INSTITUCIÓN UNIVERSITARIA ITM

Facultad de Ingenierías

Ingeniería Electrónica

Producto de laboratorio

APLICACIÓN DE APRENDIZAJE PROFUNDO “DEEP LEARNING” EN RECONOCIMIENTO DE PEATONES

Preparado por

Andrés Felipe Rúa Ortiz

Medellín, Colombia
2018

APLICACIÓN DE APRENDIZAJE PROFUNDO “DEEP LEARNING” EN RECONOCIMIENTO DE PEATONES

Trabajo de grado para optar el título de Ingeniero Electrónico

Asesor

Carlos Andrés Madrigal Gonzales

Ingeniero Electrónico

Doctor en Ingeniería

INSTITUCIÓN UNIVERSITARIA ITM

FACULTAD DE INGENIERIA

INGENIERIA ELECTRÓNICA

MEDELLÍN-ANTIOQUIA

2018

RESUMEN

En la actualidad se generan gran cantidad de accidentes de tránsito asociados al error humano por descuidos, cansancio, estados alterados de conciencia, uso del celular, distracción con los niños, ingesta de alcohol, distracción con elementos multimedia al interior de los vehículos, micro-sueños, infartos, poca experiencia al volate, entre otros. Este proyecto tiene como objetivo reducir el parámetro del error humano al mínimo a través de controles automatizados de aprendizaje profundo programados previamente. Este último se logrará a través de la implementación del "Deep Learning" con una técnica conocida como TensorFlow. En este proceso se mostrará cómo se entrena el algoritmo en procura de que "aprenda" varios patrones previamente programados reconociendo imágenes por medio de etiquetas. Para lograr dicho proceso de "aprendizaje" debemos entrenarlo mostrándole diferentes "perspectivas" de un mismo objeto de manera tal que el algoritmo reduzca al máximo la falla por interpretación de lo detectado, es decir, que se aproxime a un cien por ciento en la precisión de lo capturado. Para que esto suceda, debemos "mostrarle" como un mismo objeto puede ocupar distintas posiciones o asumir distintas formas; por ejemplo, un peatón sentado o parado, ese mismo peatón cruzando la calle, en motocicleta o en bicicleta. También se le debe "enseñar" al algoritmo a interpretar señales de tránsito, reconocer vehículos u objetos aislados a un peatón. Esto lo logramos "mostrándole" al programa una misma imagen desde diferentes posiciones y así logrará construir un mapa en 3D del objeto. Entre más le

pueda “enseñar” al algoritmo, este alcanzará una mejor interpretación del objeto y reducir falla por mala interpretación.

En este producto de laboratorio se hace una exploración del concepto de aprendizaje profundo (“Deep Learning”) y de algunas de las construcciones que se implementan en esta técnica, además, de la relación intrínseca con las Redes Neuronales Artificiales (RNA) en aplicaciones cotidianas.

RECONOCIMIENTOS

Brindo mi más sincero agradecimiento a todas las personas que hicieron posible la realización y culminación del proyecto de ingeniería Electrónica.

En primer lugar, a la **Institución Universitaria ITM** por formarme como un excelente profesional en las áreas de Electrónica y Telecomunicaciones.

A mi asesor **Carlos Andrés Madrigal Gonzales** ya que gracias a su comprensión y solidaridad se logró un buen producto de laboratorio.

A mi compañero **José Fernando Pamplona** por su tiempo y dedicación; le deseo muchos éxitos en la culminación de su maestría en Automatización y control.

A mi profesora **Sara María Yepes Zuluaga** por sus palabras, por su constante lucha de alcanzar sueños y metas que parecían imposibles. Gracias por abrirme las puertas y lograr objetivos a nivel internacional.

Al Doctor **John Fredy Hoyos Murillo** por brindarme la posibilidad de crecer profesionalmente y siempre ser una figura de admiración y respeto.

Al Doctor **Alveiro de Jesús Valencia** por brindarme siempre apoyo moral y enseñarme que “la constancia vence lo que la dicha no alcanza” y “la disciplina tarde o temprano vence la inteligencia”.

A mis **padres**, **hermana**, **sobrinos** y **cuñado** que en ellos he encontrado el valor del respecto, la responsabilidad y me han brindado todo su amor, cariño y esfuerzo para hacer de mi vida lo que soy.

Y muy especialmente a mi esposa **Janeth Isabel Agudelo Zapata** y mi hijo **Samuel Rúa Agudelo** por ser los seres más especiales de mi vida y los que me impulsan a ser cada día mejor.

ACRÓNIMOS

2D Dos Dimensiones

3D Tres Dimensiones

RNA Redes Neuronales Artificiales

ADALINE ADAptative LINear Element

ABC Lenguaje de programación

MLP Perceptron Multicapa

RBM Restricted Boltzman Machine

OPS Operation Support Systems

CNN Convolution Neural Network

RGB Red Green Blue

NVIDIA Empresa multinacional especializada en el desarrollo de unidades de procesamiento gráfico

GPU Unidad de Procesamiento Gráfico

PCL Point Cloud Library

BSD Berkeley Software Distribution

GPS Global Positioning System

SCRATCH lenguaje de programación visual

cuDNN Deep Neural Network library

TABLA DE CONTENIDO

1. INTRODUCCIÓN

Las redes neuronales artificiales (RNA) dan sus primeros pasos aproximadamente en los años 50, inicialmente se hace la presentación a la comunidad científica del perceptrón y la red ADALINE. El perceptrón consiste básicamente en reconocer ciertos tipos de modelos sencillos, por otra parte, ADALINE se basa en el uso del gradiente descendiente para entrenar sus redes, la cual es utilizada en la actualidad en la mayoría de los algoritmos de aprendizaje. (Moreno, 2002).

Actualmente, el aprendizaje profundo tiene un amplio desarrollo en las Redes Neuronales Artificiales, como es de saber, dicha técnica desempeña un papel vital en todo tipo de áreas: como por ejemplo en la medicina, se desarrolló en el año 2006 (ganando premio Nobel) una técnica para terapias humanas mejorando significativamente la salud de las personas. En la biología se desarrolló funciones de doble cadena del ADN para la generación de proteínas funcionales en el metabolismo y muy especialmente en el área de ingeniería, que para efectos de este producto de laboratorio se implementó el reconocimiento de peatones por parte de un vehículo automotor de 4 ruedas usando como plataforma TensorFlow en el sistema operativo Linux bajo la distribución Ubuntu 14.04 (Quora, 2015).

Aprovecharemos este método de aprendizaje profundo (“Deep Learning”) para lograr la detección de peatones en pro de lograr los objetivos dispuestos en este producto de laboratorio orientados a reducir la cantidad de accidentes disminuyendo

el porcentaje de mortandad en las vías, además, en implementar una aplicación a nivel de ingeniería electrónica que tendrá como meta dar los primeros pasos en una herramienta tan poderosa como el TensorFlow.

1.1. PLANTEAMIENTO DEL PROBLEMA

Las redes neuronales convolucionales han favorecido un importante avance en tareas que involucran inteligencia artificial, tales como clasificación, localización, detección y segmentación, descripción detallada de escenas, entre otras, ya sea en formato de imágenes o videos.

Los resultados que se obtienen de los sistemas que poseen esa tecnología son bastante confiables, por lo que se puede afirmar que muchas de las tareas podrían ser resueltas; un ejemplo de una aplicación y sobre la cual elaborare mi propuesta, son los sistemas para el monitoreo y control de tránsito vehicular y peatonal.

En la actualidad, se pueden apreciar muchas cámaras de vigilancia en diferentes ciudades del mundo, por lo que se convirtió en algo común y vital el estar monitoreando continuamente ciertos lugares, como, por ejemplo, los aeropuertos, los centros comerciales, los senderos peatonales, las carreteras, los bancos, supermercados, estaciones de trenes, entre otros. Por lo general, sabemos que las cámaras de control son monitoreadas en un centro de vigilancia y lo hacen personas a cargo de este rol. Por lo que el aprendizaje profundo (“Deep Learning”) puede encajarse perfectamente en pro de ayudar a los operarios de vigilancia, así reduciendo el porcentaje de error que se pueda tener a la hora de tomar decisiones en situaciones reales que se viven a diario. Dicho concepto puede también

transferirse en el ambiente vehicular, en el año 2017 se registraron 5803 accidentes de tránsito en Colombia, disminuir esta cifra puede ser un reto para la aplicación que se propone, pues es muy común que mientras una persona conduce su vehículo se quede dormido ya sea por cansancio físico, por enfermedad o por la razón que sea. De esta manera el porcentaje de accidentes se disminuiría considerablemente preservando la vida humana y se introduce una nueva cultura en la sociedad, la cultura de la inteligencia artificial.

1.2. JUSTIFICACIÓN

Con el avance actual de la electrónica y las telecomunicaciones en tanto hardware y software, se plantea la idea de una implementación de un sistema que permita la prevención de accidentes de tránsito desde un vehículo automotor de cuatro ruedas. Este desarrollo se plantea a través de una aplicación fundamentada en el Deep Learning.

Actualmente la institución Universitaria ITM cuenta con un grupo de investigación denominado automatización y control el cual posee una línea de énfasis en visión artificial y fotónica, lo que hace que este proyecto deje bases para el desarrollo de muchas aplicaciones.

El presente trabajo se enfoca en el estudio y análisis de la detención de peatones usando una RNA profunda (“Deep Learning”) que se implementa a través de la librería TensorFlow. La arquitectura evaluada fue PointNet, que ha jugado un papel vital en múltiples aplicaciones aprovechando las capacidades que tiene y que puede brindar en los modelos que se implementen usando dicha técnica.

Por lo anterior, se pretende desarrollar un primer avance en donde se capturen las características que poseen un peatón, un vehículo, una señal de tránsito, un árbol, entre otros, a partir de su representación 3D usando Matlab como software creador de la base de datos y etiquetado, buscando digitalizar los perfiles que pueda detectar un sistema orientado a generar diferencias entre los objetos. La aplicación tendrá la capacidad de utilizar esta clasificación en pro de generar alertas o

proporcionar información que permita toma de decisiones agiles al conductor o logre ejecutar acciones preventivas como frenar en caso de una colisión.

1.3. OBJETIVOS

OBJETIVO GENERAL

Aplicar el aprendizaje profundo (“Deep Learning”) en la identificación de peatones mediante TensorFlow usando nubes de puntos en Matlab adquiridas desde un sensor Velodyne LiDAR

OBJETIVOS ESPECIFICOS

Analizar los fundamentos de Deep Learning y sus aplicaciones en la detección de objetos.

Implementar un procedimiento de la instalación de TensorFlow bajo el sistema operativo Linux con el lenguaje de programación Python.

Elaborar una herramienta en Matlab y Python usando aprendizaje profundo - Deep Learning en aplicaciones de detección de peatones.

2. MARCO TEÓRICO

2.1. PYTHON

Los programas que se implementan en TensorFlow generalmente están escritos en los lenguajes de bajo nivel, sin embargo, utilizar Python reduce muchos factores pues es un lenguaje que requiere una décima parte de líneas de código. Python s un lenguaje multiplataforma y procesa pixel por pixel de una imagen. Una ventaja de este lenguaje es que en la distribución de Linux (Ubuntu) ya está instalado (solo basta con escribir en la consola la palabra “Python”) e instalar los paquetes que se necesiten (como por ejemplo el NVIDIA) para utilizar las librerías del tensor y ejecutar la base de datos que se crea a partir de la generación de características de una imagen. Python es un lenguaje muy amigable y bastante útil; no sobra resaltar que Python también es utilizado a nivel de aplicaciones y plataformas. Por último, pero no menos importante, Python es un lenguaje totalmente gratis. (Duque, 2010)

2.2. Característica del lenguaje en TensorFlow

Se destaca que Python tiene una sintaxis muy visual y práctica. En muchos lenguajes, para separar porciones de códigos se utilizan elementos como las llaves o las palabras claves. En TensorFlow, el proceso de aprendizaje en este producto de laboratorio consta del muestreo de una serie de imágenes orientadas a reducir el margen de error “interpretativo” del objeto escaneado como una imagen de un

peatón, una señal de tránsito, árbol, ciclista, entre otros teniendo como parámetro esta propiedad (Galindo, 2005).

