Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

TRACKING DE PERSONAS EN ENTORNOS INDUSTRIALES ROBOTIZADOS CON IMÁGENES RGB-D

Wilson Andrés Vargas Rojas

Ingeniería Mecatrónica

Director(es) del trabajo de grado

Carlos Andrés Madrigal

INSTITUTO TECNOLÓGICO METROPOLITANO

2018-07-31

RESUMEN

La detección de objetos es una de las aplicaciones más comunes en la visión artificial, las imágenes son procesadas para obtener relaciones que describen los objetos y su tipo, llamados clases. La detección en video es similar, sin embargo, tener un seguimiento del objeto, es otra tarea aún más compleja. Con el avance tecnológico, las imágenes y video dejaron de ser 2D y adquirieron tridimensionalidad, con un nuevo canal que estima la profundidad de los objetos.

Este trabajo pretende analizar los métodos existentes para el seguimiento de personas en imágenes de profundidad y evaluar su rendimiento en entornos industriales robotizados.

Palabras clave: RGBD, Depth, Tracking, Kinect, Detección, Visión Artificial, Aprendizaje Profundo, Nube de puntos, Deep Learning, detección de peatones.

RECONOCIMIENTOS

La realización de este trabajo fue posible al acompañamiento del Profesor Carlos Andrés Madrigal, las recomendaciones del profesor Mauricio Arias y de todo el grupo de investigación del Laboratorio de óptica, fotónica y visión artificial, familiares y amigos que me motivaron y apoyaron en su desarrollo.

ACRÓNIMOS

RGBD Canales de color y profundidad Red, Green, Blue, Depth (Rojo, Verde, Azul, Profundidad)

HOG Histogram of Oriented Gradient (Histogramas Orientados a Gradientes)

ROI Region Of Interest (Regiones de Interés)

DAG Directed Acyclic Graph (Grafo Acíclico Dirigido)

SVM Support Vector Machine (Máquina de Soporte Vectorial)

SSD Single Shot Detector (Detector de Disparo Único)

CNN Convolutional Neural Network (Redes Neuronales Convolucionales)

R-CNN Region Convolutional Neural Network (Redes Neuronales Convolucionales de Región)

YOLO You Only Look Once (Solo se mira una vez)

TABLA DE CONTENIDO

1. INTRODUCCIÓN 7
1.1. Generalidades 7
1.2. Objetivos 8
1.2.1. Objetivo general 8
1.2.2. Objetivos específicos 8
1.3. Organización del trabajo 8
2. MARCO TEÓRICO 9
2.1. Conceptos 9
2.2. Estado del arte 13
2.3. Flujo de trabajo 15
3. METODOLOGÍA 18
3.1. Software usado 18
3.2. Recursos usados 18
3.3. Implementación 18
3.4. Datasets 19
3.5. Entrenamiento 21
3.5.1. Retinanet 21
3.5.2. YOLO v3 23
3.5.3. SSD 23
3.5.4. PointNet 24
4. RESULTADOS Y DISCUSIÓN 25
4.1. Resultados 25
4.1.1. Retinanet 25
4.1.2. YOLO v3 27
4.1.3. SSD 31
4.1.4. PointNet 34

4.2. Tabla de resultados 35

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO 36

5.1. Conclusiones 36

5.1.1. Conclusión general 36

5.1.2. Conclusiones complementarias 36

5.2. Recomendaciones. 37

5.3. Trabajo futuro 37

REFERENCIAS 38

APÉNDICE 42

1. INTRODUCCIÓN

1.1. Generalidades:

El seguimiento de objetos a través de la segmentación basada en el color ha sido útil en problemas prácticos donde las escenas no son muy complejas, sin embargo, estas técnicas están limitadas para determinar la tridimensionalidad de las escenas. Debido a esta necesidad, surgieron las imágenes de profundidad, las cuales detallan mejor el espacio 3D. El seguimiento de objetos a través de las imágenes de profundidad puede generar trayectorias de movimiento en 3D, siendo a su vez efectivo para resolver muchos problemas tradicionales como cambios de iluminación, distorsión de la perspectiva, ya que la profundidad no se ve tan afectada.

En entornos de trabajo, donde se encuentran instalados brazos robóticos que interactúan con humanos, se tiene un alto riesgo de causar colisiones. Las personas circundantes limitan las maniobras del brazo e incluso vuelve improductivo el proceso. Para lograr un entorno industrial robotizado seguro, la comunidad científica viene trabajando con algunas técnicas de detección, entre ellas el sistema de detección por torque, visión artificial y sistemas híbridos. Los sistemas de visión artificial capturan la silueta de las personas poniendo en alerta al sistema cuando el brazo se aproxima a él, pero su gran falencia son los problemas tradicionales de la visión artificial (Wu, Mao, Chen, Xue & Rovetta, 2015), las técnicas que se expondrán tienen como objetivo la detección y posterior seguimiento de personas en entornos industriales con robots colaborativos, como parte de la detección de colisiones en ambientes de interacción humano-robot.

1.2. Objetivos:

1.2.1. Objetivo General:

Evaluar las técnicas de tracking a personas con imágenes RGB-D para ser aplicadas a entornos industriales robotizados.

1.2.3 Objetivos específicos:

- Realizar una revisión del estado del arte de las técnicas de tracking que permiten el uso de datos RGB-D en aplicaciones de seguimiento de peatones.
- Seleccionar e implementar por lo menos 3 técnicas de tracking usando una base de datos propia.
- Evaluarán las técnicas implementadas ante variables de precisión y tiempos de procesamiento.

1.3. Organización del trabajo:

El capítulo 2 trata sobre la revisión que se hizo de las técnicas de tracking más usadas en la comunidad de visión artificial usando datos RGB-D, haciendo una mirada del estado del arte en aplicaciones de seguimiento de personas, también se explica el flujo de trabajo común y la utilización del canal De profundidad. El capítulo 3 trata sobre la metodología utilizada para implementar y desarrollar las 4 técnicas de tracking elegidas, usando una base de datos propia y una pública, generando algunos modelos. En el cuarto capítulo se desarrolla la evaluación de las técnicas implementadas, observando la precisión y el tiempo de procesamiento. Por último, el capítulo 5 concluye sobre el trabajo realizado, los alcances y objetivos.

2. MARCO TEÓRICO

2.1. Conceptos:

Reconocimiento de objetos: Se desea saber si uno o varios objetos se encuentran en las imágenes.

Detección de objetos: Es como el reconocimiento de objetos, pero esta tarea tiene como adicional, saber las ubicaciones, lo que significa localizar regiones de interés.

Segmentación de objetos: Es aislar los objetos del resto de la escena o fondo, por lo general sabiendo toda su geometría, área y/o volumen.

Seguimiento de objetos: implica verificar la presencia de un objeto en secuencias de imágenes de video (frames) y ubicarlo con precisión para su reconocimiento, ubicación y en la mayoría de los casos, su dirección de movimiento.

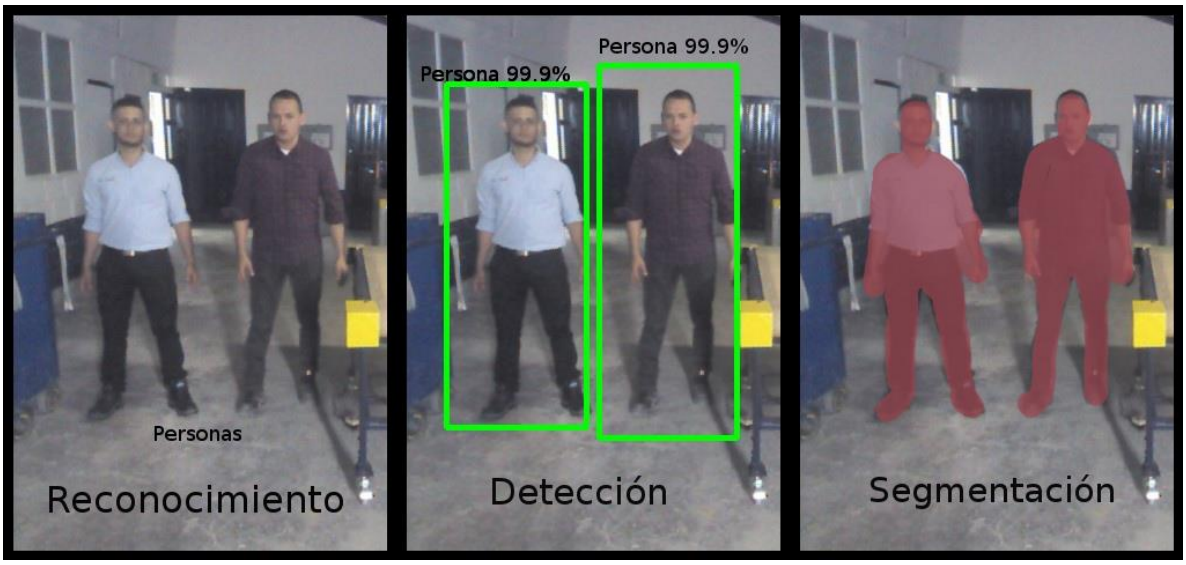


Figura 1. Conceptos claves de tareas en visión artificial.

En la figura 1, se puede observar tres tareas comunes en la visión artificial, en el lado izquierdo, se conoce que existen personas, mientras que en el cuadro del medio muestra la detección de las personas, pero limita el área en que se encuentran, en la derecha, la segmentación determina toda el área que ocupan sólo las personas.

Kinect y RGBD: Las cámaras de profundidad llevan varios años utilizándose en visión artificial, pero dado a el alto costo, acceder a ellas no era fácil, sin mencionar que la baja calidad reducía su aplicabilidad. Con el auge de los videojuegos, surge el Kinect, un sensor de profundidad a bajo costo producido por Microsoft. Las cámaras por lo general usan el espacio de color RGB, los cuales se representan en una matriz de profundidad de tres capas. RGB viene de los tres colores primarios que gracias a la síntesis aditiva generan el resto de los colores, R para el rojo, G para el verde y B para el azul.

