Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e ingeniera

Licenciatura en Ingeniería en Computación
Proyecto Tecnológico

MÓDULO DE VISIÓN PARA QUE UN ROBOT SEA CAPAZ DE LOCALIZARCE DENTRO DEL ÁREA DE JUEGO

Rafael Rayón Aldana
207206633

Asesora: Dra. Silvia Beatriz González Brambila
Trimestre 2018-I

24 de abril de 2018

.

Yo, Dra. 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.

Dra. Silvia Beatriz González Brambila

Yo, Rafael Rayón Aldana 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.

Rafael Rayón Aldana

RESUMEN

En este trabajo se detalla una metodología para el reconocimiento de objetos utilizando una cámara GoPro Hero3 simulando un ambiente de Robocup. Para lograr el objetivo se desarrollaron los módulos necesarios para que la cámara se localice e interprete los objetos dentro del terreno del juego y de acuerdo a las señales obtenidas dar recomendaciones mediante un sistema de reglas.

En la primera etapa se capturó un conjunto de imágenes de cada uno de los siguientes objetos, portería azul, portería amarilla, poste azul, poste amarillo y pelota, para entrenar una Red Neuronal Convolucional y poder reconocerlos. Posteriormente se realizaron pruebas con cada objeto para obtener un porcentaje de similitud, de una muestra de 10 fotografías por objeto se consiguió una semejanza del 70% al 90%, lo cual representa un porcentaje suficiente para el sistema considerando que es un ambiente cerrado.

En una segunda etapa la cámara captura imágenes, que son enviadas a la PC para su reconocimiento a través de la Red Neuronal, entrenada en la primera etapa, donde se clasifican y se ofrece un porcentaje de semejanza, a través del cual se generan recomendaciones, como son: caminar, patear, girar o levantarse. En un futuro estos resultados podrían enviarse a Robot que realice las acciones necesarias de acuerdo a las necesidades de un partido.

El sistema implementado contiene en una interfaz gráfica donde el usuario especifica la portería en la que se debe anotar.

La construcción del Sistema fue desarrollada en lenguaje de programación Python y bajo la plataforma Windows 10.

TABLA DE CONTENIDOS

RESUMEN ____ *3*

TABLA DE CONTENIDOS ____ *4*

1.INTRODUCCIÓN ____ *5*

2. ANTECEDENTES ____ *6*

2.1. Referencias internas. ____ **6**

2.2. Referencias externas. ____ **6**

3. JUSTIFICACIÓN ____ *7*

4. OBJETIVO ____ *8*

4.1. Objetivos particulares: ____ **8**

5. MARCO TEÓRICO ____ *9*

6. DESARROLLO DEL PROYECTO ____ *12*

6.1. Entorno: ____ **12**

6.1.1. Campo de Juego ____ 12

6.2. Módulo de Reconocimiento. ____ **14**

6.2.1. Adquisición. ____ 15

6.2.2. Fase de entrenamiento. ____ 15

6.2.3. Captura de Imagen. ____ 18

6.2.4. Reconocimiento del Objeto. ____ 19

6.2.5. Resultado. ____ 20

6.3. Módulo de Comunicación ____ **22**

6.3.1. Indicación de tomar foto ____ 22

6.3.2. Captura Fotografía ____ 23

6.3.3. Descarga la última foto tomada por la GoPro Hero 3 ____ 23

6.3.4. Interpreta la imagen ____ 23

6.4. Módulo de Localización ____ **24**

6.5. Módulo de toma de decisiones. ____ **26**

7. RESULTADOS ____ *27*

8. CONCLUSIONES ____ *31*

9. BIBLIOGRAFÍA ____ *32*

10. APÉNDICES ____ *33*

Código fuente ____ **33**

Programa principal ____ 33

Script de Evaluación ____ 37

1.INTRODUCCIÓN

El desarrollo de la robótica ha tenido gran auge y su avance es considerable con las nuevas tecnologías. En la actualidad podemos encontrar una infinidad de robots tanto en el área industrial como en la comercial y sus aplicaciones son variadas. Existen diversos proyectos donde se tratan de resolver problemas muy específicos, en esta ocasión nos referiremos solo a "RoboCup"[1][1] que es un proyecto internacional para promover competencias integradas por robots autónomos, investigación y educación sobre inteligencia artificial.

La organización RoboCup promueve la ciencia y la tecnología con los partidos de fútbol de robots y agentes de software, dentro de la categoría RoboCup de fútbol los objetivos se refieren a la investigación cooperativa multi-robots y sistemas multi-agente totalmente autónomos en entornos dinámicos de confrontación.

Existen 6 ligas en las que se pueden participar, la categoría de interés en este momento es Fútbol Liga Humanoide, sus objetivos son la confrontación de robots autónomos con un plan de cuerpo y sentidos humanos como un juego de fútbol. Dentro de esta liga los temas de investigación son: correr y patear la pelota, mantener el equilibrio, la percepción visual de la pelota, así como la detección de movimiento de la misma, detección de obstáculos (los otros jugadores), y del terreno, la auto-localización, y el juego en equipo.

En este caso se cuenta con el robot "Bioloid"[2] de Robotics[2] por lo que la parte mecánica y el equilibrio se consideran resueltos. Los temas a tratar en este proyecto son la detección de movimiento de la pelota, percepción visual del área de juego y poder dar recomendaciones sobre la toma de decisiones. Para el diseño del software se toma como referencia las reglas establecidas por Fútbol Liga Humanoide de la organización RoboCup.

[1] 1. Proyecto Internacional funda1997 para promover, a través de competencias integradas por robots autónomos la investigación y educación sobre la Inteligencia Artificial

[2] 2. Robot más completo de Robotics equipado con una serie de sensores

2. ANTECEDENTES

2.1. Referencias internas.

Dentro de la Universidad Autónoma Metropolitana no se encontraron proyectos terminales en donde se involucren a la vez problemas de visión, control y toma de decisiones, sin embargo, si existen proyectos terminales donde se manejan algunas de estas problemáticas de forma aislada como los que se mencionan a continuación:

Puesta en marcha del Robot BIOLOID ROBOTIS, "Humanoide con reconocimiento de objetos [3]. Este proyecto terminal hace uso del robot humanoide BIOLOID para el reconocimiento de objetos en movimiento de color rojo, el robot sigue al objeto hasta que se encuentre a cierta distancia de él. Para determinar la distancia utiliza un sensor ultrasónico2. A diferencia de este proyecto con el proyecto a desarrollar es reconocer más de un objeto a la vez, así como su respectivo color y forma. El robot determinara la distancia al objeto a seguir usando solo las imágenes captadas por la cámara del robot, es decir, no se usará ningún otro tipo de sensor.

Clasificador de objetos en banda infinita por medio de procesamiento digital de imágenes [4]. En este proyecto se clasifican los objetos por su perfil o forma utilizando procesamiento de imágenes a escalas de blanco y negro. A diferencia de este, el proyecto a desarrollar utilizara el procesamiento de imágenes en RGB3 y no en una banda infinita sino en un entorno delimitado con dimensiones específicas.

Clasificación de Objetos en un Sistema de Posicionamiento Automático mediante Visión Robótica Activa [5]. En este proyecto se desarrolló un sistema para la clasificación y reconocimiento de objetos de posicionamiento automático mediante visión robótica activa mediante una cámara acoplada a un robot.

2.2. Referencias externas.

Como ya se mencionó anteriormente Robocup es un proyecto internacional para promover competencias integradas por robots autónomos, por lo que es muy conocido y tiene varios participantes en distintos países.

Control por comandos de voz de un robot khepera ii para la manipulación de objetos [6]. Esta tesis aplica técnicas de procesamiento digital de imágenes para el reconocimiento de objetos. El proyecto que se propone no utilizara comandos de voz para dar órdenes al robot.

Desarrollo de un Sistema de Visión para la Localización y Navegación de Robots Humanoides [7]. Esta tesis muestra el desarrollo y los resultados obtenidos de un sistema de visión artificial que proporciona los elementos necesarios para que robots humanoides

sean capaces de localizarse y desplazarse a diferentes lugares dentro de un ambiente estructurado.

Control de un robot móvil aplicando control difuso y visión artificial desarrollado en labview [8]. Esta tesis muestra cómo controlar el prototipo de un robot móvil aplicando la teoría de control difuso en conjunto con una máquina de estados, y el reconocimiento de patrones que forma parte del campo de la visión artificial en un entorno de programación LabVIEW.

3. JUSTIFICACIÓN

Este proyecto tiene como objetivo la detección, segmentación y localización de ciertos objetos en imágenes por medio de la cámara GoProHero 3 para así dotar de visión a un robot que pueda competir en un futuro en la liga RoboCup. Un módulo de visión es de suma importancia ya que de los resultados y la velocidad con los que se obtengan se podrán tomar mejores decisiones durante un partido.

4. OBJETIVO

Diseñar y poner en operación el módulo de Visión de un Robot para el tratamiento y análisis de imágenes con el fin de poder interpretar un escenario dinámico y tomar decisiones sobre las acciones que debe realizar durante un partido de futbol.

4.1. Objetivos particulares:

- Diseñar y poner en operación el módulo que permita el reconocimiento del área de juego y de los objetos que lo conforman.
    - Se logró manejando una Red Neuronal Convolucional utilizando la librería de TensorFlow para Python

- Diseñar y poner en operación el módulo de comunicación que permita la comunicación entre la cámara GoPro Hero3 y el PC
    - Se logró mediante la utilización de la API para Python para conectar cámaras vía WIFI goProCam

- Diseñar y poner en operación el módulo de localización que permita determinar la ubicación de la cámara dentro del terreno de juego.
    - A partir de los resultados obtenidos por el módulo de reconocimiento se generaron las señales de salida estimando la posición de la cámara de acuerdo al porcentaje de reconocimiento de cada objeto alcanzado en el rango de visión de la cámara.