2.3. Redes Neuronales Artificiales (RNA)

El concepto de RNA es sencillo pues consiste en imitar las características reales de las redes neuronales de organismos vivos. La inteligencia artificial viene acompañada de parámetros que se deben tener presente a la hora de iniciar un proceso y relacionarlos, como, por ejemplo: los pesos, el ponderado y los nodos que conectan una red. Para ver un resultado de una RNA cada uno de los aspectos mencionados deben estar relacionados entre sí y lo ideal es que se repita cuantas veces sea necesario, como, por ejemplo, cuando una persona quiere aprender un idioma deberá escribir y ver una(s) palabra(s) cuantas veces sea necesario para que su red neuronal la procese y se vuelva parte de ella. En la figura 1 se observa la interpretación general de una RNA. (Herranz, s.f.)

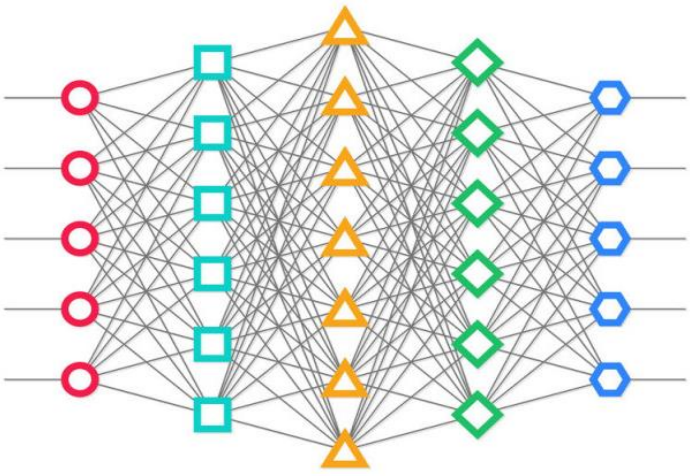


Figura 1. Red Neuronal Artificial

2.4. Arquitecturas de aprendizaje en Deep Learning

Back propagation: Es un algoritmo de entrenamiento el cual consiste en dar a conocer el error de la capa de salida hacia las capas ocultas. Por lo que el error de la capa de salida se propaga hacia atrás. En la figura 2 vemos un ejemplo de una RNA con esta arquitectura (Miller, 2015)

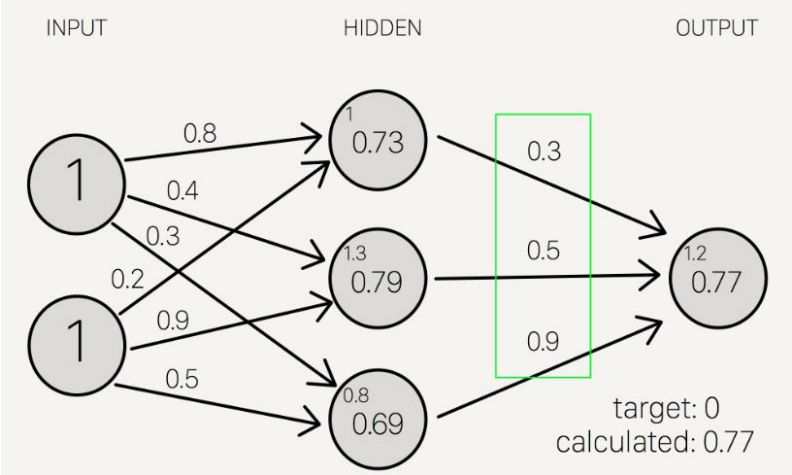


Figura 2, Perceptrón multicapa MLP

Autoencoder: Es una RNA muy parecida a la del perceptrón multicapa MLP, sin embargo, tienen ciertas diferencias. Las diferencias más destacables es que varía la MLP porque en su capa oculta presenta menos neuronas que en la capa de salida, como se muestra en la figura 3. (Torres, 2017)

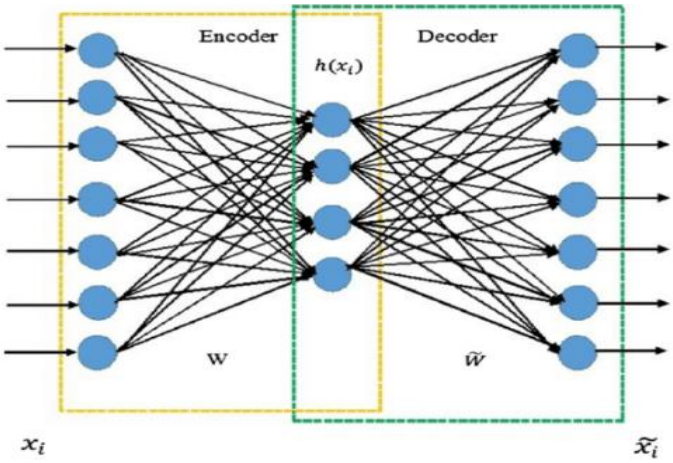


Figura 3. Arquitectura Autoencoder

2.5. Reconocimiento de imágenes digitales

Una imagen se define como una función en R^2 f (m,n), donde m y n se conocen como coordenadas espaciales. El valor de la función se interpreta como la intensidad de la imagen o fotografía (Hablando para el caso específico de este producto de laboratorio) en un punto dado. Una imagen analógica se representa mediante funciones continuas tanto en la posición como en la intensidad. La manera matemática de representarla es mediante una matriz $n * m$ que se evalúa y estudia en el álgebra lineal, su forma general es la siguiente:

$$f(m,n) = \begin{pmatrix} f(1,1)\, f(1,2) \ldots f(1,N-1)\, f(1,N) \\ f(2,1)\, f(2,2) \ldots f(2,N-1)\, f(2,N) \\ . \\ f(M-1,1)\, f(M-1,2) \ldots f(M-1,N-1)\, f(M-1,N) \\ f(M,1)\, f(M,2) \ldots f(M,N-1)\, f(M,N) \end{pmatrix}$$

Esto origina una imagen binaria y significa que cada bit puede ser 0 o 1, equivalente a decir negro o blanco, la figura 4 ilustra lo definido (Albino, 2018)

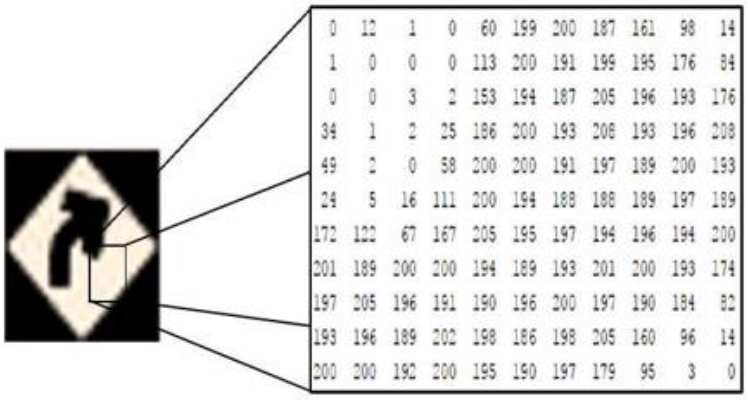


Figura 4. Pixeles de una imagen en escala de grises (8 bits por pixel)

2.6. Deep Learning

Deep Learning hace parte de un subconjunto de técnicas de Machine Learning que básicamente aprenden mediante representaciones graficas basado en datos y anteponiendo capas que a medida que se alimentan prolongan el aprendizaje siendo más exacto y significativo en una Red Neuronal Artificial.

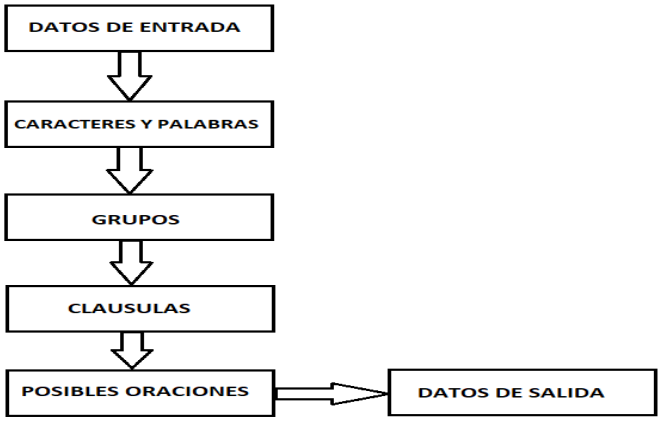


Figura 5. Deep Learning para componer un texto en español

De la figura 5, podemos interpretar cada recuadro como una capa donde se puede extraer características del idioma castellano, y las flechas indican que cuando una capa aprende una o varias características puede pasar a un siguiente nivel para

que el nivel de complejidad vaya creciendo. Es así entonces como el aprendizaje profundo (“Deep Learning”) debe funcionar en las RNA, idealizando el aprendizaje como herramienta principal de la red.

2.7. Redes neuronales convolucionales

Las redes neuronales convolucionales son una extensión de las redes neuronales en los seres vivos. Como bien sabemos de la medicina y ciencias de la salud, las redes neuronales tienen la capacidad de aprender por si solas y entre más contacto con un objeto o situación del mundo real, más se adapta al contexto. Las Redes Neuronales Convolucionales también heredan estas características dándole a cada neurona un peso y sesgo para aprender de una situación a partir de un algoritmo construido.

Por ejemplo, para la tarea de reconocer un peatón, se agregaría información acerca de cada objeto localizado dentro de una imagen, esto es, identificar y guardar el objeto con ciertas características para ir aprendiendo a reconocerlo. La figura 6 muestra cada una de las etapas (Briega, s.f.)

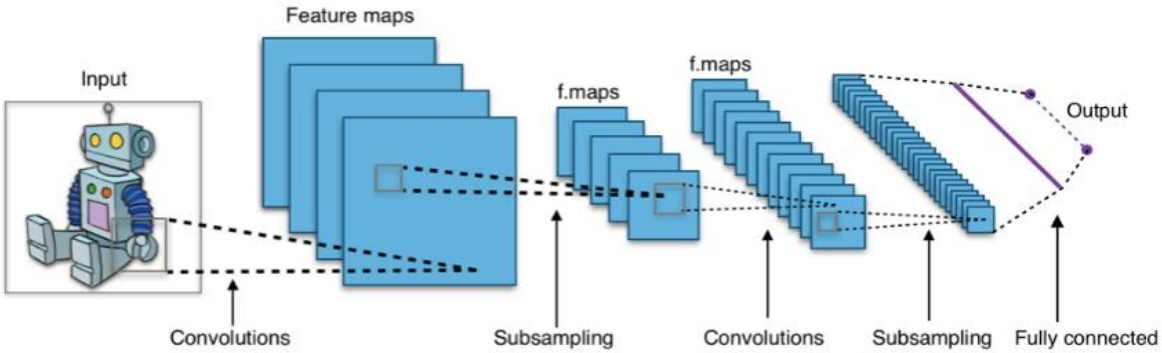


Figura 6. Arquitectura básica de una red neuronal convolucional

En las Redes Neuronales Convolucionales uno de los principales propósitos es proveer un filtro sobre la imagen o fotografía para extraer las principales características que ayuden a categorizar las propiedades más importantes de acuerdo al algoritmo construido, para este trabajo, sería el reconocimiento de peatones. Veamos en la figura 7 un ejemplo de una imagen filtrada. (Torres, 2017)