Gracias a la profundidad y el sensor visual RGB, la información en un sensor Kinect abrió nuevas oportunidades para resolver problemas en la visión artificial, incluyendo reconocimiento de objetos y actividades, seguimiento de personas, mapeo de escenas tridimensionales, localización, entre muchas otras. (Jungong Han, Ling Shao, Dong Xu & Shotton, 2013)

Figura 2. Imagen de profundidad (Depth) y color (RGB) extraídas de un kinect.

La figura 2 muestra las dos imágenes que toma un Kinect, la de la izquierda es la imagen de profundidad, entre más cercano es el objeto, más claro es el color, la imagen derecha muestra una fotografía en RGB.

Descriptor: Se le llama descriptor, como su nombre lo indica a las descripciones de las características compuesta en las imágenes (Yuan, Qiang & Yin, 2017).

Los descriptores son el primer paso para conectar la información y los píxeles, estos se describen por:

- Color
- Textura
- Forma
- Movimiento
- Ubicación

ROI: Las regiones de interés como su nombre lo indica, es la región en la cual se tiene interés en la imagen, la mayoría de las veces delimitada por un área poligonal.

Grafo: Representación simbólica de elementos conectados, mediante esquemas gráficos.

Bhattacharyya: La distancia Bhattacharyya mide la similitud de dos distribuciones de probabilidad. Es en sí, una medida de la cantidad de superposición entre dos muestras estadísticas o poblaciones.

Trackeador: Cuando se habla de trackeador, se hace referencia a la técnica con la cual se predice el movimiento de los objetos seguidos.

- **Track:** Se denomina track a cada objeto que se le hace un seguimiento.
- **Filtro Kalman:** El filtro Kalman es uno de los trackeadores más conocidos para hacer seguimiento de objetos, su nombre deriva de su desarrollador.

- **Filtro de Partículas:** El filtro de partículas es otro trackeador conocido, pero este lanza al azar puntos en la imagen que luego son renovados en los que muy probablemente se encuentre el objeto seguido.

Matching: El matching es una técnica que consiste en la comparación de una o más características, con el fin de encontrar coincidencias.

Background-subtraction: La sustracción o resta de fondo es la acción de restar la región circundante a él o los ROI.

Aprendizaje Automático (Machine Learning): El aprendizaje automático es una de las aplicaciones de la inteligencia artificial, con el fin de lograr que las computadoras aprenden sin ser programadas explícitamente. El aprendizaje automático se centra en el desarrollo de programas que pueden cambiar cuando se exponen a nuevos datos, es decir, que pueda generalizar bien (Kotsiantis, Zaharakis & Pintelas, 2007). Algunas de estas técnicas son:

- **Red Bayesiana:** Una red bayesiana, red de creencia o modelo acíclico dirigido es un modelo probabilístico que representa una serie de variables de azar y sus independencias condicionales a través de un grafo acíclico dirigido ("Red bayesiana", s.f.).
- **K Vecino más cercano:** Es un método de clasificación supervisado (Aprendizaje que estima el resultado basado en un conjunto de entrenamiento conocido) que sirve para estimar la función de densidad de los datos, relacionando lo próximos que están estos ("K vecinos más próximos", s.f.).
- **Máquina de soporte vectorial (SVM):** una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión.
- **Redes neuronales Artificiales:** Las redes de neuronas artificiales (RNA) tienen su inspiración en las neuronas del sistema nervioso animal. al igual que las neuronas biológicas, existen enlaces de neuronas que colaboran entre sí, el estímulo generado da como resultado una salida. Los enlaces tienen pesos numéricos que se adaptan según la experiencia ("Red neuronal artificial", s.f.).

- **AdaBoost:** El boosting consiste en combinar los resultados de varios clasificadores débiles para obtener un clasificador robusto, AdaBoost es una de las técnicas y es el más utilizado.

Deep Learning: El aprendizaje profundo o Deep Learning en inglés, es otra técnica de machine learning en las que los datos son más complejos. Se usan múltiples capas de procesamiento. En teoría, el aprendizaje profundo trata de basarse en técnicas acerca de cómo funciona el cerebro humano. El Deep Learning ha demostrado el gran potencial que tiene en el procesamiento de imágenes.

- **Entrenamiento:** Al igual que un niño, el aprendizaje se da basado en la experiencia que se da, el entrenamiento de tal experiencia puede ser guardada como pesos. La experiencia puede hacerse por épocas, refiriéndose a cuántas veces vuelve y se expone a la experiencia.
- **Redes Neuronales Convolucionales:** Esta es un tipo de red neuronal artificial frecuente en el procesamiento de imágenes basados en Deep Learning, su funcionamiento se basa en las neuronas ubicadas en la corteza visual cerebral.
- **Inferencia:** Después del entrenamiento y habiendo validando, el modelo generado debe ser usado para el propósito creado, la inferencia se refiere a esto, es el modelo ya preparado para realizar su cometido.
- **Transferencia de aprendizaje y Ajuste fino:** La transferencia de aprendizaje y el ajuste fino son técnicas usadas en Deep Learning para aumentar y/o mejorar las capacidades de un modelo mediante uno anterior.
- **Sobreentrenamiento:** El sobreentrenamiento es el resultado de que un modelo logre resultados muy buenos cuando se entrena, pero no generalizan bien en la inferencia.

2.2. Estado del arte:

Entre las técnicas vistas, uno de los descriptores usados es el detector basado en los histogramas orientados a gradientes (HOG) (Salas & Tomasi, 2011), (Hao Zhang, Reardon &

Parker, 2013), (Munaro, Basso & Menegatti, 2012), este, junto a los descriptores de color, forma y textura, tomaban el primer cuadro (frame) para generar la detección de las personas.
Los detectores de color generan modelos para las piernas y el torso (Jungong Han, Pauwels, de Zeeuw & de With, 2012) o color piel (Choi, Pantofaru & Savarese, 2011). Para los descriptores de forma, la detección se realiza mediante la forma de la cabeza (Muñoz-Salinas, García-Silvente & Medina Carnicer, 2008), (Choi, Pantofaru & Savarese, 2011)

Algunos no usan emparejamiento (Matching) (Salas & Tomasi, 2011), (Jungong Han, Pauwels, de Zeeuw & de With, 2012), sin embargo, la gran mayoría lo hace. (Hao Zhang, Reardon & Parker, 2013), (Ess, Leibe, Schindler & van Gool, 2009), (Muñoz-Salinas, Aguirre & García-Silvente, 2007). Para la etapa de emparejamiento, se usan varios métodos, la distancia de Bhattacharyya (Jungong Han, Pauwels, de Zeeuw & de With, 2012), (Muñoz-Salinas, García-Silvente & Medina Carnicer, 2008), Background-subtraction o Sustracción de fondo (Vo, Jiang & Zell, 2014), (Galanakis, Zabulis, Koutlemanis, Paparoulis & Kouroumalis, 2014) , método húngaro (Muñoz-Salinas, Aguirre & García-Silvente, 2007) y el template. (Satake, Chiba & Miura, 2013).

La mayoría de las técnicas usan aprendizaje automático para generar clasificadores, mientras que otros no (Salas & Tomasi, 2011), (Jungong Han, Pauwels, de Zeeuw & de With, 2012). Entre las técnicas se encuentran Redes bayesianas (Ess, Leibe, Schindler & van Gool, 2009), Adaboost (Munaro, Basso & Menegatti, 2012), (Hao Zhang, Reardon & Parker, 2013), (Muñoz-Salinas, García-Silvente & Medina Carnicer, 2008), HaarCascade (Hao Zhang, Reardon & Parker, 2013), (Muñoz-Salinas, Aguirre & García-Silvente, 2007) y Máquina de soporte vectorial (SMV) (Satake, Chiba & Miura, 2013).

Los trackeadores son casi en su totalidad el Filtro Kalman, a excepción de algunos que usan Filtro de partículas (Choi, Pantofaru & Savarese, 2011) o ninguno, que suponen velocidades constantes para generar trayectorias (Salas & Tomasi, 2011), (Jungong Han, Pauwels, de Zeeuw & de With, 2012).

Las anteriores técnicas, basan sus detectores en descriptores, sin embargo, existen aquellos que usan la técnica de Deep Learning. De estos, todos son detectores de objetos, a excepción de algunos que son clasificadores (Xue et al., 2016), o segmentan nube de puntos (Charles, Su, Kaichun & Guibas, 2017). Se dividen en dos clases, de disparo único (Liu et al., 2016), (Lin, Goyal, Girshick, He & Dollar, 2017), (Redmon, Joseph & Farhadi, 2017) y

detección de región (Girshick, Donahue, Darrell & Malik, 2014), (Girshick, 2015). Los detectores de región hacen la convolución sobre los ROI de la imagen, mientras que los de disparo único predicen simultáneamente el desplazamiento del cuadro delimitador o ROI y las probabilidades de pertenecer a una clase. Durante el entrenamiento, se hace coincidir la casilla verdadera con casillas predichas mediante el índice de Jaccard (Real & Vargas, 1996).

2.3. Flujo de trabajo:

El flujo de trabajo general de las técnicas o métodos anteriores, consisten en:

1) Generar regiones de interés.
2) Detección de las personas y/u objetos.
3) Comparación entre frames.
4) Generar trayectorias para no perder la detección.

De forma más general, la figura 3 lo representa mejor:

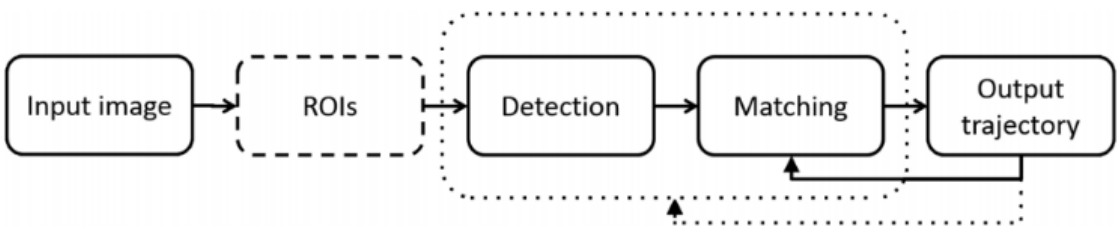


Figura 3, flujo de trabajo común en la detección de personas en imágenes. (Camplani et al., 2017)

Es de notar que la etapa más crítica y a la cual más se le hace énfasis es a la detección, aquí, el papel de los descriptores y clasificadores toman relevancia, sin embargo, el paradigma cambia con el Deep Learning, ya que la detección de los objetos y/o personas es mucho mejor que las técnicas de visión clásicas.