- Diseñar y poner en operación el módulo de toma de decisiones
    - Con la información de los módulos de localización y reconocimiento se crearon el conjunto de reglas que permite dar una recomendación. Primero se introduce al sistema la portería que se tiene que defender y a partir de ahí se generan las recomendaciones como: buscar pelota, caminar, patear, levantarse y girar.

5. MARCO TEÓRICO

El presente trabajo analiza el procesamiento digital de imágenes aplicado específicamente a un ambiente dinámico y cerrado. Es preciso aclarar algunos conceptos:

Imagen: Una imagen digital es una proyección bidimensional de una escena tridimensional, definida matemáticamente como una función f(x,y) donde x y y son las coordenadas espaciales y el valor de f representa la intensidad en dichas coordenadas. Las imágenes binarias o en escala de gris solo requieren de una función f. En cambio, una imagen a color necesita de tres funciones independientes f1, f2 y f3, en las cuales se representa la intensidad del pixel para cada componente. La resolución de una imagen se define como el número de filas y columnas que posee la imagen. En la Figura 1 la resolución de la imagen es NxM pixels. Un pixel, por su abreviatura en ingles *picture elements* es el elemento más pequeño que representa una imagen. Si el valor de la intensidad de color del pixel se representa con un bit, el rango de valores será de 0 a 1 y se le denomina imagen binaria. Para una imagen en escala de grises, el pixel se representa por un dígito de ocho bits, por lo tanto, el rango de valores es de 0 a 255, con 0 correspondiendo al negro y el 255 al blanco. Para las imágenes a color basadas en el espacio RGB[3][1] se presentan con la siguiente expresión: f(i,j) = [R,G,B]T donde cada capa de color se cuantifica por un entero de ocho bits, es decir, el rango de colores posibles representado por un pixel es de 0 a 2^8x2^8x2^8 [9].

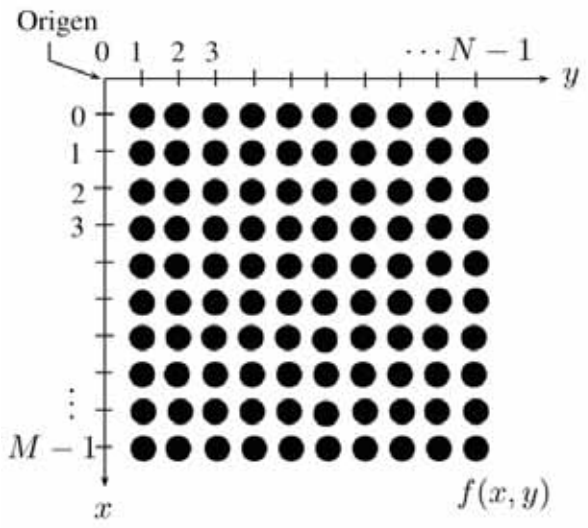


Figura 1. Representación de una imagen de NxM

Procesamiento Digital de Imágenes: El término procesamiento digital de imágenes versa sobre la manipulación y análisis de imágenes por computadora. Hace referencia a

[3] **RGB** (sigla en inglés de red, green, blue, en español «rojo, verde y azul»)

los algoritmos de computación que convierten una imagen digital adquirida en otra de mayor relevancia. El procesamiento digital de imágenes aparece tardíamente en la historia de la computación, ya que antes de pensar en ello, había que desarrollar el hardware que permitieran hacerlo. Por otro lado, los algoritmos y las técnicas de optimización que han tenido que desarrollarse para el procesamiento digital de imágenes son muy sofisticados y elaborados [10].

Inteligencia Artificial: El estudio de cómo programar computadoras que posean la facultad de hacer aquello que la mente humana puede realizar [11].

Red Neuronal: Aquellas redes en las que existen elementos procesadores de información de cuyas interacciones locales depende el comportamiento del conjunto del sistema. Tratan de emular el cerebro humano, caracterizado por el aprendizaje a través de la experiencia y la extracción de conocimiento genérico a partir de un conjunto de datos. Estos sistemas imitan esquemáticamente la estructura neuronal del cerebro, a través un programa de computadora (simulación), mediante su modelado a través de estructuras de procesamiento con cierta capacidad de cálculo paralelo (emulación), o mediante la construcción física de sistemas cuya arquitectura se aproxima a la estructura de la Red Neuronal biológica (implementación hardware RNAs) [12].

Redes neuronales convolucionales: Las redes neuronales convolucionales son muy similares a las redes neuronales ordinarias como el perceptron multicapa; se componen de neuronas que tienen pesos y sesgos que pueden aprender. Cada neurona recibe algunas entradas, realiza un producto escalar y luego aplica una función de activación. Al igual que en el perceptron multicapa también se tiene una función de pérdida o costo sobre la última capa, la cual estará totalmente conectada. Lo que diferencia a las redes neuronales convolucionales es que suponen explícitamente que las entradas son imágenes, lo que nos permite codificar ciertas propiedades en la arquitectura; permitiendo ganar en eficiencia y reducir la cantidad de parámetros en la red.

En general, las redes neuronales convolucionales están construidas con una estructura que con tres tipos distintos de capas: [16]

- Una capa convolucional, que es la de entrada y le da el nombre a la red.
- Una capa de reducción o de pooling, la cual va a reducir la cantidad de parámetros al quedarse con las características más comunes.
- Una capa clasificadora totalmente conectada, la cual da el resultado final de la red.

TensorFlow: Librería de Software libre de código abierto para computación numérica de alto rendimiento. Su arquitectura flexible permite una fácil implementación de computación en una variedad de plataformas. Desarrollado inicialmente por investigadores e ingenieros de Google cuenta con un sólido respaldo para el aprendizaje automático y el aprendizaje en profundidad [14].

Se recurrió a la tecnología de TensorFlow como librería de aplicación porque existe una gran cantidad de repositorios y gran cantidad de información desde que Google liberó el código en 2017, además tiene implementación en Python lo cual resultó conveniente porque el proyecto se realizó en ese lenguaje de programación. Resulta ser una herramienta muy útil basada en tecnologías abiertas.

Para el proyecto se utilizó el tutorial *Image Recognition*[14] el cual muestra cómo aplicar el modelo convolucional en aprendizaje profundo para reconocimiento de imágenes.

Cámara GoProHero 3: La cámara de acción **GoPro** HERO **3** como se muestra en la Figura 2 es una cámara con la cual se puede capturar fotos y videos, cuenta con conexión inalámbrica [13].

Algunas de las características de la cámara son:

Captura de vídeo panorámica:	**Yes**
Tipo sensor óptico:	CMOS
Conexión inalámbrica:	Wireless LAN
Formato vídeo digital:	H.264
Formato de grabación de imágenes:	JPEG
Interfaces proporcionadas:	HDMI, composite video/audio
Formato de grabación de imágenes:	JPEG

Figura 2. Cámara GoProHero 3 [13]

Unofficial GoPro API Library for Python: Librería de Código abierto para conectar vía WIFI cámaras GoPro [15].

Se opto por esta API ya que contiene gran variedad de funciones predeterminadas en Python para poder manipular la cámara de forma sencilla, podemos encontrar diversas funciones para integrar al proyecto como la conectividad WIFI, tomar una foto, etc. La implementación resulta sencilla para aprovechar los beneficios cuando se integra un proyecto.

6. DESARROLLO DEL PROYECTO

En este capítulo se describe cada una de las etapas desarrolladas del módulo de visión para que un robot sea capaz de localizarse dentro del área de juego. En la figura 3 se muestra la interacción de cada uno de los módulos que conforman el sistema de visión.

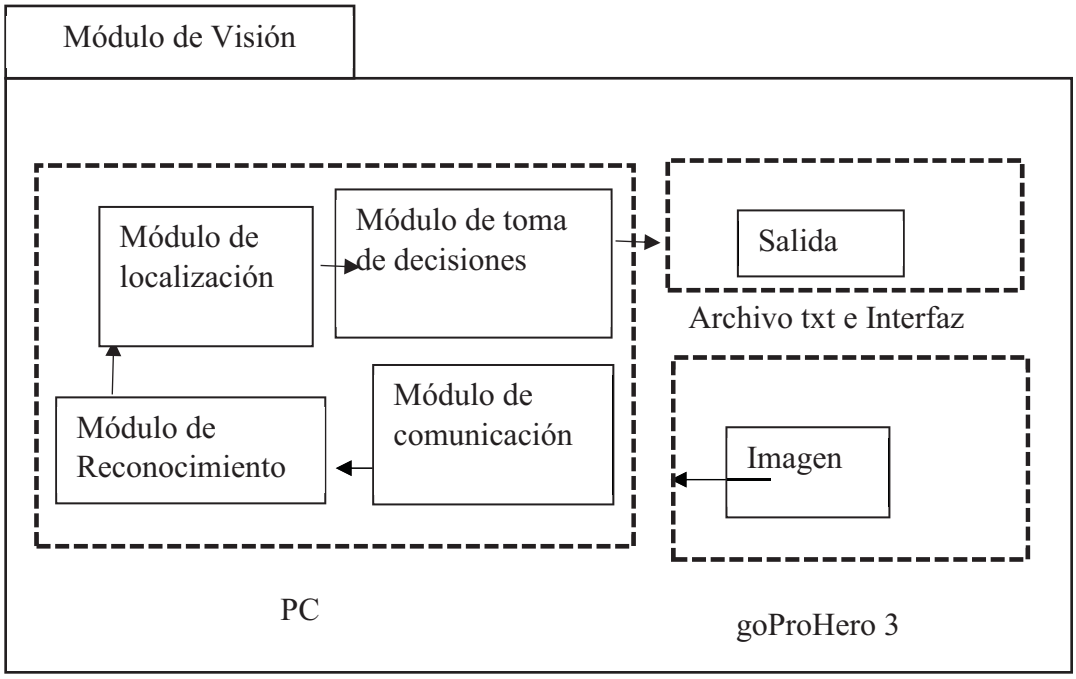


Figura 3. Panorama general del módulo de visión

El ambiente de trabajo del presente proyecto se basa en las reglas de Robocup [1] categoría KidSize, las cuales se cambian o actualizan cada año. Es importante señalar que en este proyecto se han considerado las reglas del año 2011, ya que posiblemente a la publicación de esta tesis las reglas ya hayan cambiado.

