Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Análisis comparativo de arquitecturas de aprendizaje profundo para odometría visual

Trabajo Fin de Máster

Presentado por: Tayupanta Zúñiga, Eduardo

Director: Pedraza Gómara, Luis

Ciudad: Quito
Fecha: 23 de septiembre de 2020

Resumen

Una de las principales líneas de investigación en las áreas de la robótica, es el desarrollo de técnicas que permitan que los robots sean capaces de desplazarse autónomamente en diferentes entornos. Con el fin de que el robot obtenga los valores de posición y orientación se han empleado técnicas de odometría visual, que permiten la estimación de posición y orientación a través de secuencias de imágenes. En el presente estudio se ha realizado la implementación, entrenamiento y comparativa de rendimiento de tres diferentes arquitecturas de odometría visual que emplean modelos de redes neuronales *CNN* y su combinación con *LSTM* o *GRU.* Se ha demostrado que la utilización del aprendizaje profundo en la odometría visual puede llegar a estimar valores cercanos a los valores reales de las secuencias de imágenes, por consiguiente, los resultados son competitivos con los métodos analíticos desarrollados para la estimación de posición y orientación.

Palabras Clave: Aprendizaje profundo, GRU, LSTM, Odometría visual, SLAM.

Abstract

One of the main lines of research in the areas of robotics is the development of techniques that allow robots to be able to move autonomously in different environments. For the robot to obtain the position and orientation values, visual odometry techniques have been used, which allow position estimation through image sequences. In the present study, the implementation, training, and performance comparison of three different visual odometry architectures that use CNN neural network models and their combination with LSTM or GRU have been carried out. It has been shown that the use of deep learning in visual odometry can estimate values close to the real values of the image sequences, therefore, the results are competitive with the analytical methods developed for the estimation of position and orientation.

Keywords: Deep learning, GRU, LSTM, SLAM, Visual Odometry.

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

La navegación autónoma de un agente ha sido un área de investigación en la robótica, la cual busca que el robot sea capaz de desplazarse en entornos del mundo real. Se han desarrollado diferentes métodos desde modelos analíticos hasta modelos de aprendizaje automático para la estimación de posición y orientación del robot. Los estudios de odometría visual utilizando modelos de aprendizaje profundo permiten la estimación de posición y orientación a partir de una secuencia de imágenes. Si bien existen modelos analíticos que pueden solventar el problema de la odometría visual, se observa que los modelos de aprendizaje profundo pueden tener resultados competitivos sin tener una gran cantidad de parámetros para su procesamiento, el cual depende en gran medida de las geometrías que presentan las cámaras con las que fueron capturadas las secuencias de imágenes.

Los modelos analíticos necesitan las propiedades intrínsecas de las cámaras, así como cierto tipo de condiciones al capturar las imágenes, tales como: la resolución, la cantidad de luz y la variación en contraste, entre otras. Por consiguiente, los modelos de redes neuronales han demostrado tener la capacidad de extraer características durante su entrenamiento, de esta manera resultan atractivos para la estimación de posición y orientación a partir de secuencias de imágenes (Agrawal et al., 2015).

Las redes neuronales convolucionales fueron desarrolladas a finales de los 90, pero en los últimos años han tenido un gran éxito en tareas de procesamiento de imágenes, ya que pueden extraer características concretas de estas. Su desarrollo es más complejo al utilizar una gran cantidad de capas convolucionales en cascada, de esta forma, en cada capa convolucional la red neuronal es capaz de aprender cierto tipo de características (Perez et al., 2013).

Las redes neuronales recurrentes pueden extraer información a través de series temporales, es decir, permiten tratar la dimensión del tiempo. Estos modelos fueron desarrollados en la década de los 80, en sus inicios, eran difíciles de tratar ya que necesitaban un gran esfuerzo computacional. Con los avances tecnológicos durante los últimos años se han vuelto más accesibles y su popularidad ha ido creciendo (Chen et al., 2019).

Las redes neuronales *LSTM* (*Long Short Term Memory*) son una extensión de las redes neuronales recurrentes, logran ampliar la memoria para aprender a través de sucesos que han transcurrido durante el tiempo. Este tipo de modelos mantienen información que puede ser modificada dependiendo de la importancia en la que es asignada la información (Sak et al., 2014). Otra variante de las redes neuronales recurrentes son las *GRU* (*Gated Recurrent*

Unit), que aparecieron en el año 2014 y conservan el mismo principio de las *LSTM*, con la diferencia de que son computacionalmente más eficientes que las *LSTM* (J. Chung et al., 2014).

Una vez analizados los estudios previos para solventar el problema de odometría visual, se presenta el estudio comparativo de tres tipos de arquitecturas que emplean *CNN*, *LSTM* y *GRU*. Cada arquitectura fue implementada en *Python* con la librería *TensorFlow*, debido a que no se cuenta con el código fuente desarrollado por los investigadores. El lenguaje de programación *Python* se caracteriza por ser versátil y práctico. Además, la librería *TensorFlow* permite implementar arquitecturas de redes neuronales sin necesidad de utilizar numerosas líneas de código.

1.1. Motivación

La robótica ha sido una ciencia que ha tenido un acompañamiento cada vez mayor en la vida cotidiana del ser humano, partiendo desde robots manipuladores, de servicio y entretenimiento, hasta la robótica móvil. Los robots móviles son sistemas electromecánicos que logran desplazarse autónomamente en un entorno, mediante sistemas de locomoción y cuentan con sensores que permiten monitorear las posiciones a cada instante de tiempo.

Los factores importantes para la robótica móvil son los concernientes a la evasión de obstáculos, posicionamiento y seguimiento de trayectoria. Referente al posicionamiento, se han desarrollado múltiples investigaciones que han utilizado sensores ultrasónicos para la navegación de los agentes por un entorno (J Borenstein et al., 1996).

Además, otros estudios emplean la odometría para la estimación del posicionamiento del robot. El termino odometría se refiere a la medida de las posiciones relativas de un robot móvil, se utiliza *encoders* para el cálculo de la orientación y rotación de las ruedas (Johann Borenstein & Feng, 1995).

Los *encoders* conocidos como codificadores rotatorios son dispositivos electromecánicos capaces de convertir la posición angular de un eje en código digital, se clasifican en absolutos y rotativos. Los *encoders* relativos son discos que se encuentran ubicados en el eje del motor, contiene múltiples ranuras que son codificadas a través de un sensor óptico, el patrón genera pulsos eléctricos cada vez que se interrumpe o permite el paso de luz del sensor óptico a medida que el motor gira (Venegas, 2009).

La odometría visual permite la estimación de posición y orientación de un robot, el método procesa diferentes imágenes capturadas mediante una cámara monocular o estéreo, se extrae características significativas de la imagen como: bordes, esquinas, manchas, entre otras. Posteriormente se analiza cómo estas características varían en imágenes consecutivas y se determina una estimación del desplazamiento del robot. Sin embargo, estos métodos de odometría visual para la estimación de posicionamiento de un robot, emplean combinaciones de sensores, cámaras estereotípicas, cálculos de correlación entre imágenes, *RANSAC* para la extracción de características, geometría epipolar, entre otras. Por ende, son computacionalmente exigentes (Grandón-Pastén et al., 2007).

El presente trabajo tiene como objetivo principal realizar un análisis comparativo de arquitecturas para estimar la posición de un agente a partir de una secuencia de imágenes, denominado odometría visual. Se parte de un análisis de diferentes métodos de odometría visual que emplean conceptos geométricos hasta la utilización de máquinas de aprendizaje automático.

La selección del presente trabajo responde a dos procesos:

1. El primero es que existen ciertas restricciones en los métodos tradicionales para la estimación de posición y orientación.

2. El segundo es determinar la arquitectura de aprendizaje profundo capaz de estimar la posición y orientación. De modo que los valores obtenidos sean competitivos con los métodos tradicionales.

Si bien la odometría visual ha sido solucionada a través de métodos tradicionales aplicando métodos probabilísticos y con la exploración de la geometría epipolar. Estos modelos necesitan conocimientos adicionales como los parámetros intrínsecos de las cámaras, calidad, contraste o cierta cantidad de luz en las imágenes, entre otras. Por lo tanto, se desea reducir el proceso de obtención de estos parámetros, que son necesarios para lograr una estimación que se asemeje a las posiciones reales.

De esta forma el aprendizaje profundo ha logrado ser un sistema de caja negra capaz de extraer y aprender información de secuencias de imágenes para estimar la posición del agente, sin necesidad de utilizar estrategias geométricas o condiciones especiales para solventar dicho problema, por lo que hoy en día el aprendizaje profundo ha logrado ganar más áreas de aplicación, ya que por su gran capacidad de precisión se ha convertido en una de las tecnologías de vanguardia que pueden solventar un sin número de aplicaciones.

Pero al contar con algunas arquitecturas de redes neuronales capaces de inferir la posición a través de secuencias de imagen, se pretende determinar la arquitectura que cuente con mejor desempeño y precisión al determinar las estimaciones de posición y orientación.

1.2. Planteamiento del trabajo

De acuerdo con lo descrito en el apartado anterior. Se observa que los métodos tradicionales necesitan conocimiento de parámetros adicionales, referentes a la geometría intrínseca de la cámara, calidad de la imagen y condiciones especiales para la estimación del posicionamiento de un agente robótico.

La existencia de arquitecturas de odometría visual que emplean diferentes modelos de redes neuronales, ha permitido estimar la posición del robot con tan solo secuencias de imágenes. Sin la necesidad de emplear gran cantidad de sensores, cálculo geométrico de las cámaras, así como utilizar algoritmos que conlleva mucho esfuerzo computacional como *RANSAC* y por eso se pretende realizar un estudio comparativo para definir la arquitectura con mejor desempeño en la estimación de la posición y orientación de un robot en el mundo real.

Por ende, se ha realizado un extenso estudio de diferentes arquitecturas de odometría visual que utilizan el aprendizaje profundo. Al no contar con implementaciones oficiales de las arquitecturas a comparar, se desarrolló los modelos en base al lenguaje de programación *Python* con la librería *TensorFlow*. Se empeló el conjunto de datos *KITTI* (Geiger et al., 2012), por ser un conjunto de datos extenso con un total de 11 secuencias de imágenes de alta resolución, de las cuales solo 6 secuencias fueron empleadas para entrenar los modelos.

Para reducir el tiempo de entrenamiento y aumentar la precisión de las redes neuronales se ha empleado modelos pre entrenados en las capas convolucionales, esto permite la inferencia del flujo óptimo de un par de imágenes consecutivas. Posteriormente estos valores se utilizarán como entrada de las capas *LSTM* o *GRU* para la estimación de la posición del agente robótico. Se realizan cálculos de error absoluto y el *RMSE*

Se observa que la implementación y entrenamiento de las arquitecturas pueda generar modelos capaces de inferir resultados acordes a lo esperado, de manera que se puede determinar al modelo con mayor precisión en la estimación de la posición y orientación utilizando métricas de error como: el error absoluto y el *RMSE*.

Se utilizaron estas métricas en secuencias de imágenes diferentes a las utilizadas durante el entrenamiento. Además, se aporta con la implementación de las arquitecturas para su uso

libre, así como los modelos pre entrenados de las arquitecturas, el proyecto se encuentra en el repositorio GitHub para su libre uso.

Enlace al repositorio GitHub que contiene el código desarrollado:
https://github.com/EduardoTayupanta/VisualOdometrys

1.3. Estructura de la memoria

Cabe destacar que el objetivo principal del presente trabajo es el estudio comparativo de diferentes arquitecturas de aprendizaje profundo para odometría visual. Es por ello que a continuación, se describe brevemente lo que contiene en cada uno de los capítulos, siendo el capítulo 2, donde se realiza un análisis del avance de los modelos para la estimación de la odometría visual, desde la aplicación de conceptos geométricos hasta utilizar modelos más recientes con la aplicación de técnicas de aprendizaje supervisado.

En el capítulo 3, se realiza una investigación de los diferentes métodos de odometría visual con neuronales profundas, así como apartados teóricos sobre los elementos a utilizar en el desarrollo de las arquitecturas como: *CNN*, *LSTM* y *GRU*. Adicional una descripción de la función de perdida utilizada en las arquitecturas seleccionadas, conjuntos de datos y librerías disponibles para implementar y entrenar los modelos de redes neuronales.

En el capítulo 4, se describe el objetivo que se desea alcanzar a través del desarrollo del presente trabajo, así como los objetivos específicos para lograr alcanzar el objetivo general. Adicional, se describe la metodología de trabajo, cuenta con 4 fases siendo la primera el estudio de las arquitecturas de odometría visual, la segunda se refiere al análisis del conjunto de datos a utilizar, la tercera se refiere a la implementación y entrenamiento de las arquitecturas seleccionadas y la cuarta ejecución y análisis de resultados.

En el capítulo 5, se describe las arquitecturas a utilizar para desarrollar las comparativas, así como los diferentes parámetros a ser analizados como: tiempo de entrenamiento, valores de perdida durante el entrenamiento, definición de las métricas de error absoluto y *RMSE* obtenidos para la estimación del posicionamiento del agente robótico, entre otras.

En el capítulo 6, se desarrolla con detalle la comparativa de las arquitecturas, así como los resultados obtenidos en la estimación de la posición. Se utilizó secuencias de imágenes diferentes durante la etapa de entrenamiento. Se realizó el cálculo de error en la precisión de

las arquitecturas tanto en posición como en rotación y se presentan gráficas separadas para el error de estimación de las arquitecturas.

En el capítulo 7, se aborda la discusión y el análisis de los resultados obtenidos a partir del capítulo 6, así como análisis del error y el *RMSE* de las secuencias utilizadas para la evaluación de las arquitecturas, donde se observa que la arquitectura que presenta menor error en la estimación de posición es la *PoseConvGRU*

Finalmente, en el capítulo 8, se presenta un pequeño resumen del trabajo realizado, aspectos relevantes que surgieron al comparar las diferentes arquitecturas, así como una valoración crítica de cómo los objetivos que se plantearon fueron finalmente alcanzados y se describe a los trabajos futuros que pueden ser desarrollados a partir del aporte que genera el trabajo realizado.

2. Análisis del contexto

En este capítulo se va a analizar conceptos relacionados sobre robótica móvil, autonomía robótica, el problema de la odometría visual, resaltando los métodos analíticos para la estimación de posición a partir de la odometría visual.

Las principales tendencias de investigación en la robótica es el desarrollo de técnicas para navegación autónoma en ambientes del mundo real (Saffiotti, 1997). Para que un robot sea completamente autónomo debe ser capaz de movilizarse en un entorno desconocido al mismo tiempo que construye un mapa incremental y estima su posición como se muestra en la Figura *1*, la aplicación de técnicas de localización y mapeo simultáneo (*SLAM Simultaneous Localization and Mapping*) proporcionan los medios para que un robot móvil sea realmente autónomo (Durrant-Whyte & Bailey, 2006).

Figura 1. *ORB-SLAM 2* (Mur-Artal & Tardos, 2017)

Las mejoras de rendimiento de aplicaciones desarrolladas por visión por computadora, la combinación hardware especializado como cámaras estéreo, cámaras RGB-D y sensores capaces de medir distancia desde el emisor al objeto, han hecho posible la implementación de procesos en tiempo real proporcionando estimaciones de profundidad de cada píxel (Huang et al., 2017).

Métodos basados en características dispersas que se enfoca en la visión estereotípica, presentan un gran inconveniente en la precisión al no incluir la transición de tiempo en cada cuadro. Para superar el inconveniente, se desarrolló una alternativa llamada visual SLAM que optimiza con precisión de manera continua el mapa de características al aplicar un filtro de Kalman extendido (Jeong & Lee, 2006), pero debido a la complejidad computacional y a la

dependencia de sincronización de cuadros emparejados genera errores en la estimación de movimiento (Martinez-Cantin & Castellanos, 2005).

El SLAM monocular mejora la incertidumbre al emplear un filtro de partículas, debido a su alta complejidad computacional lo que genera inconvenientes en aplicaciones en tiempo real (M. Li et al., 2008). Los modelos basados en PTAM (seguimiento y mapeo paralelo) mejora notablemente el tiempo real, el modelo utiliza la extracción de fotogramas clave (Klein & Murray, 2007).

Los métodos en base a unión de sub mapas locales han mejorado la agilidad de un sistema SLAM, se emplean filtros de información dispersos capaces de reducir el coste computacional para la construcción de mapas globales, el método es resistente al desenfoque causado por el movimiento rápido de la captura de las imágenes (Huang et al., 2009). Un último enfoque desarrolla SLAM a base de cámara monoculares, los autores centran su investigación en mapeo, localización y cierre de bucles. Utiliza funciones ORB y describe un nuevo concepto Essential Graph que acelera el proceso de correcciones de cierre de bucle (detección con corrección). Puede ser ejecutado en CPU teniendo un gran desempeño en tiempo real (Mur-Artal et al., 2015).