Con sólo utilizar información de color (RGB), el Deep Learning puede generar una buena detección, no obstante, la información de profundidad (D), no sólo apoya las condiciones del color, también puede revelar la tridimensionalidad de la escena, la figura 4 muestra los usos que brinda la información de profundidad:

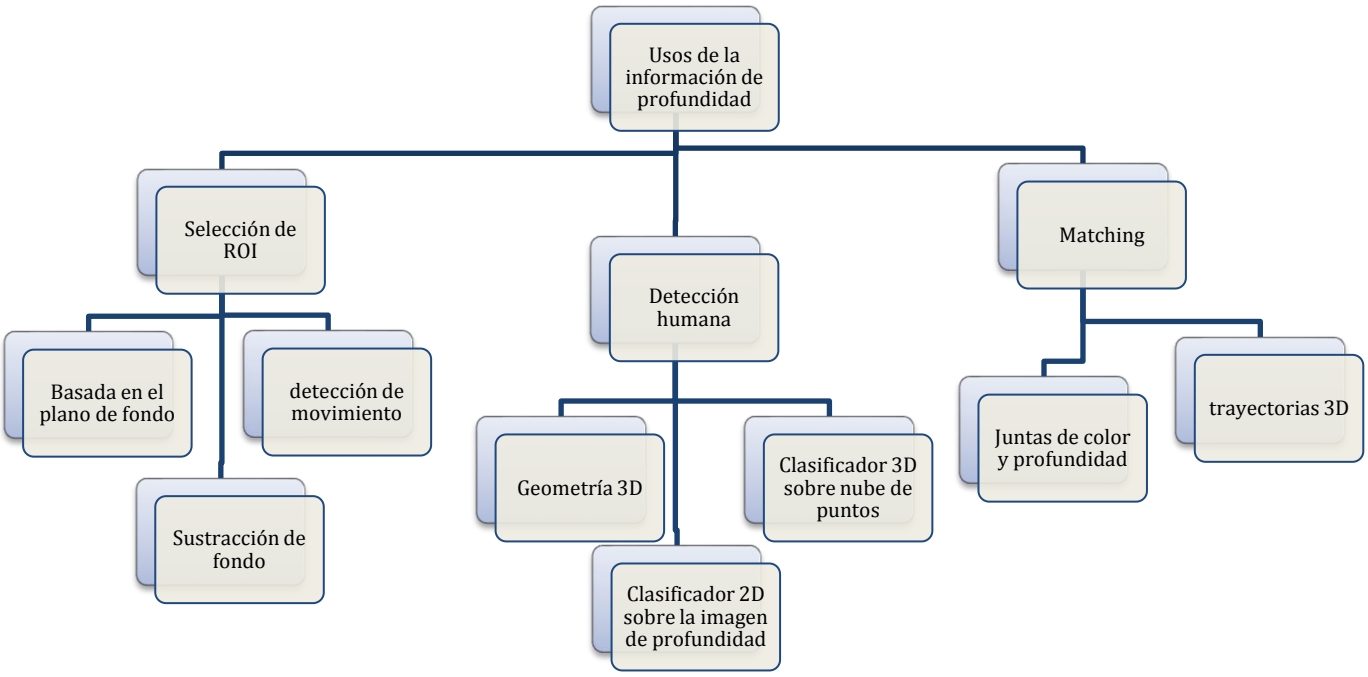


Figura 4. Usos del canal de Profundidad. (Camplani et al., 2017)

Luego de hacerse la revisión del estado del arte, se eligió 4 técnicas de object tracking para ser implementadas y obtener sus resultados.

Las técnicas seleccionadas fueron:

- Single Shot Detector SSD.
- Focal Loss Retinanet.
- You Only Look Once YOLO v3.
- PointNet.

La elección de estas responde a que SDD, Retinanet y YOLO, son excelentes detectores de color, y que han sido aplicados en entornos de producción de grandes empresas como

Google y Facebook, buscando un balance entre precisión y rapidez. Mientras que PointNet, resulta útil para expandir la detección y llegar a una segmentación tridimensional usando la profundidad.

3. METODOLOGÍA

3.1. Software usado:

Keras es un framework que corre sobre TensorFlow, ambos están diseñados para trabajar como librerías en Python cuando se trata de Deep Learning. Se trabajó con estos debido a su sencillez de implementación y de entendimiento, más la buena documentación y desarrollos de éste y además por que las técnicas escogidas trabajan con Deep Learning.

3.2. Recursos usados:

los recursos utilizados fueron 2, un computador del laboratorio de microelectrónica que entre su característica más deseable es su GPU Quadro K4000, necesaria para la tarea, además de su sistema operativo Ubuntu 16.04. Lo segundo fue un recurso de procesamiento en la nube, provisto por el servicio de Google Cloud donde se contaba con una GPU Tesla K80, su uso fue encontrado en internet. ("Google Colab Free GPU Tutorial – Deep Learning Turkey – Medium", 2018)

3.3 Implementación:

Una arquitectura de red neuronal con deep learning puede ser creada desde cero, sin embargo, debido a que esto requiere de mucho tiempo y se debe uno enfrentar a otros problemas en el entrenamiento como inicialización de pesos entre otros, se optó por utilizar modelos pre-entrenados.

Un modelo pre-entrenado es un archivo que contiene una red lista para la inferencia, pero que a su vez puede ser modificada para realizar transferencia de aprendizaje (Transfer-Learning) y/o ajuste fino (Fine-Tunnig). Para cada técnica utilizada a excepción de PointNet (SSD, Retinanet y YOLO) se utilizó estas formas de trabajo.

Los modelos pre-entrenados detectan más que sólo personas, en realidad pueden detectar más de 80 objetos, aunque pueden ser omitidos, su detección representa más recurso computacional. Con la transferencia del aprendizaje, se conservó los pesos de las capas superiores de la red, que nos identifican bordes, texturas entre otros y mediante ajuste fino, modificamos sus últimas capas para detectar sólo personas.

3.4. Datasets:

Se usaron dos dataset, uno público llamado PASCAL VOC 2007 (Everingham, Van~Gool, Williams, Winn & Zisserman, 2018) y otro creado por el laboratorio. Los dataset deben ser archivos de texto que contienen la ruta de las imágenes a usar, junto a las posiciones del ROI, por lo general, una geometría rectangular.

PASCAL 2007 es un dataset creado para una competencia del mismo nombre, sus miles de imágenes están hechas para tres retos, clasificación, detección y segmentación. Tiene 20 clases como se puede ver en la figura 5 (Personas, aves, gatos, vacas, perros, caballos, ovejas, aviones, bicicletas, barcos, autobuses, carros, motos, trenes, botellas, sillas, mesas de comedor, plantas en maceta, sofás y televisores o monitores), en sus 9.963 imágenes, que contienen 24.640 objetos anotados o delimitados.

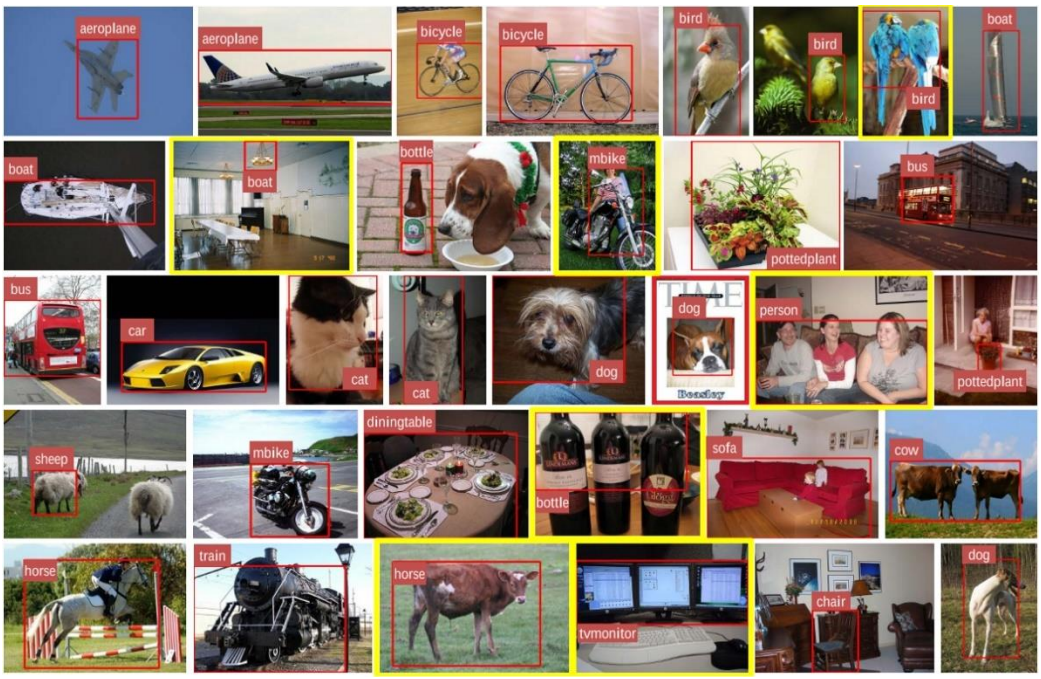


Figura 5. Pascal VOC dataset (Everingham, Van~Gool, Williams, Winn & Zisserman, 2018).

