tiene todavía algunos fallos, además, no existe mucha información que pueda ser útil, ya que generalmente los libros, documentaciones y blogs utilizan más a C y C++ como lenguaje.
En ocasiones, llegué a pensar que utilizar las funciones de OpenCV implementadas en conjunto con C o C++ de manera similar en Java podría tener éxito, en la mayoría de las ocasiones fue nulo, ya que en los lenguajes mencionados, incluso se hacían pasos de preparación para aplicar cierta función y en Java eran innecesarios, por otra parte, algunas funciones tenían un nombre diferente en Java, lo que complicaba su traducción.

Por los detalles mencionados decidí utilizar OpenCV únicamente para la detección y clasificación de choques, empleando sus funciones de grises, colores y demás similares en el manejo de imágenes.
Cabe mencionar, que el módulo de detección y clasificación de choques fue lo más inestable del proyecto, los motivos fueron variados, por ejemplo: el ambiente, la iluminación y la distancia de la cámara web a la pista en donde se desplazaban los

automóviles, que alteraban los resultados o bien no permitía generarlos, por lo tanto, experimentando, intenté adecuarlos al alcance. Entre las soluciones realizadas fueron: cambiar la distancia de la cámara a la pista, hacer pruebas de grabación en el sol y sombra, iluminar el ambiente con cuatro focos para eliminar sombras, cambiar el color de piso, cambiar el color y dureza de las barreras, incluso, se colocaron hojas y calcomanías de colores para tener mejor visibilidad y contraste de los prototipos en video.

Los resultados de la detección y clasificación de choques muestran que aún se puede perfeccionar más el rendimiento del proyecto, ya que a pesar que se construyó una función para detectar falsos positivos, en el análisis de resultados se mostró que aún existe presencia de ellos.

Para el módulo de grabación de videos utilicé JMF, que en comparación con OpenCV, no necesita codecs para guardar videos y existe más información especializada en el tema de captura y almacenamiento de videos. La programación de éste módulo me hizo apreciar los alcances de ésta biblioteca, en lo personal, no imaginaba cómo era el funcionamiento del código de un software tan básico como el que tienen integrado las computadoras personales para el manejo de su cámara web (captura de fotos y video).

El proyecto fue muy completo, generó aprendizaje en varios campos como robótica básica con LEGO, manejo de imágenes y almacenamiento de video, visión por computadora y procesamiento de videos. Actualmente, por muy sencillo que parezca, éstos temas son de interés para la tecnología moderna, aunque en México no se desarrolle investigación sobre la inteligencia artificial, siempre es bueno conocer las bases de éste estilo de temáticas.

Trabajos futuros

- Para éste mismo proyecto utilizar una cámara HD para la grabación de videos. Un motivo muy fuerte para hacerlo es detectar los colores del ambiente con mayor claridad, con ello habría una detección y clasificación de choques más óptimos.
- Utilizando las bases de éste proyecto aplicarlo a las cámaras de vigilancia de la Ciudad de México instaladas en las principales glorietas y cruces peligrosos.
  Este sería un trabajo muy complejo, a continuación las principales dificultades que se enfrentarían:
    1. Para comenzar es indispensable pensar que hay muchos modelos y colores de automóviles que sustituirán a los robots del proyecto.
    2. Los tipos de piso son muy distintos concreto, pavimento, entre otros.

3. Crear la detección de choques para una mayor cantidad de autos, a una mayor distancia (ya que las cámaras de la ciudad no se encuentran a una altura de 1.20 m como en el proyecto).
4. Automóviles que choquen estando detenidos en un semáforo.
5. El ruido que se puede causar en una imagen por clima como lluvias, día nuboso y día soleado.

Debido a la dificultad del trabajo se podría realizar algunos trabajos previos:

1. Detectar y clasificar los choques entre los prototipos usando de piso concreto y/o pavimento.
2. Analizar choques con diferentes estados de los prototipos como pueden ser detenido, avance lento, avance rápido y sin control.
3. Realizar la detección y clasificación de choques de manera exitosa bajo día soleado y día nublado.
4. Involucrar otros prototipos para realizar la detección de choques entre más de dos automóviles.