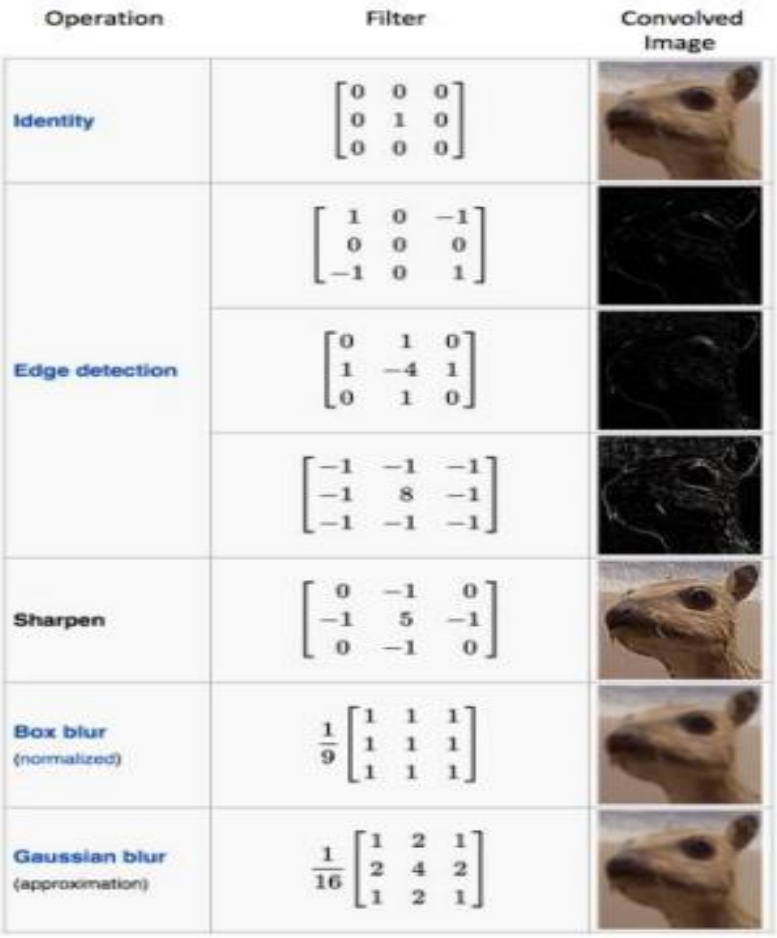


Figura 7. Diferentes filtros en una imagen

2.8. Nube de puntos

Cuando una persona adquiriere información de una situación del mundo real, el sistema neuronas automáticamente comienza a filtrar información y separar de acuerdo con los objetos que se encuentren alrededor. Por ejemplo, cuando observamos la colisión entre dos autos, lo primero que observamos es el estado de los autos para determinar qué tan leve o grave fue el accidente, luego, miramos si las personas implicadas en el accidente tienen heridas y de qué tipo son, dependiendo de esta característica el llamado de los centros de atención se prioriza o no. Todo esto sucede en cuestión de segundos, nuestro sistema activa técnicas en 3D para sintetizar el evento y reaccionar en caso de que se requiera. Pasa lo mismo en el mundo electrónico, el algoritmo detecta la imagen (se imprime en pantalla por medio de Matlab o la aplicación que se use), y se analiza por partes para ordenar la información (pixeles) en la base de datos dando categoría a lo que se tiene en el momento. (Urtasun, 2012)

Figura 8. Fotografía página oficial de KITTY

Figura 9. Porción de la Fotografía 8

De las figuras 8 y 9, vemos que un algoritmo detecta una porción de la imagen principal y entra analizar que se tiene en esa porción tan pequeña, para nuestro caso, vemos una señora con blusa blanca al lado de un árbol de rosas rojas. La información se adquiere por técnicas de 3D, es lo que denominamos en este producto de laboratorio como nube de puntos, para ello se usa la librería PCL (Point Cloud Library) que consiste en dar un tratamiento completo en Matlab para checar imágenes bajo la licencia BSD que cuenta con métodos de alta eficiencia dentro de los cuales está el filtrado.

3. METODOLOGÍA

3.1. Instalación de TensorFlow

Para la instalación y selección de librerías en TensorFlow, Se ejecutaron los pasos según la guía de instalación de la página web oficial. (Developers, 2018)

Para dicha instalación en este producto de laboratorio utilizamos los siguientes componentes:

- Máquina virtual (Virtual Box o VMware)
- Sistema operativo GNU-Linux (Ubuntu 14.04 recomendado)
- GPU instalada y funcionando
- Abrir terminal o interprete de comandos y ejecutar Python
- Procedemos a instalar paquetes de NVIDIA

```
$ sudo apt-get install cuda-command-line-tools
```

- En la siguiente librería se debe agregar la siguiente ruta:

```
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH:+${LD_LIBRARY_PATH}:}/usr/local/cuda/extras/CUPTI/lib64
```

- Luego, para evitar conflictos con el sistema operativo; mantenemos la versión del cuDNN en 7.0.5:

```
$ sudo apt-mark hold libcudnn7 libcudnn7-dev
```

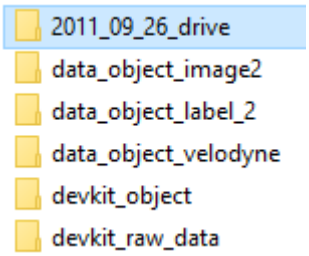
- Debemos eliminar la siguiente retención para permitir actualizaciones en el sistema.

```
$ sudo apt-mark unhold libcudnn7 libcudnn7-dev
```

3.2. Adecuación de la base de datos KITTI

Para la instalación de KITTI se deben descargar las bases de datos de la página oficial. (Urtasun, 2012). Estas últimas las describo a continuación siendo la resaltada la más importante:

2011_09_26_drive
data_object_image2
data_object_label_2
data_object_velodyne
devkit_object
devkit_raw_data



Es necesario resaltar que la descarga de estas bases de datos toma un tiempo prolongado dependiendo del ancho de banda que se disponga de conexión a internet; en mi caso, dispuse de 10 Mbps por lo que el Laptop se demoró aproximadamente 3 días en la descarga.

De estas bases de datos; obtenemos la información de las imágenes sobre las cuales Matlab introducirá la nube de puntos en el algoritmo de manera tal que la red PointNet pueda generar una “descripción” de la imagen por “aprender” en 3D.

En la página oficial de KITTI existen varias herramientas para trabajar bajo la librería de TensorFlow y diseñar algoritmos que permitan soluciones en la industria orientada a generar mejoras en la vida de las personas según el proyecto o meta establecidos; por ende, la herramienta está abierta a mejoras por parte de los estudiantes o investigadores expertos en el área de ingeniería en especial en el área de Telecomunicaciones, Electrónica e Informática.

3.3. Descripción del etiquetado en Matlab

Para hacer el etiquetado de las imágenes obtenidas de las bases de datos; debemos indicarle a Matlab la ruta del Dataset. Como se indica en el código de los anexos; se ingresa dicha ruta según se describe en las figuras 10, 11 y 12:

```
data_dir  = 'C:\Users\Personal\Desktop\Base de datos\2011_09_26_drive\2011_09_26\2011_09_26_drive_0091_sync';
calib_dir = 'C:\Users\Personal\Desktop\Base de datos\2011_09_26_drive\2011_09_26';
```

Figura 10. Rutas en Matlab

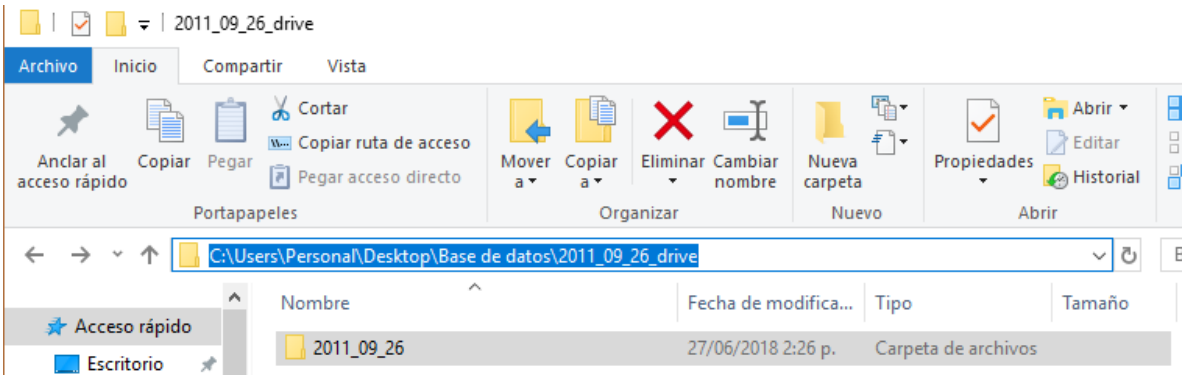


Figura 11. Rutas en el PC

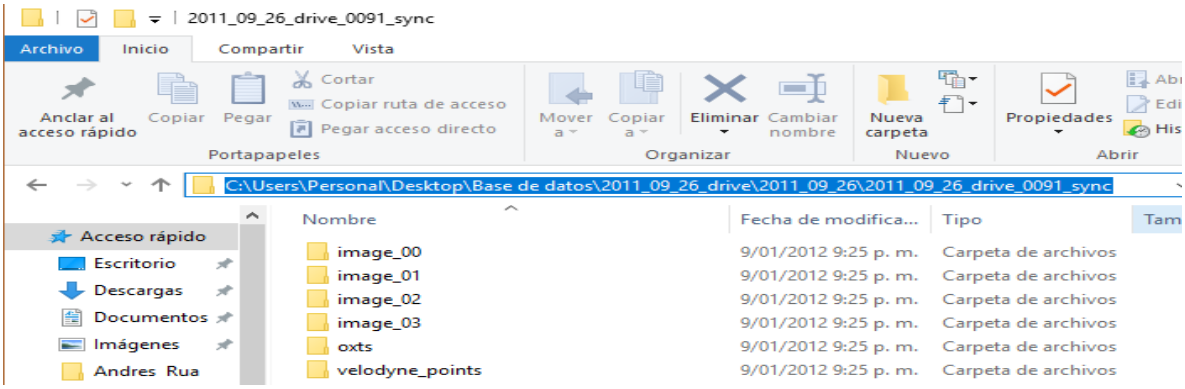


Figura 12. Rutas en el PC

Es de aclarar que el texto ingresado o ruta, debe coincidir con la ubicación de la base de datos.

Este etiquetado consiste en establecer una descripción de la imagen que proporciona la base de datos de KITTI, (Urtasun, 2012) identificando los tipos de elementos en categorías preestablecidas en Matlab. El proceso de categorización se hace fundamental pues, entre más imágenes se reconozcan y se aprendan se logrará una mayor eficiencia al momento de reconocer un objeto preestablecido como un peatón que es el objetivo principal de este producto de laboratorio.

Finalmente, cuando etiquetamos la porción de una imagen en una matriz de puntos, esta se guarda en la base de datos creada por Matlab la cual nombramos Dataset como se muestra en la figura 13.

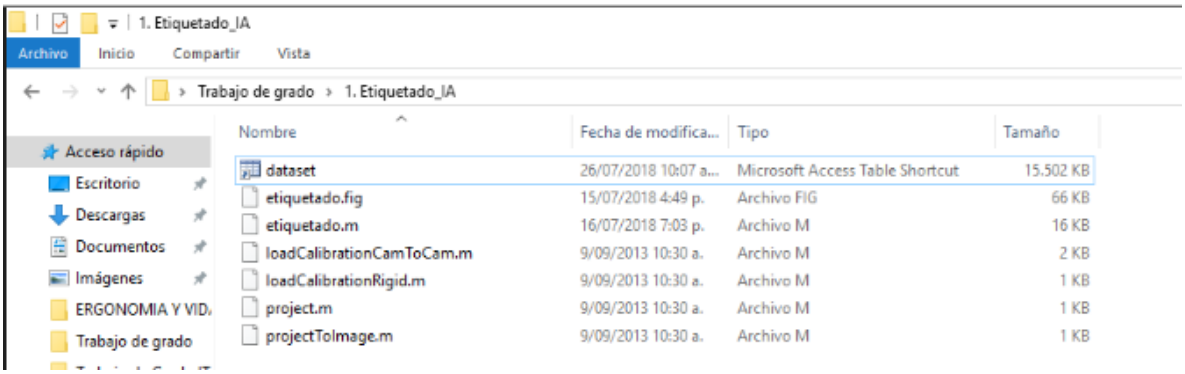


Figura 13. Base de datos (Dataset)

3.4. Implementación PointNet

Es de señalar que la base de datos no se ingresa sola. Debe transformarse en un formato h5, acomodando las clases a las que tenía la red neuronal artificial, insertando la nube de puntos y las etiquetas establecidas en Matlab, después de ello se utiliza el código de evaluación que tiene PointNet utilizando el modelo entrenado.

