Aplicación de *Deep Learning* en Robótica Móvil para Exploración y Reconocimiento de Objetos basados en Imágenes

Stevenson Contreras, Fernando De la Rosa
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Bogotá, Colombia
{js.contreras2187, fde}@uniandes.edu.co

***Resumen*—*Deep Learning* es una técnica de aprendizaje de máquina que busca definir redes neuronales basadas en el reconocimiento de patrones de los datos de entrada y lograr resultados con un nivel de confianza alto y eficiente en tiempo. En este trabajo se aplica en dos procesos que buscan mejorar la exploración de un robot móvil en espacios interiores. El primer proceso permite el reconocimiento de estructuras físicas como escaleras, ascensores y corredores a partir de las imágenes que se obtienen de una cámara instalada sobre el robot. El segundo proceso permite identificar la situación de cercanía a los obstáculos (a partir de la información de un sensor láser) para decidir la dirección y velocidad de desplazamiento del robot para navegar en un espacio interior. Este trabajo se destaca por la aplicación de *Deep Learning* en ambos procesos implementando un sistema de red neuronal genérica parametrizable, para poder crear varias instancias de red adaptables al tipo de información y su integración con un robot móvil que navega en ambientes simulados y experimentales.**

Palabras Clave*—*deep learning; machine learning; navegación en robótica; robótica móvil; procesamiento de imágenes; red neuronal; aprendizaje de máquina.

I. INTRODUCCIÓN

El aprendizaje de máquina se ha ido desarrollando en las últimas décadas con el fin de no hacer un programa específico para cada situación posible de un problema. Nuevas técnicas como *Deep Learning* (basado en redes neuronales) han surgido para tratar de simular en una máquina la forma en el que un cerebro funciona y aprende. Esta técnica consiste en desfragmentar en profundidad diferentes características de un elemento de información y de esta manera aprender por medio de patrones, que no son evidentes de reconocer para una máquina. Dentro de las aplicaciones de esta técnica se encuentra el reconocimiento de imágenes o video, reconocimiento de voz y reconocimiento de texto. En el campo de la robótica móvil, esta técnica es potencialmente muy útil para un mejor entendimiento e integración entre sensores físicos como cámaras, micrófonos, GPS, láser entre otros, y de esta forma tomar mejores decisiones sobre el entorno para las tareas que tengan que ejecutar.

El tema de estudio en este artículo consiste en utilizar *Deep Learning* para reconocer estructuras físicas (tipo escaleras, corredores, ascensores) y a su vez que el robot pueda navegar de manera segura en su espacio buscando que el robot tenga un aprendizaje y acciones que se acerquen a un comportamiento inteligente más natural.

II. TRABAJOS RELACIONADOS

A continuación, un resumen de trabajos destacados en robótica móvil que se apoyan en técnicas de aprendizaje de máquina, particularmente la técnica de *Deep Learning*.

El trabajo de Maturana y Scherer [1] presenta un sistema de detección de obstáculos en un terreno con vegetación. En este proyecto se implementó una técnica de *Deep Learning* la cual utiliza una red neuronal convolucional para calcular la región más adecuada en la que un helicóptero pueda hacer un aterrizaje satisfactorio. Esta red reduce la cantidad de parámetros a aprender para lograr una mayor eficiencia en el reconocimiento de imágenes, aprovechando la estructura espacial de una imagen.

Chen *et al.* presentan en [2] el problema de demostrar que se puede reconocer de manera acertada las puertas de un sitio interior mediante la técnica de *Deep Learning* y que esta es adecuada para la robótica móvil. En sitios interiores es necesario que el robot pueda reconocer las diferentes posiciones y ángulos de una puerta cuando está recorriendo un sitio; de esta manera puede crear un plan de ruta con base en las puertas que va reconociendo. Primero se recolectan muchas imágenes de puertas en diferentes posturas e imágenes de falsos positivos para que la red neural se pueda entrenar.

El proyecto realizado por Buitrago-Martínez *et al.* [3] presenta una forma de navegación de un robot móvil desde una posición inicial a una posición objetivo mediante la generación de una ruta sin colisiones, con un proceso de aprendizaje el cual puede ser aplicado satisfactoriamente en ambientes desconocidos. Aplicando la técnica de *Q-learning*, un método de aprendizaje por refuerzo, la cual por medio de una función de optimización, una acción en un determinado estado obtiene una recompensa en un rango que permite determinar el próximo movimiento con cierta probabilidad. De esta manera,

978-1-5090-2966-2/16/$31.00 ©2016 IEEE

el conjunto de acciones del robot no recae en ciclos sin salida, sino que para poder optimizar la función objetivo, puede caer en un estado que no la mejore pero que genera acciones (con buenas recompensas) que permiten salir de mínimos locales para llegar al mínimo global.

III. Problemá de Estudio y Solución Propuesta

Este trabajo busca que un robot móvil logre analizar y navegar un espacio interior a partir de imágenes y de medidas de proximidad a obstáculos. A partir de la bio-inspiración del cerebro humano, las redes neuronales permiten que las máquinas puedan aprender sin necesidad de tener que programar todas las posibles situaciones que puedan encontrar, aspecto tedioso en una solución en el mundo real, ya que hay demasiadas variables que una función no puede calcular de forma completa o se tardaría mucho en ejecutar a través de un computador. A la luz de esta problemática, una red neuronal, le permitirá al robot identificar y generalizar los objetos obtenidos de una cámara digital por medio de patrones. Así mismo razonar y relacionar los datos obtenidos de distancia con su tarea de navegación en su espacio.