Para el cálculo de la localización de un robot móvil o para drones (Qin et al., 2018) se puede emplear el método de odometría visual, que se refiere a la estimación de movimiento a partir de una secuencia sucesiva de imágenes, se emplea una arquitectura geométrica de hipótesis y pruebas (Nistér et al., 2004). El término odometría visual fue escogido por guardar cierta similitud con la odometría de ruedas, que se refiere a la estimación de movimiento progresivo de un vehículo mediante el número de revoluciones por minuto (Scaramuzza & Fraundorfer, 2011). La odometría visual puede combinarse con información de otras fuentes como GPS, sensores de proximidad, codificadores de rueda, entre otros como se muestra en la Figura *2*.

Figura 2. *Odometría Visual conjunto de datos KITTI* (Geiger et al., 2012)

A continuación, se presentan diferentes métodos para el cálculo de la odometría visual como el estudio *Real-time stereo visual odometry for autonomous ground vehicles* (Howard, 2008) que describe un algoritmo de odometría visual a partir de pares consecutivos de imágenes estéreo, no realiza suposiciones del movimiento de la cámara y opera con imágenes de

disparidad densa, el error en la estimación de posición es del 0.25% del total de la distancia recorrida utilizando imágenes de 512×384 de dimensión.

Para el método de extracción de características geométricas de las imágenes monoculares, se aplica un riguroso procesamiento matemático, gráficamente se puede observar en la Figura *3*, para así obtener la posición a partir de la odometría visual (Jiao et al., 2018). Sin embargo, las cámaras presentan propiedades internas llamados parámetros intrínsecos que son valores que derivan de la geometría epipolar (Du & Brady, 1993).

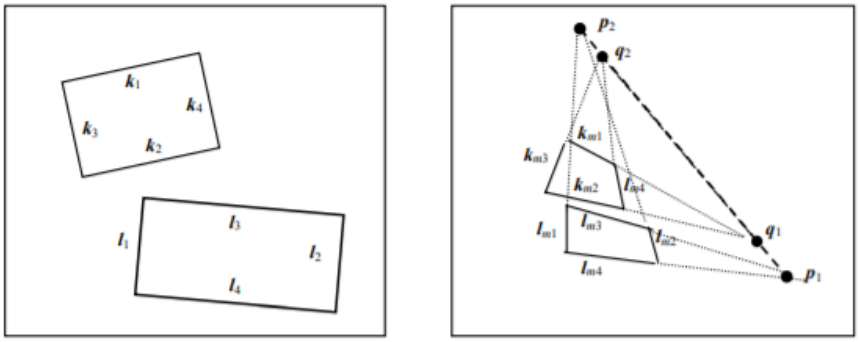

Figura 3. *Geometría epipolar* (Fu-chao et al., 2003)

Existen diferentes métodos de calibración de cámaras para la obtención de parámetros intrínsecos y extrínsecos como (Dawson-Howe & Vernon, 1994) que define los parámetros de la cámara en una matriz que define el mapeo 3D al plano de la imagen, emplea un único objeto de calibración con la utilización de transformaciones de *Hough* para el cálculo de líneas de cuadriculas de calibración Figura *4*. Otro método es desarrollado por (Strobl & Hirzinger, 2011), que presenta un método novedoso con la utilización de objetos planos precisos, estimación máxima de probabilidades en el caso de objetos inexactos.

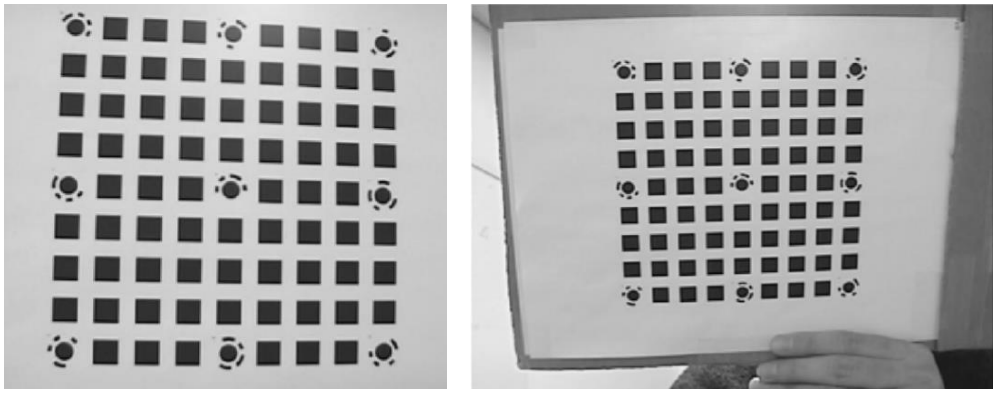

Figura 4. *Método de calibración de cámaras* (Z. Zhang, 2000)

Los métodos directos para odometría visual, no emplean cálculos para determinar descriptores o puntos clave en las imágenes siendo su principal objetivo el estimar movimiento

a partir de la iluminación que presenta el píxel, los métodos suponen la invarianza de la imagen en escala de grises lo que provee mayor robustez y precisión.

Métodos basados en cámara monoculares realizan cálculos para estimar la postura de la cámara utilizando un mapa de profundidad semidenso, se puede ejecutar en tiempo real en una CPU, pero genera incertidumbre de posición al no emplear detección de cierre de bucle (Engel, 2013). Otro método se basa en la predicción probabilística de la profundidad de cada píxel es sensible a cambios de iluminación, pero reduce la incertidumbre en la estimación de la posición debido a que no se basa en la uniformidad de intensidad del píxel (Newcombe et al., 2011).

El método de odometría visual monocular semidirecto elimina la costosa tarea de extraer características y coincidencias entre imágenes para estimación de movimiento, extrae bloques en la imagen mejorando la solidez, implementa métodos probabilísticos de mapeo para la estimación de puntos 3D y el alto procesamiento de 55 cuadros por segundo para un sistema a bordo y a 300 cuadros por segundo para una computadora de escritorio, a demostrado ser robusto en escenarios con alta frecuencia y poca textura repetitiva (Forster et al., 2014).

Al ser un método rápido logra tener éxito cuando la cámara se mueve despacio ya que la búsqueda del gradiente puede dar como resultado que el algoritmo falle, de modo que el método directo es sensible al cambio de iluminación. Los problemas de los métodos convencionales para estimar la posición y orientación a partir de imágenes consecutivas como: extracción de características de la imagen, alto costo computacional, presentan sensibilidad a la iluminación, se requiere un alto grado de texturas y un elevado número de cuadros por segundo (Jiao et al., 2018).

Actualmente el aprendizaje profundo ha sido ampliamente desarrollado en el área de visión por computadora, la aplicación de redes neuronales convolucionales ha generado varias aplicaciones con cámaras monoculares como, por ejemplo: la clasificación de imágenes, detección de objetos, reconocimiento facial, entre otras. A pesar del gran éxito que ha tenido el uso del aprendizaje profundo con imágenes, son pocas las alternativas desarrolladas para solventar los inconvenientes presentados con los métodos tradicionales de la odometría visual (Steffen et al., 2017).

3. Estado del arte

En el presente apartado se presenta un análisis de aprendizaje profundo y su aplicación en odometría visual, así como diferentes arquitecturas de modelos de red neuronal compuestos por varias capas interconectadas que logran que la red aprenda representaciones de datos con varios niveles de abstracción.

3.1. Aprendizaje profundo

Un subconjunto de algoritmos de aprendizaje automático es el aprendizaje profundo. El aprendizaje profundo son algoritmos que permiten procesar grandes conjuntos de datos para resolver de manera eficiente los problemas tradicionales de inteligencia artificial. Las recientes mejoras en *hardware* han fortalecido el estudio de diferentes arquitecturas de redes neuronales, permitiendo un menor coste computacional y reduciendo el tiempo de procesamiento. Han tenido éxito en transferencia de aprendizaje, estimación de profundidad, odometría visual, visión por computadora, sistemas de control, robótica móvil, entre otras (Guo et al., 2016).

El aprendizaje profundo ha sido ampliamente utilizado en el área de visión por computadora ya que emplean redes neuronales convolucionales para la detección y extracción de patrones en imágenes (Brunetti et al., 2018), gracias al cómputo paralelo ha fortalecido la aplicación de redes convolucionales y las redes neuronales recurrentes motivando soluciones en tiempo real para problemas de la robótica móvil (Y. Zhang, 2006).

3.1.1. Red neuronal convolucional

Las redes neuronales convolucionales han demostrado ser capaces de solventar varios retos de visión por computadora, ya que son capaces de superar a las máquinas de aprendizaje automático. Una red neuronal convolucional es un tipo de red neuronal artificial para el procesamiento de imágenes ya que son especialmente apropiadas para el procesamiento de las matrices de pixeles que conforman la imagen.

El término convolución se refiere a la operación matricial de aplicar filtros que recorren la imagen (Steffen et al., 2017).

La Figura *5* muestra una capa convolucional, se puede visualizar la capa como pequeñas cuadriculas llamados *kernels* que recorren la imagen en búsqueda de patrones. Si el píxel coincide con los patrones del *kernel* el resultado es un valor positivo y si no coincide de vuelve

un valor cercano a cero (Albawi et al., 2018). Un parámetro principal para las capas convolucionales es el uso de *stride*, que se refiere al salto del *kernel* entre píxeles.

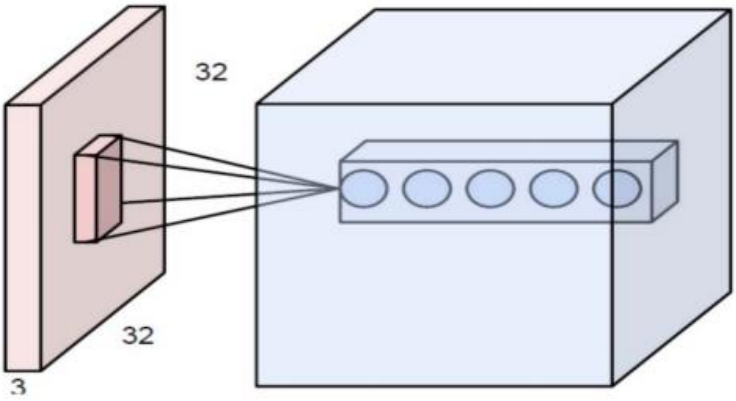

Figura 5. *Operación de convolución en CNN* (Albawi et al., 2018).

Como se observa en Figura *6* el valor de *stride* es de 1, esto se refiere a que el *kernel* se va a deslizar un píxel hacia la derecha hasta llegar al final horizontal, posteriormente el *kernel* saltará un píxel vertical hacia abajo e iniciará de nuevo en la posición izquierda y repetirá el deslizamiento horizontal y así sucesivamente hasta completar toda la imagen.

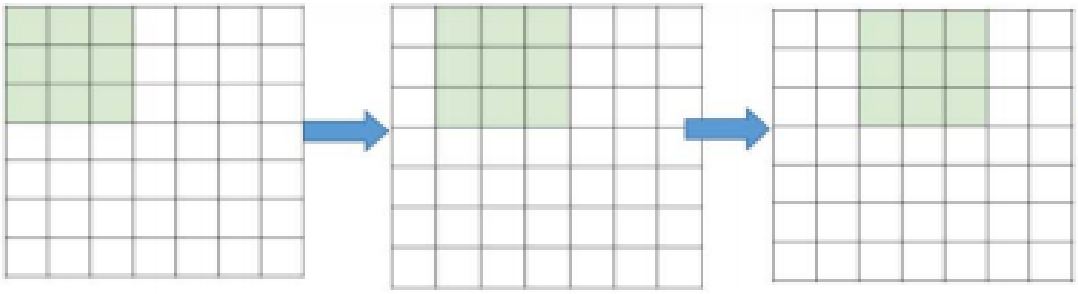

Figura 6. *Ejemplo de paso del filtro en la imagen con valor Stride=1* (Albawi et al., 2018)

El cambio de valor del *stride* permite la reducción de tamaño de salida de la capa convolucional, el valor viene dado por la siguiente ecuación:

$$O = 1 + \frac{N - F}{S}$$

Donde N se refiere a la dimensión de la imagen, F al tamaño del *kernel* y S se refiere al valor de *stride*.

Por ejemplo, se emplea la siguiente matriz de 5×5 pixeles que representa la imagen y un *kernel* 3×3 pixeles, como se muestra en la Figura *7*:

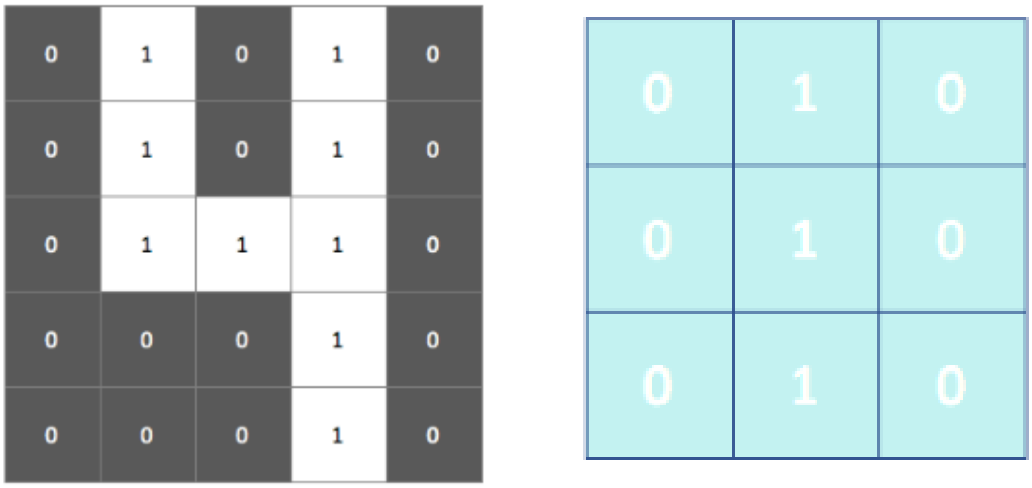

Figura 7. *Imagen de muestra de tamaño 5x5 y kernel de tamaño 3x3* (Jordan, 2017)

Para el ejemplo se emplea un valor de *stride* de 1 píxel. Siguiendo la formula anterior y reemplazando los valores del ejemplo:

$$O = 1 + \frac{5 - 3}{1} = 3$$

Esto se refiere a que la salida de la operación de convolución será de tamaño 3×3. El cálculo de la operación se muestra en la Figura *8*, donde se observa que el *kernel* se traslada de izquierda a derecha y de abajo hacia arriba con un *stride* de 1 píxel por la imagen generando la matriz de salida de tamaño 3×3 pixeles.

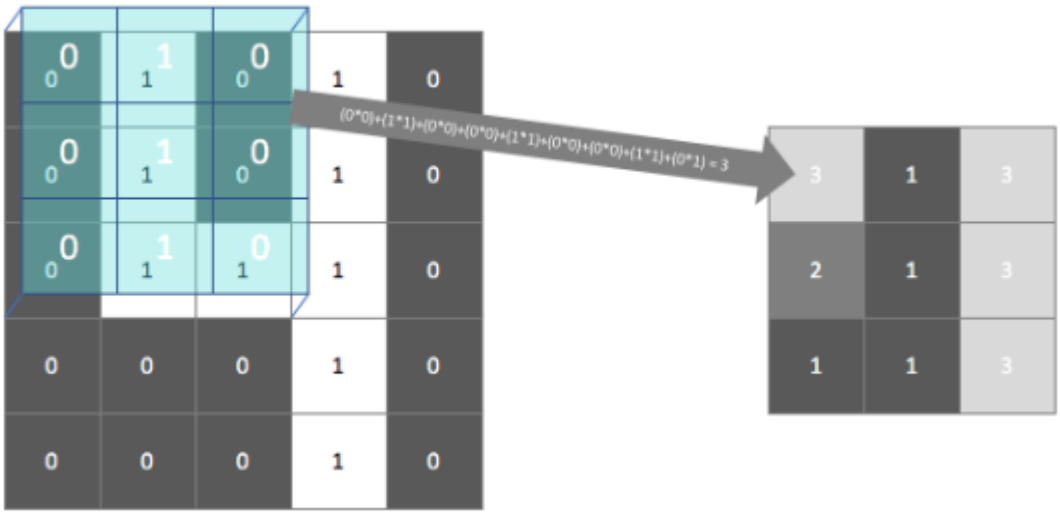

Figura 8. *Operación de convolución* (Jordan, 2017)

El cambio del valor de *stride* genera pérdida de información en los bordes. Para solventar el inconveniente, se añaden con valores de cero a los bordes de la imagen llamado *padding.*

Para el cálculo del tamaño de salida de la operación convolucional se aplica la siguiente ecuación.

$$O = 1 + \frac{N + 2P - F}{S}$$

Donde N se refiere a la dimensión de la imagen, P al tamaño de *padding*, F el tamaño del *kernel* y S se refiere al valor de *stride*.