Este código está disponible en la página oficial de GitHub como se muestra en la figura 14. (Charles, 2007)

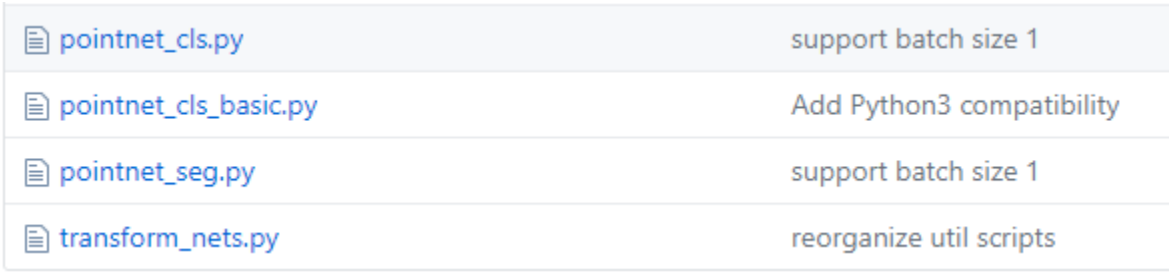


Figura 14. Modelos de PointNet

4. RESULTADOS Y DISCUSIÓN

4.1. Evaluación de detección de objetos en 3D

A continuación, veremos un ejemplo de punto de nubes sobre una imagen (figura 15) descargada de KITTI.

Figura 15. Fotografía página oficial de KITTY

Al aplicar el algoritmo run_demoVelodyne.m (Ubicado en el Apéndice A de este trabajo) obtendremos una nube de puntos de la siguiente forma en las figuras 16 y 17:

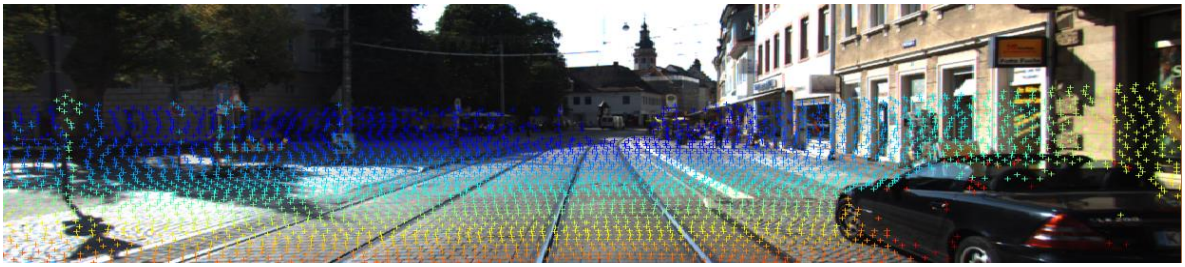


Figura 16. Nube de puntos

Manipulando el algoritmo en Matlab, veremos más puntos sobre la imagen, como se muestra a continuación.

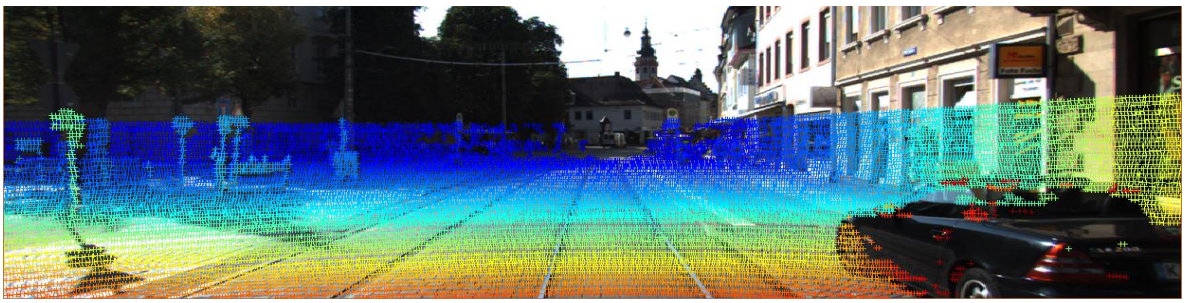


Figura 17. Nube de puntos

Adicionalmente, al aplicar el algoritmo run_demoVehiclePath.m (Ubicado en el apéndice B de este trabajo) obtendremos la ruta del vehículo en el tiempo como se aprecia en la figura 18.

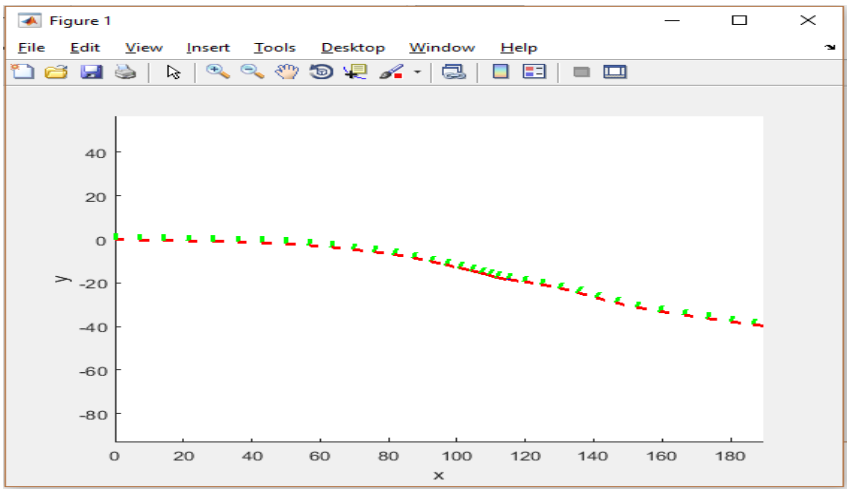


Figura 18. Ruta del vehículo

4.2. Etiquetado en Matlab

La idea de etiquetar es identificar y estar completamente seguro de la clase de un objeto; en caso de no estar seguro, se presiona el botón "*Don't save*". La etiqueta se muestra en la figura 19.

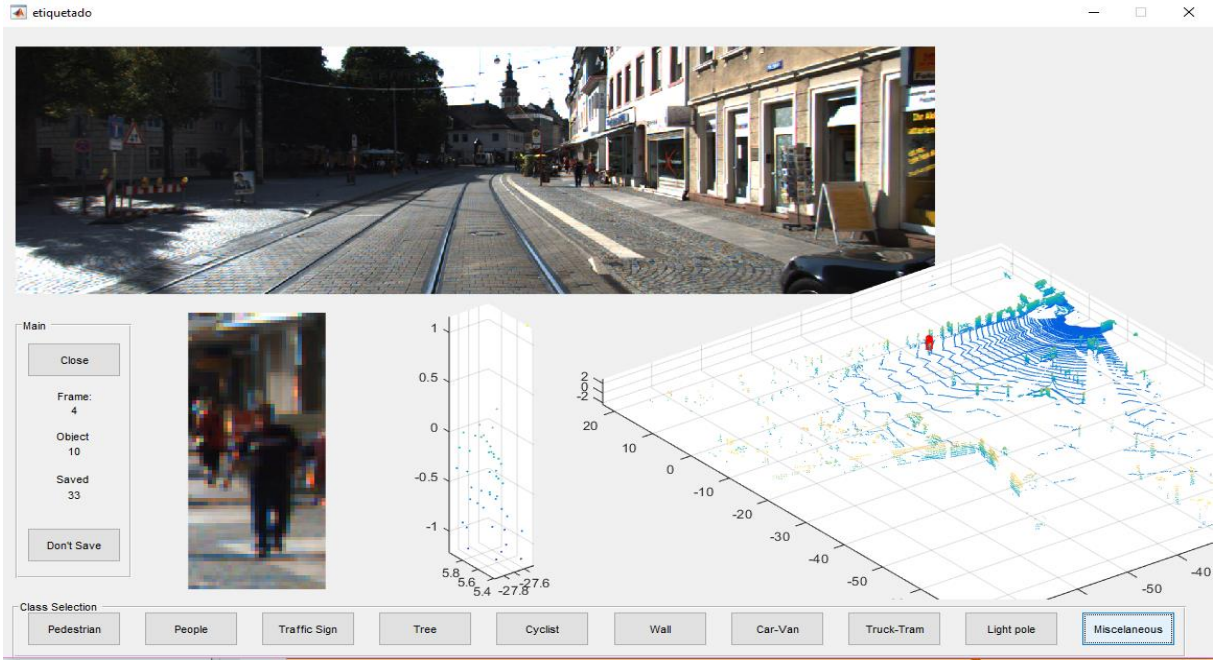


Figura 19. Etiquetado en Matlab

En la imagen se ven diferentes etiquetas, mencionemos brevemente cada una. En *misceláneos* se etiquetan cosas como carteles de publicidad, basureros y cosas que en general no entran en las categorías que definen. La siguiente etiqueta es *postes de luz*, cada que el programa detecte dicho objeto lo almacenamos, la siguiente etiqueta es *camión o camiones*, luego sigue *carro y/o van*, luego *paredes y/o muros*, luego *ciclistas*, *arboles*, *señales de tránsito*, *personas y senderos peatonales*. A veces el programa en Matlab puede resultar un poco lento por qué está procesando objetos que están fuera de la imagen, por lo que se debe esperar a que aparezca el nuevo objeto a etiquetar, sobre todo, con los últimos objetos del frame. A medida que se va alimentando el programa, la imagen avanza; quizás sea insuficiente el cambio que hace, pero cuando se almacena muchos datos de imágenes se ve el cambio, como si estuviera caminando en cámara lenta y analizando la imagen que aparece acorde al avance de la posición.

La etiqueta tiene varias características para un correcto funcionamiento a nivel de software.

Lo primero que se debe tener en cuenta a la hora de correr el programa son algunas funciones que se obtienen en la página web oficial de KITTI, como se muestra en la figura 20 (Urtasun, 2012).

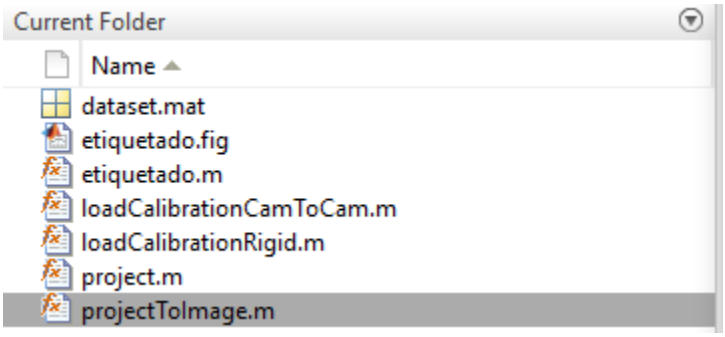


Figura 20. Funciones en Matlab para etiquetar la base de datos.

Como pueden observar, estas funciones deben estar en el Current folder del Matlab, *etiquetado.fig* y .m es el entorno donde analizamos cada una de las imágenes y secuencias del programa, tenemos también *loadCalibrationCamToCam.m* y *loadCalibrationRigid.m* que nos ayuda en el análisis de la imagen, su pixelado, y por su puesto su frame desde varios puntos de vista. Si alguna de estas funciones no está dentro del programa, Matlab no lo ejecutara correctamente y tendremos varios errores.