A. *Deep Learning*

Se propone un sistema de red neuronal genérica. Bajo este modelo se generan dos instancias, una para el reconocimiento de imágenes y la otra para el de navegación. Esto implica una etapa de entrenamiento que las dos redes reutilizan (con datos adecuados para cada reconocimiento), con una buena cantidad de datos de calidad. Una red neuronal es sensible a la parametrización que se le defina, si se utilizan valores exagerados, puede sobre-entrenarse, haciendo que no se adapte ante situaciones nuevas, o si sucede el caso contrario, no podrá realizar con éxito el reconocimiento.

Es por esto que cada red tiene una configuración diferente. El sistema de red se puede adaptar a las necesidades de cada reconocimiento. Una de las redes está enfocada en el proceso de aprendizaje de las diferentes estructuras físicas en un espacio interior, como escaleras, corredores y ascensores. Esta red está compuesta por dos capas de neuronas que convolucionan la imagen, es decir, descomponen la imagen aplicando filtros para resaltar bordes de los objetos, y capas de *subsampling*, las cuales reducen la resolución de la imagen para reducir la carga de procesamiento y cinco capas de neuronas totalmente conectadas las cuales realizan la clasificación (Fig. 1). La otra red aprende a predecir la dirección y la velocidad para moverse utilizando los datos de proximidad del sensor láser, es por esto que no tiene capas de convolución ni *subsampling*, ya que estos procesos son para tratamiento de imágenes. A continuación, se presenta una descripción detallada de cómo implementar estas redes neuronales propuestas.

En estas redes la activación de cada neurona se determina por una función sigmoidea equivalente, que será eficiente para los cálculos sobre GPU. Esta función tendrá como entrada la matriz de pesos y los datos variables que provienen de las imágenes o proximidades; estas dimensiones tienen que ser coherentes para una efectiva multiplicación entre las matrices.

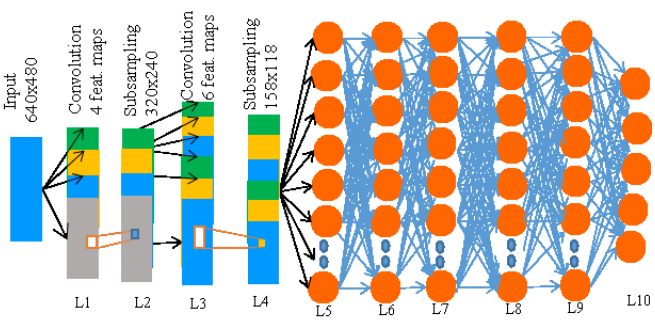


Fig. 1. Red neuronal convolucional [4].

Al inicio, la imagen comienza con 640×480 pixeles que se convierte a un arreglo de 307200 valores (pixeles), luego se convoluciona en 4 *features maps* para resaltar los bordes y formas de la imagen. Después se reduce la resolución de la imagen con *subsampling*, dejando así los *features maps* con una resolución de 320×240 pixeles. Por último, se vuelve a aplicar la convolución y *subsamplig*, obteniendo 6 *features maps* con resolución de 158×118 pixeles. De esta manera, estos datos ingresan a la red neuronal totalmente conectada, cada valor x_i se multiplica con un peso θ_i y se suma de la siguiente manera $y=\theta_1x_1+\theta_2x_2...+\theta_nx_n$. El valor resultante se normaliza entre 0 y 1 mediante la función sigmoidea $g(X, \theta')$, con el fin de convertir los datos en un porcentaje de activación de la neurona, la cual se define como:

$$g(X,\theta') = \frac{1+\frac{\tanh(X*\theta'+b)}{2}}{2}$$

con $X \in R^{1 \times numPixeles}$ y θ' (matriz θ transpuesta) $\in R^{numPixeles \times numNeuronas}$ obtendremos una probabilidad de activación $P(Y=i \mid X, \theta, b)$ donde X son los valores esperados, b es el vector de parcialidad o corte con Y definida por la fórmula lineal $Y = \theta X + b$. La salida de esta capa será la entrada para la siguiente capa determinada por:

$$X \in R^{1 \times numNeuronas_{capa-1}}$$

$$\theta_{capa} \in R^{numNeuronas_{capa-1} \times numNeuronas_{capa}}$$

Para el caso de los θ_{out} de la capa de salida sus dimensiones se determinarán por el número de objetos o acciones a clasificar:

$$\theta_{out} \in R^{numNeuronas_{capa-1} \times numObjetos_{capa}}$$

Para el reconocimiento de imágenes se consideran cuatro estructuras: escaleras, ascensores, corredores anchos y corredores estrechos. Para la navegación se producen tres acciones: giro izquierda, giro derecha y movimiento al frente.