6.1. Entorno:

En las siguientes secciones se detallan los elementos de Robocup categoría KidSize que influyen en el diseño y la implementación de los algoritmos y de los módulos previamente mencionados.

6.1.1. Campo de Juego

La competencia tiene lugar en una superficie rectangular con dos porterías, dos postes de ubicación y líneas de color blanco para delimitar la región de juego, como se muestra en la Figura 4. Las dimensiones se pueden observar en la Tabla 1. En la Figura 5 se muestra la recreación del ambiente de trabajo. La superficie es de color verde, el suelo debe ser

plano y horizontal, como se puede apreciar se colocaron paredes de color blanco en las orillas, con la intención de obtener el menor ruido posible en las imágenes que capta la cámara.

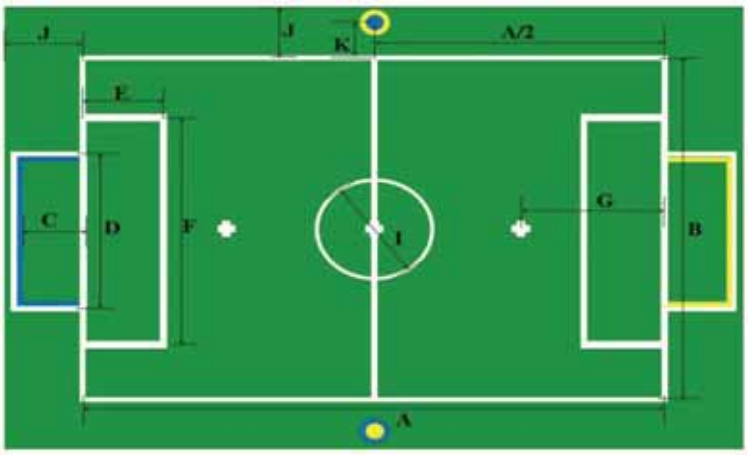


Figura 4. Campo de Juego

	kIdSize	Cm
A	Largo	450
B	Ancho	300
C	Profundidad de la portería	50
D	Ancho de la portería	150
E	Largo del área de Gol	50
F	Ancho del área de Gol	190
G	Distancia de penalti	120
I	Diámetro del circuíto central	90
J	Ancho entre borde y franja	60
K	Distancia entre el poste y el campo	40

Tabla 1. Dimensiones del campo de juego

Figura 5. Recreación del ambiente de trabajo construido para que el robot pueda navegar.

6.2. Módulo de Reconocimiento.

En este módulo se desarrolla el sistema de reconocimiento que permite identificar objetos dentro del campo de juego, para este se utilizó TensorFlow un proyecto de software libre desarrollado por Google para aprendizaje automático a través de un rango de tareas. En la Figura 6 se muestran las fases que se utilizaron para el reconocimiento de los objetos.

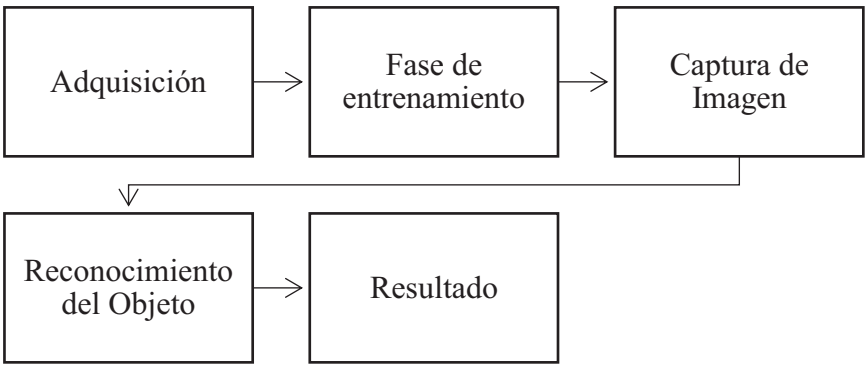


Figura 6. Fases del Módulo de Reconocimiento de imágenes

6.2.1. Adquisición.

Una vez definido el espacio de trabajo se procedió a capturar imágenes usando la cámara con las siguientes características:

1. Resolución: 5 MP
2. Formato: JPG
3. Dimensiones: 2592 x 1944

Las imágenes se tomaron bajo condiciones ambientales normales y considerando una perspectiva del robot.

6.2.2. Fase de entrenamiento.

Se tomaron una cantidad considerable de imágenes de cada objeto del terreno de juego (Porterías, postes, pelota, adversario) con la finalidad de que la Red Neuronal definiera cada objeto de interés y pudiera considerar los objetos claves de interés. En la Figura 7 se muestran los puntos desde donde se consideró conveniente capturar fotografías, desde cada uno de estos puntos de tomaron aproximadamente 20 fotografías distintas, con pequeñas variaciones de ángulos, distancias y enfoques.

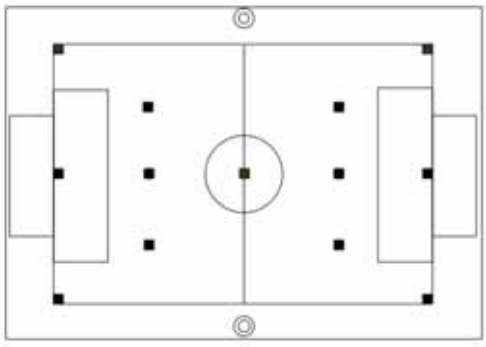


Figura 7. Posiciones Clave desde donde se tomaron fotografías

En la Figura 8 se presentan varias muestras de fotografías que fueron tomadas, donde pueden observarse variaciones de las fotografías.

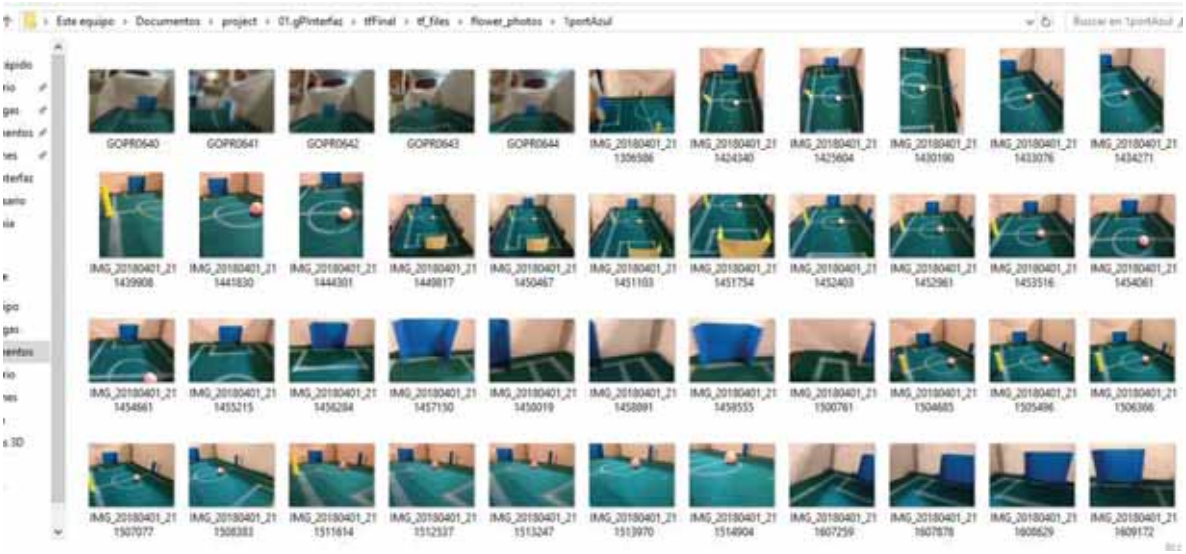


Figura 8. Imágenes de muestra para la fase de entrenamiento

Una vez obtenidas las imágenes de cada objeto se procedió a entrenar la Red Neuronal para poder clasificarlas y utilizarlas dentro del sistema de visión, el código de la Red Neuronal debe estar guardado en el directorio del proyecto y después seguir los siguientes pasos:

1. Crear una carpeta con el nombre del objeto a identificar que contiene las imágenes con las que se entrenara la Red Neuronal. Los nombres de las carpetas son los siguientes:
    a. Portería Azul
    b. Portería Amarilla
    c. Pelota
    d. Adversario
    e. Poste Azul
    f. Poste Amarillo
2. Abrir una consola y dirigirnos al directorio del proyecto, como se muestra en la Figura 9