Las CNN están constituidas por múltiples capas convolucionales, capas no lineales, capas de agrupación y capas completamente conectadas (Albawi et al., 2018), que han tenido mucho éxito para la extracción de características, reconocimiento de objetos sin tomar en cuenta las condiciones de iluminación gracias a las mejoras de velocidad de procesamiento ya que el cómputo se puede paralelizar gracias a la utilización de hardware especializado (Krizhevsky et al., 2012).

En la actualidad existen arquitecturas de redes neuronales convolucionales complejas que han sido entrenadas con una gran cantidad de datos para aplicaciones específicas, se puede utilizar modelos pre entrenados para el reconocimiento de objetos como *ResNet* (He et al., 2016), *SENet* , etc., o como los modelos de detección como *SDD Mobilenet*, *SDD Inception*, entre otras (Wu, 2017). Ya que la odometría visual está más relacionada con la geometría, arquitecturas de *CNN* como *VGGNet* (C. Chung et al., 2018) y *GoogleNet* (Szegedy et al., 2015) no pueden ser empleadas para este propósito.

Un ejemplo de arquitectura empleada para obtener un efecto característico efectivo para la odometría visual es *FlowNet* (Sen et al., 2017). *FlowNet* emplea *CNN* para estimaciones de flujo óptico con alta precisión hasta 10 fotogramas por segundo, ya que no se cuenta con grandes cantidades de conjuntos de datos la *CNN* fue entrenada con conjuntos de datos de simulaciones *Flying Chairs,* que han podido generalizar conjuntos de datos como *KITTI* y *Sentel* (Dosovitskiy et al., 2015).

Por ejemplo, se describe la arquitectura *AlexNet*, fue presentada en el desafío *ImageNet* en 2012 para clasificar 1000 imágenes. Es una de las primeras redes neuronales con una precisión de 84.7%, consta de 5 capas convolucionales y 3 totalmente conectadas, se utilizó la unidad de activación *ReLU* que aceleró 6 veces la convergencia de la red neuronal a diferencia de utilizar la función no lineal *tanh* (Alom et al., 2018), como se muestra en la Figura *9*.

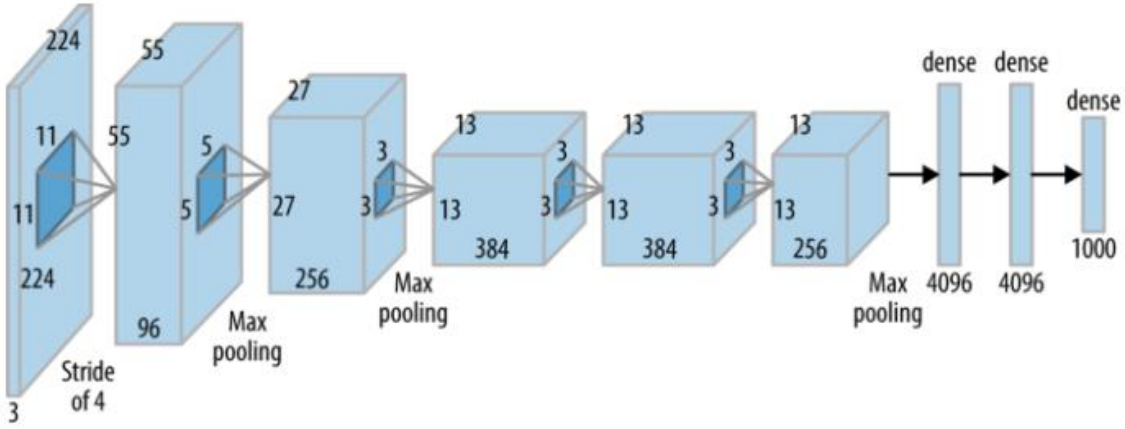

Figura 9. *AlexNet* (Anwar, 2019)

La red cuenta con 62 millones de parámetros que se modificaran durante el entrenamiento de la red neuronal. El entrenamiento de la red neuronal emplea imágenes *RGB* con dimensión de $227 \times 227 \times 3$ y genera un vector de probabilidades de 1000×1. Se empleó *Data Augmentation* para el aumento de la variación de imágenes en el conjunto de datos de entrenamiento. Además, se observa capas de agrupación máxima o *MaxPooling* después de la primera, segunda y quita capa convolucional, la capa *MaxPooling* es de tamaño 3×3 y con *stride* de 2 (Anwar, 2019).

3.1.2. Red neuronal recurrente

Las redes neuronales recurrentes *RNN* han sido desarrolladas desde la década de 1990. Su diseño permite el aprendizaje de patrones de tiempo variable o secuenciales, con un sistema de retroalimentación con conexiones de bucle cerrado. Permite que las salidas previas sean utilizadas como entradas mientras se mantengan los estados ocultos como se muestra en la Figura *10* (Hecht-Nielsen, 1989).

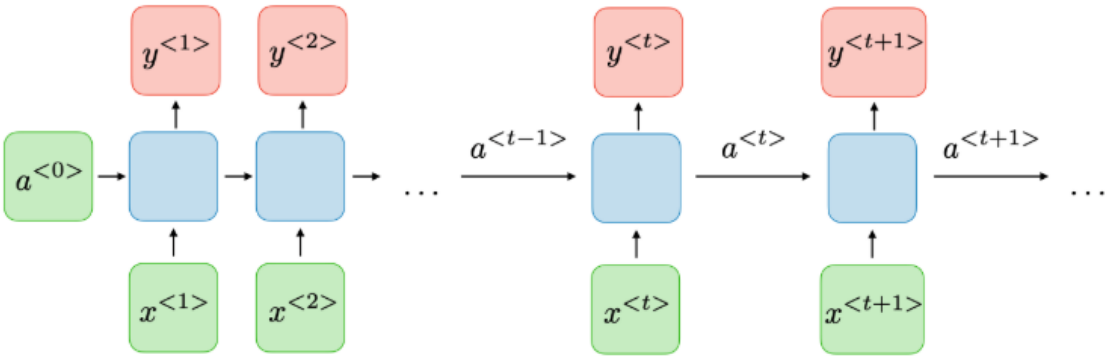

Figura 10. *Arquitectura de una RNN* (Amidi & Amidi, 2017)

Internamente las *RNN* presentan una sola capa $g2$ con unidad de activación $tanh$ que genera la salida y el estado $a^{<t>}$ se encuentra disponible para su utilización en la capa consecutiva, como se muestra en la Figura *11*.

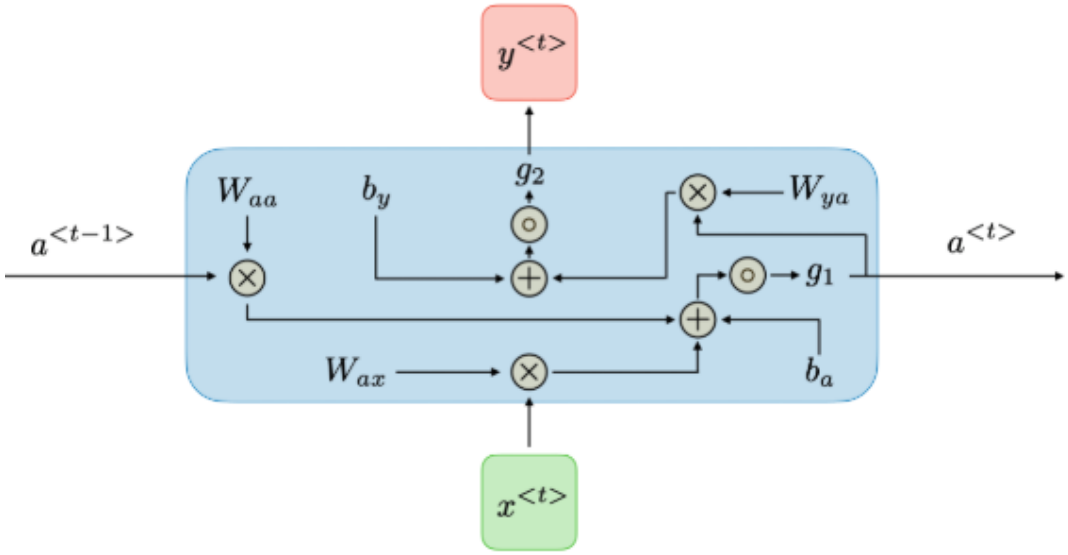

Figura 11. *Estructura interna RNN* (Amidi & Amidi, 2017)

Donde para cada paso en el tiempo t existe una activación $a^{<t>}$ que produce una salida $y^{<t>}$, se expresa esta transición de estados de acuerdo con las siguientes ecuaciones.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Donde W_{aa}, W_{ax}, W_{ya}, b_a y b_y se refiere a los coeficientes que son compartidos temporalmente y g_1, g_2 se refiere a las funciones de activación dentro de cada estado (Sak et al., 2014).

La aplicación de redes neuronales recurrentes ha demostrado superar significativamente a modelos como las redes neuronales convolucionales tradicionales. Se presenta mejoras en precisión, la entrada al modelo puede ser de cualquier longitud, el tamaño del modelo no depende de la longitud de la entrada, para las estimaciones cuenta con información histórica lo que permite que los pesos sean compartidos a lo largo del tiempo. Pero aún existen problemas en la implementación por su gran coste computacional. No se puede acceder a información que ha sido procesada hace mucho tiempo (Sen et al., 2017).

Aplicaciones como la generación de imágenes emplea *RNN* presentando un marco de auto codificación secuencias para construir iterativamente imágenes complejas (Gregor et al., 2015).

La combinación de *CNN* con *RNN* permite la estimación de poses de una secuencia de imágenes en *RGB*, estimaciones de profundidad y para odometría visual, a más de aprender características a través de las *CNN* puede modelar y relacionar temporalmente las dinámicas secuenciales de imágenes sucesivas (Wang et al., 2019).

3.1.3. LSTM

La arquitectura *LSTM (Long Short-Term Memory)* (Hochreiter & Schmidhuber, 1997) fue desarrollada para mejorar el error existente en las *RNN.* Se demostró que los retrasos presentes en la arquitectura eran inaccesibles debido a que el error al retro propagarse decae o explota exponencialmente (Hochreiter et al., 2001).

Las capas *LSTM* se encuentran formada por múltiples bloques de memoria conectados. Estos bloques contienen varias celdas de memoria conectada de forma recurrente y tres unidades multiplicativas denominadas *input*, *output* y *forget gate* que permiten operaciones continuas de lectura, escritura y reinicio de la celda de memoria, de forma que la *input* de cada celda es multiplicada por la unidad de activación de la puerta de entrada, el *output* es multiplicada por la puerta de salida y los valores correspondientes a las celdas anteriores son multiplicadas por la *forget gate* (Graves & Schmidhuber, 2005).

Las *LSTM* presentan la misma estructura interna de las *RNN*, pero con la variación de que el módulo de repetición presenta 4 capas que interactúan. De estas cuatro capas tres permiten el control del estado, como se muestra en la Figura *12*.

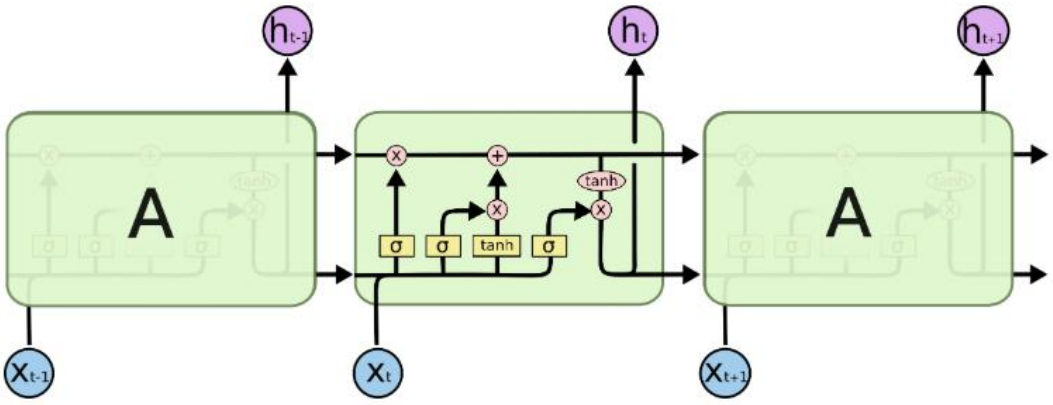

Figura 12. *Estructura interna LSTM* (Olah, 2015)

Por ejemplo, dada una entrada x_k en el tiempo k, con el estado oculto h_{k-1} en la celda de memoria c_{k-1} de la celda *LSTM* anterior, la *LSTM* se actualiza según la siguiente formula:

$$i_k = \sigma(W_{xi}x_k + W_{hi}h_{k-1} + b_i)$$

$$f_k = \sigma\left(W_{xf}x_k + W_{hf}h_{k-1} + b_f\right)$$

$$g_k = \tanh\left(W_{xg}x_k + W_{hg}h_{k-1} + b_g\right)$$

$$c_k = f_k \odot c_{k-1} + i_k \odot g_k$$

$$o_k = \sigma(W_{xo}x_k + W_{ho}h_{k-1} + b_o)$$

$$h_k = o_k \odot \tanh\left(c_k\right)$$

Donde $\odot$ se refiere al producto por elemento de cada vector, σ es la unidad de activación $sigmoid$, $tanh$ es la tangente hiperbólica no lineal, W se refiere a las matrices de peso correspondiente, b se refiere al *bias* y los términos i_k, f_k, g_k, c_k y o_k se refiere a la puerta *input, forget gate, memory cell* y *output gate* para un tiempo k respectivamente (Sen et al., 2017).

La principal característica de las *LSTM* es permite que el estado se ejecute por toda la cadena, presentando algunas operaciones lineales de multiplicación y adición. Esto permite que la información fluya a lo largo de ella sin cambios. Además, la celda permite agregar o eliminar información a través de su estructura llamada *gates*. Las *gates* permiten que la información transite opcionalmente, se utiliza la unidad de activación $sigmoid,$ generando valores entre 0 y 1. Lo que se refiere a que si el valor obtenido es 0 significa no dejar pasar la información y 1 que la información pase (Olah, 2015).

3.1.4. Bi-LSTM

Las siglas *Bi-LSTM* hace referencia a *(Bidirectional Long Short-Term Memory).* Investigadores denotan que las *RNN* pueden memorizar a corto plazo, lo que permite conectar información que ha sido retenida en memoria para el proceso actual (Jiao et al., 2018).

Siendo la información x_t en el tiempo t, la *RNN* se actualiza mediante la siguiente fórmula:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

Donde:

- h_t se refiere a la variable de estado de la capa en el tiempo t.
- y_t se refiere a la variable de resultado.
- W_{hh} y W_{xh} se refiere a las matrices de peso de la capa.
- b_h y b_y se refiere a vectores de polarización de la capa.
- f se refiere a la función de activación.

Las *RNN* generan problemas para el descenso de gradiente y su anulación en el proceso de propagación inversa, para solventar el inconveniente de las *RNN* se aplica el modelo *LSTM*. Para ello se añade al modelo *RNN* compuertas adicionales, pero los modelos *LSTM* solo pueden contener una función de memoria para la secuencia hacia adelante Figura *13*.

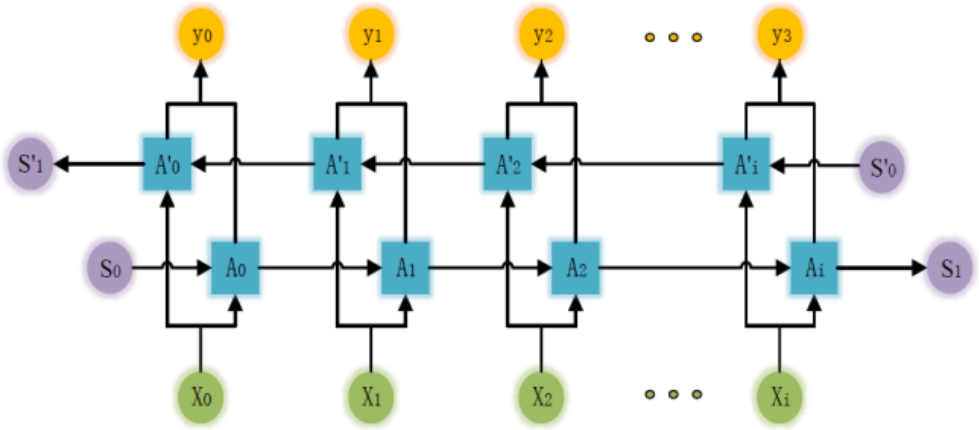

Figura 13. *Estructura Bi-LSTM* (Jiao et al., 2018)

Siendo la información x_t en el tiempo t:

$$s_t = f(Ux_t + Ws_{t-1})$$

$$s'_t = f(U'x_t + W's'_{t+1})$$

$$A_t = f(WA_{t-1} + Ux_t)$$

$$A'_t = f(W'A'_{t+1} + U'x_t)$$

$$y_t = g(VA_2 + V'A'_2)$$

Donde:

- s_t y s'_t se refiere a las variables de memoria hacia adelante y hacia atrás en el tiempo t.