El etiquetado tiene varios frames: La imagen tomada por una cámara, nos muestra objetos alrededor. Debajo de la imagen aparecen 3 partes; la primera es el recorte de la imagen donde está el objeto que se va a analizar, luego vemos una figura en 3D que muestra solo puntos, dichos puntos representan los objetos más lejanos de la imagen y por ultimo vemos la imagen genérica, pero en 3D. A medida que analizamos cada tramo

vemos como sectoriza lo que captura en cada clic que damos, ya sea guardando o no guardando el fragmento analizado.

Veamos nuevamente el etiquetado, pero un poco más adelantado en la figura 21.

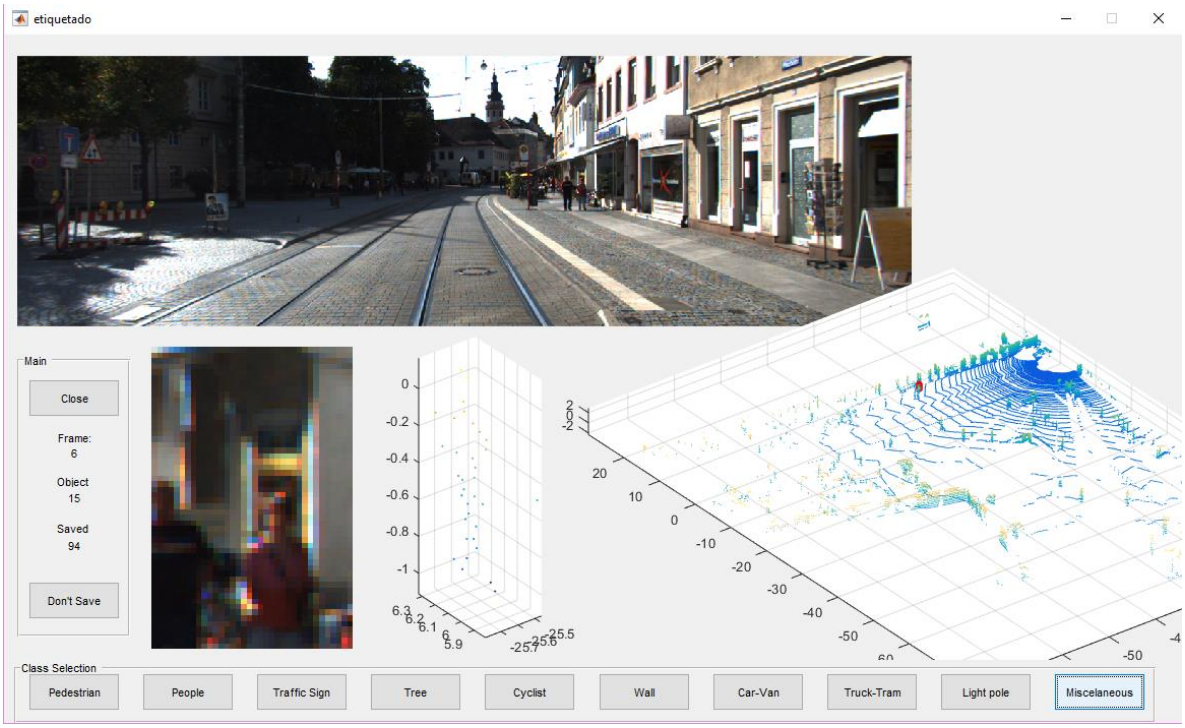


Figura 21 Etiquetado en Matlab más detallado

Como se observa en la figura 21, la imagen está un poco más adelantada con respecto a la imagen anterior.

Cuando terminamos el análisis de una imagen y se ha guardado correctamente las características de lo capturado, Matlab nos proporciona una nube de puntos de la imagen como se ve a continuación en la figura 22 y 23.

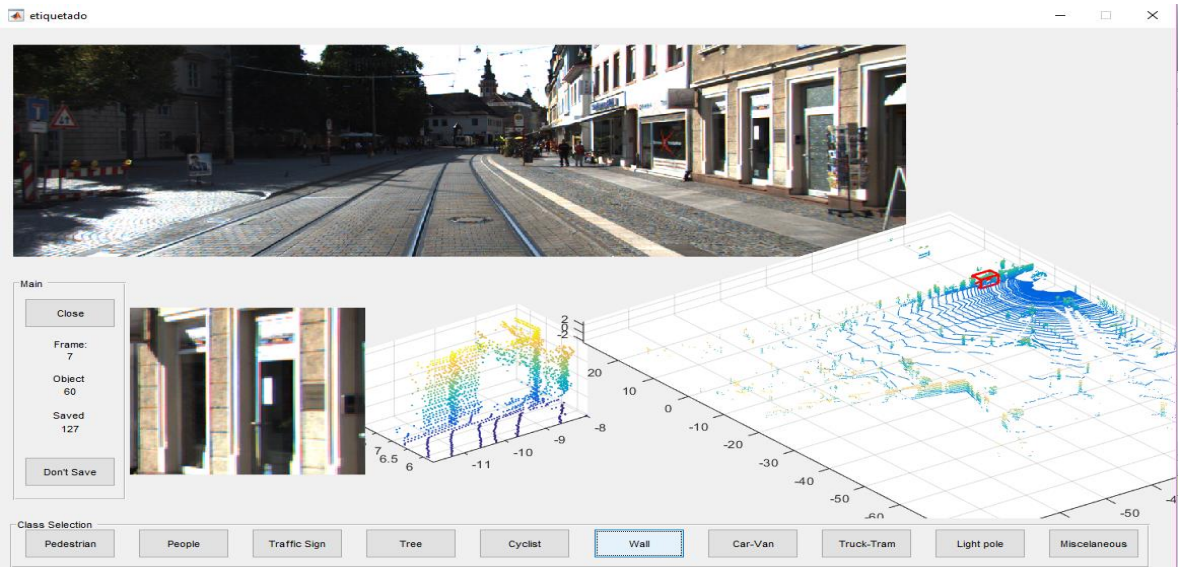


Figura 22. Etiquetado en Matlab más detallado

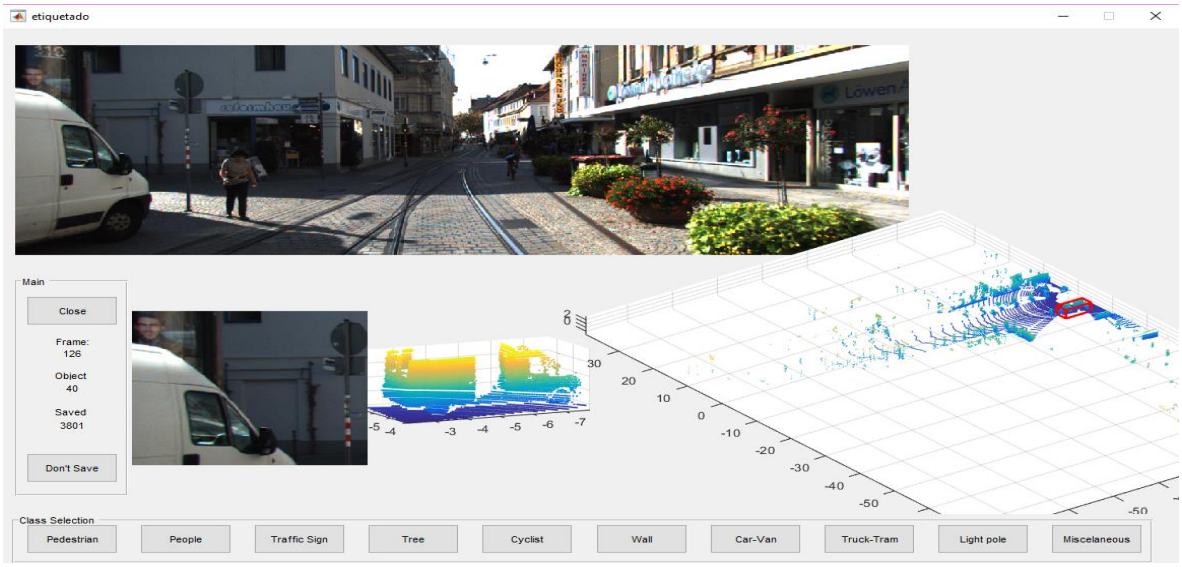


Figura 23. Etiquetado en Matlab más detallado

En el apéndice C encontraremos el código de los etiquetados y de las funciones de Matlab (Chapman, 2016).

4.3. PointNet

La nube de puntos es un tipo importante de estructura de datos geométricos. Debido a su formato irregular, la mayoría de los centros de investigación convierten los datos en cuadrículas de vóxeles en 3D. En este producto de laboratorio, utilizamos un nuevo tipo de red neuronal que consume directamente nubes de puntos. Esta red, llamada PointNet, proporciona una arquitectura unificada para aplicaciones que van desde la clasificación de objetos (peatones, ciclistas, arboles, señales de tránsito), la segmentación de partes, hasta el análisis semántico de escenas de acuerdo a la imagen actual en el ejecutable de Matlab. Aunque simple, PointNet es altamente eficiente y efectivo (Qi, 2017).

4.4. Resultados del Dataset en PointNet

A lo largo de este trabajo, se ha venido almacenando un Dataset de la captura de imágenes en Matlab, como se mencionó anteriormente, el etiquetado nos proporciona información exacta de una imagen donde se muestran factores como carros, señales de tránsito, peatones, ciclistas, tiendas, letreros, senderos peatonales, entre otros. El etiquetado tiene estas opciones para determinar si una fracción de la imagen corresponde a un grupo determinado.

A continuación, se detalla como quedo seccionado cada grupo en las etiquetas en la figura 24.

Figura 24. Menú del Etiquetado en Matlab

Cada que seleccionamos un determinado grupo, la imagen corta otra fracción para almacenarla nuevamente, como podemos ver en el recuadro rojo de la figura 25 y la imagen del recuadro en la figura 26.

Figura 25. Imagen completa

Figura 26. Imagen sectorizada con nube de puntos

Vemos en la imagen algunas características, como por ejemplo un árbol con flores rojas y amarillas, también vemos una señora de blusa blanca con una bicicleta, adicional, vemos en la parte superior derecha un letrero de alguna tienda o almacén. La magia de la construcción de la base de datos es tomar la decisión de donde se va almacenar, alguien podría elegir que es un árbol, pero también se podría elegir que es una persona o incluso un ciclista.

El gráfico al lado de la porción de imagen, muestra la nube de puntos de la porción y el que está más a la derecha muestra la imagen total en 3D. Cuando se toma la decisión veremos que la porción de imagen cambia y en la imagen 3D el cubo de color rojo cambia de posición como se aprecia en la figura 27.

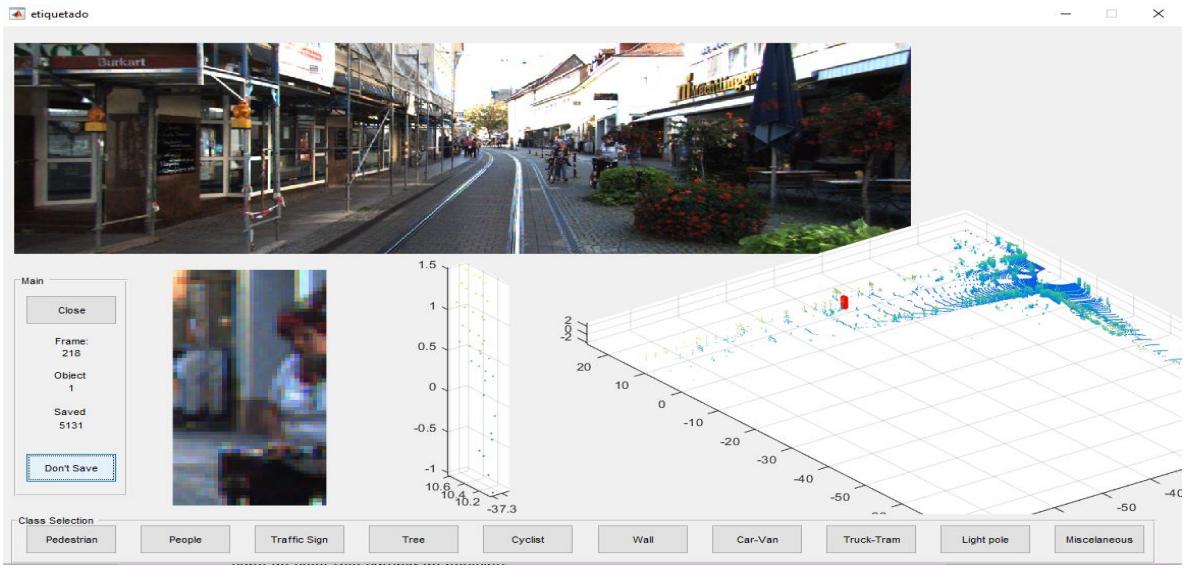


Figura 27. Vista general del etiquetado en otra posición

Observe que es la misma imagen, pero ubicada en una porción diferente, así, a medida que vamos avanzando poco a poco en la construcción de la base de datos le ayudamos a la red neuronal a identificar un peatón en sus diferentes formas: sentado, parado, en bicicleta, de espaldas, de lado cruzando un semáforo.