B. *BackPropagation*

Para el proceso de entrenamiento y aprendizaje de la red neuronal se usa el algoritmo de *BackPropagation*, el cual a partir del resultado obtenido de una imagen o valor de entrada se genera un error con respecto al valor esperado y de esta

manera, se retrocede por la red ajustando todos los pesos con relación a este error global. Teniendo esto en mente, es necesario calcular la función de costo regularizada $J(\theta)$, que representa el costo o la diferencia entre el resultado de la función de activación $h_\theta(x^{(i)})_k$ y el valor esperado $y^{(i)}{}_k$ por medio de una función logística que tiene en cuenta el número de imágenes a entrenar m, la cantidad de neuronas de una capa K, la constante de regulación λ, el número de neuronas de las capas intermedias N y el número de capas intermedias de la red L:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}(-y_k^{(i)}\log(h_\theta(x^{(i)}))_k$$

$$-(1-y_k^{(i)})\log(1-h_\theta(x^{(i)})_k) + \frac{\lambda}{2m}\sum_{\theta=1}^{\theta}\sum_{j=1}^{N}\sum_{l=1}^{L}(\Theta_{j,l}^{\theta})^2$$

Así mismo $y^{(i)}{}_k \in \{0, 1\}$ donde 1 significa un valor positivo para el objeto que se está entrenando. Por ejemplo si se quiere clasificar correctamente la imagen de una escalera, el conjunto de entrenamiento tendrá imágenes de escaleras y de no escaleras, de esta manera cada imagen estará asociado a un valor $y^{(i)}{}_k$ que será 1 o 0 determinando si es o no una escalera.

A partir de esta función de costo $J(\theta)$ es necesario hallar el gradiente, que básicamente es la derivada de esta función, lo que nos determina que tan buena es esta función de costo. Una vez calculado el gradiente se puede entrenar la red neuronal, minimizando la función de costo con el algoritmo optimizador de gradiente (Fig. 2).

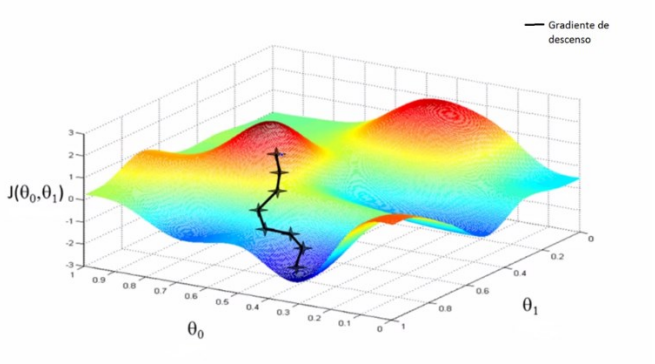


Fig. 2. Optimización gradiente de descenso [4].

Este gradiente es hallado después de ejecutar el algoritmo de *BackPropagation*, ya que si se hace de la forma matemática usual (hallando la derivada), el cálculo computacional para muchas entradas es ineficiente y costoso, mientras que *BackPropagation* optimiza este cálculo.

El proceso de entrenamiento de las redes neuronales empieza, inicializando de forma aleatoria y simétrica los parámetros θ, ya que es una estrategia efectiva de generar valores aleatorios en un rango $[-\varepsilon_{init}, \varepsilon_{init}]$ donde

$$\varepsilon_{init} = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}}$$

con L_{in} el número de entradas (pixeles) y L_{out} el número de neuronas de la primera capa. De este modo los pesos se definen como:

$$\theta = rand(L_{out}, 1 + L_{in})2\varepsilon_{init} - \varepsilon_{init}$$

Luego para ejecutar este algoritmo *BackPropagation* se empieza ejecutado el algoritmo hacia adelante *Forward Pass* a partir del conjunto de entrenamiento $(x^{(t)}, y^{(t)})$, con $x^{(t)}$ los valores de entrada (pixeles o proximidades) y $y^{(t)}$ el valor binario esperado, mencionando si $x^{(t)}$ es (1) o no (0) el dato a aprender. De esta forma se calculan todas las activaciones a través de la red neuronal incluyendo los valores de salida $h_\theta(x)$. En este punto, se quiere calcular el error $\delta_j^{(l)}$ por cada neurona j en la capa l, este le asigna a cada neurona un grado de responsabilidad en el error que se obtuvo en la salida. Para cada salida en la neurona k se obtiene el error mediante:

$$\delta_k^{(out)} = a_k^{(out)} - y_k$$

donde $y_k \in \{0, 1\}$, 1 si es la clasificación deseada y 0 si es una clasificación falsa al elemento a entrenar. Para las capas ocultas este error se calcula como:

$$\delta^{(l)} = (\theta^{(l)})^T \delta^{(l+1)} g'(z^{(l)})$$

donde $g'(z)$ es la derivada de la función sigmoidea:

$$g'(z) = g(z)(1 - g(z))$$

Calculado los errores se acumulan en un delta de error mediante:

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

La derivada de la función de costo se obtiene como:

$$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta) = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\theta_{ij}^{(l)}$$

para $j \geq 1$ donde $j \in Neurona$, $i \in entrada$, $l \in capa$. Por último, teniendo la función de costo y su derivada se ejecuta el algoritmo de optimización de los parámetros para minimizar el costo, es decir que el gradiente hallado por el algoritmo de *BackPropagation* tienda a 0.