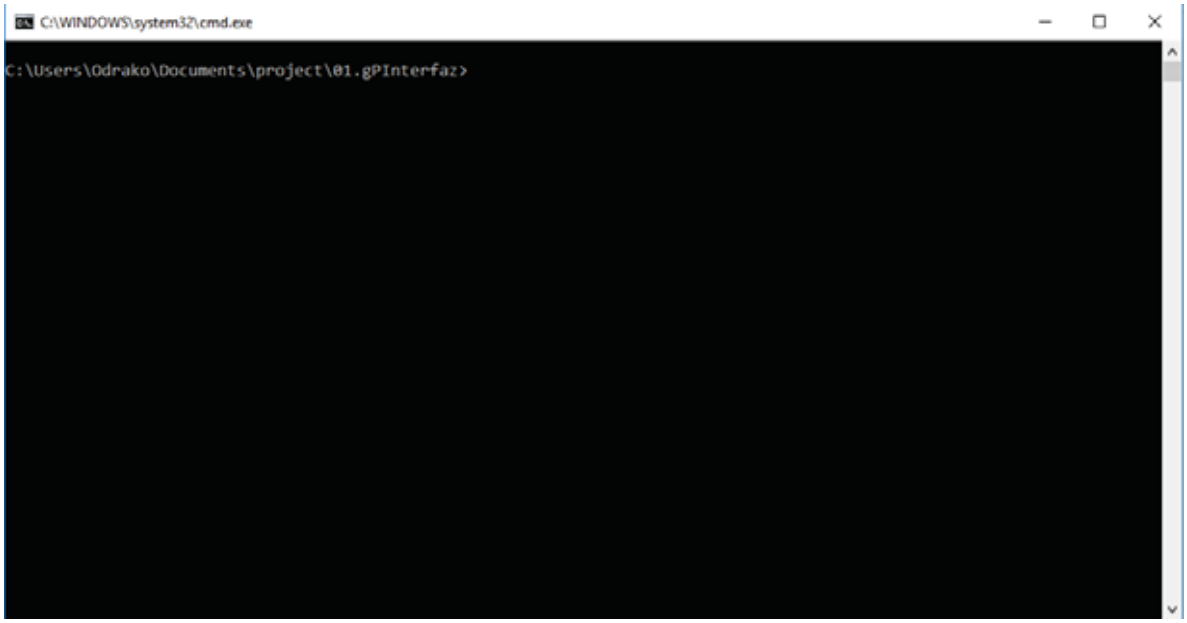


Figura 9. Directorio del proyecto en la consola de Windows 10

3. Teclear el siguiente comando de Python para que comience el entrenamiento de la Red Neuronal,

   python -m scripts.retrain *// Este comando inicia el script, extrae la información de las carpetas y subcarpetas cada una llena de imágenes. El nombre de las carpetas es importante porque tomará los nombres que se le dieron a las carpetas como etiquetas.*

   --bottleneck_dir=tf_files/bottlenecks *// En este directorio se guarda los archivos de bottlenecks que en la fase de entrenamiento se refiere a una capa previa antes de la capa de salida y en el mismo se guardan los resultados que se vuelven a reutilizar si se ejecutara el script nuevamente*

   --how_many_training_steps=500 *// Se consideran 500 interacciones*

   --model_dir=tf_files/models/ *//Se indica el directorio donde se guardara el modelo*

   --summaries_dir=tf_files/training_summaries/mobilenet_0.50_224 *//Se indica donde guardar el modelo que se utiliza, en esta ocasión utilizamos el modelo mobilenet en su versión 5 que es un modelo pequeño que corre significativamente más rápido*

   --output_graph=tf_files/retrained_graph.pb *// se genera el archivo de salida del grafo*

   --output_labels=tf_files/retrained_labels.txt *// Este comando solo genera un archivo de texto con el nombre de las etiquetas*

   --architecture=mobilenet_0.50_224 *// aquí se indica el modelo y el tamaño de imágenes, modelo: mobilenet 5 y el tamaño de la imagen 224*

--image_dir=tf_files/photos *// Aquí se indica el directorio donde se guardan las fotos de los objetos*

4. Verificar que el entrenamiento haya sido exitoso. En la Figura 10 se muestra el entrenamiento final de la última capa de la red. Mientras se entrena se mostrarán una serie de salidas escalonadas con la siguiente información:
   - **Accuracy:** Porcentaje de las imágenes utilizadas del conjunto de imágenes para el entrenamiento y fueron etiquetadas con la clase correcta:
   - **Validation accuracy:** Porcentaje de imágenes correctamente etiquetadas seleccionadas de un grupo al azar del conjunto de imágenes.
   - **Cross entropy:** Función de pérdida que da una idea de qué tan bien está progresando el proceso de aprendizaje.

   Al final del entrenamiento tendremos los siguientes resultados:
   Final Test Accuracy**:** Una vez completados todos los pasos de entrenamiento se ejecuta la evaluación de precisión, esta evaluación de prueba proporciona la mejor estimación de como el modelo entrenado realizara la tarea de clasificación.

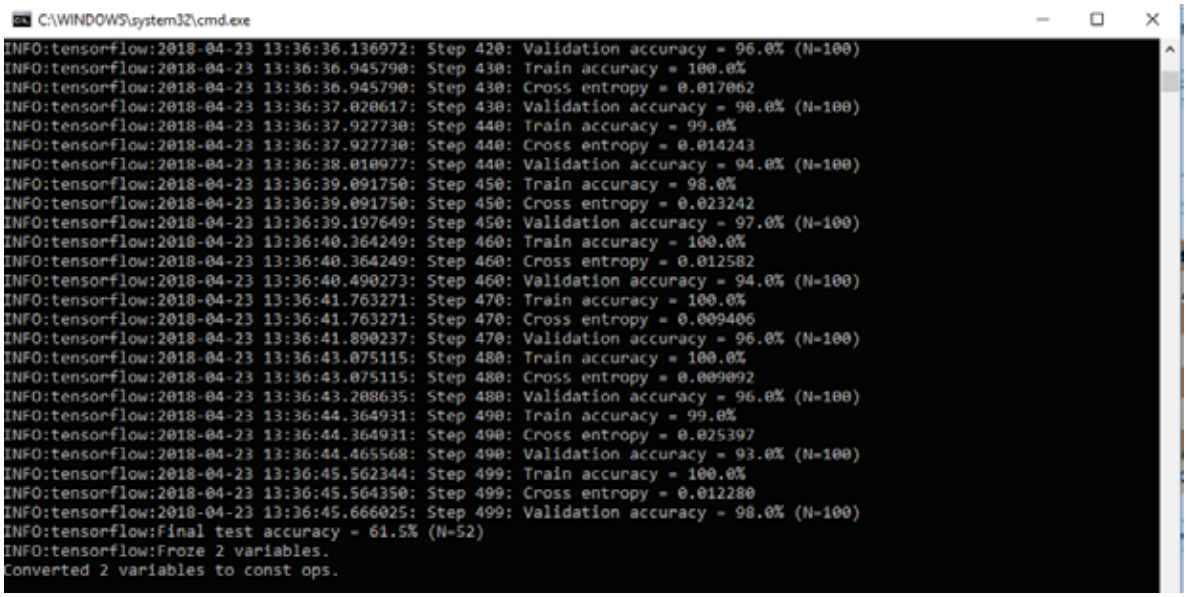


Figura 10. Iteraciones realizadas por la Red Neuronal

6.2.3. Captura de Imagen.

Para que la cámara capturare fotografías de manera inalámbrica debe utilizarse el modo WIFI de la cámara GoProHero3 y al conectarla a la PC debe aparecer como se muestra en la Figura 11.

Figura 11. GoPro Hero3 conectada vía WIFI con la PC

En este caso la cámara se llama odkGoPro y para estar seguros que está conectada debajo del nombre de la cámara aparecerá **Sin internet, segura,** sin este paso no será posible capturar imágenes.

Se utiliza Unofficial GoPro API Library for Python [15] para realizar el módulo de comunicación, para comenzar se importan las siguientes librerías en Python:

from goprocam import GoProCamera
from goprocam import constants

Con el siguiente código se inicializa la cámara y se parametriza en modo fotografía.

gpCam = GoProCamera.GoPro() //Inicializamos el objeto cámara
gpCam.mode(constants.Mode.PhotoMode) //Parametrizamos en modo fotografía

Se considero un retraso de 10seg entre foto y foto ya que la calidad de la imagen es alta (5mp) y debe considerarse el tiempo en que la Red Neuronal procesa la imagen.

6.2.4. Reconocimiento del Objeto.

Teniendo la imagen capturada se procede a reconocerla con ayuda de la Red Neuronal, esto lo logramos en Python con ayuda de un subproceso, que recibe tres argumentos:

1. ***scripts.label_image*** *// Este script contiene las etiquetas de los objetos previamente entrenados*
2. ***--graph=tf_files****/retrained_graph.pb // El árbol de entrenamiento*
3. ***–image=ruta*** *// Imagen que se desea reconocer*

Se utiliza la clase subproceso de Python para lo cual previamente se debe importar la librería:

- ***import*** *subprocess*

con los argumentos quedaría de la siguiente manera:

```
p = subprocess.run
(["python", "-m", "scripts.label_image" ,"--graph=tf_files/retrained_graph.pb","--image="+ruta],
        stdout=subprocess.PIPE,
        shell = True,
        universal_newlines = True,
        encoding = "cp850")
    out = p.stdout
```

El resultado se guarda en la variable **out** la cual contiene cada una de las etiquetas de la Red Neuronal con un porcentaje de coincidencia del objeto. Se consideró para el sistema un porcentaje de coincidencia de un 80% y el ejemplo de la salida lo podemos visualizar en la Figura 12, la cual también arroja el tiempo que se tarda en realizar la evaluación.

```
Evaluation time (1-image): 0.309s

2portamarilla           0.935803
1portazul               0.0583246
4adversario             0.00584997
6postamarillo           1.53589e-05
5postazul               6.52351e-06
```

Figura 12. Ejemplo del resultado del reconocimiento de la Red Neuronal

6.2.5. Resultado.

Los resultados se pueden visualizar en tiempo real por medio de la interfaz gráfica Figura 13. la cual se construyó utilizando Pygame[4] y la salida de la evaluación en un archivo de texto que guarda todas las secuencias de imágenes y de ahí podemos utilizar los resultados para la toma de decisiones.