La base de datos creada por el laboratorio tiene 563 imágenes RGB de una sola persona en un entorno industrial robotizado, más otras 563 en Depth, en el mismo entorno, así

mismo, 1.174 imágenes (RGB y Depth cada una) para dos personas, entre 3 y 4 personas contiene 3.548 y finalmente tiene 5.260 imágenes de negativos, es decir, entornos industriales sin personas. La Figura 6, muestra algunas de sus imágenes.

Figura 6. Base de datos creada en el laboratorio, personas en entornos industriales.

3.5. Entrenamiento:

3.5.1. Retinanet.

Para el entrenamiento de Retinanet utilizamos el framework creado por los mismos, su framework y modelo pre-entrenado se puede encontrar en la página de su proyecto ("keras-retinanet", n.d.). Este framework lee el Dataset con la siguiente forma:

path/to/image.jpg,x1,y1,x2,y2,class_name

Donde, path/to/image.jpg, es la ruta de la imagen, x1,y1,x2,y2 son las posiciones del recuadro delimitador o ROI y class_name se refiere a la clase del objeto detectado, debido a que los objetos de interés que necesitamos son personas, nuestro class_name es persona. Se realizó dos etapas de entrenamiento, en la primera, usamos el dataset público PASCAL de imágenes ya etiquetadas con su ROI para realizar la detección de personas en cualquier ambiente, luego, un entrenamiento con la base de datos propia con personas en entornos industriales en ambientes robotizados.

Retinanet usa redes neuronales convolucionales conocidas para la extracción de características, llamadas backbone. Especificamente se utilizó la implementación Resnet 30 (He, Zhang, Ren & Sun, 2016) al ser la predeterminada y la cual más precisión se obtiene.

El entrenamiento al ser una tarea que requiere tiempo fue dividida, las primeras épocas del entrenamiento en su primer etapa, fueron hechas con el recurso computacional del laboratorio y luego en procesamiento en la nube. En total fueron 20 épocas entrenadas con un total de aproximadamente 2 horas por cada época para esta primera etapa.

Debido a que había problemas de desconexión, volcado de memoria, entre otros, se guardaba cada modelo después de realizar cada época evitando así la pérdida total del trabajo. La segunda etapa fue hecha en el mismo computador de la institución, otras 10 épocas fueron sumadas.

En total fueron 30 épocas con un batch-size de 10 y steps o pasos de 500 para esta técnica.

3.5.2. YOLO v3:

Mientras Retinanet fue elegido por su precisión, YOLO fue seleccionado por su velocidad, su versión tercera revela mayor precisión que sus predecesores, por lo cual, la versión 3 fue la trabajada. Su framework y modelo pre-entrenado puede ser encontrado en su Github. ("keras-yolo3", n.d.)

El entrenamiento de YOLO fue similar a retinanet, salvo que la forma de recibir los datos de entrenamiento cambia un poco para sus creadores:

image_file_path box1 box2 ... boxN

Image_file_path al igual que en Retinanet significa la ruta de la imagén, boxN se refiere al ROI y la clase, el cual va de la siguiente forma:

x_min,y_min,x_max,y_max,class_id

Esta parte es similar a la de Retinanet, a excepción de class_id, que representa un número, en este caso le pondremos 1.

Entonces, para imágenes con más de 1 persona en la escena, sería así:

image_file_path x_min,y_min,x_max,y_max,class_id x_min,y_min,x_max,y_max,class_id

Obsérvese que hay un espacio entre class_id, para mejor entendimiento, se pone la imagen del dataset en la figura 6.

```
JPEGImages/000021.jpg 210,36,336,482,1 46,82,170,365,1 11,181,142,419,1
JPEGImages/000023.jpg 2,1,117,369,1 3,2,243,462,1 225,1,334,486,1
JPEGImages/000023.jpg 2,1,117,369,1 3,2,243,462,1 225,1,334,486,1
JPEGImages/000023.jpg 2,1,117,369,1 3,2,243,462,1 225,1,334,486,1
JPEGImages/000030.jpg 51,160,150,292,1 295,138,450,290,1
JPEGImages/000030.jpg 51,160,150,292,1 295,138,450,290,1
JPEGImages/000032.jpg 195,180,213,229,1 26,189,44,238,1
JPEGImages/000032.jpg 195,180,213,229,1 26,189,44,238,1
JPEGImages/000035.jpg 1,96,191,361,1 218,98,465,318,1
JPEGImages/000035.jpg 1,96,191,361,1 218,98,465,318,1
JPEGImages/000041.jpg 216,92,307,302,1 164,148,227,244,1
JPEGImages/000041.jpg 216,92,307,302,1 164,148,227,244,1
JPEGImages/000048.jpg 2,1,302,500,1
```

Figura 7. Dataset para YOLO v3.

Al igual que en Retinanet, el entrenamiento fue dividido del mismo modo, dos etapas, dos formas de entrenar (GPU computador institucional, GPU en la Nube). Esta red fue entrenada en su primera etapa con 10 épocas en la institución y 30 epocas en la nube, en su segunda etapa fue entrenada otras 10 en la institución.

En total se realizaron 50 épocas con pasos del número de imágenes en entrenamiento dividido en las épocas de cada etapa y un batch-size en primera etapa de 30 y en segunda de 10.

3.5.3. SSD:

La elección de esta técnica, responde a que la arquitectura de la red es predecesora de las dos anteriores, por lo cual, vale la pena compararla con Retinanet y YOLO v3. su extractor de características consta de 9 capas convolucionales VGG, una arquitectura de red convolucional para imágenes a gran escala (Liu & Deng, 2015) su modelo pre-entrenado se encuentra en su repositorio. ("ssd_keras", n.d.)

La forma de leer los datos es un poco particular debido a que lee de manera similar en el formato de retinanet.

El modo de entrenamiento es el mismo que en Retinanet y YOLO v3, salvo que todo el entrenamiento fue realizado en la institución, mismas dos etapas, primera etapa con 10

épocas, y la segunda etapa con otras 10, en total fueron 20 épocas con un batch-size de 10 y steps o pasos de 500.

3.5.4. PointNet:

Para el entrenamiento con PointNet, fue necesario pasar las imágenes RGB-D a nube de puntos, este programa se realizó teniendo los parámetros intrínsecos con los cuales fue tomada las imágenes y visualizarlas con la librería de Python Open3D. Generada la nube de puntos, fue necesario etiquetar cada punto de esta, para su la labor se realizó un algoritmo utilizando la librería Open3D para generar cada etiqueta, usando KDTree integrada en dicha librería. Como dicha labor requiere de mucho tiempo, se recurrió a otra base de datos de nube de puntos, el algoritmo se anexará al final en el apéndice A. Se entrenaron 10 épocas con 800 pasos.

4. RESULTADOS Y DISCUSIÓN

4.1. Resultados:

Cada modelo se evaluó bajo dos criterios, velocidad de procesamiento y precisión, la precisión es tomada bajo dos parámetros de entrenamiento llamado pérdida en la detección (Loss) y pérdida en la clasificación (val_loss), estos nos muestran cuanto la efectividad del modelo cuando se sometió a entrenamiento, mientras que la velocidad de procesamiento se observa mirando el tiempo de ejecución.

4.1.1. Retinanet:

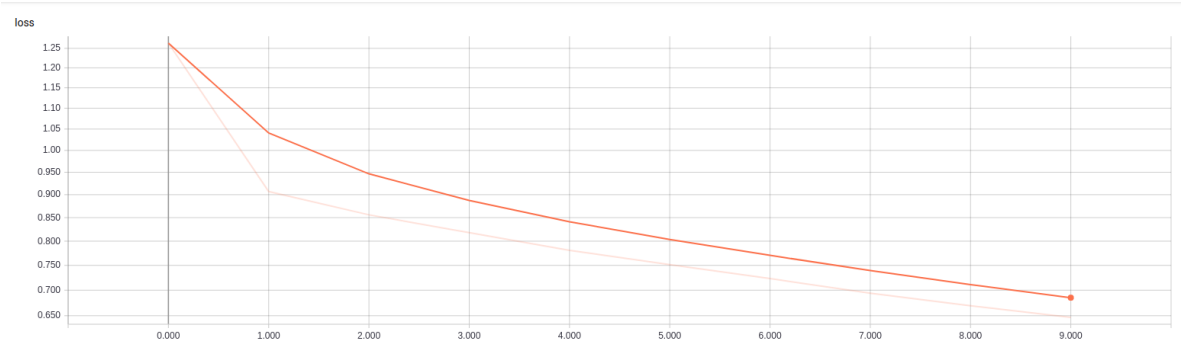


Figura 8. Pérdida focal de Retinanet.

La figura 8 nos muestra la precisión correspondiente a Retinanet, fue rápida mientras avanzaba las épocas, en un total de 10 épocas, la precisión se estableció cerca al 0.7 de porcentaje de error en cuanto establecer bien la región de detección se refería. Sólo dos baches fueron necesarios para lograr este resultado (líneas naranjas).

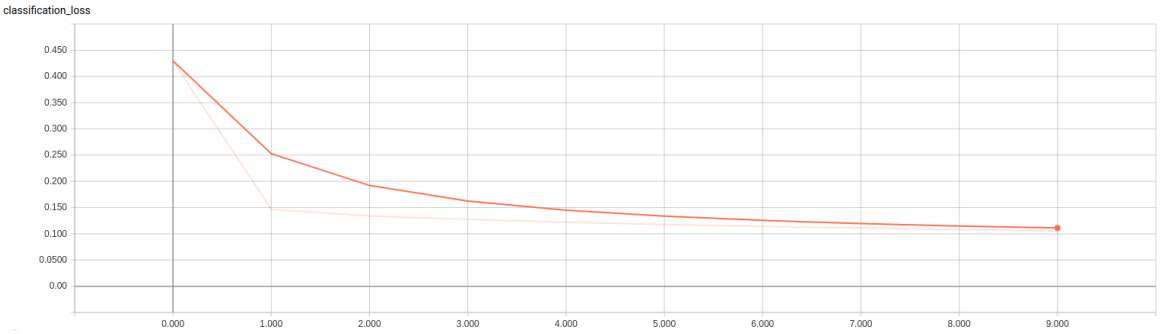


Figura 9. Pérdida de clasificación Retinanet.