C. *Arquitectura del Sistema*

Para el problema de interés se plantea un sistema compuesto por las dos redes neuronales (Fig. 3) controlado principalmente por el módulo *NeuralDriver*, el cual integra los componentes *ImageProcessor* y *RobotSystem* que se utilizan para poder obtener la información de la cámara y el robot respectivamente. De esta forma, *NeuralDriver* puede entrenar las redes utilizando el módulo *Training*, el cual a su vez emplea el módulo *NeuralLearning* para realizar todo el proceso de aprendizaje, el cual depende de *NeuralNetwork* para obtener todos los datos de la red neuronal. Finalmente, la información resultante se reporta a *TargetManager*, encargado de persistir la información de la red neuronal entrenada y/o generar una acción para el modo *Online*. En modo *Online* (tiempo real) se obtiene la información de los sensores (cámara y láser) para

reportar la información obtenida y dar las ordenes de acción al robot (girar izquierda, girar derecha o mover al frente).

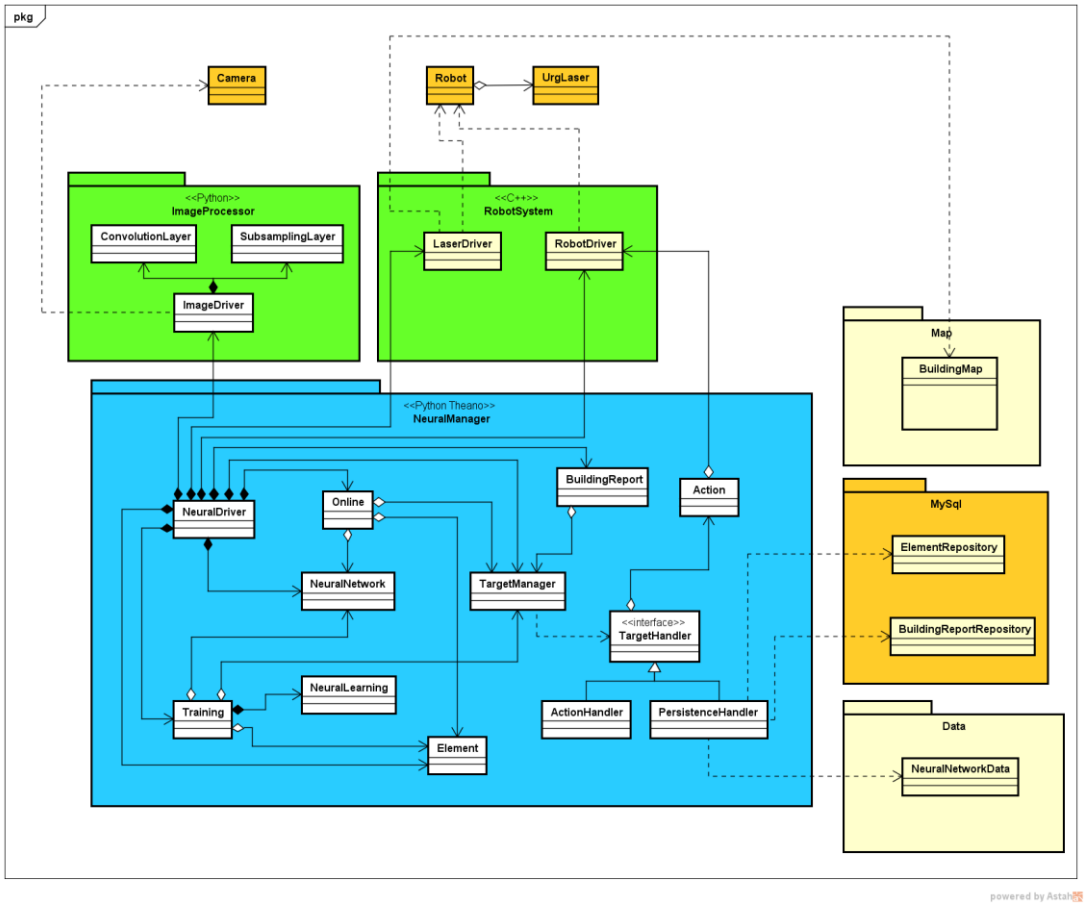


Fig. 3. Arquitectura general del sistema.

Debido a que el sistema de redes neuronales es genérico se pueden crear varias instancias con parámetros diferentes. Para la red de reconocimiento de imágenes, el tamaño de la entrada es la imagen tratada en el controlador *ImageDriver*, cuenta con 5 capas de 100 neuronas conectadas entre sí y una capa de salida de 5 neuronas relacionadas cada una con escaleras, corredores anchos, corredores estrechos, ascensores y un objeto de control, es decir, imágenes que no son las estructuras a reconocer. Se usó esta configuración debido a la gran cantidad de datos de pixeles a procesar y entrenar, que aunque requiere mayor poder de cómputo, la clasificación es más acertada y confiable. Este análisis se toma en cuenta dependiendo de los recursos disponibles y la cantidad de datos a analizar, los cuales aumentaron progresivamente de 200 hasta 800 imágenes etiquetándolas con el valor esperado usando el módulo *Training* mediante el proceso de configurar el objeto a entrenar y obtener imágenes de la cámara, preguntando al usuario si es una imagen válida para el objeto. Las dimensiones de la red completamente conectada se definen a continuación:

capaIn[arreglo_pixels_158×118, 100], capa2[100, 100], capa3[100, 100], capa4[100, 100], capa5[100, 100], capaOut[100, 5]