[4] Módulos del lenguaje Python que permiten la creación de videojuegos en dos dimensiones.

Figura. 13. Interfaz Grafica

En la Figura 13. se muestran los elementos que conforman la interfaz gráfica a continuación una breve explicación.

1. **Campo de juego:** En la Figura 14 se muestran los posibles estados los cuales podría tomar de acuerdo al resultado del reconocimiento.

Figura 14. Estados de la cancha

2. **Contador:** En pantalla igual se muestra el tiempo de retraso entre cada toma de fotografía por parte de la cámara

3. **Imagen en tiempo real.:** Aquí podemos visualizar la foto tal cual tomo la cámara.

4. **Aquí el resultado del reconocimiento**

6.3. Módulo de Comunicación

El sistema cuenta con la capacidad de conectividad inalámbrica el cual supone que están conectadas la cámara y la PC sin necesidad de cables, cubriendo las necesidades del proyecto. La cámara GoProHero3 cuenta con conectividad WIFI, esta tecnología permite solucionar el problema como se muestra en la Figura 15 donde se muestra el ciclo que realiza el sistema.

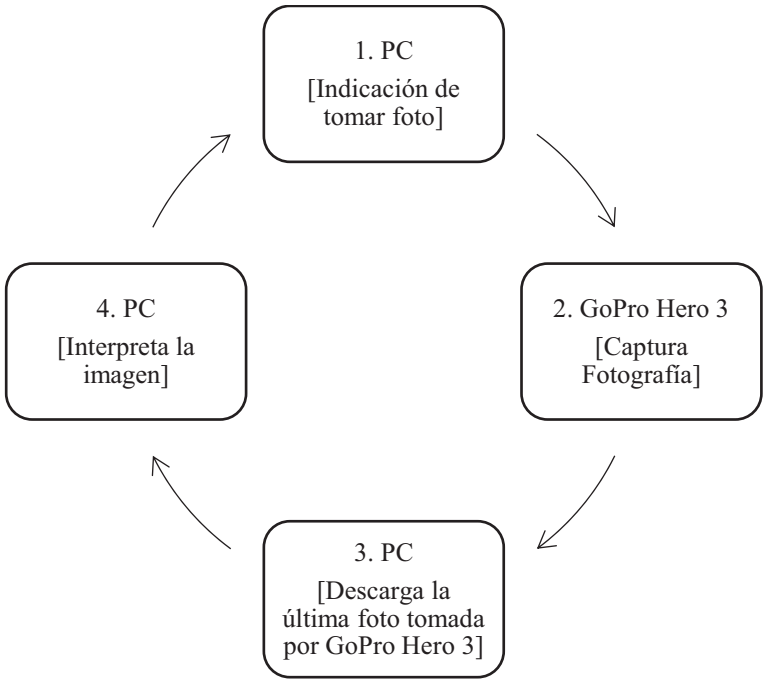


Figura 15. Fases del Módulo de comunicación

6.3.1. Indicación de tomar foto

Como el proyecto se enfoca únicamente en dotar al robot de visión se consideraron dos tiempos de retraso para realizar pruebas de 10s y 20s. El sistema manda la señal de tomar foto en el tiempo de intervalo que se haya indicado previamente y continua hasta cerrar la aplicación, se creó la siguiente función que toma la fotografía y regresa el nombre del archivo .JPG.

def foto(): //Nombre de la función

gpCam.shutter(constants.start) // Parámetro para indicar a la cámara tomar la fotografía

gpCam.downloadLastMedia() // Son esta función descargamos a la PC la última fotografía

return gpCam.getMedia() // Se regresa el nombre del archivo

La función está dentro de un ciclo de 10seg tomado en cuenta para el proyecto.

6.3.2. Captura Fotografía

La cámara captura la fotografía al recibir la indicación por la PC y la guarda en la memoria interna. Las características de la cámara ayudan a capturar imágenes de gran calidad y dado que está diseñada para poder capturar imágenes en ambientes difíciles es de gran ayuda para este proyecto ya que es un ambiente cerrado.

Al utilizar Unofficial GoPro API Library for Python [15] contamos con la siguiente función para tomar una fotografía en Python

- *gpCam.shutter(constants.start)*

A la función se le debe pasar el argumento *constants.start* para capturar la fotografía de la Gopro Hero3.

6.3.3. Descarga la última foto tomada por la GoPro Hero 3

Una vez capturada la imagen es necesario descargarla a la PC para su procesamiento, se manda la señal a la cámara por medio de la siguiente función:

- *gpCam.downloadLastMedia()*

Así se consigue descargar la última fotografía capturada a la PC, la función por default descarga la imagen en el directorio donde se encuentra guardado el proyecto y utilizando la siguiente función se consigue obtener el nombre con el que se descargó:

- *gpCam.getMedia()*

6.3.4. Interpreta la imagen

Con la imagen descargada en el PC y conociendo el nombre se envía la señal al módulo de Reconocimiento explicado previamente en el apartado 6.2 el cual se encargar de interpretar la imagen y generar los resultados y con ello se cierra el ciclo. Para conseguirlo se creó una función llamada *redN()* la cual se encargara de mandar los resultados a la interfaz gráfica y a un archivo de texto y retorna el resultado.

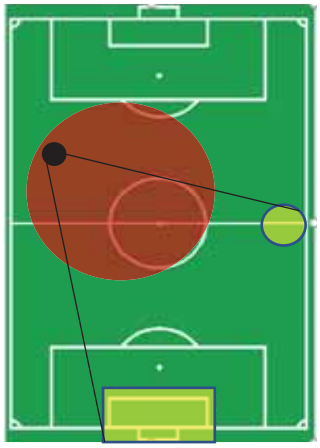
def redN(): //Nombre de la función

ruta="100GOPRO-"+filename // guardamos el nombre del archivo y su dirección

```
p = subprocess.run(["python", "-m", "scripts.label_image" ,"--
graph=tf_files/retrained_graph.pb","--image="+ruta],// subproceso para la
Red Neuronal
        stdout=subprocess.PIPE,
        shell = True,
        universal_newlines = True,
        encoding = "cp850")
out = p.stdout // Termina subproceso
print(out)//Salida en consola del resultado
 with open("salida.txt", 'w') as f:  //Se crea el archivo de texto con la salida
                    f.write(out)
return out
```

6.4. Módulo de Localización

Se propone ubicar el robot dentro del campo de juego por el método de triangulación, el cual requiere de conocer la posición de dos objetos fijos para estimar la posición de un tercero. Dentro del ambiente de Robocup se conocen cuatro objetos fijos que son las dos porterías y los dos postes de ubicación. Los objetos que cambian de posición son el balón y los robots del equipo contrario, por lo que para utilizar este método es necesario que el robot identifique al menos dos objetos fijos que se determinarán de acuerdo a los resultados que arroje el módulo de reconocimiento y a partir de ahí se harán las recomendaciones de acuerdo a una serie de reglas, tomando en cuenta los estados de la cancha, las reglas son las siguientes:



Zona 1: Como se muestra en la figura 16, la cámara percibe dos objetos el poste y la portería amarilla

Zona 2: Como se muestra en la figura 17, la cámara percibe dos objetos el poste y la portería azul

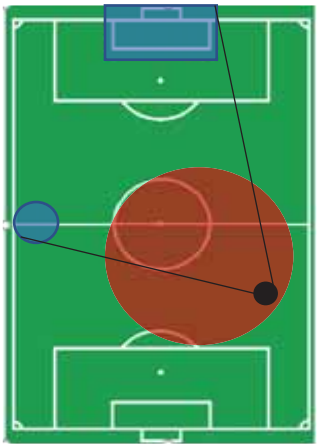
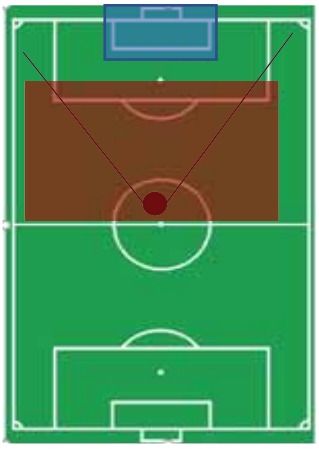


Figura 16. Zona1

Figura 17. Zona 2



Zona 3: Como se muestra en la figura 18, la cámara percibe la portería azul

Zona 4: Como se muestra en la figura 19, la cámara percibe la portería amarilla

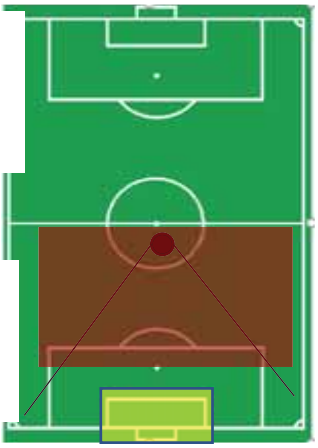
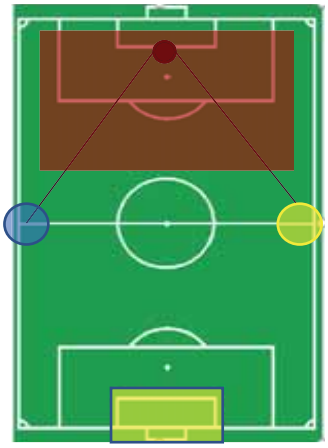