- A_t y A'_t se refiere a las variables que se encuentran conectadas a las capas hacia adelante y hacia atrás en el tiempo t.
- y_t se refiere a la variable de salida del sistema.
- U, U', W, W', V y V' se refieren a la matriz de pesos.
- f y g se refiere a las funciones de activación no lineales.

Una aplicación de las *Bi-LSTM* para navegación de robots es para estimación del desplazamiento de un agente en un tiempo específico, para el entrenamiento de la red neuronal se emplean desplazamientos y mediciones inerciales donde la red ha demostrado aprender las características de movimiento, construir trayectorias con alta precisión a partir de una entrada de sensor (Chen et al., 2019).

Las arquitecturas *Bi-LSTM* pueden correlacionar las imágenes con mayor precisión que solo al utilizar las *LSTM*, un nuevo marco para el desarrollo de la odometría visual es la implementación de *CNN* con *Bi-LSTM*, de modo que se conecta el *Bi-LSTM* atrás del *CNN*, esto genera que la *RNN* pueda obtener información posterior y previa de la secuencia de entrenamiento (Jiao et al., 2018), de forma semejante se puede utilizar las *CNN* con *Bi-LSTM* para estimaciones de distancia recorrida a partir de una secuencia sucesiva de imágenes como *DistanceNet* (Kreuzig et al., 2019) que emplea este enfoque donde explota el orden de las distancias mediante la aplicación de regresiones.

Además, las *Bi-LSTM* son empleadas para el reconocimiento de acciones detalladas en video, permite modelar las dinámicas de movimiento temporales (Singh et al., 2016)

3.2. Funciones de pérdida

Para lograr la predicción de la posición se utiliza el cálculo de la probabilidad condicional para el problema de odometría visual. El cálculo hace refierencia a la probabilidad de algún evento Y_t, dada la ocurrencia de algún otro evento X_t, descrito como, la probabilidad de Y_t dada X_t.

Siendo $Y_t = (y_1, y_2, \dots, y_t)$ la pose a predecir y $X_t = (x_1,\ x_2, \dots, x_t)$ las secuencias de cuadros monoculares. El cálculo es el siguiente:

$$p(Y_t|X_t) = p(y_1, y_2, \dots, y_n|x_1, x_2, \dots, x_n) = \frac{p(y_1, y_2, \dots, y_n \cap x_1, x_2, \dots, x_n)}{p(x_1, x_2, \dots, x_n)}$$

Es decir, la probabilidad de predecir la pose suponiendo que las secuencias de cuadros monoculares haya ocurrido (Jiao et al., 2018).

Los investigadores proponen el cálculo del valor θ^* óptimo, maximizando $p(Y_t|X_t)$, de este modo se pretende realizar la predicción de la rotación φ_k en ángulos de Euler y la traslación p_k. Donde para cada par (p_k, φ_k) corresponde un (p'_k, φ'_k) (Jiao et al., 2018). La función de pérdida se denota como:

$$\theta^* = \arg_\theta min \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{3} \|p_k - p'_k\|_2^2 + w\|\varphi - \varphi'_k\|_2^2$$

Donde:

- $\|.\|$ se refiere a la norma.
- k se refiere al número de estados en cada posición, como se cuenta con tres ángulos de Euler, k va de 1 a 3.
- N se refiere al número de secuencias de imágenes por lote.
- w se refiere al peso, para equilibrar la posición y traslación.
- θ^* el valor se obtiene al aplicar descenso de gradiente de forma iterativa.

3.3. Algoritmos de odometría visual basados en aprendizaje profundo

Con el auge de las redes neuronales profundas ha generado métodos para resolver los problemas de odometría visual. No se requiere un gran cálculo geométrico como los métodos descritos anteriormente, sino más bien modelos descriptivos y concisos.

La aplicación de modelos basados en redes neuronales profundas provee resultados en la predicción de cambios de dirección y la velocidad de la cámara. Modelos emplean el aprendizaje automático para predecir movimiento y profundidad, a partir de un conjunto de datos de cámaras estéreo, posteriormente se concatenan 5 fotogramas consecutivos para ingresar al modelo de red neuronal como se muestra en la Figura *14*.

El modelo toma las imágenes estéreo, se procesa por separado tanto la imagen izquierda como derecha en la primera capa convolucional. Posteriormente se aplica la multiplicación del anterior resultado. El resultado de la multiplicación es procesado por una segunda capa convolucional para finalmente estimar la posición y el cambio de dirección en una capa completamente conectada.

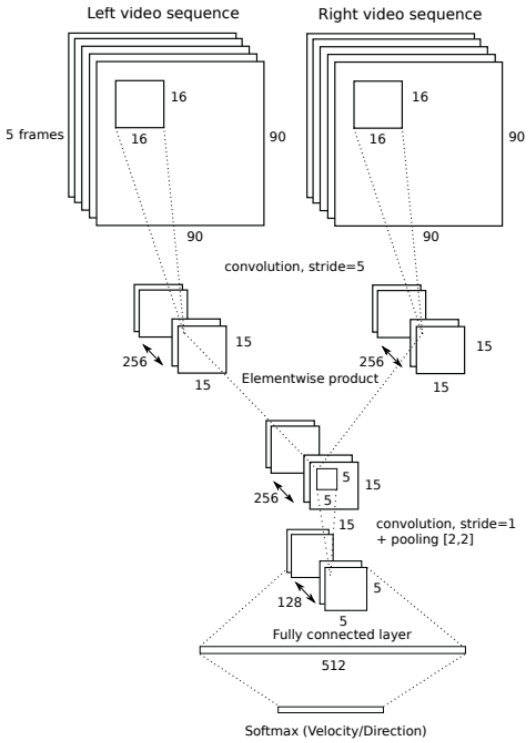

Figura 14. *Arquitectura de CNN* (Konda & Memisevic, 2005)

El modelo se encuentra estructurado para estimar la velocidad y el cambio de dirección, como se muestra en el diagrama de flujo de la Figura *15*. Con la información de la estimación de velocidad y el cambio de la dirección se logra producir la trayectoria de la secuencia de imágenes completa (Konda & Memisevic, 2005).

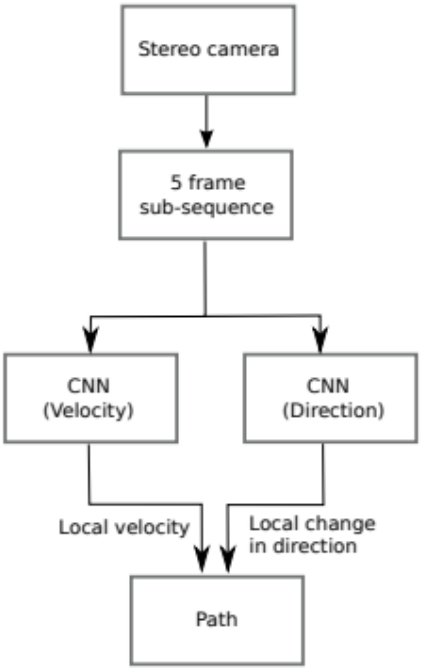

Figura 15. *Diagrama de flujo para la estimación de trayectoria* (Konda & Memisevic, 2005)

Investigadores emplearon redes neuronales convolucionales para aprender representaciones ópticas de pares de imágenes consecutivas, para lo cual extrae la información del flujo óptico denso empelando el cálculo la variación total.

Flujo óptico se refiere al patrón que existe durante el movimiento de objetos dentro de un par de imágenes consecutivas, se representa como un campo vectorial de dos dimensiones donde existe un vector de desplazamiento que es capaz de mostrar el movimiento de puntos desde la primera imagen a la segunda imagen, como se puede observar en la *Figura 16*. Se puede observar un vector de desplazamiento de 5 imágenes consecutivas (OpenCV, 2017).

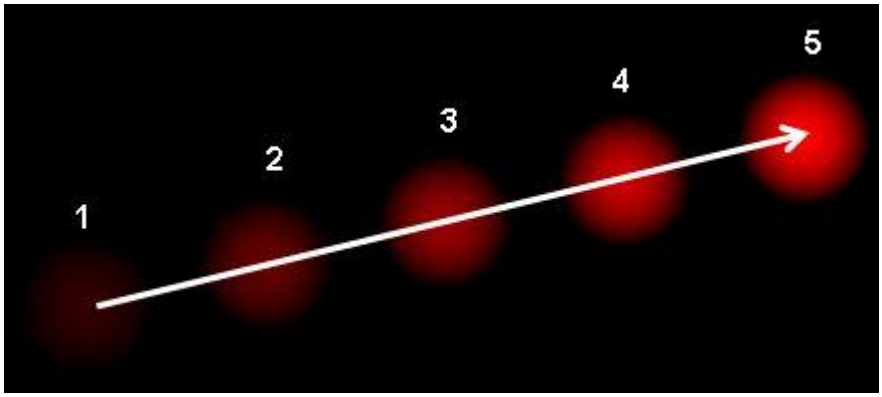

Figura 16. Demostración de flujo óptico (OpenCV, 2017)

En la *Figura 17* se muestra en la imagen superior la extracción de representaciones ópticas, graficadas en línea roja, donde se hace referencia al desplazamiento de un píxel entre imágenes. La imagen inferior se muestra la estimación del flujo óptico siendo los colores más cálidos zonas donde ha existido un mayor desplazamiento entre pixeles.

Figura 17. *Campo de flujo óptico denso* (Costante et al., 2016)

Se aplica *CNN* donde la imagen es dividida en 4 cuadrantes y cada uno representa a un banco de filtros descritos por *CNN1* seguida de una capa *MaxPooling* y *CNN2* seguida de una capa *MaxPooling*. Posteriormente se concatenan para producir características más fuertes como se muestra en la *Figura 18*. El modelo fue capaz de procesar imágenes con desenfoque debido al movimiento entre imágenes y a cambios bruscos de iluminación (Costante et al., 2016).

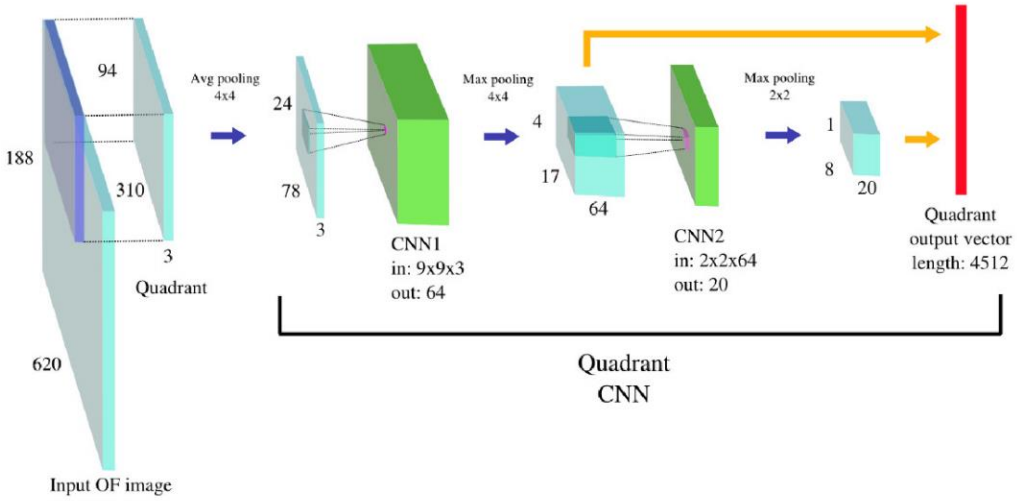

Figura 18. *Arquitectura Quadrant CNN* (Costante et al., 2016)

Otro ejemplo, se emplearon dos fotogramas de video consecutivos para la estimación del flujo óptico, la estimación muestra el movimiento directo hacia adelante como se observa en la *Figura 19*. Donde los colores más cálidos como el rojo describen la existencia de un mayor cambio en los pixeles en línea recta:

Figura 19. *Estimación de flujo óptico* (Muller, 2016)

Investigaciones desarrolladas para odometría visual, emplearon conjuntos de datos *RGB-D* para implementar una red neuronal capaz de estimar la profundidad, transformaciones globales, transformación de píxel, generando una biblioteca de software *GVNN* (visión geométrica con red neuronal) (Handa et al., 2016).

En comparación con los algoritmos tradicionales para el cálculo de estimación de pose a través de imágenes monoculares, los algoritmos de aprendizaje profundo han demostrado que pueden ser eficientes para la extracción de información de movimiento entre secuencias de cuadros, reemplazando la complejidad de las fórmulas matemáticas y el esfuerzo computacional para lograr extraer características para buscar confidencias en las imágenes siguientes (Jiao et al., 2018).

Recientes investigaciones han demostrado que, al ignorar las relaciones presentes entre las secuencias de imágenes, conlleva a la pérdida de precisión. Por consiguiente existen problemas al emplear secuencias de imágenes diferentes al conjunto de entrenamiento, ya que el modelo no es capaz de generalizar las secuencias de imágenes (Steffen et al., 2017).

Modelos de redes neuronales han propuesto la solución de odometría visual a partir del flujo óptico, creando interpolación y extrapolación entre los cuadros de imágenes, con la implementación de redes neuronales convolucionales para predecir campos de flujo a partir de una secuencia de imágenes de entrada.

Para ello se ha empleado redes neuronales convolucionales, para la predicción densa de pixeles el modelo contiene una parte de expansión para refinar inteligentemente el flujo para alta resolución.

La arquitectura muestra que la entrada a la red se concatenan las imágenes una seguida de la otra formando así seis canales, que corresponde al par de imágenes en formato *RGB*, lo que permite que la red sea capaz de extraer la información de movimiento por sí misma, los detalles de la *CNN* se detallan en la Tabla *1*.

Tabla 1. Estructura de la CNN

Capa	Tamaño de Kernel	Padding	Stride	Número de canales
Conv1	7x7	3	2	64
Conv2	5x5	2	2	128
Conv3	5x5	2	2	256
Conv3_1	3x3	1	1	256
Conv4	3x3	1	2	512
Conv4_1	3x3	1	1	512
Conv5	3x3	1	2	512
Conv5_1	3x3	1	1	512
Conv6	3x3	1	2	1024

(Jiao et al., 2018)

La red se refiere a contraer y expandir los partes que se entrenan como un todo utilizando retro propagación, la arquitectura se muestra a continuación en la Figura *20*:

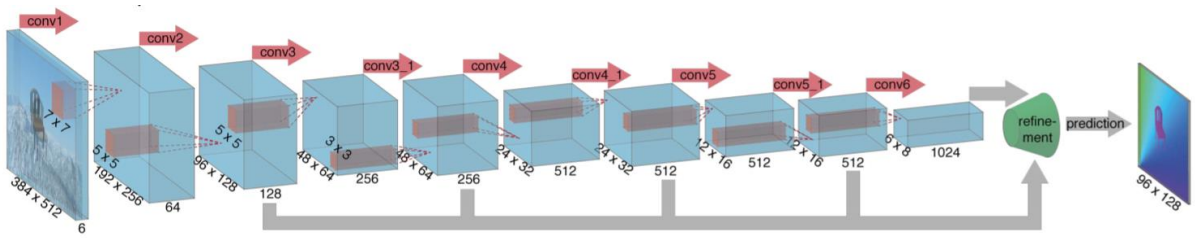

Figura 20. *FlowNetSimple* (Dosovitskiy et al., 2015)

Para la obtención de valores verdaderos se ha identificado como una tarea difícil ya que las verdaderas correspondencias del mundo real entre pixeles no se han podido determinar fácilmente ya que se observa un pequeño número de conjuntos de datos disponibles para ser empleado para esta tarea Tabla *2*.

Tabla 2. Conjunto de datos de Flujo óptico

Conjunto de datos	Pares de imágenes con su correspondiente valor de verdad	Densidad de verdad por imagen
KITTI	194	~ 50%
Sintel	1041	100%
Flying Chairs	22872	100%

(Dosovitskiy et al., 2015)

A continuación, en la Figura *21*, se muestra una representación de flujo óptico a partir de la utilización del modelo *FlowNet,* se utiliza el conjunto de datos *Flying Chairs*. El conjunto de datos utilizado es sintético, por lo cual se simulan imágenes estereotípicas para capturar representaciones de sillas ubicadas en el aire. No se cuenta con desenfoque de movimiento ya que no son imágenes reales, pero a su vez cuentan con la suficiente información para poder entrenar y evaluar el modelo.