Para la red neuronal de navegación se cuenta con 3 capas de 25 neuronas conectadas entre sí y una capa de salida de 3 neuronas relacionadas con las posibles acciones de movimiento del robot. Debido a que la cantidad de datos de proximidad es mucho menor (1500 veces más pequeño que los datos de los pixeles), se configuró una red neuronal sin convolución ni *subsampling*, con un número de neuronas reducido en comparación a la red de reconocimiento; esto debido a que se puede llegar a una clasificación confiable, optimizando recursos y tiempo de ejecución. Las dimensiones de esta red se definen a continuación:

capaIn[{arreglo_sensor_123, objeto_Reconocido_1}, 25], capa2[25, 25], capa3[25, 25], capaOut[25, 3]

Como se puede observar, para la entrada de la red de navegación se tiene el buffer del sensor de proximidad y el objeto reconocido por la otra red lo cual define una dependencia entre redes y permite definir un comportamiento (movimiento) diferente por cada estructura reconocida. De esta manera se le agrega un factor más inteligente y contextualizado

a la acción a tomar para moverse. Por ejemplo, si está en un corredor amplio se podría mover más seguro y rápido, si está en un corredor estrecho podría moverse más lento o no entrar y si reconoce unas escaleras, intenta evitarlas y además trataría de buscar un ascensor (depende del entrenamiento que se realice por cada estructura reconocida).

IV. Implementación y Resultados

El sistema integrado por las 2 redes neuronales definidas se implementó bajo el lenguaje Python 2.7 ya que esta versión es compatible con las demás tecnologías a utilizar: OpenCV, el ambiente instalado Anaconda [5] que incluye las librerías de apoyo para Theano y el IDE de desarrollo PyCharm [6].

Se utilizó el *framework* Theano [7] hecho bajo Python, especializado en definir, optimizar y ejecutar expresiones matemáticas para arreglos multidimensionales. Con Theano se pudo ejecutar los diferentes cálculos de la red neuronal sobre GPU, realizando cálculos de matrices en paralelo haciendo más eficiente el entrenamiento. Para utilizar la GPU se requiere tener una tarjeta gráfica NVIDIA con el *toolkit* CUDA [8] instalado y Visual Studio 2012 [9] o superior.

En experimentación se trabajó con un robot móvil P3-DX [10], un sensor láser URG [11] y una cámara web Logitech C210 [12] de 640×480 pixeles usando OpenCV (Python) [13], de esta forma se obtuvo la matriz de datos de los pixeles de la imagen a 30 *frames* por segundo. La conexión al robot (Fig. 4) se hace con la librería Aria (Python) [14], la cual facilita la integración del sistema en un mismo lenguaje. Así mismo, se utiliza el simulador MobileSim (Fig. 5) [15], para el entrenamiento y pruebas. En la tele-operación del robot se usó un Joystick Genius F-16U [16], el cual facilita el control de movimiento del robot durante el entrenamiento.

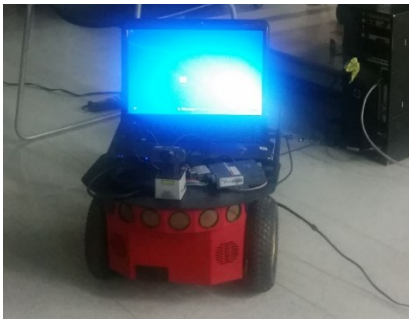


Fig. 4. Robot P3-DX, con sensor láser URG y portátil.

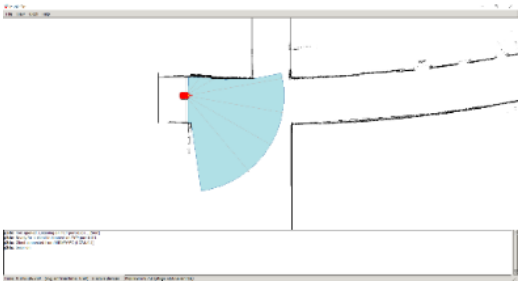


Fig. 5. MobileSim: Ambiente de simulación con robot P3-DX.

El sistema tiene dos estados básicos: modo *Training* (entrenamiento) y modo *Online* (Tiempo real). Para el entrenamiento se dispuso de un computador Windows 10 con procesador i7 de 12 CPUs 3.7GHz, memoria RAM de 16 GB y una tarjeta gráfica (GPU) MSI GTX 980 de 4GB RAM. Por otra parte el modo *Online*, se utilizó un portátil Windows 7 AMD de 2 CPUs, 2GHz con memoria RAM de 4GB, y GPU integrada la cual solamente es compatible con la función CPU de Theano.

Para tener un aprendizaje de todas las estructuras físicas fue necesario entrenar con toda la información obtenida ya que si se aprendía fragmentado, ocurría un desbalanceo en la red neuronal, ocasionando que al final generara un resultado único sobre la última estructura entrenada, independientemente de la imagen o dato del sensor mostrado durante el modo *Online*.