Figura 18. Zona3

Figura 19. Zona 4



Zona 5: Como se muestra en la figura 20, la cámara percibe tres objetos: postes y la portería amarilla

Zona 21: Como se muestra en la figura 19, la cámara percibe tres objetos: ambos postes y la portería azul

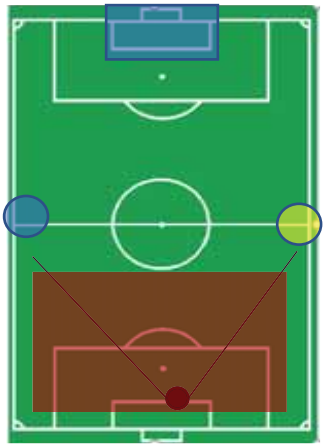


Figura 20. Zona 5

Figura 21. Zona 6

Se dividió la cancha es seis zonas para poder identificar aproximadamente la posición de la cámara y de acuerdo a la zona realizar recomendaciones.

6.5. Módulo de toma de decisiones.

Se propuso la siguiente solución:

Localización: Es el resultado del módulo de localización que arroja una de las seis zonas posibles en las que se encuentra la cámara.

Localizado: Conociendo la zona en la que se encontra la cámara el status localizado se activa.

Las posibles recomendaciones a dar son las siguientes:

- Encontrar balón
- Caminar
- Encontrar portería
- Patear

El algoritmo se muestra la Figura 22.

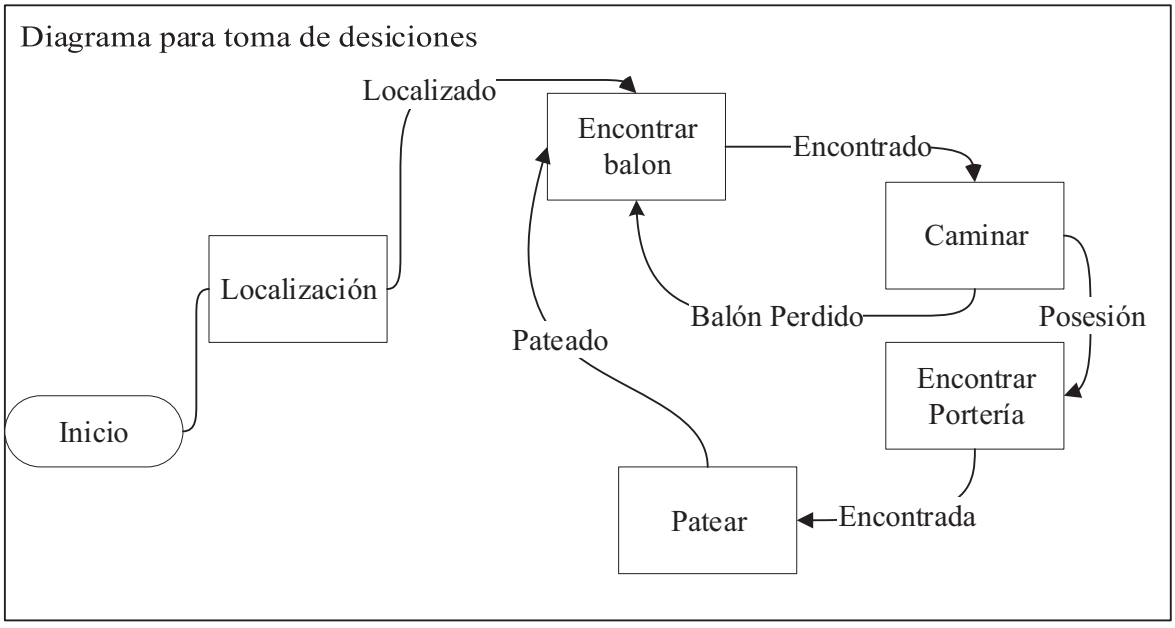


Figura 22. Diagrama de toma de decisiones

El módulo de localización es de gran importancia porque arroja la zona en la que se encuentra y así inicia el algoritmo hasta arrojar posibles recomendaciones. Al inicio de la aplicación se configura con pulsando el número **"1"** si la portería a defender es azul o **"0"** si la portería a defender es la amarilla y de esa manera iniciaremos el sistema.

Para lograrlo los módulos anteriores deben arrojar información certera. Los datos enviados de la computadora hacia el robot deberán ser codificados de tal forma que lo pueda interpretar el Robot, en este proyecto solo se tomara en cuenta la recomendación. Las acciones a realizar y que puede ejecutar el robot son solo cuatro: encontrar balón, encontrar portería, caminar, y patear.

7. RESULTADOS

En este apartado se presentan los resultados obtenidos de la metodología propuesta.

En la Figura 23 vemos la aplicación funcionando y este es el resultado final con la funcionalidad propuesta.

Figura 23. Interfaz Gráfica

En la Figura 23. se muestran los elementos que conforman la interfaz gráfica a continuación una breve explicación.

1. **Campo de juego:** En la Figura 24. Se muestran los posibles estados los cuales podría tomar de acuerdo al resultado del reconocimiento.

Figura 24. Estados de la cancha

2. **Contador:** En pantalla igual se muestra el tiempo de retraso entre cada toma de fotografía por parte de la cámara

3. **Imagen en tiempo real.:** Aquí podemos visualizar la foto tal cual tomo la cámara.

4. **Aquí el resultada del reconocimiento**

En la Figura 25 podemos observar el archivo de texto con los resultados del reconocimiento, la zona donde se encuentra y una recomendación. Cabe destacar que para fines del proyecto y en modo de interpretación encontraremos los resultados en un lenguaje sencillo y fácil de entender, se propone que para futuros proyectos se consideren señales binarias para el envío de información hacia un Robot. Para encontrar la zona y dar una recomendación se toman dos evaluaciones del módulo de reconocimiento.

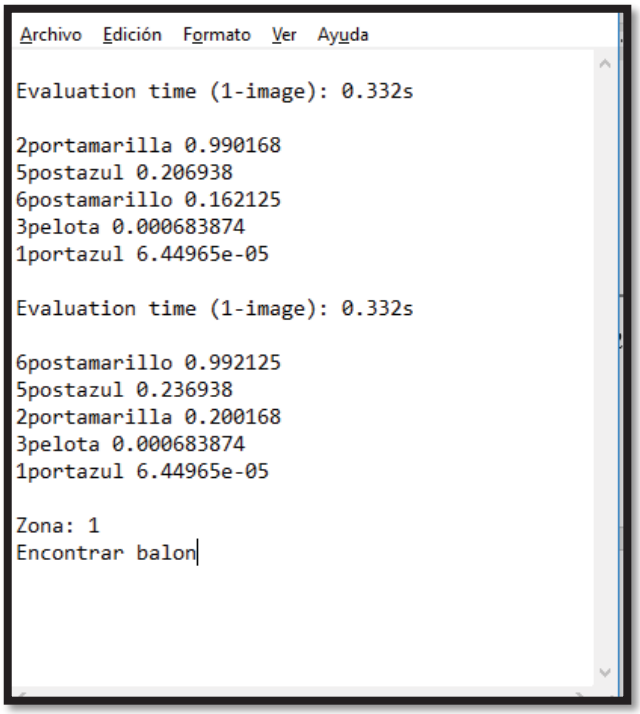


Figura 25. Archivo de Texto

En la Figura 26 podemos visualizar el resultado del módulo de comunicación que descarga los archivos de la cámara con la función *gpCam.downloadLastMedia()*

Figura 26. Imágenes obtenidas por el módulo de comunicación.

Se puede observar que todas las imágenes están formadas por **"100GOPRO-"** ,en el programa se tuvo que concatenar (*fileName= 100GOPRO- & gpCam.getMedia())* esta constante para poder obtener el nombre final del archivo ya que la función *gpCam.getMedia()* únicamente regresa el nombre del archivo después del guion.

En la Figura 27 observamos las etiquetas que utilizó la Red Neuronal para el entrenamiento, así como una colección de imágenes también necesarias por el módulo de reconocimiento.