Al ejecutar la base de datos (Dataset) en Python utilizando como sistema operativo Linux-Ubuntu y la librería de TensorFlow nos arroja en siguiente resultado que se observa en la figura 28.

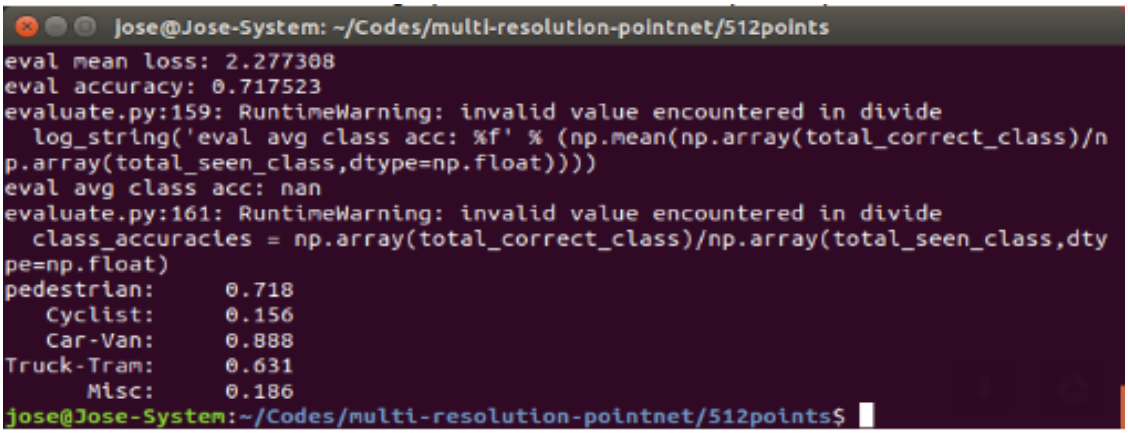


Figura 28. Resultado final en TensorFlow bajo Linux en Python

Mostrando en números reales la cantidad de objetos que detecto en el recorrido de la base de datos almacenada en Matlab durante la selección. Para tener una información más precisa y más acertada se debe alimentar la base de datos un par de días o semanas más, este resultado es bajo la cantidad de 5130 imágenes, se deberá entonces almacenar al menos unas 15000 imágenes para ver esos porcentajes más cercanos a un 100% de lo esperado.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

5.1. Conclusiones

En este trabajo de laboratorio, revisamos el estado del arte de los métodos de reconocimiento de objetos. Después de este estudio, podemos afirmar que el aprendizaje profundo “Deep Learning” está superando a los paradigmas tradicionales; además, observamos que los métodos basados en datos 3D generalmente superan a los basados en 2D gracias a la dimensión adicional de la información. Esta revisión estableció la motivación para este trabajo: explorar las posibilidades de la representación de objetos 3D reconociendo que tipo de clase es y qué características tiene, empleando una RNA.

La construcción de la base de datos se debe “alimentar” con una cantidad considerable de información con intención de reducir el porcentaje de falla por interpretación. En el proceso de “alimentación” de la base de datos de este proyecto, se ingresaron 5130 imágenes. Luego de evaluar el resultado, se puede concluir que es necesaria una base de datos con un mayor número de imágenes para lograr un desempeño más alto en el proceso de “aprendizaje”.

5.2. Recomendaciones

Trabajar en este ambiente suele ser tedioso por la falta de herramientas informáticas disponibles en la web, por ello es importante contar con una GPU y un equipo de buena calidad con sistema operativo adicional (Linux-Ubuntu) para experimentar este tipo de RNA, adicionalmente, si no se cuenta con recursos propios deberán utilizar los equipos disponibles en el ITM para este tipo de trabajos.

Continuar construyendo la base de datos en el Dataset, pues las RNA profundas debemos alimentarlas con volúmenes de información muy grandes.

5.3. Trabajo futuro

El presente trabajo deja las puertas abiertas para los estudiantes de Ingeniería Electrónica y posiblemente de Telecomunicaciones, además, los estudiantes de Maestría en Automatización y Control en la mejora de la aplicabilidad del reconocimiento de peatones. En la actualidad adquirir un buen sensor de Velodyne es costoso, por lo que alimentar las simulaciones (a nivel de código) nos puede llevar por caminos interesantes llenos de ideas con el propósito de mejorar algunos aspectos del prototipo utilizado; este trabajo es el inicio de un gran proyecto que llenara de orgullo a nuestro ITM y seguramente a nuestra ciudad Medellín colocándola en la vanguardia de ciudad innovadora pues la Inteligencia Artificial es un tema que ya camina en la vida de casi todo el mundo.

6. REFERENCIAS

Raúl Gonzáles Duque, (2010), Python para todos, 1ª edición, 1-160, Creative Commons, Barcelona España.

Google, 2015. TensorFlow, Estados Unidos. https://www.tensorflow.org/

Velodyne LiDAR, (2016), Estados Unidos, San Jose. http://www.velodynelidar.com/

Urtasun, Andreas Geiger and Philip Lenz and Raquel, (2012), Estados Unidos, Conference on Computer Vision and Pattern Recognition (CVPR), http://www.cvlibs.net/datasets/kitti/index.php

Torres, Alvaro Casas Martinez y Alejandro Jose del Real, (2017), Reconocimiento de imágenes con redes convolucionales en C, Sevilla-España, Escuela técnica superior de Ingeniería.

Albino, José Martin Flores, (2018), Deep Learning para la Detección de Peatones y Vehículos, Toluca, México, Universidad Autónoma del estado de México.

Charles R. Qi, (2007), Estados Unidos, https://github.com/charlesq34/pointnet/tree/master/models, Stanford University.

Miller Steven, (2015), Mind: How to Build a Neural Network (Part One), https://stevenmiller888.github.io

Juan José Montaño, (2002), Redes Neuronales Artificiales Aplicadas al Análisis de Datos, Tesis Doctoral, Universidad de las Islas Baleares, Palma de Mallorca.

Quora, (2015), Estados Unidos, https://www.quora.com/Is-Linux-better-than-Windows-for-using-TensorFlow

Chapman, Sthephen J, (2016), MATLAB PROGRAMMING FOR ENGINEERS, Estados Unidos, International Student Edition.

Charles R. Qi, (2017), Estados Unidos, California, PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation.

7. APÉNDICE

Los códigos escritos (Apéndice A y B) en esta sección del trabajo de laboratorio requieren de una base de datos que debe ser descargada de la página web KITTI de acuerdo sea la necesidad de trabajo en las RNA empleando TensorFlow como técnica. (Urtasun, 2012)

Apéndice A

```
function run_demoVelodyne (base_dir,calib_dir)
% KITTI RAW DATA DEVELOPMENT KIT
%
% Demonstrates projection of the velodyne points into the image plane
%
% Input arguments:
% base_dir .... absolute path to sequence base directory (ends with _sync)
% calib_dir ... absolute path to directory that contains calibration files

% clear and close everything
close all; dbstop error; clc;
disp('======= KITTI DevKit Demo =======');

% options (modify this to select your sequence)
if nargin<1
  base_dir  = 'C:\Users\Personal\Desktop\Base de
datos\2011_09_26_drive\2011_09_26\2011_09_26_drive_0091_sync';
end
if nargin<2
  calib_dir = 'C:\Users\Personal\Desktop\Base de
datos\2011_09_26_drive\2011_09_26';
end
cam       = 2; % 0-based index
frame     = 0; % 0-based index

% load calibration
calib = loadCalibrationCamToCam(fullfile(calib_dir,'calib_cam_to_cam.txt'));
Tr_velo_to_cam =
loadCalibrationRigid(fullfile(calib_dir,'calib_velo_to_cam.txt'));

% compute projection matrix velodyne->image plane
R_cam_to_rect = eye(4);
R_cam_to_rect(1:3,1:3) = calib.R_rect{1};
P_velo_to_img = calib.P_rect{cam+1}*R_cam_to_rect*Tr_velo_to_cam;

% load and display image
img = imread(sprintf('%s/image_%02d/data/%010d.png',base_dir,cam,frame));
```

```
fig = figure('Position',[20 100 size(img,2) size(img,1)]); axes('Position',[0
0 1 1]);
imshow(img); hold on;

% load velodyne points
fid =
fopen(sprintf('%s/velodyne_points/data/%010d.bin',base_dir,frame),'rb');
velo = fread(fid,[4 inf],'single')';
%velo = velo(1:5:end,:); % remove every 5th point for display speed
fclose(fid);
% pcshow(velo(:,1:3))
% remove all points behind image plane (approximation
idx = velo(:,1)<5;
velo(idx,:) = [];

% project to image plane (exclude luminance)
velo_img = project(velo(:,1:3),P_velo_to_img);

% plot points
cols = jet;
for i=1:size(velo_img,1)
  col_idx = round(64*5/velo(i,1));

plot(velo_img(i,1),velo_img(i,2),'o','LineWidth',4,'MarkerSize',1,'Color',col
s(col_idx,:));
end
```

Apéndice B

```
function run_demoVehiclePath (base_dir)
% KITTI RAW DATA DEVELOPMENT KIT
%
% Plots OXTS poses of a sequence
%
% Input arguments:
% base_dir .... absolute path to sequence base directory (ends with _sync)

% clear and close everything
clear all; close all; dbstop error; clc;
disp('======= KITTI DevKit Demo =======');

% sequence base directory
if nargin<1
  base_dir = 'C:\Users\Personal\Desktop\Base de
datos\2011_09_26_drive\2011_09_26\2011_09_26_drive_0091_sync';
end

% load oxts data
oxts = loadOxtsliteData(base_dir);

% transform to poses
pose = convertOxtsToPose(oxts);
```

```
% plot every 10'th pose
figure; hold on; axis equal;
l = 3; % coordinate axis length
A = [0 0 0 1; l 0 0 1; 0 0 0 1; 0 l 0 1; 0 0 0 1; 0 0 l 1]';
for i=1:10:length(pose)
  B = pose{i}*A;
  plot3(B(1,1:2),B(2,1:2),B(3,1:2),'-r','LineWidth',2); % x: red
  plot3(B(1,3:4),B(2,3:4),B(3,3:4),'-g','LineWidth',2); % y: green
  plot3(B(1,5:6),B(2,5:6),B(3,5:6),'-b','LineWidth',2); % z: blue
end
xlabel('x');
ylabel('y');
zlabel('z');
```