Para validar los resultados es necesario obtener el conjunto de probabilidades de confianza y el costo que la red neuronal retorna al comparar el resultado con el valor esperado en cada iteración; de esta manera, se visualiza que tan bien entrenada esta la red. Así mismo, se debe utilizar diferentes tasas de aprendizaje y conjuntos de datos de entrenamiento para entender el comportamiento de aprendizaje de las redes neuronales. Por otra parte es necesario tomar los tiempos de entrenamiento para hacer comparaciones entre la diferencia de ejecución sobre CPU y GPU. Por último, se debe estimar con pruebas reales que tan acertado es ante nuevas muestras de imágenes, y con la exploración visualizar y evaluar la cantidad de terreno recorrido, ante las acciones esperadas.

Al principio, durante el entrenamiento para las dos redes neuronales falló su convergencia por tener un conjunto de entrenamiento pequeño y una tasa de aprendizaje grande, quedando inservibles. Después de esto, se tomaron muestras mayores para la navegación, incluyendo 1500 muestras del sensor láser experimental, donde cada muestra contiene 123 detecciones/distancias posibles de proximidad, con resolución de 1 grado en el campo visual (barrido horizontal), con una tasa de aprendizaje de 0.001 (Fig. 6).

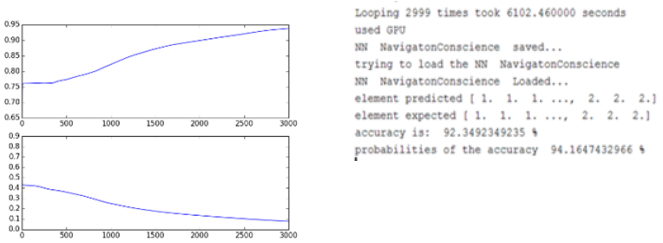


Fig. 6. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Proceso de aprendizaje navegación.

Se definieron 6 redes neuronales independientes con entrenamientos diferentes en la navegación, en algunas impulsando la curiosidad del robot y otras manteniendo la seguridad. Se obtuvieron las gráficas de aprendizaje que se ilustran en Fig. 7 – 10.

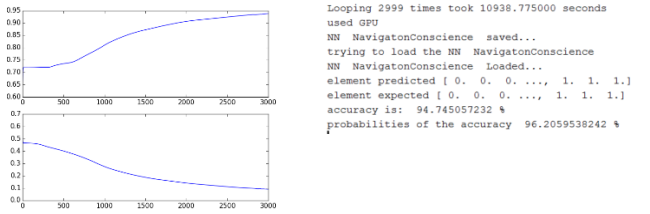


Fig. 7. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Re Aprendizaje navegación 1.

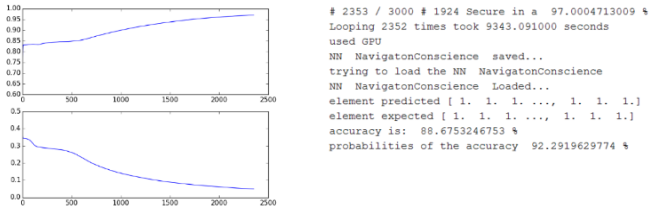


Fig. 8. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Re Aprendizaje navegación 2.

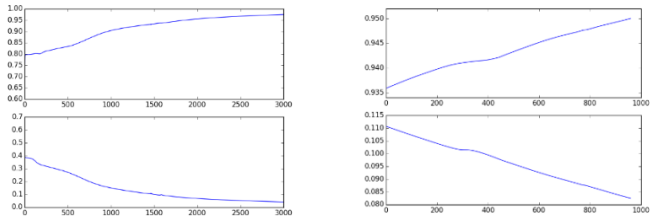


Fig. 9. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Re Aprendizaje fallido navegación.

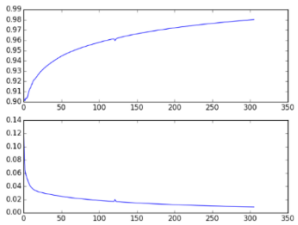


Fig. 10. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Re Aprendizaje fallido navegación.

Luego de probar estas 6 redes neuronales en simulación y en el ambiente experimental, se obtuvo una red funcional (Fig. 7) que se comportó muy bien en simulación, presentando 5 colisiones en todo el recorrido, navegando en el 60% aproximadamente del espacio (Fig. 11 – 14). Las zonas amarillas son los lugares donde el robot colisionó, y la zona azul es donde se redirigió hacia el corredor estrecho. El resto de lugares el sistema lo exploró automáticamente. Sin embargo, con el robot experimental solo se pudo recorrer un 40% del lugar, con 5 colisiones y muchas indecisiones al moverse. Esto se debió a que los datos obtenidos por el sensor láser de proximidad generan errores considerables comparados con lo que se simuló.

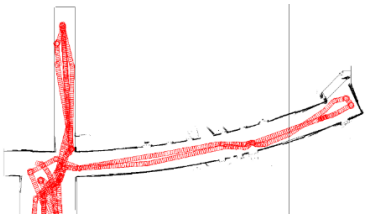


Fig. 11. Vista norte del mapa.