Figura 21. *Flying Chairs* (Dosovitskiy et al., 2015)

Enfoques para el cálculo de la odometría visual a través de cámaras monoculares, es la utilización de redes neuronales convolucionales *CNN* en combinación con redes recurrentes *LSTM* o con redes bidireccionales con unidades de memoria a largo plazo *Bi-LSTM*, que genera una pose de escala absoluta con 6 grados de libertad a partir de secuencia de imágenes monoculares continuas (Jiao et al., 2018).

3.4. Conjuntos de datos

En general se parte de conjuntos de datos, que se encuentran a disposición de otros investigadores para sus investigaciones, así se pueden dividir en conjunto de datos de entrenamiento y validación, para lo cual se presentan las siguientes alternativas utilizadas por diferentes autores (Ilg et al., 2017; R. Li et al., 2018) como se describen en los siguientes párrafos.

La página web *KITTI* (Geiger et al., 2012) provee conjuntos de datos para visión estéreo, flujo óptico, odometría visual, detección de objetos, etc. El conjunto de datos KITTI Visual Odometry provee 22 secuencias estéreo, datos adicionales del conjunto de datos es que la tasa de almacenamiento de las imágenes es a 10 fps, la velocidad con la que iba el automóvil para recorrer el escenario es de 90 km/h, como se muestra en la Figura *22*.

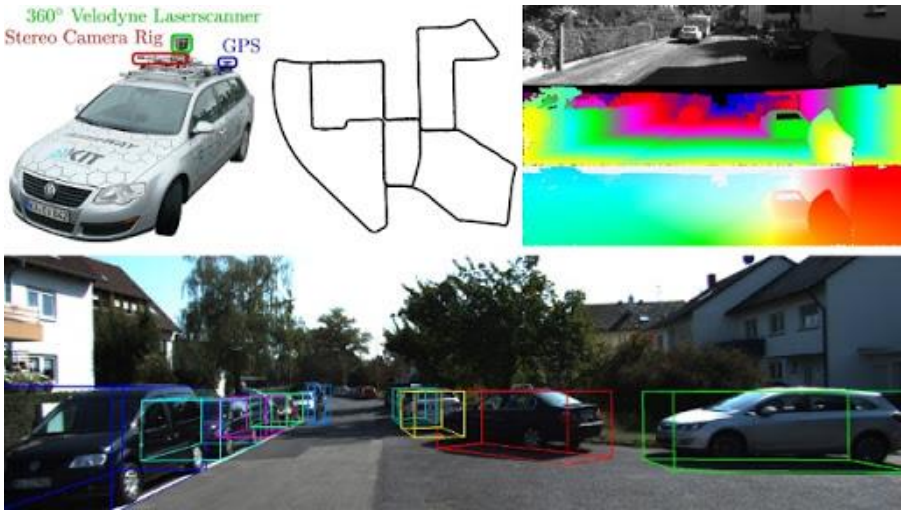

Figura 22. *Suite de referencia KITTI* (Geiger et al., 2012)

El Departamento de Ingeniería Mecánica y de Procesos de la Escuela Politécnica Federal de Zúrich (Engineering, 2020), provee conjuntos de datos *ASL*, donde su principal objetivo es el de facilitar datos para la comunidad robótica para desarrollo, comparación de resultados y evaluación. Las imágenes son capturadas por un dron donde las imágenes contienen una severa inquietud por el movimiento de la cámara Figura *23*.

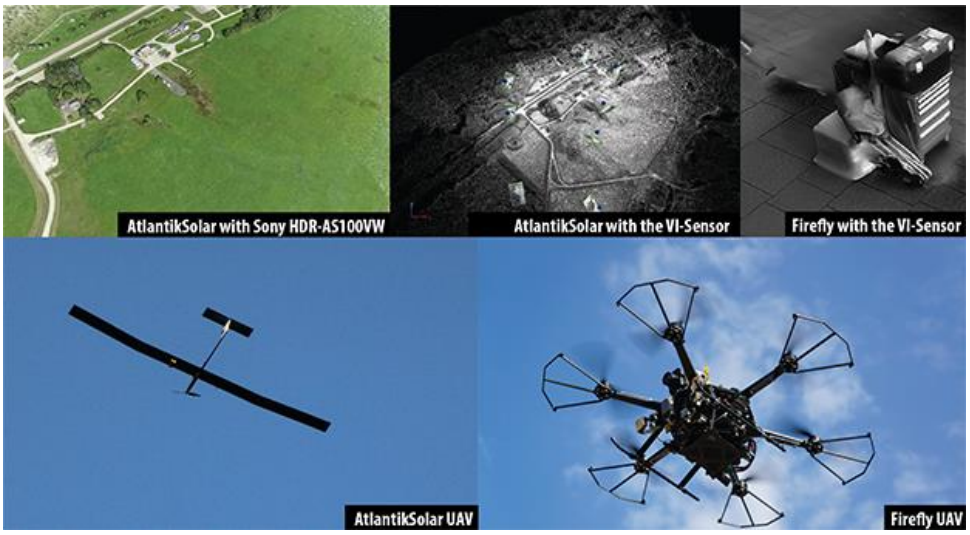

Figura 23. *Conjunto de datos ASL* (Engineering, 2011)

Otro conjunto de datos disponible gratuitamente es del departamento de *Computer Science* de la Universidad de Oxford (Chen et al., 2018), donde se pretende explotar los datos inerciales de navegación peatonal para estimación de movimiento y ubicación. El conjunto de datos presenta mediciones en base a IMU (Unidad de medición Inercial) de diferentes escalas de dispositivos móviles con diferentes recursos, proporciona directrices para el uso de datos en bruto de IMU Figura *24*.

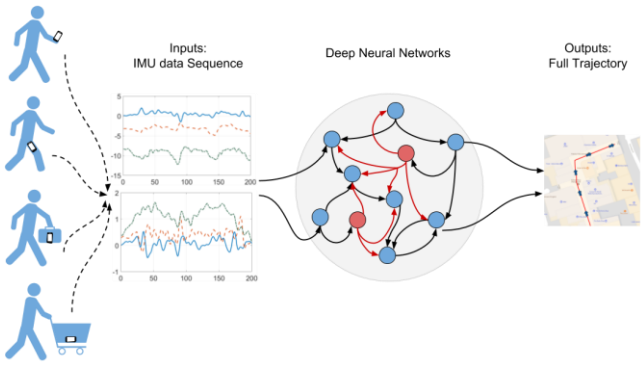

Figura 24. *Conjunto de datos de odometría inercial de Oxford* (Chen et al., 2018)

3.5. Librerías para aprendizaje profundo

Para el aprendizaje automático se dispone de una amplia gama de arquitecturas para cumplir con diferentes tareas. En la actualidad existe diferentes librerías desarrolladas en diferentes lenguajes de programación como *C++, Python, Lua*, entre otras (Erickson et al., 2017). Cada una fue diseñada para solventar los problemas presentes en las arquitecturas de las redes neuronales.

3.5.1. TensorFlow

Es una librería desarrollada por *Google*, admite múltiples arquitecturas de ejecución como *CPU* y *GPU*, debido a su amplia utilización y popularización en la investigación emplea lenguaje Python. Adicional presenta una red para monitorear el rendimiento del entrenamiento de redes neuronales como es *Tensorboard* (Erickson et al., 2017). Es de libre acceso y cuenta con varios colaboradores para solventar problemas presentes en la ejecución de código. Adicional se encuentran varios modelos pre entrenados para su uso e implementación en diferentes proyectos de investigación.

3.5.2. Pytorch

Es un framework de *Python* para el motor *Torch*, cuenta con soporte tanto para *CPU* como *GPU*, es más amigable en su utilización ya que emplea integración profunda para mayor flexibilidad en la implementación de arquitecturas de redes neuronales (Erickson et al., 2017). Cuenta con soporte para compilación para núcleos *CUDA* en su última versión liberada. De manera similar a *TensorFlow* cuenta con modelos pre entrenados para su utilización.

3.5.3. Keras

Es una librería de alto nivel, es empleado por *TensorFlow* en su *core*, es fácil de implementar arquitecturas ya que con cada línea de código se puede crear una capa de la red. Provee varios algoritmos de vanguardia para su alto rendimiento, cuenta con una amplia documentación disponible en su página oficial (Erickson et al., 2017). La librería cuenta con el soporte y colaboración por parte de *Google*.

4. Objetivos y metodología de trabajo

Una vez realizado el análisis en los anteriores apartados donde se describe los diferentes métodos de odometría visual, que van desde los métodos analíticos hasta la utilización del aprendizaje profundo, en el presente apartado se describe el objetivo general y los objetivos específicos para el desarrollo del presente trabajo.

4.1. Objetivo general

De acuerdo con el análisis presentado en los anteriores apartados se ha observado que la utilización del aprendizaje profundo ha logrado ser una alternativa para el problema de odometría visual. Por consiguiente, el presente trabajo tiene como objetivo: **Realizar un estudio comparativo de las diferentes arquitecturas de aprendizaje profundo para la estimación de la odometría visual**.

4.2. Objetivos específicos

1. Identificar y describir las arquitecturas de aprendizaje profundo disponibles para odometría visual.
2. Identificar los conjuntos de datos disponibles para el entrenamiento y prueba de la odometría visual.
3. Definir las métricas de éxito para la odometría visual.
4. Implementar las arquitecturas seleccionadas para odometría visual.
5. Calcular y comparar el tiempo de entrenamiento entre las diferentes arquitecturas para un mismo conjunto de datos.
6. Evaluar las diferentes arquitecturas con diferentes conjuntos de datos.
7. Comparar el resultado de estimación de la odometría visual de las diferentes arquitecturas.

4.3. Metodología del trabajo

Para alcanzar el objetivo presente en este trabajo se presenta una metodología por fases:

1. Se realizará un estudio profundo de las arquitecturas de odometría visual. En esta fase se desarrollará una búsqueda bibliográfica enfocada a modelos de aprendizaje profundo que pueden contribuir en la estimación de posición a partir de una secuencia de imágenes. Adicional, se identificarán los diferentes parámetros como sus funciones de perdida, funciones de activación y optimizadores empleados en cada arquitectura.

2. Se analizarán los conjuntos de datos existentes para para odometría visual con el fin de escoger el óptimo para este estudio.

3. En esta fase se enfocará en los siguientes puntos:

3.1. Desarrollo y en la implementación de las arquitecturas en código Python.

3.2. Optimización de carga de los conjuntos de datos seleccionados.

3.3. Optimización de procesamiento de datos para el computador empleado para el presente estudio.

4. En esta última fase se realizarán los siguientes puntos:

4.1. Ejecución de las arquitecturas de odometría visual con el conjunto de datos seleccionado.

4.2. Se desarrollará la comparativa de los diferentes modelos en tiempo de ejecución, así como precisión en la estimación de la posición.

4.3. Una vez desarrollada la comparativa se determinará el modelo óptimo para la estimación de posición a partir de secuencia de imágenes a color.

5. Planteamiento de la comparativa

En el presente apartado se presenta modelos de redes neuronales capaces de solventar el problema de la odometría visual, se ha analizado el conjunto entrenamiento, evaluación y otro para evaluación. También se realiza un análisis los criterios de éxito para el desarrollo de la comparativa.

5.1. Arquitecturas de aprendizaje profundo

A continuación, se describen arquitecturas para solventar el problema de odometría visual, así como el conjunto de datos utilizado, dimensiones de las imágenes, número de parámetros de las capas, así como las dimensiones del resultado de cada capa de la red neuronal.

5.1.1. Arquitectura DeepVO

La aplicación *DeepVO* (Sen et al., 2017) emplea redes neuronales convolucionales en combinación con redes neuronales recurrentes *RCNN*, donde no solo el modelo es capaz de aprender representaciones de características efectivas para el problema de odometría visual sino que también es capaz de modelar las dinámicas secuenciales con la utilización de las redes neuronales recurrentes. Los autores emplearon el conjunto de datos *KITTI* para odometría visual por ser un conjunto de datos extenso con 10 conjuntos de datos para entrenamiento y 11 conjuntos de datos para prueba.

La arquitectura como se muestra en la Figura *25*, presenta un redimensionamiento de las imágenes a $(1280 \times 384 \times 3)$, se aplica el par de imágenes a la entrada del modelo, la capa *CNN* se encarga de producir características de odometría visual, la capa convolucional consta de 9 capas convolucionales donde se toma imágenes en *RGB* como entrada para la red, posteriormente se aplana la salida matricial a un vector que será introducido en las capas *LSTM* para producir un aprendizaje secuencial. La arquitectura emplea el modelo pre entrenado *FlowNet* para acelerar el tiempo de entrenamiento de la red neuronal:

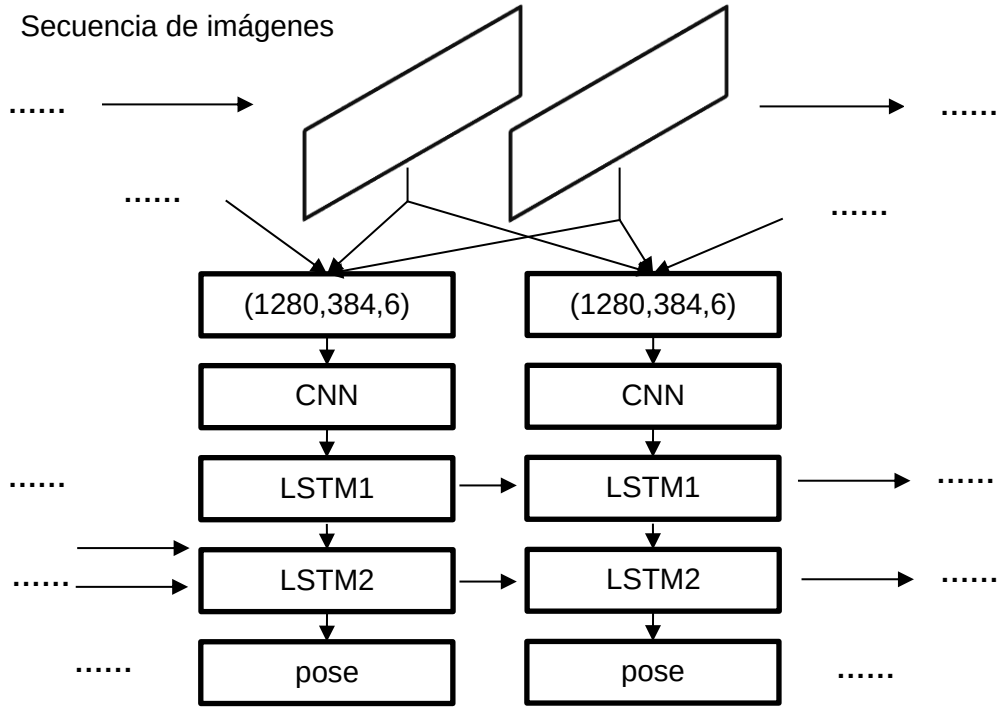

Figura 25. *Estructura del modelo DeepVO*

Se pretende que el modelo sea capaz de extraer características simultaneas y secuenciales a través de la combinación de las capas *CNN* y *RNN* ya que la estimación de la posición actual puede ser beneficiada de información representada en anteriores pares de imágenes, por lo cual se emplea dos capas *LSTM* de 1000 estados ocultos cada una. La capa *LSTM* genera estimaciones en cada paso basado en lo obtenido en las capas *CNN* anteriores la estructura del modelo.

Se presenta en la Tabla *3*, donde se observa el resultado de las dimensiones del procesamiento de cada capa. Como se puede observar, al contar con imágenes de entrada de dimensiones de $1280 \times 384 \times 3$ genera un resultado de $20 \times 6 \times 1024$, El modelo pre entrenado *FlowNet*. Este resultado debe ser convertido en un vector para que pueda ser procesada en las capas *LSTM,* la conversión genera un vector de 1×122880.

Al contar con una gran cantidad de parámetros, se requiere un gran procesamiento computacional, por lo que utilizaron la tarjeta gráfica *NVIDIA Tesla K40* de 12 *GB* de *RAM* para el entrenamiento de la red neuronal por 200 épocas. Para el entrenamiento se empleó las secuencias 00, 02, 08 y 09 porque son las secuencias relativamente extensas de todo el conjunto. Para las pruebas del modelo se emplearon las secuencias 03, 04, 05, 06, 07 y 10.