Retinanet logró 0.1 % de error en la clasificación en sus primeras 10 épocas como se ve en la figura 9. Un porcentaje muy bajo de error.

```
processing time:  2.406602382659912
processing time:  2.3962559700012207
processing time:  2.4072625637054443
processing time:  2.4078586101531982
processing time:  2.402517318725586
processing time:  2.406869411468506
```

Figura 10. Tiempo de procesamiento Retinanet.

Retinanet parece ser muy preciso, no obstante, su tiempo de procesamiento es alto como lo muestra la figura 10, cerca de 3 segundos por cuadro corriendo sobre la GPU institucional, lo que puede derivar en que se necesita una capacidad de cómputo mayor.

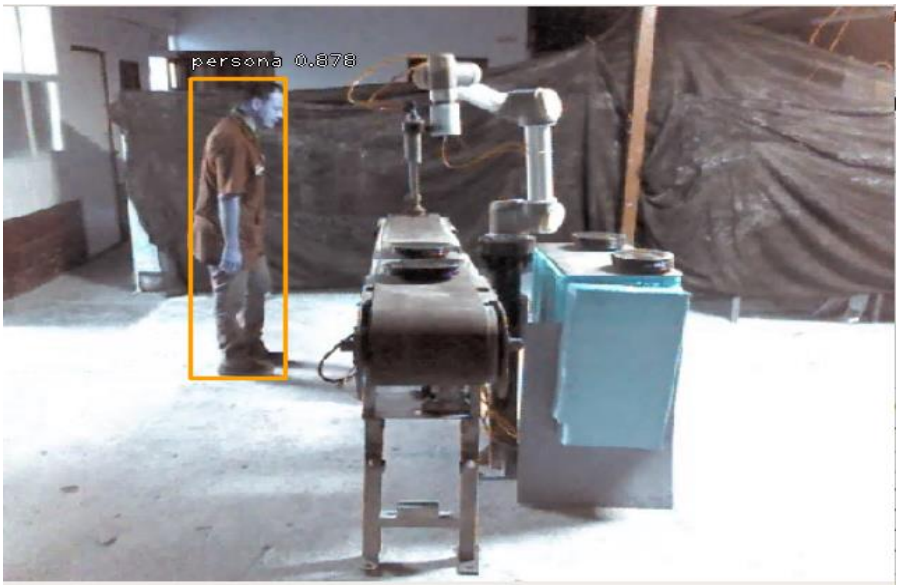


Figura 11. Detección de Retinanet.

La figura 11 muestra una imagen detectada de una persona en un entorno industrial robotizado controlado, se puede ver como el cuadro limitador es preciso.

4.1.2. YOLO V3:

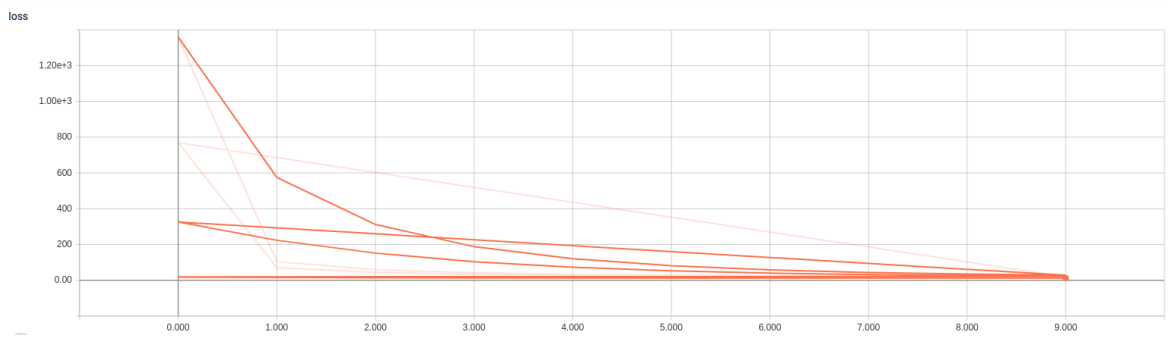


Figura 12. Pérdida de YOLO v3, primeras épocas.

La figura 12 muestra como YOLO tuvo que utilizar 10 baches para aproximarse a tener un error decente, se puede ver como desde el principio estaba lejos de ser un buen detector

ya que sus tres primeras épocas tenían un error de más 400%. Sin embargo, después de 10 épocas, mejoró notablemente.

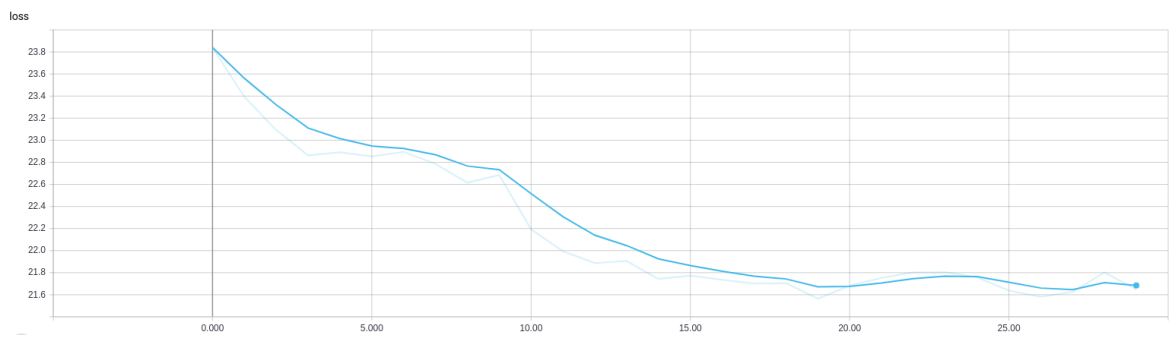


Figura 13. YOLO v3. últimas épocas.

Aunque YOLO fue quien más épocas de entrenamiento recibió, no fue diferencialmente mejor, como la figura 13 muestra, sus épocas finales lo acercaron sólo al 21% de error, lo que su precisión no es la mejor de todas.

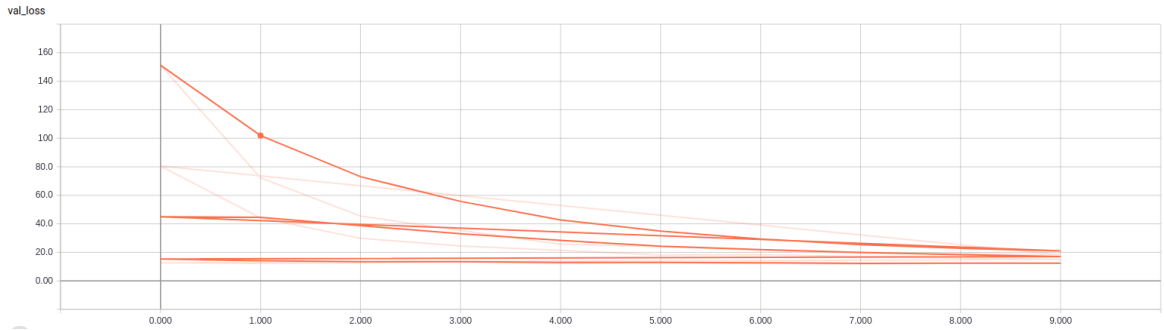


Figura 14. Pérdida de clasificación YOLO v3.

La pérdida de clasificación fue igual a la de pérdida de detección, la figura 14 muestra los 10 baches usados, para llegar a un aproximado de 22% de error.

```
(416, 416, 3)
Found 0 boxes for img
0.33678509799938183
(416, 416, 3)
Found 0 boxes for img
0.335761621000529
(416, 416, 3)
Found 0 boxes for img
0.33771610299936583
(416, 416, 3)
Found 1 boxes for img
persona 0.34 (332, 0) (632, 377)
0.3366764600004899
(416, 416, 3)
Found 1 boxes for img
persona 0.39 (332, 0) (633, 377)
0.3465054680000321
(416, 416, 3)
Found 1 boxes for img
persona 0.49 (333, 0) (632, 374)
0.33897564600001081
(416, 416, 3)
Found 1 boxes for img
persona 0.36 (331, 0) (632, 376)
0.3383246559997133
(416, 416, 3)
Found 0 boxes for img
0.33766149599978235
(416, 416, 3)
```

Figura 15. Tiempo de procesamiento YOLO v3.

Aunque YOLO palidece con respecto a Retinanet en términos de precisión, YOLO lo supera en tiempo de procesamiento, la figura 15 lo demuestra. YOLO obtiene cerca de 0.4 segundos en procesar la imagen, eso son entre 2 y 3 frames por segundo.

Figura 16. Detección de YOLO v3.

El ROI de la persona detectada en figura 16, recorta un poco la pierna, sin embargo, la región está en un estándar aceptable.