Apéndice C

Etiquetado.m

```
function varargout = etiquetado(varargin)
% ETIQUETADO MATLAB code for etiquetado.fig
%      ETIQUETADO, by itself, creates a new ETIQUETADO or raises the existing
%      singleton*.
%
%      H = ETIQUETADO returns the handle to a new ETIQUETADO or the handle to
%      the existing singleton*.
%
%      ETIQUETADO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ETIQUETADO.M with the given input
arguments.
%
%      ETIQUETADO('Property','Value',...) creates a new ETIQUETADO or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before etiquetado_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to etiquetado_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help etiquetado

% Last Modified by GUIDE v2.5 15-Jul-2018 15:02:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @etiquetado_OpeningFcn, ...
                   'gui_OutputFcn',  @etiquetado_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
```

```
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before etiquetado is made visible.
function etiquetado_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to etiquetado (see VARARGIN)
global data_dir;
global FrameCount;
global saved;
global obj;
global ObjCount;
global P_velo_to_img;
% Choose default command line output for etiquetado
handles.output = hObject;
data_dir  =
'C:\Users\user\Desktop\Bases_de_Datos\2011_09_26_drive\2011_09_26\2011_09_26_
drive_0091_sync';
calib_dir =
'C:\Users\user\Desktop\Bases_de_Datos\2011_09_26_drive\2011_09_26';
% load calibration
calib = loadCalibrationCamToCam(fullfile(calib_dir,'calib_cam_to_cam.txt'));
Tr_velo_to_cam =
loadCalibrationRigid(fullfile(calib_dir,'calib_velo_to_cam.txt'));
% compute projection matrix velodyne->image plane
R_cam_to_rect = eye(4);
R_cam_to_rect(1:3,1:3) = calib.R_rect{1};
P_velo_to_img = calib.P_rect{3}*R_cam_to_rect*Tr_velo_to_cam;

if exist('dataset.mat','file')
    load dataset.mat
    set(handles.text6,'String',ObjCount);
    set(handles.text7,'String',saved);
    set(handles.text5,'String',FrameCount);
else
    saved=1;
    FrameCount=1;
    ObjCount=1;
    save 'dataset.mat' saved FrameCount ObjCount
end

NewFrame(handles);
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes etiquetado wait for user response (see UIRESUME)
% uiwait(handles.figure1);

function NewObject (handles)
global P_velo_to_img;
global ObjCount;
global FrameCount;
global velo;
global metersBBox;
global img;
global saved;
global obj
siz=size(metersBBox);
if ObjCount<=siz(1)
    i=ObjCount;
    axes(handles.axes1);
    hold off;
    imshow(img);
    hold on;

    xmi=metersBBox(i,1);
    xma=(metersBBox(i,1)+metersBBox(i,3));
    xpos=find((velo(:,1)>xmi)&(velo(:,1)<xma));
    ymi=metersBBox(i,2);
    yma=(metersBBox(i,2)+metersBBox(i,4));
    ypos=find((velo(xpos,2)>ymi)&(velo(xpos,2)<yma));
    obj(saved).points=velo(xpos(ypos),1:3)';
    maxobj=max(obj(saved).points(3,:));
    minobj=min(obj(saved).points(3,:));
    altura=maxobj-minobj;
    if altura>0.7
        axes(handles.axes4);
        try
            pcshow(obj(saved).points','MarkerSize',16);
        catch
        end

        axes(handles.axes2);
        hold off
        try
            pcshow(velo(:,1:3));
        catch
        end
        hold on;
        numpoints=size(ypos);
        obj(saved).numpoints=numpoints(1);

obj(saved).distance=sqrt(obj(saved).points(1,1)^2+obj(saved).points(2,1)^2);
        obj(saved).reflex=velo(xpos(ypos),4)';
```

```
    %Draw a 3D Bounding Box around each object
        bbx=[metersBBox(i,1) metersBBox(i,1)+metersBBox(i,3)
metersBBox(i,1)+metersBBox(i,3) metersBBox(i,1)...
             metersBBox(i,1) metersBBox(i,1)+metersBBox(i,3)
metersBBox(i,1)+metersBBox(i,3) metersBBox(i,1)];
        bby=[metersBBox(i,2) metersBBox(i,2) metersBBox(i,2)+metersBBox(i,4)
metersBBox(i,2)+metersBBox(i,4)...
              metersBBox(i,2)+metersBBox(i,4) metersBBox(i,2)+metersBBox(i,4)
metersBBox(i,2) metersBBox(i,2)];
        bbz=[maxobj maxobj maxobj maxobj minobj minobj minobj minobj];
        bbx1=bbx([1 4 5 8 1]);
        bbx2=bbx([2 3 2 3 2]);
        bby1=bby([2 3 4 7 2]);
        bbz1=bbz([3 4 5 6 3]);
        axes(handles.axes2);
        plot3(bbx,bby,bbz,'LineWidth',2,'Color','r')
        plot3(bbx1,bby1,bbz1,'LineWidth',2,'Color','r')
        plot3(bbx2,bby1,bbz1,'LineWidth',2,'Color','r')

        bb_corners=[min(bbx) min(bby) min(bbz); min(bbx) max(bby) max(bbz)];
        bb_img = project(bb_corners,P_velo_to_img);
        if (bb_img(1,2)-6)>=1 && (bb_img(1,1)-
6)>=1&&(bb_img(2,2)+12)<=375&&bb_img(2,1)+12<=1242
            axes(handles.axes3);
            imshow(img(bb_img(1,2)-6:bb_img(2,2)+12,bb_img(1,1)-
6:bb_img(2,1)+12,:))
        else
            ObjCount=ObjCount+1;
            set(handles.text6,'String',ObjCount);
            save 'dataset.mat' ObjCount '-append';
            NewObject (handles)
        end
    else
        ObjCount=ObjCount+1;
        set(handles.text6,'String',ObjCount);
        save 'dataset.mat' ObjCount '-append';
        NewObject (handles)
    end
else
    ObjCount=1;
    FrameCount=FrameCount+1;
    set(handles.text6,'String',ObjCount);
    set(handles.text5,'String',FrameCount);
    save 'dataset.mat' ObjCount FrameCount '-append';
    NewFrame(handles)
end


function NewFrame(handles)
    global data_dir;
    global FrameCount;
    global velo;
    global metersBBox;
    global img
```

```
    img =
imread(sprintf('%s/image_%02d/data/%010d.png',data_dir,2,FrameCount));

    fid =
fopen(sprintf('%s/velodyne_points/data/%010d.bin',data_dir,FrameCount),'rb');
    velo = fread(fid,[4 inf],'single')';
    fclose(fid);
    %
    errorPoints=(velo(:,3)>-2.5);
    velo=velo(errorPoints,:);
    errorPoints=(velo(:,1)>0);
    velo=velo(errorPoints,:);
    velo=[-velo(:,1) -velo(:,2) velo(:,3) velo(:,4)];

    % vpointpos=find(velopoints(:,1)>0);
    % velo=velopoints(vpointpos,1:4);
    %% Modeling the ground plane and erase it
    ptCloud=pointCloud(velo(:,1:3));
    maxDistance = 0.1;
    referenceVector = [0,0,1];
    maxAngularDistance = 1;
    [~,inlierIndices,outlierIndices] =
pcfitplane(ptCloud,maxDistance,referenceVector,maxAngularDistance);
    floorLess = select(ptCloud,outlierIndices);
    si=size (inlierIndices);
    while (si(1)>9000)          %%%justificar el numero de puntos minimo en
el plano
        [~,inlierIndices,outlierIndices] =
pcfitplane(floorLess,maxDistance,referenceVector,maxAngularDistance);
        floorLess = select(floorLess,outlierIndices);
        si=size (inlierIndices);
    end

    %% Create the ocupation matrix (20cm by 20cm) form x and y coordinates
    Xmax=max(velo(:,1));
    Ymax=max(velo(:,2));
    Xmin=min(velo(:,1));
    Ymin=min(velo(:,2));
    numPoints=size(floorLess.Location);
    occGrid=zeros(ceil((Ymax-Ymin)/0.2)+1,ceil((Xmax-Xmin)/0.2)+1);
    for i=1:numPoints
        occGrid(ceil((floorLess.Location(i,2)-
Ymin)/0.2)+1,ceil((floorLess.Location(i,1)-Xmin)/0.2)+1)=
occGrid(ceil((floorLess.Location(i,2)-
Ymin)/0.2)+1,ceil((floorLess.Location(i,1)-Xmin)/0.2)+1)+1;
    end

    % Retire the spaces (20cm by 20cm) where there are less than 4 points
    occGrid(occGrid<6)=0;

    %% Find conected regions and erase those thar only have one pixel(20 by
20cm)
    bwGrid=zeros(size(occGrid));
```

```
    bwGrid(occGrid>0)=1;
    blobsGrid = bwconncomp(bwGrid,8);

    for i=1:blobsGrid.NumObjects
        if size(blobsGrid.PixelIdxList{i})==1
            occGrid(blobsGrid.PixelIdxList{i})=0;
        end
    end
    % figure(3)
    % imshow(occGrid*10)
    bwGrid=zeros(size(occGrid));
    bwGrid(occGrid>0)=1;
    blobsGrid = bwconncomp(bwGrid,8);

    %% Defining the bounding boxes and transforming its coordinates to meters
    bBoxes=regionprops(blobsGrid,'BoundingBox');
    bbSize=size(bBoxes);
    for i=1:bbSize(1)
        metersBBox(i,1)=((bBoxes(i).BoundingBox(1)-2)*0.2)+Xmin;
        metersBBox(i,3)=((bBoxes(i).BoundingBox(3)+1)*0.2);
        metersBBox(i,2)=((bBoxes(i).BoundingBox(2)-2)*0.2)+Ymin;
        metersBBox(i,4)=((bBoxes(i).BoundingBox(4)+1)*0.2);
    end
    NewObject (handles)

% --- Outputs from this function are returned to the command line.
function varargout = etiquetado_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pedestrian.
function pedestrian_Callback(hObject, eventdata, handles)
% hObject    handle to pedestrian (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='pedestrian';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)
```

```
% --- Executes on button press in trafficsign.
function trafficsign_Callback(hObject, eventdata, handles)
% hObject    handle to trafficsign (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='sign';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in tree.
function tree_Callback(hObject, eventdata, handles)
% hObject    handle to tree (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='tree';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in cyclist.
function cyclist_Callback(hObject, eventdata, handles)
% hObject    handle to cyclist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='cyclist';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in wall.
function wall_Callback(hObject, eventdata, handles)
% hObject    handle to wall (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global ObjCount
global obj
global saved
obj(saved).label='wall';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in carvan.
function carvan_Callback(hObject, eventdata, handles)
% hObject    handle to carvan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='car-van';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in lightpole.
function lightpole_Callback(hObject, eventdata, handles)
% hObject    handle to lightpole (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='light pole';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in misc.
function misc_Callback(hObject, eventdata, handles)
% hObject    handle to misc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='misc';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
```

```
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in nosave.
function nosave_Callback(hObject, eventdata, handles)
% hObject    handle to nosave (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
save 'dataset.mat' ObjCount '-append';
NewObject (handles)

% --- Executes on button press in trucktram.
function trucktram_Callback(hObject, eventdata, handles)
% hObject    handle to trucktram (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='truck-tram';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)

% --- Executes on button press in people.
function people_Callback(hObject, eventdata, handles)
% hObject    handle to people (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ObjCount
global obj
global saved
obj(saved).label='peoplo';
ObjCount=ObjCount+1;
set(handles.text6,'String',ObjCount);
set(handles.text7,'String',saved);
save 'dataset.mat' saved obj ObjCount '-append';
saved=saved+1;
NewObject (handles)


% --- Executes on button press in close.
function close_Callback(hObject, eventdata, handles)
% hObject    handle to close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
close(handles.output);
```