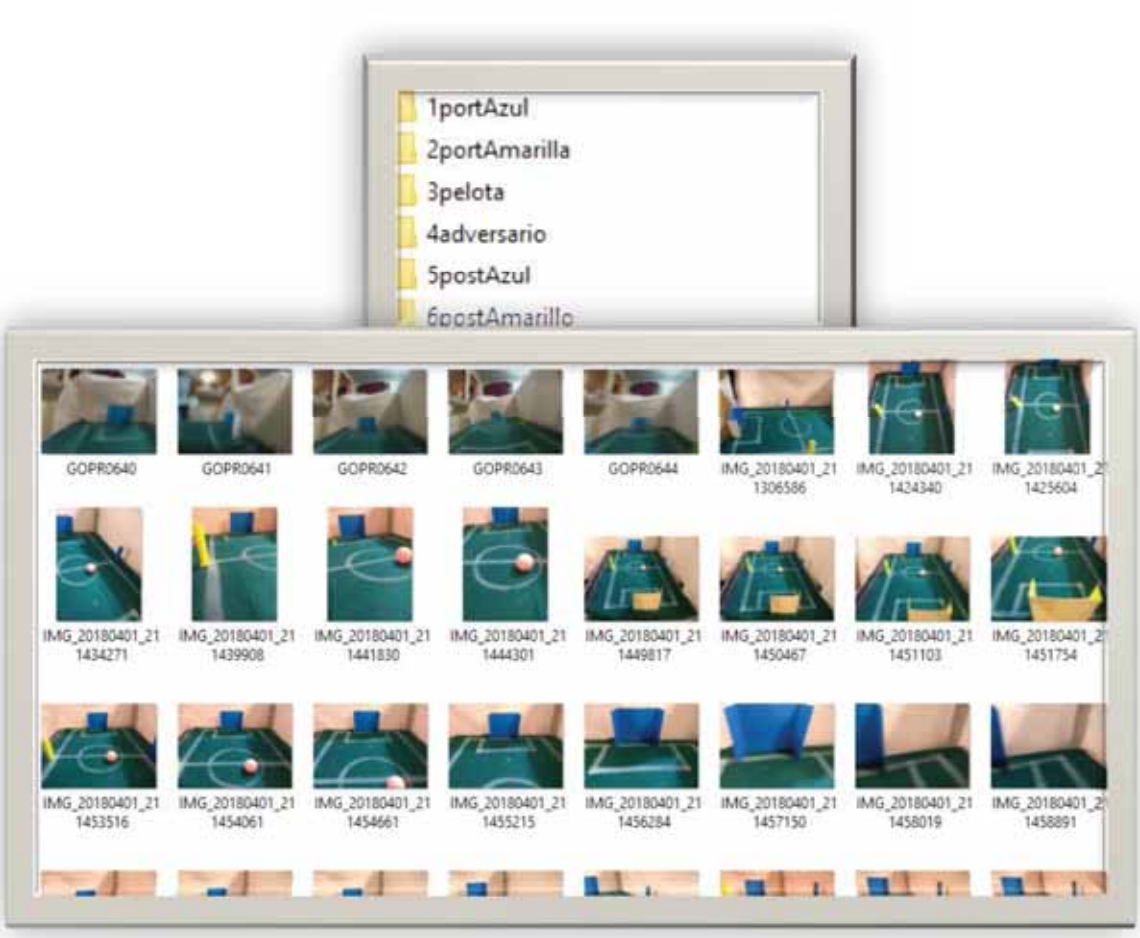


Figura 27. Imágenes y etiquetas de entrenamiento.

Para el proyecto se tomaron 292 fotografías y 6 etiquetas, es una muestra pequeña pero aun así da resultados confiables ya que el proyecto se realiza en un ambiente cerrado y el ruido de las imágenes disminuye considerablemente.

En la tabla 2 se muestra el número total de imágenes utilizadas por cada objeto para el entrenamiento de la Red Neuronal.

Objeto	No. Fotos
Poste Azul	30
Poste Amarillo	31
Portería Amarilla	102
Porteria Azul	85
Pelota	26
Adversario	16

Tabla 2. Fotos tomadas por cada objeto

8. CONCLUSIONES

En este trabajo se describió una metodología que permite reconocer objetos utilizando una cámara GoPro Hero 3 considerando un ambiente dinámico similar al RoboCup. Los objetos de interés son porterías, postes de ubicación y un balón. Se propuso una interfaz gráfica para visualizar los resultados y permita interpretarlos fácilmente por el usuario.

Se propuso el módulo de visión el cual es capaz de capturar una imagen inalámbricamente utilizando la cámara GoProHero 3 lo cual cumple con lo propuesto con el módulo de comunicación, el único inconveniente que se tuvo es el tamaño de la imagen de 5 Mega Pixeles, la cual por el tamaño de la misma limita el tiempo de procesamiento por lo cual se recomendaría utilizar otra cámara que permita resoluciones bajas y poder trabajar con tiempos de procesamiento bajos. Se acoplaron los diferentes módulos logrando un 80% de efectividad en las pruebas realizadas.

Se propuso el módulo de localización mediante zonas ubicando objetos claves que permitieron una aproximación de la ubicación y estos resultados a su vez alimentaran el módulo de toma de decisiones que y se definió los estados posibles por los que un robot eventualmente podría encontrar en una competición.

El proyecto esta acotado considerablemente para participar en una competencia de RoboCup, solo se enfocó en los ojos del Robot, el reto es importante y conveniente para desarrollar trabajos futuros y fomentar la investigación en Redes Neuronales y Robótica.

9. BIBLIOGRAFÍA

[1] RoboCup http://www.robocup.org/ 2018

[2] Robotis http://www.robotis.us/bioloid-1/ 2018

[3] Arturo Herrerías Mata. Puesta en marcha del Robot BIOLOID ROBOTIS, "Humanoide con reconocimiento de objetos" Proyecto terminal, Ingeniería en Electrónica, Universidad Autónoma Metropolitana- Unidad Azcapotzalco 2010.

[4] Díaz Cabrera Fausto Mario. Clasificador de objetos en banda infinita por medio de procesamiento digital de imágenes. Proyecto terminal, Ingeniería en Computación, Universidad Autónoma Metropolitana- Unidad Azcapotzalco 2010.

[5] Eric Miguel González Jiménez, "Clasificación de Objetos en un Sistema de Posicionamiento Automático mediante Visión Robótica Activa" Proyecto terminal, Ingeniería en Computación

[6] Lara Luis Hómer Albert, "Control por comandos de voz de un robot Khepera II para la manipulación de objetos". Tesis para Maestría en Ciencias en Ingeniería de Computo. Instituto Politécnico Nacional.

[7] Erick Rogelio Cruz Hernández," Desarrollo de un Sistema de Visión para la Localización y Navegación de Robots Humanoides". Tesis para Maestro en Ciencias de la ingeniería , Tecnológico de Monterrey.

[8] Hugo Israel Alcaraz Herrera, "Control de un robot móvil aplicando control difuso y visión artificial desarrollado en labview". Tesis, Ingeniería en Computación. Universidad Nacional Autónoma de México.

[9] Gonzalo Pajares Martinsanz "Imágenes Digitales: procesamiento práctico con Java" RA-MA Editorial,2003

[10] Gonzalez, R. C. y R. E. Woods, "Digital Image Processing". PrenticeHall,Inc, 2002.

[11]Francisco Escolano Ruiz, Miguel Ángel Cazorla Quevedo, "Inteligencia Artificial Modelos técnicas y Áreas de aplicación". Mcgraw-hill, 2008

[12] Raquel Flores López, José Miguel Fernández, "Las Redes Neuronales Artificiales", NETBIBLO, 2008

[13] GoPro Hero 3 https://gopro.com/update/hero3 2018

[14] TesnsorFlow https://www.tensorflow.org/ 2018

[15] Unofficial GoPro API Library for Python https://github.com/KonradIT/gopro-py-api 2018

[16] Introducción al Deep Learning https://iaarbook.github.io/deeplearning/ 2018

10. APÉNDICES

Código fuente

Programa principal

```
1.  import pygame
2.  import tkinter
3.  import pygame as pg
4.  from pygame.locals import *
5.  from goprocam import GoProCamera
6.  from goprocam import constants
7.  import os, sys
8.  import subprocess
9.  def main():
10. pg.init()
11. lines=["","","","","","","","",""]
12. pre=.80
13. #portAzul=''
14. #portAma=''
15. postAzul=''
16. postAma=''
17. Pelota=''
18. rival=''
19. screen = pg.display.set_mode((900, 665))
20. pg.display.set_caption('Vision Artificial Rafael Rayon Aldana')
21. font = pg.font.Font(None, 40)
22. myfont = pygame.font.SysFont("monospace", 15)
23. gray = pg.Color('white')
24. blue = pg.Color('dodgerblue')
25. salida=''
26. canchai =
    pg.image.load('C:/Users/Odrako/Documents/project/image/cancha.jpg')
27. cancha1 =
    pg.image.load('C:/Users/Odrako/Documents/project/image/canPortA.jpg')
28. cancha2 =
    pg.image.load('C:/Users/Odrako/Documents/project/image/canPortAma.jpg')
29. cancha3=
    pg.image.load('C:/Users/Odrako/Documents/project/image/canPostAz.jpg')
30. cancha4 =
    pg.image.load('C:/Users/Odrako/Documents/project/image/canPostAma.jpg')
```