4.1.3. Single Shot Detector SDD:

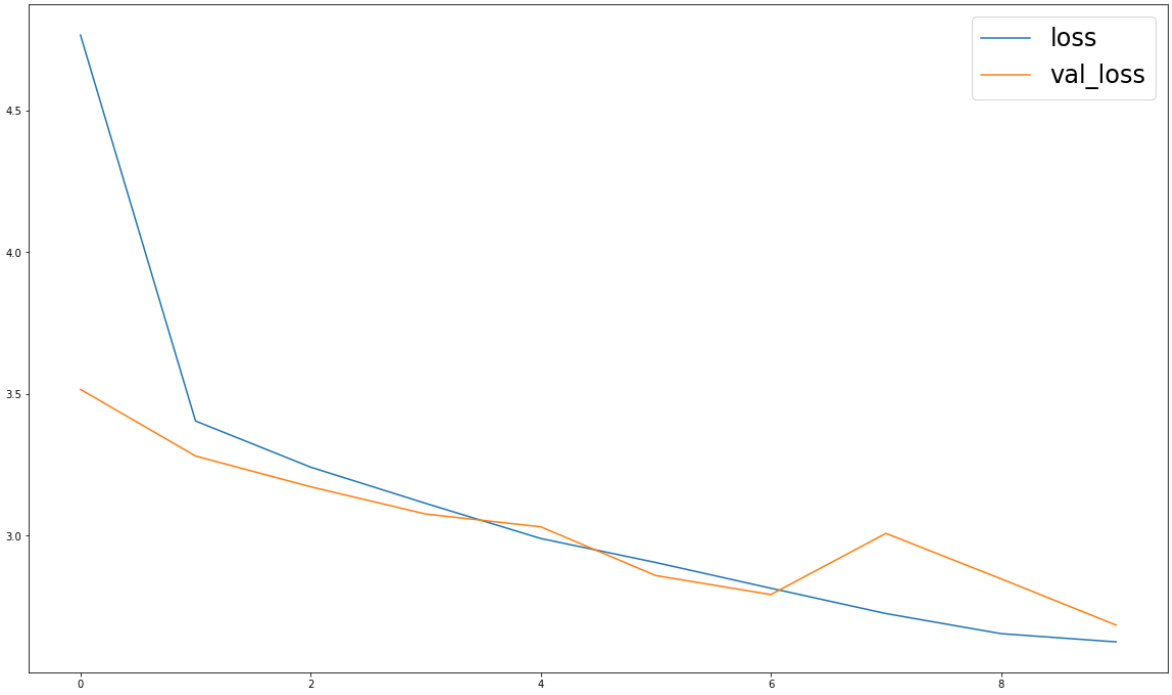


Figura 17. Pérdida de detección y clasificación Single Shot Detector SDD, primeras épocas.

En la figura 17, se puede observar como la línea en azul que representa la pérdida de SSD, inicia con más de 5% de error, pero después de 10 épocas se puede observar cómo va disminuyendo por debajo del 3%. La clasificación vista en naranja muestra cómo inicia desde 3.5% y desciende a 3%.

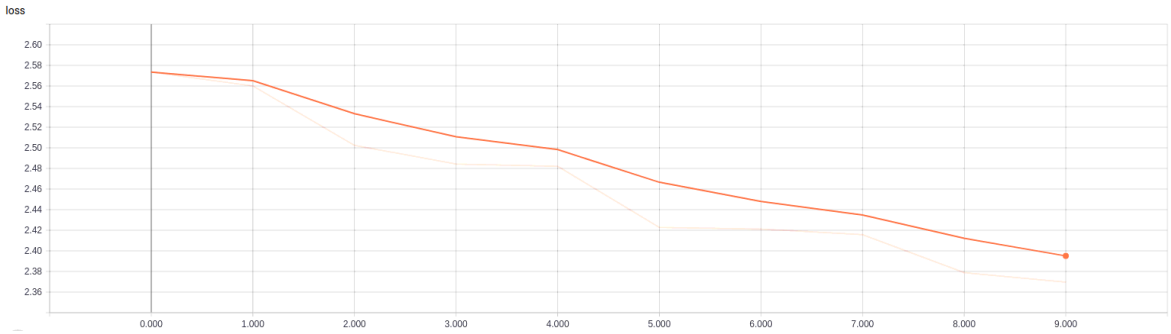


Figura 18. Pérdida de Single Shot Detector, épocas finales.

Para sus épocas finales, SSD logra una precisión, de detección cercana al 2.4%, como lo muestra la figura 18 estando en el medio de Retinanet y YOLO.

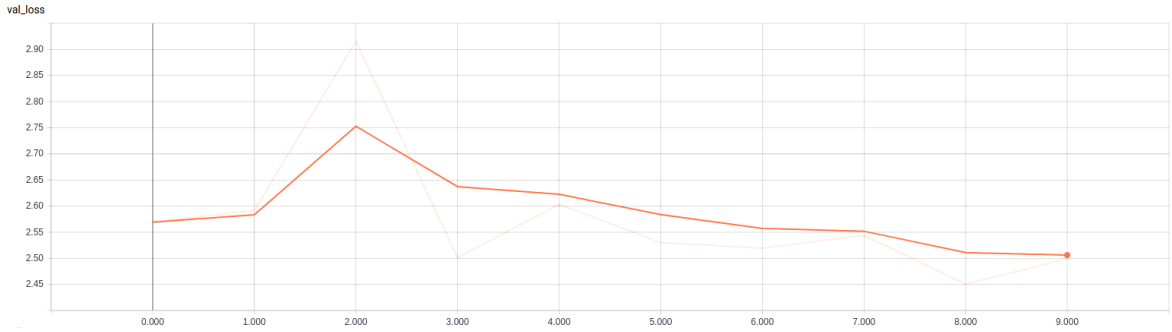


Figura 19. Pérdida de clasificación Single Shot Detector.

SSD parece tener leves problemas para clasificar, quizás sus pocas capas convolucionales no logran ser muy robustas, la figura 19 muestra como en la época 3 sube de 2.6 a 2.75% el error, aunque realmente no es mucho, esto puede derivar en la pérdida del objeto fácilmente.

```
time:  0.2705988883972168
time:  0.26885437965393066
time:  0.2720310688018799
time:  0.2699911594390869
time:  0.26658082008361816
time:  0.26711583137512207
time:  0.26813673973083496
time:  0.26901769638061523
time:  0.26677870750427246
time:  0.26787376403808594
time:  0.27106237411499023
time:  0.2682771682739258
time:  0.2670779228210449
```

Figura 20. Tiempo de procesamiento Single Shot Detector.

El tiempo de procesamiento de SSD es de aproximadamente 0.3 segundos, similar a YOLO, son alrededor de 3 a 4 cuadros por segundo tal como muestra la figura 20.

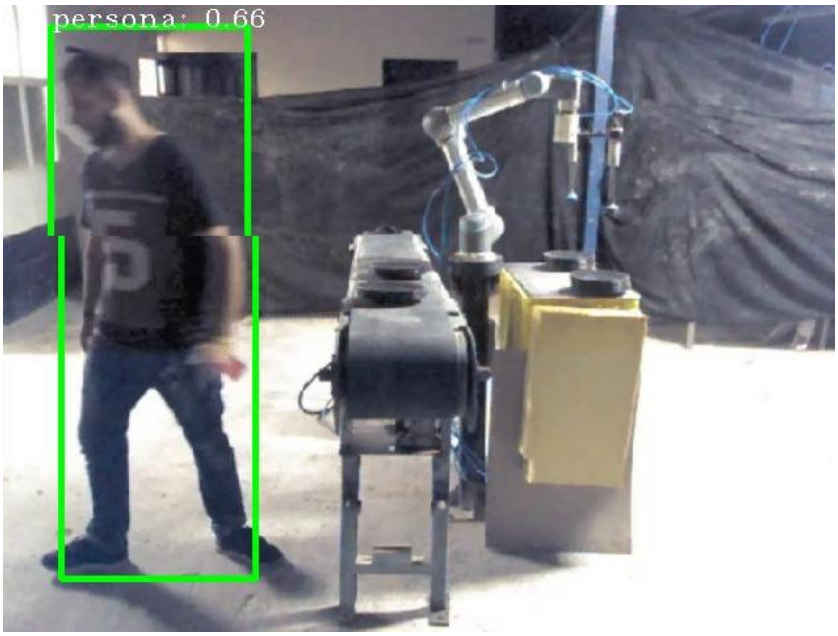


Figura 21. Detección de Single Shot Detector SDD.

La figura 21 muestra la detección de SSD, similar a YOLO, parte de las piernas no están dentro del cuadro delimitador, pero sigue estando en un nivel muy aceptable.

4.1.4. PointNet:

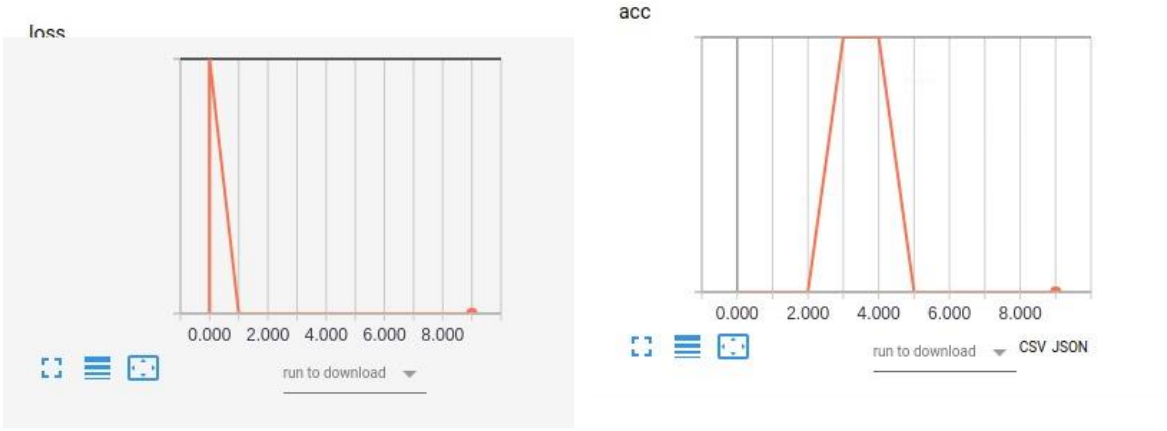


Figura 22 y 23. Pérdida y Precisión de PointNet.