loadCalibrationCamtoCam.m

```
function calib = loadCalibrationCamToCam(filename)

% open file
fid = fopen(filename,'r');

if fid<0
  calib = [];
  return;
end

% read corner distance
calib.cornerdist = readVariable(fid,'corner_dist',1,1);

% read all cameras (maximum: 100)
for cam=1:100

  % read variables
  S_       = readVariable(fid,['S_' num2str(cam-1,'%02d')],1,2);
  K_       = readVariable(fid,['K_' num2str(cam-1,'%02d')],3,3);
  D_       = readVariable(fid,['D_' num2str(cam-1,'%02d')],1,5);
  R_       = readVariable(fid,['R_' num2str(cam-1,'%02d')],3,3);
  T_       = readVariable(fid,['T_' num2str(cam-1,'%02d')],3,1);
  S_rect_  = readVariable(fid,['S_rect_' num2str(cam-1,'%02d')],1,2);
  R_rect_  = readVariable(fid,['R_rect_' num2str(cam-1,'%02d')],3,3);
  P_rect_  = readVariable(fid,['P_rect_' num2str(cam-1,'%02d')],3,4);

  % calibration for this cam completely found?
  if isempty(S_) || isempty(K_) || isempty(D_) || isempty(R_) || isempty(T_)
    break;
  end

  % write calibration
  calib.S{cam} = S_;
  calib.K{cam} = K_;
  calib.D{cam} = D_;
  calib.R{cam} = R_;
  calib.T{cam} = T_;

  % if rectification available
  if ~isempty(S_rect_) && ~isempty(R_rect_) && ~isempty(P_rect_)
    calib.S_rect{cam} = S_rect_;
    calib.R_rect{cam} = R_rect_;
    calib.P_rect{cam} = P_rect_;
  end
end

% close file
fclose(fid);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function A = readVariable(fid,name,M,N)

% rewind
fseek(fid,0,'bof');

% search for variable identifier
success = 1;
while success>0
  [str,success] = fscanf(fid,'%s',1);
  if strcmp(str,[name ':'])
    break;
  end
end

% return if variable identifier not found
if ~success
  A = [];
  return;
end

% fill matrix
A = zeros(M,N);
for m=1:M
  for n=1:N
    [val,success] = fscanf(fid,'%f',1);
    if success
      A(m,n) = val;
    else
      A = [];
      return;
    end
  end
end
```

loadCalibrationRigid.m

```
function Tr = loadCalibrationRigid(filename)

% open file
fid = fopen(filename,'r');

if fid<0
  error(['ERROR: Could not load: ' filename]);
end

% read calibration
R  = readVariable(fid,'R',3,3);
T  = readVariable(fid,'T',3,1);
Tr = [R T;0 0 0 1];

% close file
```

```
fclose(fid);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function A = readVariable(fid,name,M,N)

% rewind
fseek(fid,0,'bof');

% search for variable identifier
success = 1;
while success>0
  [str,success] = fscanf(fid,'%s',1);
  if strcmp(str,[name ':'])
    break;
  end
end

% return if variable identifier not found
if ~success
  A = [];
  return;
end

% fill matrix
A = zeros(M,N);
for m=1:M
  for n=1:N
    [val,success] = fscanf(fid,'%f',1);
    if success
      A(m,n) = val;
    else
      A = [];
      return;
    end
  end
end
```

project.m

```
function p_out = project(p_in,T)

% dimension of data and projection matrix
dim_norm = size(T,1);
dim_proj = size(T,2);

% do transformation in homogenuous coordinates
p2_in = p_in;
if size(p2_in,2)<dim_proj
  p2_in(:,dim_proj) = 1;
end
p2_out = (T*p2_in')';
```

```
% normalize homogeneous coordinates:
p_out = p2_out(:,1:dim_norm-1)./(p2_out(:,dim_norm)*ones(1,dim_norm-1));
```

projectToImage.m

```
function pts_2D = projectToImage(pts_3D, K)
% PROJECTTOIMAGE projects 3D points in given coordinate system in the image
% plane using the given calibration matrix K.

% project in image
pts_2D = K * pts_3D(1:3,:);

% scale projected points
pts_2D(1,:) = pts_2D(1,:)./pts_2D(3,:);
pts_2D(2,:) = pts_2D(2,:)./pts_2D(3,:);
pts_2D(3,:) = [];
```

8. ANEXOS

8.1. Instalación de TensorFlow en una GPU

- **Instalación de máquina virtual:**

Se debe instalar una máquina virtual, bien sea Virtual Box o VMware con buenos recursos de maquina (procesador, memoria RAM y bus de datos)

Realizar una buena configuración de la máquina virtual instalada orientado a que esta última posea acceso a internet para la instalación de librerías necesarias en el software por instalar. En la figura 29 se observa la instalación de varios sistemas operativos.

Figura 29. Virtual Box

- **Instalación de Ubuntu 14.04**

Luego de instalada la máquina virtual y estar en ejecución; se debe ingresar la imagen o archivo .iso del sistema operativo **Ubuntu 14.04**. La máquina virtual está en capacidad de leer este tipo de archivos y ejecutarlo.

Durante este proceso, se debe realizar una debida instalación como la asignación apropiada de memoria swap y espacios en disco necesario en los directorios requeridos para que el software se ejecute sin problemas.

En la figura 30, se aprecia el entorno del sistema operativo Ubuntu

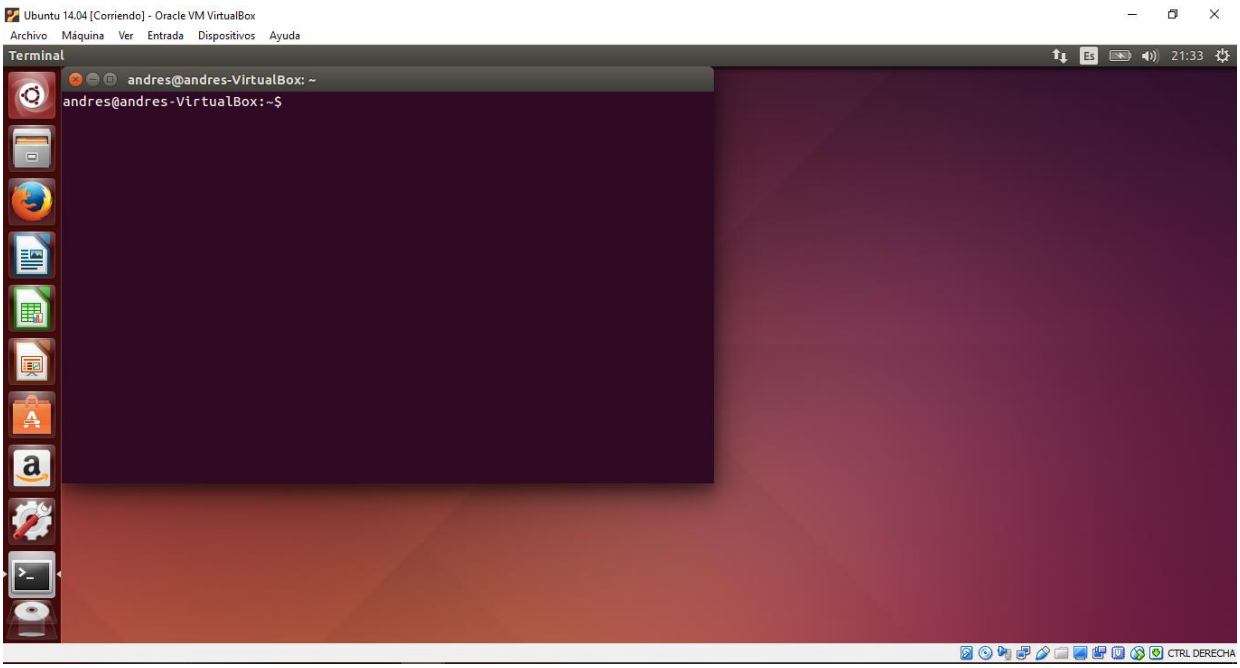


Figura 30. Ubuntu-14.04

- **Verificación Python**.

Solo basta con escribir la palabra Python en la terminal del sistema operativo Ubuntu para validar que esté instalado y cuál es su versión como se ve en la figura 31:

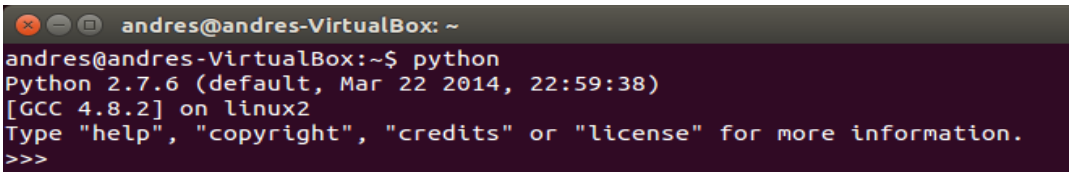


Figura 31. Python

Se procede a validar en Python que el paquete *pip* esté instalado con el comando: **pip -V**, sino está instalado; se debe proceder a ejecutar el siguiente comando en la terminal: **sudo apt-get install python3-pip** como se observa en la figura 32

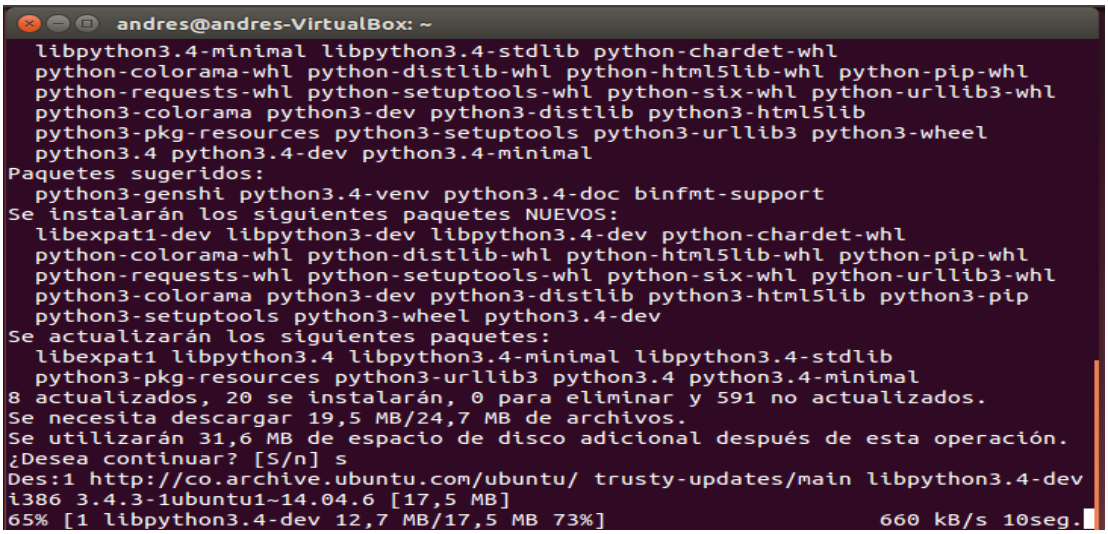


Figura 32. Instalación de paquetes Ubuntu-14.04

Dependiendo de la versión de Python se procede con la instalación del *pip* de la siguiente forma:

```
$ sudo apt-get install python-pip python-dev python-virtualenv   # for Python 2.7
$ sudo apt-get install python3-pip python3-dev python-virtualenv # for Python 3.n
```

Según la página web referenciada debemos generar los directorios apropiados para así poder instalar el TensorFlow según los pasos que indica dicha referencia. (Developers, 2018)

Luego de terminado el proceso de instalación según parámetros dispuestos en la guía referenciada. Debemos instalar una **GPU** (Graphic Process Unit) para mejorar o independizar el procesamiento que implica una imagen de la CPE de la máquina virtual. (Developers, 2018)

```
$ nvidia-docker run -it -p hostPort:containerPort TensorFlowGPUImage
```

```
$ nvidia-docker run -it tensorflow/tensorflow:latest-gpu bash
```

```
$ nvidia-docker run -it -p 8888:8888 tensorflow/tensorflow:latest-gpu
$ nvidia-docker run -it -p 8888:8888 tensorflow/tensorflow:0.12.1-gpu
```

Validar instalación y correr un procedimiento sencillo dispuesto en la referencia. (Developers, 2018).

```
# Python
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

Y el resultado debe ser:

```
Hello, TensorFlow!
```

Al realizar todos estos pasos podrás usar la poderosa herramienta de TensorFlow y disfrutar de los beneficios que esta trae.