- Usar la detección de choques en las cámaras de vigilancia, agregando un módulo de cálculo de velocidad, con ello sabríamos si un choque ocurrió por exceso de velocidad.

Bibliografía

Se muestra la lista de referencias utilizadas en la elaboración del proyecto.

Bibliografía

[1] Jiménez Gerardo, “Cámaras de video-vigilancia, una mira de largo alcance” Enero 2013. [En línea]. Disponible en: http://www.excelsior.com.mx/2013/01/28/881432 [Consultada 19 de mayo de 2013]

[2] INEGI, “Accidentes de tránsito terrestre en zonas urbanas y conurbanas” [En línea]. Disponible en: http://www.inegi.org.mx/est/contenidos/Proyectos/registros/economicas/accidentes/default.aspx [Consultada 13 de mayo de 2014]

[3] J.A. Hernández García, “Detección de movimiento a través de secuencias de video”, Tesis de Maestría, ESIME Culhuacán, Instituto Politécnico Nacional, D.F., México, 2006.

[4] Z. Kai Ku et al., “Shape-based Recognition and Classification for Common Objects - An Application in Video Scene Analysis”, in Second International Conference on Computer Engineering and Technology (ICCET), 2010 © IEEE Xplore. DOI: 10.1109/ICCET.2010.5485747

[5] Yang Changjiang et al., “Fast Multiple Object Tracking via a Hierarchical Particle Filter”, in Tenth IEEE International Conference on Computer Vision (ICCV’05), 2005 © IEEE.

[6] Treptow André et al., “Real-Time Object Tracking for Soccer-Robots without Color Information”, Department of Computer Science, University of Tuebingen, Tuebingen, Germany.

[7] “Download source”, 2013. [En línea]. Disponible en: http://www.downloadsource.es/1771449/acer-crystal-eye-webcam/

[8] LEGO group “LEGO Mindstorms”, 2012. [En línea]. Disponible en: http://legomindstorms.es [Consultada 12 de noviembre de 2013]

[9] Catalán Sergio, “Java Media Framework” [En línea]. Disponible en: http://profesores.elo.utfsm.cl/~agv/elo330/2s03/projects/JavaMediaFramework/principal.htm [Consultada 13 de mayo de 2014]

[10] Parra Gerardo, “JConferencia: Audio-Conferencia en Java” [En línea]. Disponible en:

http://www.hpca.ual.es/~vruiz/docencia/redes_globales/proyectos/Audio-conferencia/JConferencia:%20Audioconferencia%20en%20Java%20-%20Gerardo%20Parra%20Juan%20de%20la%20Cruz/JConferencia.pdf [Consultada 13 de mayo de 2014]

[11] García, Antony, "Multiprocesos en Java: Como usar hilos (Threads)" [En línea]. Disponible en: http://panamahitek.com/multiprocesos-en-java-como-usar-hilosthreads/ [Consultada 13 de mayo de 2014]

[12] OpenSource Initiative, "OpenSource Initiative" [En línea]. Disponible en: http://opensource.org [Consultada 01 de agosto de 2014]

[13] Sanchez, Sergio, et al. "Introducción a las librerías OpenCV" [En línea]. Disponible en: http://web-sisop.disca.upv.es/imd/cursosAnteriors/2k3-2k4/copiaTreballs/serdelal/trabajoIMD.xml [Consultada 13 de mayo de 2014]

[14] Perdomo Pablo, Kit básico LEGO MindStormsEducation "Instructivo para el armado del kit básico LEGO MindStorms Education" [En línea]. Disponible en: https://eva.fing.edu.uy/pluginfile.php/51097/mod_resource/content/0/Instructivo_Kit_basico_LEGO_MindStorms_Education.pdf [Consultada 14 de noviembre de 2013]

[15] Santamaría, Rodrigo, "Java Threads" [En línea]. Disponible en: http://vis.usal.es/rodrigo/documentos/aso/seminarios/javaThread.pdf [Consultada 13 de mayo de 2014]

[16] Oracle, "OTN Community, Capture device in use" [En línea]. Disponible en: https://community.oracle.com/thread/1280675?start=0&tstart=0 [Consultada 13 de mayo de 2014]

[17] G. Bradski, "Learning OpenCV", Primera edición, Sebastopol California, O' Reilly , 2008.