Las figuras 22 y 23 resultado de PointNet no resultan ser entendibles, la pérdida cae abruptamente en la primera época, a un valor tan pequeño, esto es posiblemente a que el modelo quedó sobreentrenado, la siguiente imagen lo ilustra mejor:

```
 32/800 [>.............................] - ETA: 2:47 - loss: 1.1921e-07 - acc: 1
 64/800 [=>............................] - ETA: 1:47 - loss: 1.1921e-07 - acc: 1
 96/800 [==>...........................] - ETA: 1:25 - loss: 1.1921e-07 - acc: 1
128/800 [===>..........................] - ETA: 1:13 - loss: 1.1921e-07 - acc: 1
160/800 [=====>........................] - ETA: 1:05 - loss: 1.1921e-07 - acc: 1
192/800 [======>.......................] - ETA: 58s - loss: 1.1921e-07 - acc: 1.
224/800 [=======>......................] - ETA: 53s - loss: 1.1921e-07 - acc: 1.
256/800 [========>.....................] - ETA: 49s - loss: 1.1921e-07 - acc: 1.
288/800 [=========>....................] - ETA: 45s - loss: 1.1921e-07 - acc: 1.
320/800 [===========>..................] - ETA: 41s - loss: 1.1921e-07 - acc: 1.
352/800 [============>.................] - ETA: 38s - loss: 1.1921e-07 - acc: 1.
384/800 [==============>...............] - ETA: 35s - loss: 1.1921e-07 - acc: 1.
416/800 [===============>..............] - ETA: 32s - loss: 1.1921e-07 - acc: 1.
448/800 [================>.............] - ETA: 29s - loss: 1.1921e-07 - acc: 1.
480/800 [==================>...........] - ETA: 26s - loss: 1.1921e-07 - acc: 1.
512/800 [===================>..........] - ETA: 23s - loss: 1.1921e-07 - acc: 1.
```

Figura 24. Sobre-entrenamiento de PointNet.

Se puede apreciar como en la figura 24 el entrenamiento, la pérdida o porcentaje de error es muy poco, del orden de las micras y como el accuracy o exactitud es 1, o sea un 100%,

algo que no es lo esperado para un entrenamiento, un sesgo grande de aprendizaje fue evidenciado.

4.2. Tabla de resultados.

Técnica	error de precisión en Detección (%)	error de precisión en Clasificación (%)	Tiempo procesamiento (s).
Retinanet	0.7	0.1	2.4
YOLO v3	21.5	21	0.34
SSD	2.4	2.9	0.27
PointNet	NA	NA	NA

Tabla 1. Comparativa de técnicas.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

5.1. Conclusiones:

5.1.2. **Conclusión general:** Los algoritmos y modelos generados durante la realización del trabajo realizan muy bien la tarea de tracking sin la necesidad de trackeadores, su detección cuadro a cuadro es alta, aunque hay falsos positivos en algunos de estos, su desempeño sigue siendo aceptable. Dichos modelos son muy generalizantes, es decir, pueden ser utilizado no sólo en entornos industriales, sino también en varios contextos según se le entrene.

5.1.3. Conclusiones Complementarias:

- En la revisión del estado del arte, nos dimos cuenta de la importancia de la fase de detección de objetos y personas. Los trabajos se enfocan más en generar detectores perfectos, que trackeadores más robustos, es por eso que se elige apostar por la detección.
- Las técnicas elegidas son actualmente parte de los mejores detectores, aunque el recurso computacional sigue siendo costoso, nuevas tecnologías lo compensan como es el procesamiento en la nube, un dataset público resulta ser muy variado y bueno para el entrenamiento generalizante, pero el dataset propio es más homogéneo y ayuda a enfocarlo más a ciertos entornos.
- Cada técnica evaluada tiene su fortaleza en alguna u otra variable (Precisión y tiempo de ejecución), es decir, una buena precisión resulta en un tiempo de procesamiento alto y viceversa.

5.2. Recomendaciones:

- Entre las recomendaciones está usar una unidad de cómputo mayor en la GPU, un requisito necesario para que el laboratorio continúe con desarrollos en la materia.
- Se debe usar un dataset propio más heterogéneo para evitar problemas de sobre-entrenamiento y reducir la probabilidad que los modelos queden sesgados.
- Se debe generar una base de datos con nube de puntos en las que se encuentren personas etiquetadas, ya que actualmente no existe una.

5.3. Trabajo futuro:

- Queda como trabajo futuro el usar los modelos y algoritmos generados de manera paralela para que pueda revelarse la tridimensionalidad de la escena y poder predecir trayectorias tridimensionales para evitar las colisiones entre humano robot.
- Se puede extender la investigación para evadir obstáculos, ya sea para personas con visibilidad nula o reducida, o en robótica móvil.

REFERENCIAS

- Berclaz, J., Fleuret, F., Turetken, E. & Fua, P. (2011). Multiple Object Tracking Using K-Shortest Paths Optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(9), pp.1806-1819.
- Bourdev, L. & Malik, J. (2009). Poselets: Body part detectors trained using 3D human pose annotations. 2009 IEEE 12th International Conference on Computer Vision, pp.1365–1372.
- Camplani, M., Paiement, A., Mirmehdi, M., Damen, D., Hannuna, S., Burghardt, T., & Tao, L. (2017). Multiple human tracking in RGB-depth data: a survey. IET Computer Vision, 11(4), 265-285. doi: 10.1049/iet-cvi.2016.0178
- Charles, R., Su, H., Kaichun, M., & Guibas, L. (2017). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. 2017 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2017.16
- Choi, W., Pantofaru, C. & Savarese, S. (2011). Detecting and tracking people using an RGB-D camera via multiple detector fusion. 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp.1076–1083.
- Ess, A., Leibe, B., Schindler, K., & van Gool, L. (2009). Robust Multiperson Tracking from a Mobile Platform. IEEE Transactions On Pattern Analysis And Machine Intelligence, 31(10), 1831-1846. doi: 10.1109/tpami.2009.109
- Everingham, M., Van~Gool, L., Williams, C., Winn, J., & Zisserman, A. (2018). The PASCAL Visual Object Classes Challenge 2007 (VOC2007). Retrieved from http://host.robots.ox.ac.uk/pascal/VOC/voc2007/
- Fleuret, F., Berclaz, J., Lengagne, R. & Fua, P. (2008). Multicamera People Tracking with a Probabilistic Occupancy Map. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30(2), pp.267-282.

- Galanakis, G., Zabulis, X., Koutlemanis, P., Paparoulis, S., & Kouroumalis, V. (2014). Tracking persons using a network of RGBD cameras. Proceedings Of The 7Th International Conference On Pervasive Technologies Related To Assistive Environments - PETRA '14. doi: 10.1145/2674396.2674467
- Girshick, R. (2015). Fast R-CNN. 2015 IEEE International Conference On Computer Vision (ICCV). doi: 10.1109/iccv.2015.169
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference On Computer Vision And Pattern Recognition. doi: 10.1109/cvpr.2014.81
- Google Colab Free GPU Tutorial – Deep Learning Turkey – Medium. (2018). Retrieved from https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d
- Hao Zhang, Reardon, C., & Parker, L. (2013). Real-Time Multiple Human Perception With Color-Depth Cameras on a Mobile Robot. IEEE Transactions On Cybernetics, 43(5), 1429-1441. doi: 10.1109/tcyb.2013.2275291
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2016.90
- Jungong Han, Ling Shao, Dong Xu, & Shotton, J. (2013). Enhanced Computer Vision With Microsoft Kinect Sensor: A Review. IEEE Transactions On Cybernetics, 43(5), 1318-1334. doi: 10.1109/tcyb.2013.2265378
- Jungong Han, Pauwels, E., de Zeeuw, P., & de With, P. (2012). Employing a RGB-D sensor for real-time tracking of humans across multiple re-entries in a smart environment. IEEE Transactions On Consumer Electronics, 58(2), 255-263. doi: 10.1109/tce.2012.6227420
- K vecinos más próximos. Retrieved from https://es.wikipedia.org/wiki/K_vecinos_más_próximos
- keras-retinanet. Retrieved from https://github.com/fizyr/keras-retinanet

- keras-yolo3. Retrieved from https://github.com/qqwweee/keras-yolo3
- Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160, 3-24.
- Lin, T., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature Pyramid Networks for Object Detection. 2017 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2017.106
- Lin, T., Goyal, P., Girshick, R., He, K., & Dollar, P. (2017). Focal Loss for Dense Object Detection. 2017 IEEE International Conference On Computer Vision (ICCV). doi: 10.1109/iccv.2017.324
- Liu, S., & Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. 2015 3Rd IAPR Asian Conference On Pattern Recognition (ACPR). doi: 10.1109/acpr.2015.7486599
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. (2016). SSD: Single Shot MultiBox Detector. Computer Vision – ECCV 2016, 21-37. doi: 10.1007/978-3-319-46448-0_2
- Muñoz-Salinas, R., García-Silvente, M. & Medina Carnicer, R. (2008). Adaptive multi-modal stereo people tracking without background modelling. Journal of Visual Communication and Image Representation, 19(2), pp.75-91.
- Muñoz-Salinas, R., García-Silvente, M., & Medina Carnicer, R. (2008). Adaptive multi-modal stereo people tracking without background modelling. Journal Of Visual Communication And Image Representation, 19(2), 75-91. doi: 10.1016/j.jvcir.2007.07.004
- Real, R., & Vargas, J. (1996). The Probabilistic Basis of Jaccard's Index of Similarity. Systematic Biology, 45(3), 380-385. doi: 10.1093/sysbio/45.3.380
- Red bayesiana. (2018). Retrieved from https://es.wikipedia.org/wiki/Red_bayesiana
- Red neuronal artificial. Retrieved from https://es.wikipedia.org/wiki/Red_neuronal_artificial

- Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2017.690
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR). doi: 10.1109/cvpr.2016.91
- Redmon, Joseph & Farhadi, Ali (2018). YOLOv3: An Incremental Improvement. arXiv
- Salas, J., & Tomasi, C. (2011). People Detection Using Color and Depth Images. Lecture Notes In Computer Science, 127-135. doi: 10.1007/978-3-642-21587-2_14
- Satake, J., Chiba, M., & Miura, J. (2013). Visual Person Identification Using a Distance-dependent Appearance Model for a Person Following Robot. International Journal Of Automation And Computing, 10(5), 438-446. doi: 10.1007/s11633-013-0740-y
- Special issue on computer vision for RGB-D sensors: Kinect and its applications. (2012). IEEE Transactions On Systems, Man, And Cybernetics, Part B (Cybernetics), 42(4), 1295-1296. doi: 10.1109/tsmcb.2012.2207010
- ssd_keras. Retrieved from https://github.com/pierluigiferrari/ssd_keras
- Vo, D., Jiang, L., & Zell, A. (2014). Real time person detection and tracking by mobile robots using RGB-D images. 2014 IEEE International Conference On Robotics And Biomimetics (ROBIO 2014). doi: 10.1109/robio.2014.7090411
- Wu, X., Mao, X., Chen, L., Xue, Y., & Rovetta, A. (2015). Depth image-based hand tracking in complex scene. Optik - International Journal For Light And Electron Optics, 126(20), 2757-2763. doi: 10.1016/j.ijleo.2015.07.027.
- Xue, H., Liu, Y., Cai, D. & He, X. (2016). Tracking people in RGBD videos using deep learning and motion clues. Neurocomputing, 204, pp.70-76.
- Yuan, D., Qiang, J., & Yin, J. (2017). Image segmentation via foreground and background semantic descriptors. Journal Of Electronic Imaging, 26(05), 1. doi: 10.1117/1.jei.26.5.053004