Tabla 3 Estructura de DeepVO

Capas		Dimensiones
Imágenes apiladas		1280x384x3
		1280x384x3
Capa convolucional	Conv 1	640x192x64
	Conv 2	320x96x128
	Conv 3	160x48x256
	Conv 3_1	160x48x256
	Conv 4	80x24x512
	Conv 4_1	80x24x512
	Conv 5	40x12x512
	Conv 5_1	40x12x512
	Conv 6	20x6x1024
Capa LSTM	LSTM1	1000
	LSTM2	1000
Capa completamente conectada		6
Resultado - Pose		6

La aplicación de *RCNN* para estimar la odometría visual se puede verificar que genera resultados precisos sin la utilización de los parámetros intrínsecos de la cámara. En la Figura *26* se muestra una estimación de la odometría visual al utilizar la arquitectura *DeepVO*, donde en línea entrecortada el recorrido real de la secuencia de datos y con línea azul se muestra la estimación del recorrido. Para esta prueba se empleó la secuencia 00 del conjunto de datos *KITTI,* como se puede observar. Lo que pretende los autores es que el presente modelo no sea reemplazo del enfoque clásico de la estimación de la odometría visual sino más bien como un complemento y así aportar a la comunidad científica con nuevas alternativas en implementación.

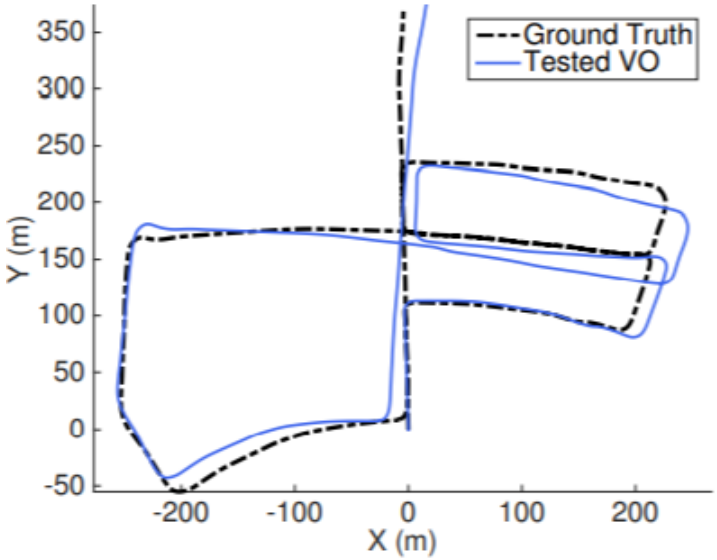

Figura 26. *Estimación DeepVO conjunto de datos KITTI secuencia 00* (Sen et al., 2017)

5.1.2. Arquitectura MagicVO

La aplicación *MagicVO* (Jiao et al., 2018) los autores partieron del conjunto de datos *KITTI* y *ETH-asl cla*. El modelo presenta un sistema que no requiere parámetros intrínsecos de la cámara, el cálculo de la posición es absoluto, la combinación de redes convolucionales con redes bidireccionales con unidades de memoria a largo plazo que permite a más de la propiedad de aprender características del par de imágenes aprende la relación entre secuencia de imágenes antes y después del cuadro actual, de este modo genera un modelo con alta precisión.

La arquitectura muestra un preprocesamiento de la secuencia de imágenes, realizando un redimensionamiento de escala a $(640 \times 192 \times 3)$, se superpone el tercer canal dos cuadros adyacentes al actual, se realiza normalización de los valores de los pixeles, como primera parte se presenta una *CNN* seguida de la capa *Bi-LSTM* y finalmente una capa completamente conectada como se muestra en la Figura *27*. La respuesta final del modelo es la posición con 6 grados de libertad (Jiao et al., 2018). Los modelos pre entrenados no han logrado aprender relaciones geométricas entre imágenes por lo que los autores han empleado el modelo pre entrenado FlowNet para acelerar la convergencia (Dosovitskiy et al., 2015).

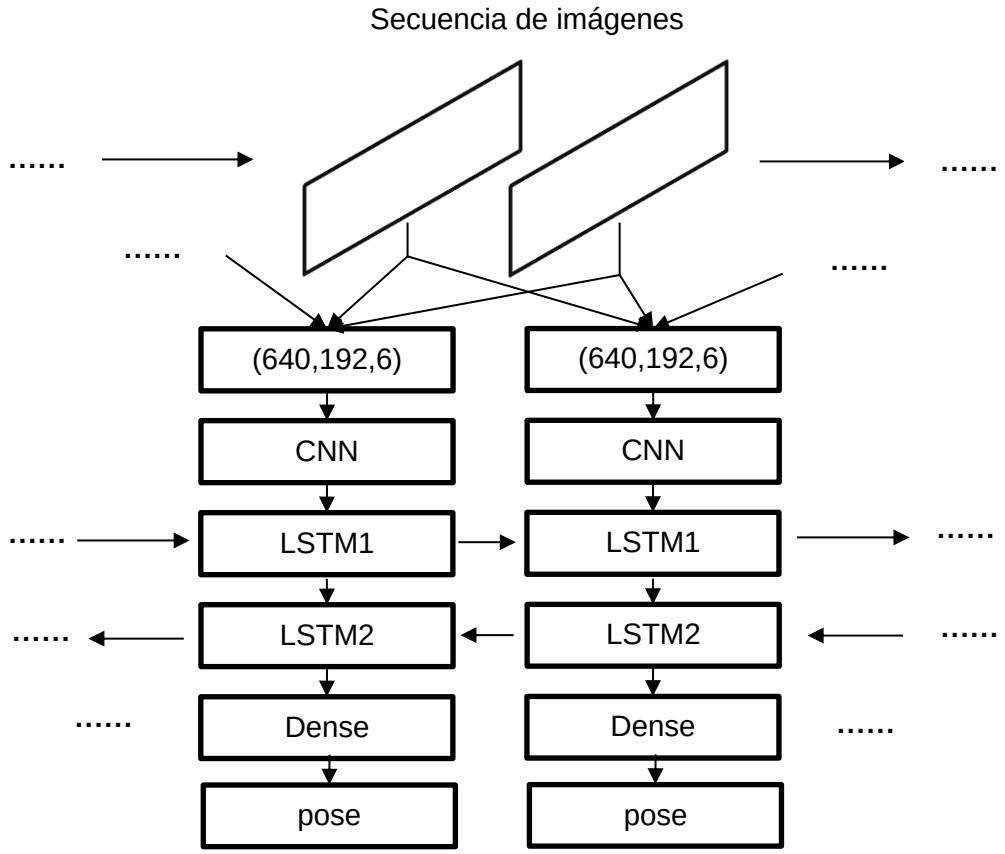

Figura 27. *Estructura del modelo MagicVO*

Se presenta capas ocultas de *RNN* con *Bi-LSTM*, así la correlación existente entre cuadros de imagen puede ser utilizada para estimar la estimación de poses. La entrada al modelo presenta imágenes de dimensiones de $640 \times 192 \times 3$, que al procesar por las capas convolucionales genera un resultado de $10 \times 3 \times 1024$, que debe ser convertido en un vector de 1×30720. Para ser introducido en las capas *LSTM* con 1000 nodos hacia adelante y 1000 hacia atrás. Posteriormente se presenta dos capas completamente conectadas para integrar la información procedente de las capas *Bi-LTSM* para estimar con precisión la pose final con 6 grados de libertad como se muestra en Tabla *4*. Siendo el resultado los ángulos de Euler y los puntos de traslación (Jiao et al., 2018).

Para el entrenamiento de la red neuronal emplearon la tarjeta gráfica *NVIDIA TITAN X con 64 GB* de RAM, se ejecutó con las secuencias 00, 02, 04, 06, 08 y 10 y para verificación se emplearon las secuencias 03, 05, 07, 09 del conjunto de datos *KITTI*.

Tabla 4. Estructura de MagicVO

Capas		Dimensiones
Imágenes apiladas		640x192x3
		640x192x3
Capa convolucional	Conv 1	320x96x64
	Conv 2	160x48x128
	Conv 3	80x24x256
	Conv 3_1	80x24x256
	Conv 4	40x12x512
	Conv 4_1	40x12x512
	Conv 5	20x6x512
	Conv 5_1	20x6x512
	Conv 6	10x3x1024
Capa Bi-LSTM	LSTM1	1000
	LSTM2	1000
Capa completamente conectada		256
		6
Resultado - Pose		6

El modelo resultante se comparó con *ORB_SLAM2* (Mur-Artal et al., 2015) y VINS-Mono (Qin et al., 2018). Los resultados demostraron que el modelo *MagicVO* tiene un buen desempeño en estimar la traslación, con errores por debajo al 6%, en cambio los errores de en la rotación varían ya que se encuentran a la par con el modelo *ORB_SLAM2.* La Figura *28* muestra con un cuadrado de color negro el inicio de la secuencia 00, la secuencia de imágenes pertenece al conjunto de datos *KITTI*, la línea entrecortada de color negro pertenece a las posiciones reales, mientras que la línea de color rojo representa a las estimaciones de la arquitectura *MagicVO* y la línea punteada representa a la estimación al utilizar el modelo *ORB_SLAM2*.

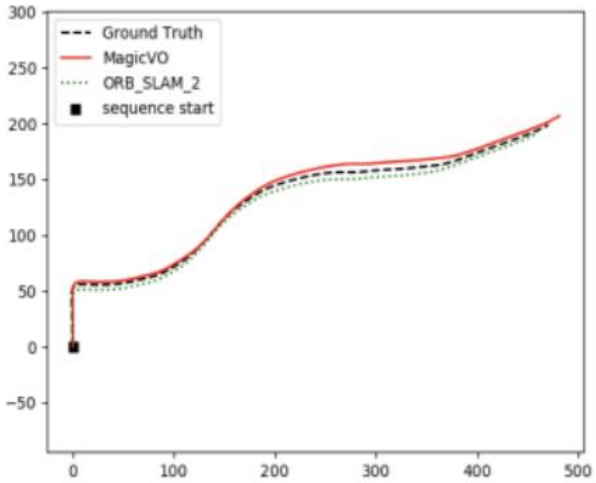

Figura 28. *Estimación MagicVO conjunto de datos KITTI secuencia 03* (Jiao et al., 2018)

5.1.3. Arquitectura PoseConvGRU

PoseConvGRU (Zhai et al., 2019) presenta una novedosa red neuronal convolucional recurrente con la utilización de arquitecturas *GRU*, el modelo es capaz de codificar de un par de imágenes secuenciales funciones de movimiento a corto plazo, así el modelo propaga información en el tiempo. El modelo cuenta con tres partes como se muestra Tabla *5*, se emplea la arquitectura de *FlowNet* para la extracción de información de movimiento y estimación de pose seguida de una capa *MaxPooling*, posteriormente se cuenta con un módulo *ConvGRU*.

Tabla 5. Estructura de PoseConvGRU

Capas		**Dimensiones**
Imágenes apiladas		1280x384x3
		1280x384x3
Capa convolucional	Conv 1	640x192x64
	Conv 2	320x96x128
	Conv 3	160x48x256
	Conv 3_1	160x48x256
	Conv 4	80x24x512
	Conv 4_1	80x24x512
	Conv 5	40x12x512
	Conv 5_1	40x12x512
	Conv 6	20x6x1024
	MaxPooling	10x3x1024
Capa GRU	GRU	3
Capa completamente conectada		4096
		1024
		128
		6
Resultado - Pose		6

La entrada de la *CNN* son las imágenes *RGB* de un par de imágenes con redimensionadas a $(1280\text{x}384\text{x}3)$, estos pares de imágenes son sometidos a 10 capas convolucionales siendo la última capa de *MaxPooling*. El uso de una *MaxPooling* en la última capa *CNN* reduce de manera significativa los parámetros de salida de *FlowNet* aumentando la complejidad de entrenamiento de la red neuronal.

El módulo *ConvGRU* es capaz de almacenar a largo plazo una representación visual de la secuencia de imágenes, ya que permite que la red neuronal aprenda las relaciones intrínsecas de poses, se emplea la arquitectura *GRU* para que la red pueda recordar estados de sucesos

históricos como las relaciones geométricas provenientes de las secuencias anteriores del par de imágenes, se emplea la arquitectura *GRU* para reducir la cantidad de parámetros ya que presenta un rendimiento similar a la arquitectura *LSTM*. Posteriormente se aplica capas densas para la estimación de la posición relativa de las imágenes con *6* grados de libertad Figura *29*.

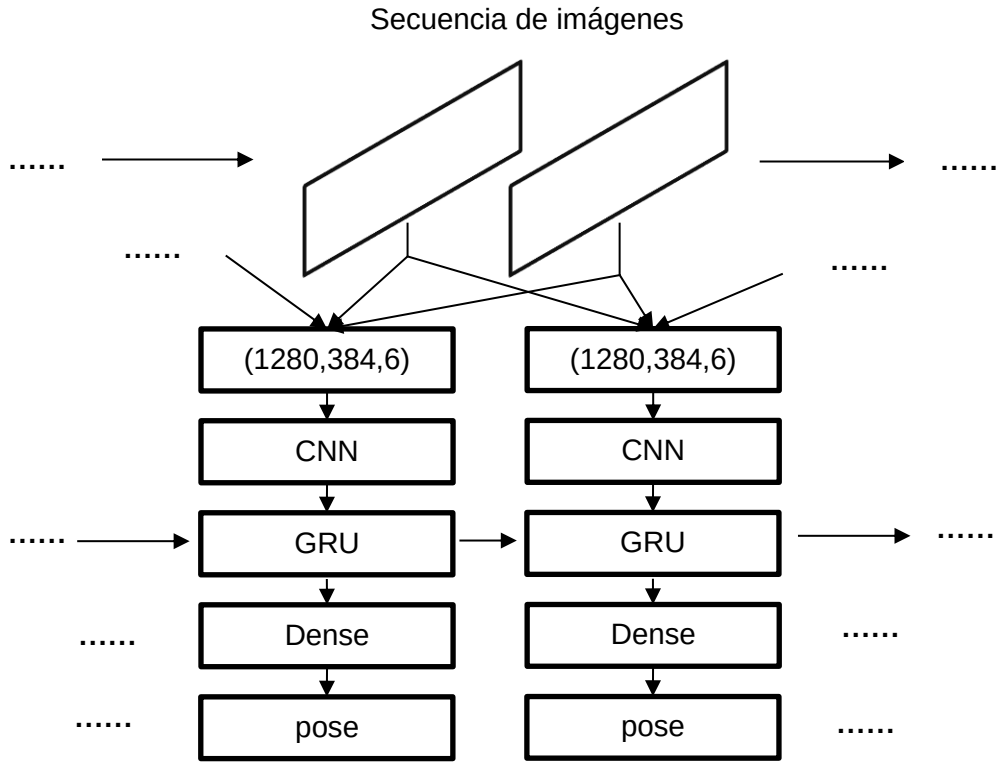

Figura 29. *Estructura del modelo PoseConvGRU*

Los autores presentan un modelo capaz de ser ejecutado en tiempo real porque se presentan una cantidad reducida de cálculos presentes en la arquitectura, las capas *ConvGRU* pretende una regresión relativa para un par de imágenes consecutivas. La Figura *30* muestra la estimación de posición y orientación para diferentes épocas de entrenamiento del modelo, donde para la época 15 se muestra el recorrido de color verde claro, para la época 55 de color verde oscuro, para la época 75 de color verde y para la época 100 de color rojo. Para ello se empleó la secuencia 00 del conjunto de datos *KITTI*. Como se observa a medida que aumenta las épocas en el entrenamiento del modelo, la estimación se apega a los valores reales del recorrido.

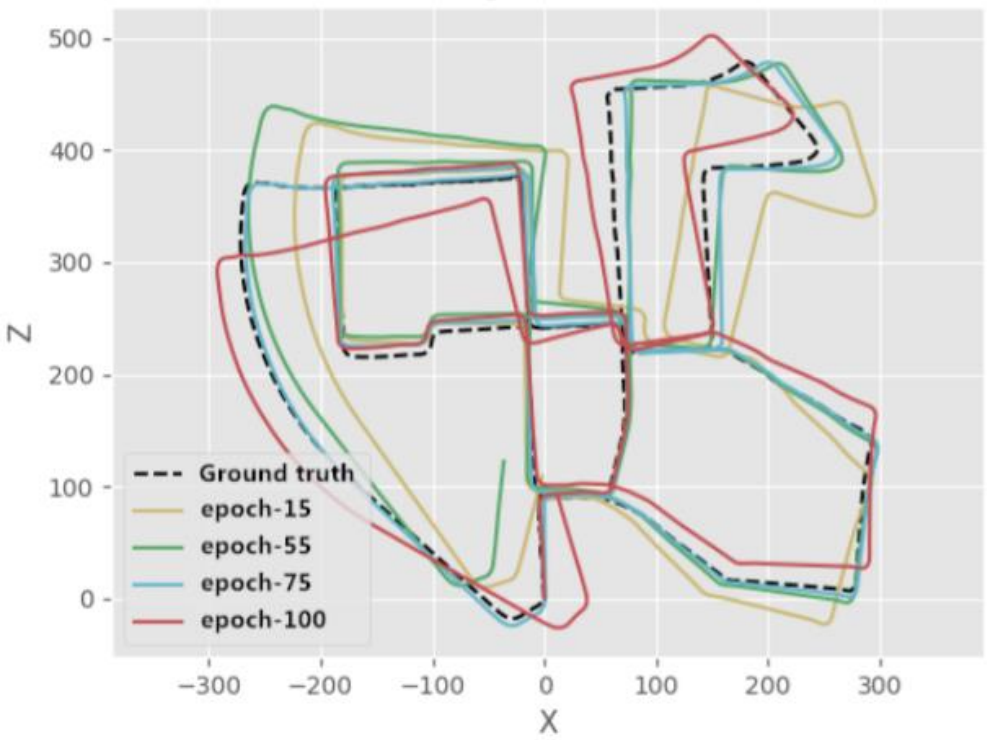

Figura 30. *Estimación PoseConvGRU conjunto de datos KITTI secuencia 00* (Zhai et al., 2019)

5.2. Conjuntos de datos

A continuación, se describe una comparativa de conjuntos de datos para desarrollar el presente trabajo, se toman como ejemplo los conjuntos de datos descritos en el capítulo 3.