31. path="C:/Users/Odrako/Documents/project/01.gPInterfaz/tf - copia/100GOPRO-"
32. clock = pg.time.Clock()
33. timer = 20 # Decrementara el contador en 10
34. i=0
35. dt = 0 # diferencia de tiempo
36. gpCam = GoProCamera.GoPro()
37. gpCam.mode(constants.Mode.PhotoMode)
38. def foto():
39. gpCam.shutter(constants.start)
40. gpCam.downloadLastMedia()
41. return gpCam.getMedia()
42. def cancha(x,y):
43. screen.blit(canchai, (x,y))
44. def canchaPorA(x,y):
45. screen.blit(cancha1, (x,y))
46. def canchaPorAma(x,y):
47. screen.blit(cancha2, (x,y))
48. def canchaPostAz(x,y):
49. screen.blit(cancha3, (x,y))
50. def canchaPostAma(x,y):
51. screen.blit(cancha4, (x,y))
52. def captcha(x,y):
53. screen.blit(pygame.transform.scale(captura, (400, 300)), (x,y))
54. def redN():
55. ruta="100GOPRO-"+filename
56. print("perro",ruta)
57. #p = subprocess.run(["python", "-m", "scripts.label_image" ,"--graph=tf_files/retrained_graph.pb","--image=test4.jpg"],
58. p = subprocess.run(["python", "-m", "scripts.label_image" ,"--graph=tf_files/retrained_graph.pb","--image="+ruta],
    a. stdout=subprocess.PIPE,
    b. shell = True,
    c. universal_newlines = True,
    d. encoding = "cp850")
59. out = p.stdout
60. print(out)
61. with open("salida.txt", 'w') as f:
62. f.write(out)
63. return out
64. done = False
65. while not done:
66. for event in pg.event.get():
67. if event.type == pg.QUIT:

```
        a.  done = True
68. ##=========contador========
69. timer -= dt
70. if timer <= 0:
71. timer = 20  # Reseteamos el timer en 20.
72. foto()#Se llama a la funcion que toma lo foto directo de la camara
73. salida=redN()
74. archivo = open("perro.txt", "r")
75. miFuente = pygame.font.SysFont("monospace", 15)
76. lines=archivo.readlines()
77. print("peroo",filename)
78. #print ("perro: ",portAzul)
79. archivo.close()
80. #=========================
81. screen.fill(gray)
82. #==========Etiquetas=========================
83. txt = font.render(str(round(timer, 2)), True, blue)
84. labelPostAma=myfont.render("Poste Amarillo--------", 1, (0,0,0))
85. #============================================
86. name=gpCam.getMedia()
87. fin=len(gpCam.getMedia())
88. filename=name[fin-12:fin]
89. label = myfont.render(path + filename, 1, (0,0,0))
90. label2 = myfont.render(salida, 1, (0,0,0))
91. evTime= myfont.render(lines[1], 1, (0,0,0))
92. #=========================================
93. k=lines[3].find(" ")
94. newline=lines[3][k+1:len(lines[3])-1]
95. porcentPortA=lines[3][k+1:len(lines[3])-1]
96. if porcentPortA=="":
97. porcentPortA=".00001"
98. portAzul= myfont.render(newline, 1, (0,0,0))
99. labelPortAzul=myfont.render(lines[3][0:k], 1, (0,0,0))
100.        bandera=lines[3][0:1]
101.        #print(bandera)
102.        k=lines[4].find(" ")
103.        newline=lines[4][k+1:len(lines[4])-1]
104.        porcentPortAma=lines[3][k+1:len(lines[3])-1]
105.        if porcentPortAma=="":
106.        #labelPortAzul=""
107.        porcentPortAma=".00001"
108.        PortAma= myfont.render(newline, 1, (0,0,0))
109.        labelPortAma=myfont.render(lines[4][0:k], 1, (0,0,0))
110.        k=lines[5].find(" ")
```

```
111.        newline=lines[5][k+1:len(lines[5])-1]
112.        porcentPelot=lines[3][k+1:len(lines[3])-1]
113.        if porcentPelot=="":
114.        #labelPortAzul=""
115.        porcentPelot=".00001"
116.        Pelota= myfont.render(newline, 1, (0,0,0))
117.        labelPelota=myfont.render(lines[5][0:k], 1, (0,0,0))
118.        k=lines[6].find(" ")
119.        newline=lines[6][k+1:len(lines[6])-1]
120.        porcentAdv=lines[3][k+1:len(lines[3])-1]
121.        if porcentAdv=="":
122.        #labelPortAzul=""
123.        porcentAdv=".00001"
124.        Adv= myfont.render(newline, 1, (0,0,0))
125.        labelAdv=myfont.render(lines[6][0:k], 1, (0,0,0))
126.        k=lines[7].find(" ")
127.        newline=lines[7][k+1:len(lines[7])-1]
128.        porcentPostAz=lines[3][k+1:len(lines[3])-1]
129.        if porcentPostAz=="":
130.        #labelPortAzul=""
131.        porcentPostAz=".00001"
132.        PostAzul= myfont.render(newline, 1, (0,0,0))
133.        labelPostAzul=myfont.render(lines[7][0:k], 1, (0,0,0))
134.        #k=lines[8].find(" ")
135.        #newline=lines[8][k+1:len(lines[8])-1]
136.        #PostAma= myfont.render(newline, 1, (0,0,0))
137.        #========Etiquetas en Ventana==========
138.        screen.blit(evTime, (450,350))
139.        screen.blit(portAzul, (650,390))
140.        screen.blit(PortAma, (650,410))
141.        screen.blit(Pelota, (650,430))
142.        screen.blit(Adv, (650,450))
143.        screen.blit(PostAzul, (650,470))
144.        screen.blit(labelPortAzul, (450,390))
145.        screen.blit(labelPortAma, (450,410))
146.        screen.blit(labelPelota, (450,430))
147.        screen.blit(labelAdv, (450,450))
148.        screen.blit(labelPostAzul, (450,470))
149.        screen.blit(txt, (600,0))
150.        cancha(0,0)
151.        if float(porcentPortA) >pre and int(lines[3][0:1])==1:
152.        canchaPorA(0,0)
153.        if float(porcentPortA) >pre and int(lines[3][0:1])==2:
154.        canchaPorAma(0,0)
```

```
155.        if float(porcentPortA) >pre and int(lines[3][0:1])==5:
156.        canchaPostAz(0,0)
157.        if float(porcentPortA) >pre and int(lines[3][0:1])==6:
158.        canchaPostAma(0,0)
159.        screen.blit(pygame.transform.scale(pygame.image.load(path +
    filename), (400, 300)), (450,30))
160.        pg.display.flip()
161.        dt = clock.tick(30) / 1000  # / 1000 to convert to seconds.
162.        if __name__ == '__main__':
163.        main()
164.        pg.quit()
```

Script de Evaluación

```
1.  from __future__ import absolute_import
2.  from __future__ import division
3.  from __future__ import print_function

4.  import argparse
5.  import sys
6.  import time

7.  import numpy as np
8.  import tensorflow as tf

9.  def load_graph(model_file):
10. graph = tf.Graph()
11. graph_def = tf.GraphDef()

12. with open(model_file, "rb") as f:
13. graph_def.ParseFromString(f.read())
14. with graph.as_default():
15. tf.import_graph_def(graph_def)

16. return graph

17. def read_tensor_from_image_file(file_name, input_height=299,
    input_width=299,
                                    a.  input_mean=0, input_std=255):
18. input_name = "file_reader"
19. output_name = "normalized"
20. file_reader = tf.read_file(file_name, input_name)
21. if file_name.endswith(".png"):
22. image_reader = tf.image.decode_png(file_reader, channels = 3,
```

```
                i.  name='png_reader')
elif file_name.endswith(".gif"):
image_reader = tf.squeeze(tf.image.decode_gif(file_reader,
                    1.  name='gif_reader'))
elif file_name.endswith(".bmp"):
image_reader = tf.image.decode_bmp(file_reader, name='bmp_reader')
else:
image_reader = tf.image.decode_jpeg(file_reader, channels = 3,
                    1.  name='jpeg_reader')
float_caster = tf.cast(image_reader, tf.float32)
dims_expander = tf.expand_dims(float_caster, 0);
resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
sess = tf.Session()
result = sess.run(normalized)

return result

def load_labels(label_file):
label = []
proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
for l in proto_as_ascii_lines:
label.append(l.rstrip())
return label

if __name__ == "__main__":
file_name = "tf_files/flower_photos/daisy/3475870145_685a19116d.jpg"
model_file = "tf_files/retrained_graph.pb"
label_file = "tf_files/retrained_labels.txt"
input_height = 224
input_width = 224
input_mean = 128
input_std = 128
input_layer = "input"
output_layer = "final_result"

parser = argparse.ArgumentParser()
parser.add_argument("--image", help="image to be processed")
parser.add_argument("--graph", help="graph/model to be executed")
parser.add_argument("--labels", help="name of file containing labels")
parser.add_argument("--input_height", type=int, help="input height")
parser.add_argument("--input_width", type=int, help="input width")
parser.add_argument("--input_mean", type=int, help="input mean")
parser.add_argument("--input_std", type=int, help="input std")
```

```
60. parser.add_argument("--input_layer", help="name of input layer")
61. parser.add_argument("--output_layer", help="name of output layer")
62. args = parser.parse_args()

63. if args.graph:
64. model_file = args.graph
65. if args.image:
66. file_name = args.image
67. if args.labels:
68. label_file = args.labels
69. if args.input_height:
70. input_height = args.input_height
71. if args.input_width:
72. input_width = args.input_width
73. if args.input_mean:
74. input_mean = args.input_mean
75. if args.input_std:
76. input_std = args.input_std
77. if args.input_layer:
78. input_layer = args.input_layer
79. if args.output_layer:
80. output_layer = args.output_layer

81. graph = load_graph(model_file)
82. t = read_tensor_from_image_file(file_name,
        i.  input_height=input_height,
        ii.  input_width=input_width,
        iii.  input_mean=input_mean,
        iv.  input_std=input_std)

83. input_name = "import/" + input_layer
84. output_name = "import/" + output_layer
85. input_operation = graph.get_operation_by_name(input_name);
86. output_operation = graph.get_operation_by_name(output_name);

87. with tf.Session(graph=graph) as sess:
88. start = time.time()
89. results = sess.run(output_operation.outputs[0],
        i.  {input_operation.outputs[0]: t})
90. end=time.time()
91. results = np.squeeze(results)

92. top_k = results.argsort()[-5:][::-1]
93. labels = load_labels(label_file)
```

94. print('\nEvaluation time (1-image): {:.3f}s\n'.format(end-start))

95. for i in top_k:
96. print(labels[i], results[i])