APÉNDICE

Apéndice A:

```
import numpy as np
import copy
from open3d import *
import sys
import os
from PIL import Image
import math as m
import pandas as pd
import cv2

def CrearArchivo(NombreArchivo):
    archivo=open(NombreArchivo, 'w')
    archivo.close()

def generate_pointcloud(rgb_file,depth_file,ply_file):
    rgb = Image.open(rgb_file)
    rgb = rgb.convert('RGB')
    depth = Image.open(depth_file)
    depth = depth.convert('I')

    if rgb.size != depth.size:
        raise Exception("Color and depth image do not have the same resolution.")
    if rgb.mode != "RGB":
```

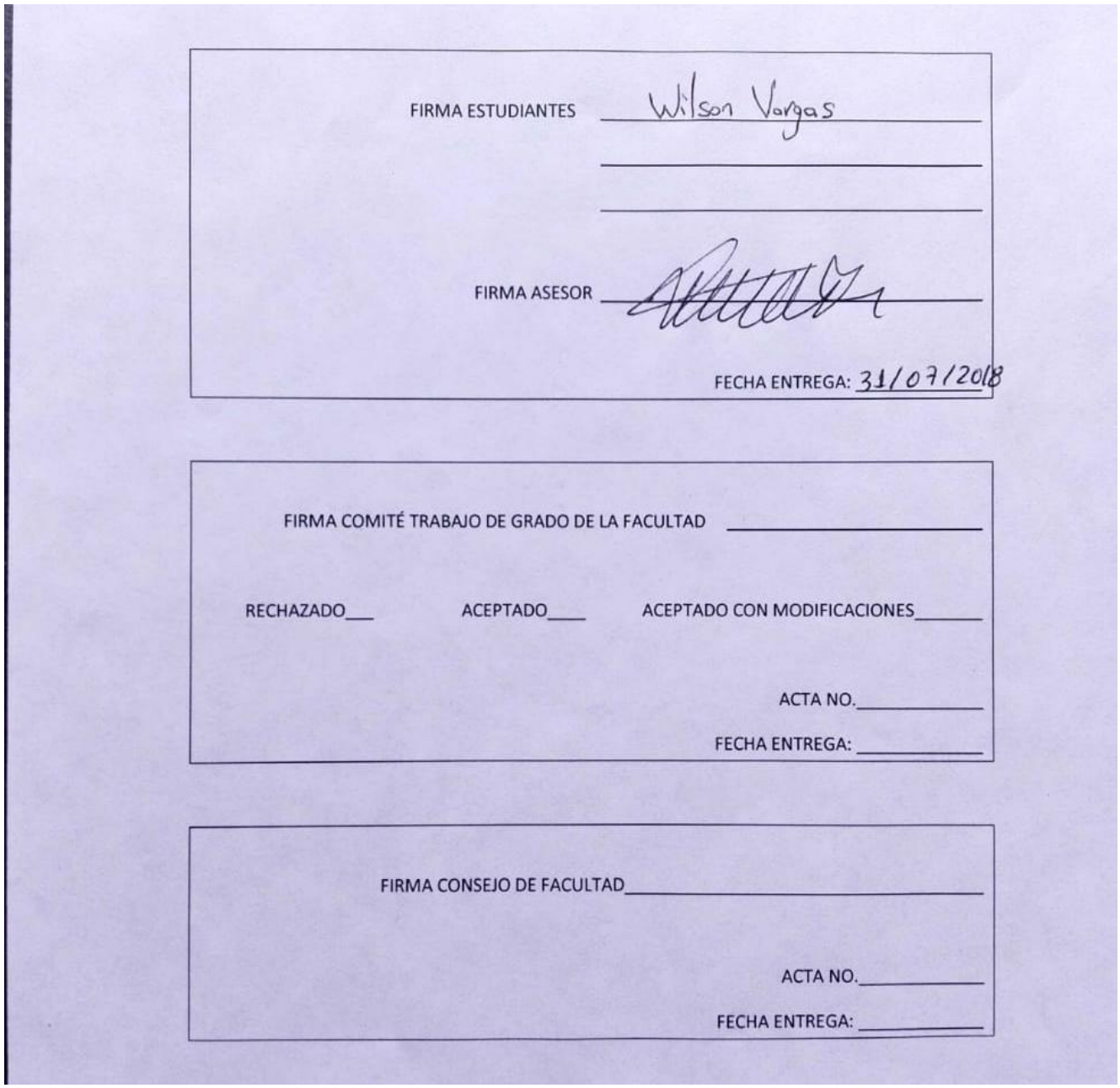
```
    raise Exception("Color image is not in RGB format")
  if depth.mode != "I":
    raise Exception("Depth image is not in intensity format")


  points = []
  for v in range(rgb.size[1]):
    for u in range(rgb.size[0]):
      color = rgb.getpixel((u,v))
      Z = depth.getpixel((u,v)) / scalingFactor
      d = depth.getpixel((u,v))
      #Z = 0.1236*(m.tan(d/(2842.5+1.1863)))
      if Z==0: continue
      X = (u - centerX) * Z / focalLength
      Y = (v - centerY) * Z / focalLength
      points.append("%f %f %f %d %d %d 0\n"%(X,Y,Z,color[0],color[1],color[2]))
      #points.append("%f %f %f 1\n"%(X,Y,Z))
  file = open(ply_file,"w")
  file.write('''ply
format ascii 1.0
element vertex %d
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
property uchar alpha
end_header
```

```
%s
'''%(len(points),"".join(points)))
    #file.write("".join(points))
    file.close()
    #print("succesfull")


def pick_points(pcd):
    print("")
    print("1) Please pick at least three correspondences using [shift + left click]")
    print("   Press [shift + right click] to undo point picking")
    print("2) Afther picking points, press q for close the window")
    vis = VisualizerWithEditing()
    vis.create_window()
    vis.add_geometry(pcd)
    vis.run() # user picks points
    vis.destroy_window()
    print("")
    return vis.get_picked_points()


def pickedPoins (ply_file, root):
    source = read_point_cloud(ply_file)
    picked_id_source = pick_points(source)
    #picked_id_target = pick_points(target)
    #print("len",len(source.points[:]))
    labels = np.zeros((len(source.points[:]),1))
    corr = np.zeros((len(picked_id_source),1))
    corr[:,0] = picked_id_source
    #corr[:,1] = picked_id_target
```

```
    a = int(corr[0,0])

    pcd_tree = KDTreeFlann(source)

    print("Find its neighbors with distance less than 0.2, paint green.")
    [k, idx, _] = pcd_tree.search_radius_vector_3d(source.points[a], 0.2)
    np.asarray(source.colors)[idx[1:], :] = [0, 1, 0]
    labels[idx]=1
    #print(labels)
    name = ply_file.split('/')[-1]
    name = name.split('.')[0]
    np.savetxt(root + "/labels/" + str(name) + ".txt", labels, fmt='%d')
    print("Visualize the point cloud.")
    draw_geometries([source])

if __name__ == "__main__":
    #demo_crop_geometry()

    focalLength = 368.096588
    centerX = 261.696594
    centerY = 202.522202
    scalingFactor = 1000.0

    cols = ["frame", "xmin", "ymin", "xmax", "ymax", "label"]
    datargb = pd.read_csv('/home/itm/BaseDatosRGB-D-
Junio20/PointsClouds/ETIQUETACOLOR.txt', names=cols, delimiter=',')
    datadepth = pd.read_csv('/home/itm/BaseDatosRGB-D-
Junio20/PointsClouds/ETIQUETADEPTHtxt', names=cols, delimiter=',')
```

```
xmin = datargb.ymin.values
ymin = datargb.ymin.values
xmax = datargb.xmax.values
ymax = datargb.ymax.values
frames = datargb.frame.values
dframes = datadepth.frame.values

ruta = frame[0].split('/')[:-2]
if not os.path.exists(ruta):
   os.makedirs(ruta)
rutapoint = "pointclouds"
rgb_files= []
depth_files = []

for i in range(len(xmin)):
   rgb_path = cv2.imread('/home/itm/Downloads/' + frames[i])
   depth_path = cv2.imread('/home/itm/Downloads/' + dframes[i])

   rgb_path = rgb_path[ymin[i]:ymax[i],xmin[i]:ymin[i],:]
   depth_path = rgb_path2[ymin[i]:ymax[i],xmin[i]:ymin[i],:]
   rgb_files.append(rgb_path)
   depth_files.append(depth_path)

   for f in range(len(depth_files)):
      root = os.path.join(ruta, rutapoint,"pointclouds")
      CrearArchivo(root + str(f) + '.ply')
      rgb_file= rgb_files[f]
```

```
        depth_file= depth_files[f]
        ply_file=root + str(f) + ".ply"
        generate_pointcloud(rgb_file,depth_file,ply_file)
        pickedPoins(ply_file, ruta)

  print("succesfull")

#parametros intr. fx 368.096588, fy 368.096588, cx 261.696594, cy 202.522202
```



FIRMA ESTUDIANTES Wilson Vargas

FIRMA ASESOR

FECHA ENTREGA: 31/07/2018

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD

RECHAZADO___ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____