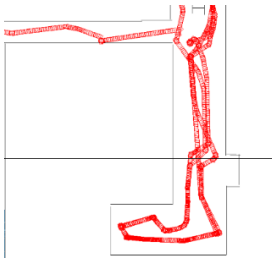


Fig. 12. Vista sur del mapa.

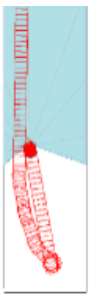


Fig. 13. Patrón de corredor interior.

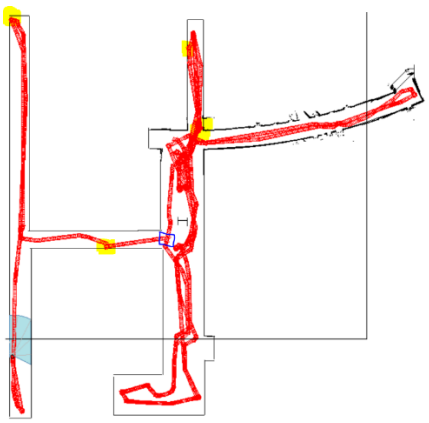
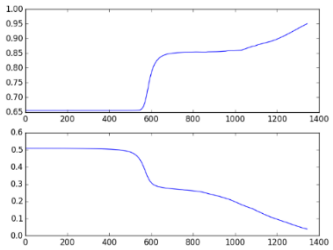


Fig. 14. Vista general del mapa.

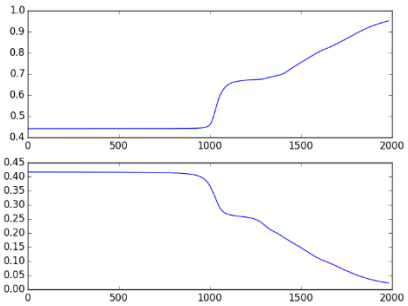
Con respecto al reconocimiento de estructuras (escaleras, corredores anchos, corredores estrechos y ascensores) a partir de imágenes no se presentaron muchos problemas después de ajustar los parámetros con una tasa de aprendizaje de 0.001. En principio se utilizaron 500 imágenes, las cuales la red neuronal aprendió después de 1343 iteraciones con un porcentaje del 94.9% de confianza (Fig. 15).



```
warning!!  0.949  ##  0.949
# 1343 / 3000 secure 94.9809839554
# 1344 / 3000 secure 95.0149925143
# 1344 / 3000 # 527 Secure in a  95.0149925143 %
Looping 3000 times took 5282.195000 seconds
used GPU
ArRobot::myPacketReader: Timed out (4) at 5836769 (200 into cycle afte
ArRobot::myPacketReader: Timed out (4) at 19713910 (200 into cycle aft
```

Fig. 15. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Aprendizaje final imágenes 1.

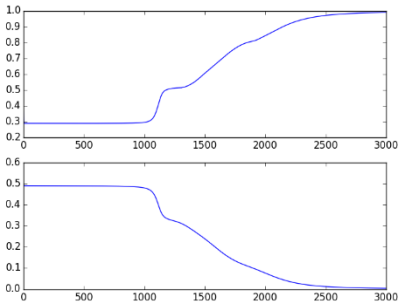
Pero al probarlo en el ambiente experimental se percibió que la cantidad de reconocimientos acertados hacia las escaleras y ascensores era bajo. Entonces se tomó más imágenes para estos objetos y se reentrenó para una siguiente prueba. En la Fig. 16 se observa que al tener 800 imágenes de estructuras, el sistema necesito más iteraciones para llegar al mismo nivel de confianza.



```
# 1983 / 3000 # 785 Secure in a  95.0141979614 %
Looping 3000 times took 11244.109000 seconds
used GPU
```

Fig. 16. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Aprendizaje final imágenes 2.

Para mejorar el aprendizaje, se necesitó 2999 iteraciones y se entrenó a un 98% de confianza (Fig. 17), esto generó un reconocimiento acertado de 80% de los objetos mostrados en tiempo real, de esta manera se logró que la red neuronal funcionara mientras que el robot se movía.



```
Looping 2999 times took 4842.658000 seconds
used GPU
```

Fig. 17. Gráficas Iteraciones vs Confianza (superior) e Iteraciones vs Costo (inferior): Aprendizaje final imágenes 3.

Debido a que la navegación no tuvo el mismo resultado satisfactorio porque el entrenamiento se realizó sobre simulación, el sensor láser virtual estaba mal calibrado con respecto al sensor láser real del robot, lo cual generó inconsistencias y dudas al momento de moverse. Se intentó reajustar estos parámetros, pero por diferencias en el sensor láser experimental y las ambigüedades presentadas en los datos obtenidos, la navegación no fue la esperada. Una prueba complementaria para el reconocimiento de imágenes se realizó tele-operando el robot con un nivel de confianza de 89%. Algunos objetos mostrados al sistema fueron (Fig. 18 – 21):

Fig. 18. Imágenes de puertas de ascensor (2 imágenes).

Fig. 19. Imágenes de escaleras (2 imágenes).

Fig. 20. Imágenes de corredor estrecho (2 imágenes).

Fig. 21. Imagen de corredor amplio.