5.2.1. KITTI Odometry

De acuerdo al análisis presentado en (Watson, 2020), el conjunto de datos presenta imágenes de cuatro cámaras dos en color y dos en escala de grises. Las cámaras fueron ubicadas en la parte superior de un vehículo y se realizaron recorridos a través de ambientes urbanos. Las cámaras se encontraban separadas por 54 cm hacia la parte frontal del vehículo, también el automóvil cuenta con algunos sensores *GPS*, escáner de detección y rango de luz *LIDAR* y receptores de unidad de medición inercial *IMU.* La configuración del vehículo se muestra en la Figura *31*. El conjunto de datos para cada secuencia proporciona la matriz de calibración de la cámara.

El conjunto de datos proporciona 22 secuencias diferentes de las cuales las 11 primeras se cuenta con las posiciones reales asociadas al vehículo. Las otras 11 secuencias solo

presentan las secuencias de imágenes que es utilizada para validación de los modelos de odometría visual.

Figura 31. vehículo de adquisición de datos (Watson, 2020)

Las diferentes secuencias presentan una cantidad notable en la cantidad de fotogramas adquiridos por las cámaras, en la Tabla *6* se muestra la cantidad de imágenes por secuencia. Siendo las secuencias 00, 02 y 08 las secuencias con mayor número de imágenes.

Tabla 6. Numero de imágenes por secuencia

Secuencia	Número de imágenes
00	4541
01	1101
02	4661
03	801
04	271
05	2761
06	1101
07	1101
08	4071
09	1591
10	1201

Modelos de imagen a color de la secuencia 00 se muestra en la Figura *32*. Durante las secuencias se observa que el vehículo se dirige en línea recta hacia adelante, además de realizar giros en algunas intersecciones.

Figura 32. *Secuencia 00 conjunto de datos KITTI*

Cada imagen se encuentra rectificada y sin distorsión de adquisición, los valores de la imagen se encuentran entre 0 y 255, además presentan la peculiaridad que las imágenes son una parte de la imagen original, por lo que las imágenes tienen un tamaño consistente durante el recorrido de la secuencia.

Las posiciones reales se calcularon en base a los sistemas de medición GPS/IMU, de modo que el conjunto de datos provee estos valores en un archivo por secuencia. Cada fila representa a las posiciones de una imagen, se cuenta con 12 valores por fila que corresponde

a la representación de la matriz espacial euclidiana *SE(3).* Las posiciones son absolutas referentes a la posición inicial del recorrido de la secuencia.

5.2.2. Oxford RobotCar

Es un conjunto de datos que durante un año (mayo de 2014 a diciembre de 2015) realizar un recorrido por el centro de Oxford, se utilizó la plataforma *Oxford RobotCar* de la marca *Nissan LEAF,* consta con un recorrido de más de 1000 km recogiendo alrededor de 20 millones de imágenes de 6 cámaras monoculares sobre el vehículo. Además, se instaló varios sensores *LIDAR*, *GPS* y *INS* para el cálculo de las posiciones de verdad, siendo la distribución de los sensores como se muestra en la Figura *33* (Maddern et al., 2017).

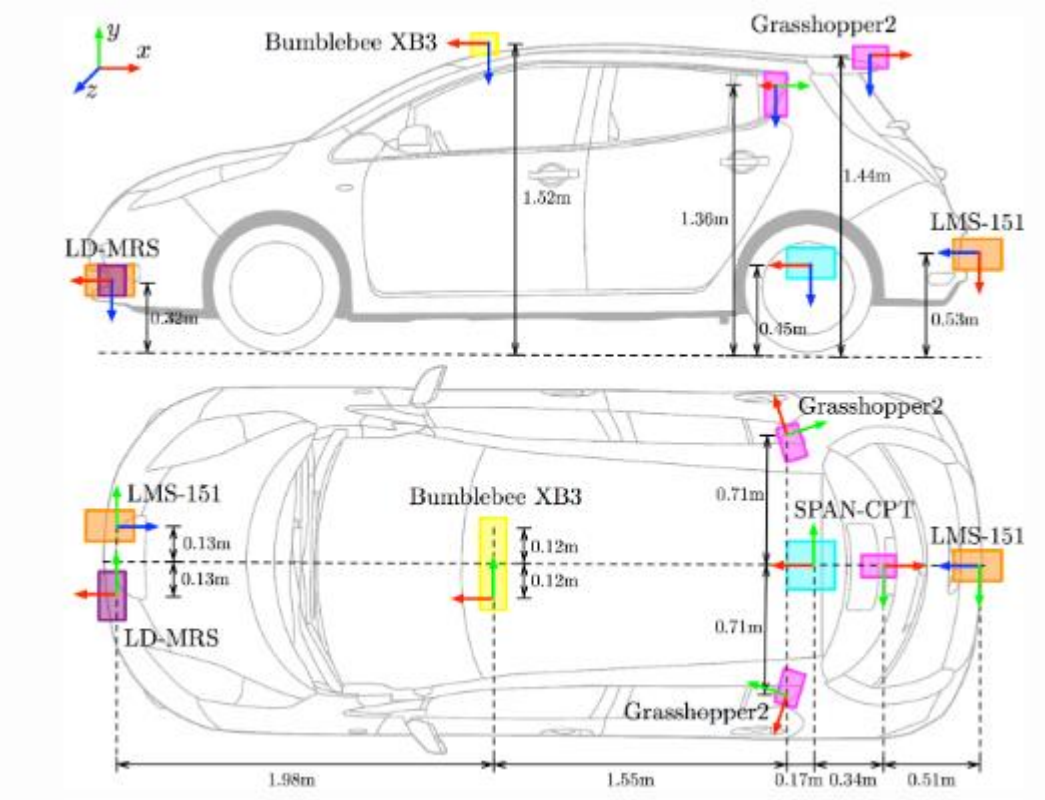

Figura 33. *Ubicación de los sensores* (Maddern et al., 2017)

Los datos fueron obtenidos en una variedad de condiciones climáticas, siendo en la noche, lluvia intensa, nieve y luz solar directa como se observa en la Figura *34*. Permitiendo la investigación de localización y mapeo a largo plazo del vehículo autónomo en entornos del mundo real, como durante el periodo existió cambios en las rutas donde se movilizo el vehículo (Maddern et al., 2015).

Figura 34. *Secuencias de imágenes* (Maddern et al., 2017)

Las secuencias de imágenes no se encuentran pre procesadas, es decir se encuentran con distorsión y no estar calibradas. Los datos provenientes del *GPS* y los sensores inerciales *SPAN-CPT* se proveen en el conjunto de datos en formato ASCII, separados en dos archivos con valores del *GPS* de la latitud $[deg]$, longitud $[deg]$, altitud $[m]$ e incertidumbre $[m]$ que fueron tomados a una frecuencia de 5 Hz y de la unión de los valores provenientes del *GPS+Inercial* que representan a los 3 valores de posición $[m]$, velocidad $[m/s]$ y rotación $[rad]$.

5.2.3. OxIOD

El conjunto de datos presenta secuencias de imágenes con valores de verdad obtenidos por una *IMU*, para odometría inercial. Se cuenta con 158 secuencias que recorren alrededor de 42 km de distancia total. El conjunto presenta secuencias de movimiento de una persona al

realizar una caminata, correr, caminata lenta, se ubicaron los sensores en la mano, bolsillo o en un vehículo (Chen et al., 2018).

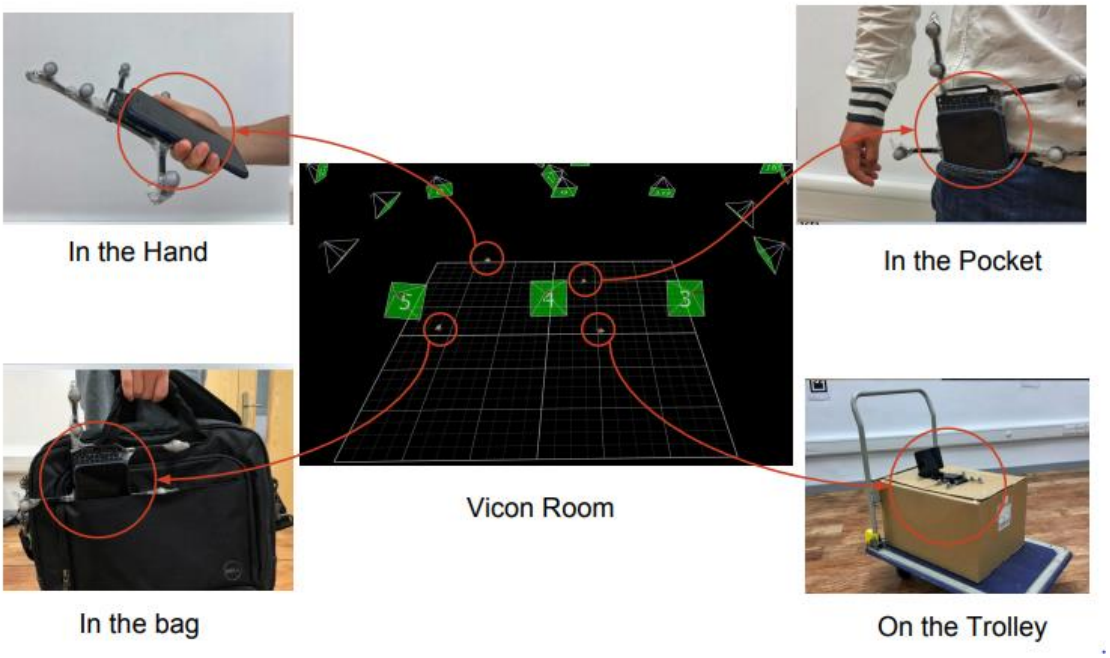

Figura 35. Esquema de estimación de movimiento (Chen et al., 2018)

5.2.4. Selección de conjunto de datos

A continuación, se presenta una comparativa de los conjuntos de datos representados en la Tabla 7, donde:

1. Para el conjunto de datos *KITTI* presenta el sensor *OXTS RT300v3* para obtener las posiciones y orientaciones (OXTS, 2013). Este sensor presenta:

   1.1. $0.03°$ de precisión en pitch y roll

   1.2. 1 *cm* de precisión en la posición

   1.3. 250 Hz velocidad máxima de adquisición

2. Para el conjunto de datos *Oxford RobotCar* presenta el sensor *NoovAte SPAN* para obtener las posiciones y orientaciones (NovAtel, 2017). La descripción se muestra a continuación:

   2.1. $< 10°$ de precisión en pitch y roll

   2.2. < 10 m de precisión en la posición

2.3. 5 Hz velocidad máxima de adquisición

3. Para el conjunto de datos *OxIOD* utiliza el sensor *InvenSense 20600* para capturar las posiciones y orientaciones (InvenSense, 2016). La descripción se muestra a continuación:

3.1. 5° de precisión en pitch

3.2. 100 Hz velocidad máxima de adquisición

Tabla 7. Comparativa de conjuntos de datos

Conjunto de datos	Año	Entorno	Accesorio	Tipo de IMU	Frecuencia de adquisición	Posición verdadera	Tamaño de recorrido
KITTI Odometry	2013	Exteriores	Carro	OXTS RT300v3	250 Hz	GPS/IMU	22 secu, 39.2 km
Oxford RobotCar	2016	Exteriores	Carro	NovAtel SPAN	5 Hz	GPS/IMU	1010.46 km
OxIOD	2018	Interiores	Humano	InvenSense ICM-20600	100 Hz	Captura de movimeinto	158 secu, 42.587 km

(Chen et al., 2018)

Se ha seleccionado el conjunto de datos *KITTI* para el desarrollo del presente trabajo, por ser el conjunto de datos popular en las implementaciones de odometría visual, el conjunto de datos presenta una gran resolución de 1241×376. Las imágenes están en formato PNG que pueden ser procesados por cualquier lenguaje de programación.

Se cuenta con archivos que presentan múltiples filas de 12 valores que representan a las matrices de rotación y rotación. Estos valores corresponden a cada fotograma capturado durante el recorrido de las diferentes secuencias. A diferencia de los otros conjuntos de datos que presentan los valores puros provenientes de los sensores, lo que puede generar discontinuidades en la reconstrucción de las secuencias.

Estas 12 secuencias son de diferentes escenarios con elementos de tráfico del mundo real, que van desde autopistas hasta áreas rurales, con objetos dinámicos y estáticos. Adicional se muestra marcas de tiempo síncronas con la captura de las imágenes, se ha utilizado sistemas de localización de última generación con gran precisión en comparación a los demás conjuntos de datos, presentan más de 200 km de recorrido.

Además, el conjunto de datos cuenta con librerías para la utilización de las imágenes como los datos de las posiciones y orientaciones en lenguaje Python, a diferencia de los anteriores conjuntos de datos ya que se debe generar el código desde cero.

5.3. Métricas de éxito

Se ha observado el conjunto de datos *KITTI* es el ideal para el entrenamiento y prueba de las diferentes arquitecturas. Se empleó un equipo con CPU i7-9700KF de 3.60GHz, 16 *GB* de memoria *RAM*, tarjeta gráfica *NVIDIA GTX* 1660 super de 6 *GB*, como el equipo cuenta con tarjeta gráfica *NVIDIA* se instaló *CUDA Toolkit* 10.2 para obtener soporte para *GPU* en *TensorFlow.* De esta forma se optimiza tiempos de ejecución de los modelos por la paralelización que presenta la arquitectura de la tarjeta gráfica.

5.3.1. Dimensión de imágenes

Debido a las limitaciones del equipo se decidió reducir el tamaño de las imágenes de 1241×376 a 640×192, de esta forma se reduce el tiempo de entrenamiento y no satura al equipo durante la ejecución del entrenamiento de las arquitecturas. Sin embargo, debido a la reducción en la dimensión de las imágenes podría causar perdidas de información en el entrenamiento de las arquitecturas porque las relaciones entre pixeles serían inconsistentes.

5.3.2. Secuencias de entrenamiento

Se emplean las secuencias 00, 02, 04, 06, 08 y 10 cuentan con 4541, 4661, 271, 1101, 4071 y 1201 pares imágenes, las trayectorias reales se presentan en la Figura *36* respectivamente.

Se ha escogido estas secuencias para el entrenamiento por presentar un mayor número de imágenes para los diferentes recorridos, la secuencia 02 es la que cuenta con el mayor número de datos de entrenamiento. Además de presentar en los recorridos cambios de dirección hacia adelante. Para la gráfica se ha tomado los ejes *X* y *Z* por la disposición de los ejes la cámara de video.

5.3.3. Secuencias de prueba

Para la evaluación de las diferentes arquitecturas se emplean las secuencias 03, 05 y 07 cuenta con 801, 2761 y 1101 pares imágenes respectivamente, las trayectorias se presentan en la Figura 37.

Se ha escogido estas secuencias para la evaluación de los modelos, por presentar un menor número de imágenes para los diferentes recorridos. Además de presentar en los recorridos cambios de dirección hacia adelante. Para las gráficas se ha tomado los ejes *X* y *Z* por la disposición de la cámara de video.