Para el caso de los corredores anchos y estrechos, a veces ocurrían ambigüedades, dado a que un corredor estrecho lo es dependiendo de su perspectiva, es por esto que el nivel de confianza bajaba a un 80%, ya que es muy parecido a un corredor amplio.

V. Conclusiones y Trabajo Futuro

En este trabajo se investigó sobre los métodos de aprendizaje de máquina actuales, utilizando uno de los más recientes como lo es *Deep Learning* de manera supervisada, es decir enseñándole únicamente los objetos de interés para el caso de estudio. Se diseñó una arquitectura, pensada en utilizar redes neuronales genéricas, las cuales a futuro, se podría usar el mismo sistema para reconocer sonido u otros tipos de imágenes en otros contextos. Esta arquitectura se diseñó para adaptarse a distintos equipos físicos, portátiles, robots, servidores, dependiendo de la necesidad a la hora de la toma y aprendizaje de datos. Se probó de diferentes maneras y con diferentes datos y se llegó a la definición de dos redes neuronales: una de reconocimiento de imágenes con un alto nivel de confianza y otra de reconocimiento de situaciones de proximidad a obstáculos con un nivel de confianza aceptable. Las fallas se presentan cuando no se logra un buen entrenamiento, es decir es necesario enseñarle de manera adecuada los datos a reconocer; si no hay una muestra que en conjunto logre definir de manera correcta los objetos o acciones a reconocer, las fallas aumentan considerablemente.

Se identificó una diferencia importante entre la información obtenida por el sensor láser simulado (usado en entrenamiento) y el sensor láser real (usado en experimentación) que afectó la tasa de reconocimiento en la red neuronal de navegación.

Como trabajos futuros, se espera trabajar en mejorar cada una de las redes neuronales en cuanto a su información sensorial de entrada para disminuir los casos de ambigüedad. Con respecto a la red de navegación se espera caracterizar mejor el funcionamiento del sensor láser con un modelo más preciso y con un campo visual de 180 grados. Y en relación a la red de reconocimiento se quiere probar con una cámara de mayor resolución 2D y también con una cámara infrarroja (con información 3D). Se espera que la captura de imágenes con mayor resolución 2D o 3D disminuya el error de detección.

Referencias

[1] D. Maturana y S. Scherer. 3D Convolutional Neural Networks for Landing Zone Detection from LiDAR. 2015 IEEE Int. Conf. on Robotics and Automation (ICRA), Marzo 2015, pp. 3471-3478.

[2] W. Chen, T. Qu, Y. Zhou, K. Weng, G. Wang y G. Fu. Door Recognition and Deep Learning Algorithm for Visual based Robot Navigation. 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO), Diciembre 2014, pp. 1793-1798.

[3] A. Buitrago-Martinez, F. De la Rosa y F. Lozano-Martinez. Hierarchical Reinforcement Learning Approach for Motion Planning in Mobile Robotics. 2013 IEEE Latin American Robotics Symposium and Competition (LARS/LARC), Octubre 2013, pp. 83-88.

[4] Convolutional Neural Networks (LeNet) — DeepLearning 0.1 documentation. [En línea] Disponible en: http://deeplearning.net/tutorial/lenet.html. [Accedido: 5-jul-2016].

[5] Continuum Analytics. Download Anaconda now!. [En línea] Disponible en: https://www.continuum.io/downloads. [Accedido: 5-jul-2016].

[6] JetBrains. PyCharm: Python IDE for Professional Developers. [En línea] Disponible en: https://www.jetbrains.com/pycharm/. [Accedido: 5-jul-2016].

[7] Welcome — Theano 0.8.2 documentation. [En línea] Disponible en: http://deeplearning.net/software/theano/. [Accedido: 5-jul-2016].

[8] NVIDIA. CUDA Zone. [En línea] Disponible en: https://developer.nvidia.com/cuda-zone. [Accedido: 5-jul-2016].

[9] Microsoft. Visual Studio - Tools for every developer and every app. [En línea] Disponible en: https://www.visualstudio.com/. [Accedido: 5-jul-2016].

[10] Adept MobileRobots. Robotic Control Platforms and Applications. [En línea] Disponible en: http://www.mobilerobots.com/Mobile_Robots.aspx. [Accedido: 5-jul-2016].

[11] Hokuyo Automatic Co. Scanning range finder URG-04LX. [En línea] Disponible en: https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx.html. [Accedido: 5-jul-2016].

[12] Logitech. Webcam C210. [En línea] Disponible en: http://www.logitech.com/assets/33983/webcam-c210-gsw.pdf. [Accedido: 5-jul-2016].

[13] Itseez. OpenCV. [En línea] Disponible en: http://opencv.org/. [Accedido: 5-jul-2016].

[14] Adept MobileRobots. ARIA. [En línea] Disponible en: http://robots.mobilerobots.com/wiki/ARIA. [Accedido: 5-jul-2016].

[15] [14] Adept MobileRobots. MobileSim. [En línea] Disponible en: http://robots.mobilerobots.com/wiki/MobileSim. [Accedido: 5-jul-2016].

[16] Genius. Joystick USB de ordenador con Turbo. [En línea] Disponible en: http://www.geniusnet.com/Genius/wSite/ct?xItem=19497&ctNode=1309. [Accedido: 5-jul-2016].