Figura 36. *Secuencias de entrenamiento*

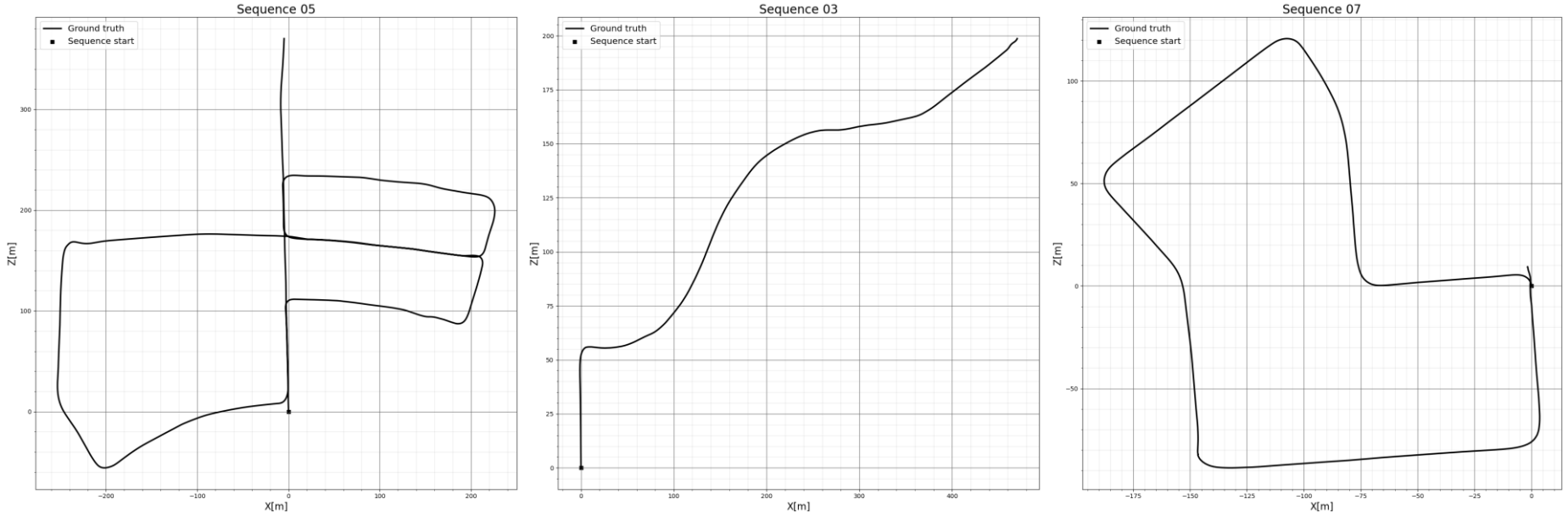

Figura 37. Secuencias de prueba

5.3.4. Pérdida durante el entrenamiento

Durante el entrenamiento de las arquitecturas de odometría visual se pretende que el modelo sea capaz de determinar buenos valores para todos los *weights* y *bias* de las posiciones de verdad del conjunto de datos. En el aprendizaje supervisado un algoritmo, un algoritmo de aprendizaje automático crea modelos a partir de los ejemplos e intenta encontrar las mejores métricas para reducir la pérdida, denominado minimización del riesgo empírico (Google, 2020).

El termino pérdida se refiere a la penalización al producir una mala predicción, por lo que la pérdida es un marcador para mostrar lo mal que el modelo ha calculado en un ejemplo. Si el modelo ha logrado predecir el valor de verdad de forma correcta el valor de la perdida es cero, de lo contrario el valor es mayor.

En general el objetivo del entrenamiento es el de encontrar los valores de *weights* y *bias* que generen una pérdida baja o cercana a cero.

5.3.5. Error absoluto en el cálculo de posición y orientación

Dado que cualquier estimación presente en los cálculos internos que realiza las arquitecturas de odometría visual, por lo que es atractivo en el mundo científico resaltar el resultado de la estimación junto con la medida real. Por lo que la incertidumbre en la medida es un valor numérico resultado dos cálculos que son el error absoluto y el error relativo.

El error absoluto es la medida de realizar la diferencia del valor real entre el valor estimado, puede ser positivo o negativo y presenta las mismas unidades de medida. Es un indicador de la imprecisión que tiene el modelo al realizar una predicción:

$$\varepsilon_a = y - prediccion(x)$$

Siendo:

- (x, y) ejemplo
    - x se refiere al conjunto de características, que el modelo emplea para realizar las predicciones
    - y es el valor real del ejemplo
- $prediccion(x)$ se refiere al modelo que va a realizar la estimación del valor real

5.3.6. RMSE para el cálculo de posición y orientación

Es la raíz cuadrada de la perdida cuadrática media de todo el conjunto de datos. Para el cálculo del *RMSE* se realiza la raíz cuadrada de la suma de todas las perdidas cuadradas para los ejemplos individuales que posteriormente se dividirá para el número total de ejemplos (Google, 2020).

$$RMSE = \sqrt{\frac{1}{N} \sum_{(x,y)\in D} (y - prediccion(x))^2}$$

Siendo:

- (x, y) ejemplo
    - x se refiere al conjunto de características, que el modelo emplea para realizar las predicciones
    - y es el valor real del ejemplo
- $prediccion(x)$ se refiere al modelo que va a realizar la estimación del valor real
- D se refiere al conjunto de datos etiquetados por (x, y)
- N número de ejemplos en el conjunto de datos D

6. Desarrollo de la comparativa

En la presente sección se presenta una comparativa de los modelos *DeepVO*, *MagicVO* y *PoseConvGRU* tanto en el número de parámetros que cuenta cada arquitectura, estimación de posición y orientación en los conjuntos de datos de prueba, tiempo de inferencia de la arquitectura, etc.

6.1. Conjunto de datos

Para el entrenamiento de las diferentes arquitecturas se ha elegido el conjunto de datos *KITTI Visual Odometry* (Geiger et al., 2012) como se discutió en el apartado anterior.

Se presentan 11 secuencias de imágenes con su respectivo valor de verdad, cada secuencia de imágenes cuenta con dos conjuntos de imágenes izquierda y derecha, ya que se ha utilizado cámaras estero con dimensión de 1241×376 pixeles.

Como se mencionó en el anterior apartado, cada secuencia representa diferentes recorridos, los cuales se encuentran con entornos donde se observan, objetos estáticos como dinámicos durante el recorrido. Por lo cual se ha utilizado el conjunto de imágenes de la cámara izquierda de las secuencias 00, 02, 04, 06, 08 y 10 por ser secuencias relativamente extensas. Con un total de 15846 imágenes en total para el entrenamiento y las secuencias 03, 05 y 07 se han utilizado para evaluación.

Se ha realizado un preprocesamiento de las imágenes de las cuales se describen a continuación.

```python
def decode_img(self, img):
    image = tf.image.decode_png(img, channels=3)
    image = image[..., ::-1]
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, [self.height, self.width])
    return image

def load_image(self, filename, odometry):
    img1 = tf.io.read_file(filename[0])
    img2 = tf.io.read_file(filename[1])
    img1 = self.decode_img(img1)
    img2 = self.decode_img(img2)
    img = tf.concat([img1, img2], -1)
    return img, odometry
```

Figura 38. *Preprocesamiento de imagen*

En la Figura *38* se crearon dos funciones, la primera *load*_img hace referencia a la lectura y decodificación de las imágenes:

1. Se ha utilizado pares de imágenes para generar los valores de entrada para los modelos.

2. La lectura de archivos con *TensorFlow*.

3. Como la librería *TensorFlow* realiza la lectura de archivos, se debe decodificar las imágenes para obtener los 3 canales *RGB,* que se refiere a las capas rojo, verde y azul que se encuentra formada la imagen.

Posteriormente, existe una subfunción *decode_img*, donde se realiza:

1. Como las diferentes arquitecturas se emplea el modelo pre entrenado *FlowNet*, se debe cambiar las imágenes a 3 canales *BGR.*

2. Se ha cambiado el formato de la variable que contiene la imagen a tensores para el procesamiento de los modelos.

3. Se ha redimensionado las imágenes a 640×192 pixeles para reducir el tiempo de entrenamiento de los modelos.

4. El par de imágenes consecutivas se concatena una detrás de otra, de forma que las variables de entrada a los modelos son de $640 \times 192 \times 6$ como se muestra en la Figura *39*.

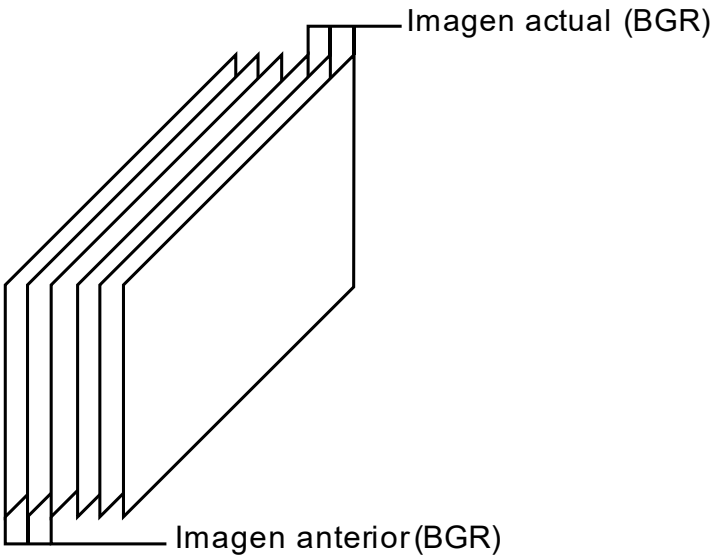

Figura 39. *Estructura de la variable de la imagen*

Los valores reales para cada imagen se encuentran en formato de vector 1×12 que representan a la matriz de rotación:

$$[a_{11} \quad a_{12} \quad a_{13} \quad t_x \quad a_{21} \quad a_{22} \quad a_{23} \quad t_y \quad a_{31} \quad a_{32} \quad a_{33} \quad t_z]$$

Para lo cual, es necesario convertir la matriz de 1×12 a 3×4

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \end{vmatrix}$$

Siendo

- Matriz de rotación $\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$
- Vector de traslación $\begin{vmatrix} t_x \\ t_y \\ t_z \end{vmatrix}$

Para el entrenamiento de los diferentes modelos se ha empleado la posición relativa de un cuadro con respecto al otro, para ello se utilizó la matriz de rotación homogénea del par de imágenes consecutivas Figura *40*.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Y la transformación lineal $Rt_1^{-1} * Rt_2$, siendo Rt_1 la matriz de rotación de la imagen anterior y Rt_2 la matriz de rotación de la imagen actual. Posterior se calcula los ángulos de Euler de la matriz de rotación resultante para concatenarlo en un solo vector junto con el vector de traslación de la siguiente forma:

$$[t_x \quad t_y \quad t_z \quad \alpha \quad \beta \quad \gamma]$$

Este proceso se realiza de manera iterada hasta que se complete el recorrido de las secuencias

```python
def rotationMatrixToEulerAngles(self, R):
    assert (self.isRotationMatrix(R))
    sy = math.sqrt(R[0, 0] * R[0, 0] + R[1, 0] * R[1, 0])
    singular = sy < 1e-6

    if not singular:
        x = math.atan2(R[2, 1], R[2, 2])
        y = math.atan2(-R[2, 0], sy)
        z = math.atan2(R[1, 0], R[0, 0])
    else:
        x = math.atan2(-R[1, 2], R[1, 1])
        y = math.atan2(-R[2, 0], sy)
        z = 0
    return np.array([x, y, z], dtype=np.float32)

def matrix_rt(self, p):
    return np.vstack([np.reshape(p.astype(np.float32), (3, 4)), [[0., 0., 0., 1.]]])

def get_data(self):
    images_paths = []
    odometries = []
    for index, sequence in enumerate(self.sequences):
        for i in range(self.sizes[index] - 1):
            images_paths.append([self.get_image_paths(sequence, i), self.get_image_paths(sequence, i + 1)])
            pose1 = self.matrix_rt(self.poses[index][i])
            pose2 = self.matrix_rt(self.poses[index][i + 1])
            pose2wrt1 = np.dot(np.linalg.inv(pose1), pose2)
            R = pose2wrt1[0:3, 0:3]
            t = pose2wrt1[0:3, 3]
            angles = self.rotationMatrixToEulerAngles(R)
            odometries.append(np.concatenate((t, angles)))
    return np.array(images_paths), np.array(odometries)
```

Figura 40. *Preprocesamiento de valores reales*

6.2. Implementación de arquitecturas

Se ha implementado las arquitecturas *DeepVO*, *MagicVO* y *PoseConvGRU* en *Python*, se utilizó la librería de alto rendimiento de *TensorFlow* con el submódulo de *Keras* para la

creación de redes neuronales profundas porque cuenta con una gran variedad de documentación y soporte para implementar diferentes modelos de redes neuronales.

Para el conjunto de datos seleccionado se ha creado código para que el *pipeline* de carga de datos sea dinámico al momento de realizar el entrenamiento, esto se debe a que almacenar todas las secuencias en memoria *RAM* es ineficiente y satura los recursos del computador.

Para ello se ha utilizado la librería *tf.data* de *TensorFlow* que permite la carga dinámica de datos y la selección de imágenes aleatorias durante el entrenamiento de redes neuronales profundas. De forma que se carga un mini lote de 8 pares de imágenes en memoria para que sean procesadas, durante ese tiempo se carga el siguiente mini lote en memoria y así sucesivamente hasta completar todas las secuencias de los datos de entrenamiento. El entrenamiento de cada arquitectura se realizó durante 200 épocas para realizar una comparativa a la par con el mismo conjunto de datos.

Para acelerar el entrenamiento se ha utilizado el modelo pre entrenado de *FlowNet*, ya que es la primera capa en la cual se lleva el procesamiento con redes neuronales convolucionales. El modelo pre entrenado *FlowNet* tiene la peculiaridad de que fue entrenado con unidades de activación *Leaky ReLU* con el coeficiente *alpha* de 0.1, a diferencia de los desarrollados en las arquitecturas a *DeepVO*, *MagicVO* y *PoseCnvGRU* que emplean funciones de activación *ReLU* en la capa de convolución.

Adicional se ha modificado las unidades de activación en las capas con redes neuronales completamente conectadas a *Leaky ReLU* para ayudar a la convergencia de los modelos para la estimación de odometría visual. También se ha modificado la última capa que cuenta con la unidad de activación *Linear* para generar los valores de posición y rotación relativa.

6.2.1. Capa convolucional FlowNet

Las arquitecturas elegidas han empleado *FlowNet* para la extracción de características de flujo óptico. Existen múltiples variaciones de *FlowNet* las cuales cuentan con múltiples capas de correlación, convolucionales y de refinamiento para la estimación del flujo óptico. Por ende, se ha utilizado *FlowNet S* que es la versión con un número reducido de parámetros y la cual presenta una mayor velocidad de estimación de valores.

Se ha utilizado los parámetros de *weights* y *biases* alojada en *GitHub*[1], que es una implementación no oficial de (Dosovitskiy et al., 2015), el modelo pre entrenado se encuentra

[1] FlowNet https://github.com/sampepose/flownet2-tf

desarrollado con *TensorFlow* v1.x por lo que se ha logrado extraer los valores de *weights* y *biases* para ser utilizadas en la nueva versión de *TensorFlow* que es la v2.2.

Se ha utilizado la versión 2.2 ya que es fácil de usar por su simplicidad en la creación de modelos, eliminación de colecciones gráficas, eliminación de creación de sesiones para realizar procesamiento de tensores y cambios en el procesamiento de variables.

Para la extracción de los valores de *weights* y *biases* de la versión 1.x se ha cargado el modelo almacenado con el buffer de protocolo *MetaGraphDef* el cual permite la recreación de todas las conexiones de la red neuronal por nombres de las diferentes capas, como se muestra en la Tabla *8*.

Tabla 8. Nombre de las capas convolucionales de FlowNet

Capa
Conv1
Conv2
Conv3
Conv3_1
Conv4
Conv4_1
Conv5
Conv5_1
Conv6

Los valores de *weights* y *biases* extraídos se han cargado al nuevo modelo secuencial en la versión 2.2 de *TensorFlow* con igual estructura como se describe en la Figura *41*. Se cuentan con un total de 14612544 parámetros que no van a ser modificados durante el entrenamiento de las diferentes arquitecturas. Adicional se han añadido las unidades de activación *Leaky ReLU* a la salida de cada capa convolucional.

Las salidas de las capas convolucionales tienen las siguientes dimensiones:

1. *conv1* es de dimensión $96 \times 320 \times 64$, siendo el valor de 64 la cantidad de *kenels* de la capa con 18880 parámetros.

2. *conv2* es de dimensión $48 \times 320 \times 128$, siendo el valor de 128 la cantidad de *kernels* de la capa con 204928 parámetros

3. *conv3* es de dimensión $24 \times 80 \times 256$, siendo el valor de 256 la cantidad de *kernels* de la capa con 819456 parámetros

4. *conv3_1* es de dimensión $24 \times 80 \times 256$, siendo el valor de 256 la cantidad de *kernels* de la capa con 590080 parámetros

5. *conv4* es de dimensión $12 \times 40 \times 512$, siendo el valor de 512 la cantidad de *kernels* de la capa con 1180160 parámetros

6. *conv4_1* es de dimensión $12 \times 40 \times 512$, siendo el valor de 512 la cantidad de *kernels* de la capa con 2359808parámetros

7. *conv5* es de dimensión $6 \times 20 \times 512$, siendo el valor de 512 la cantidad de *kernels* de la capa con 2359808 parámetros

8. *conv5_1* es de dimensión $6 \times 20 \times 512$, siendo el valor de 512 la cantidad de *kernels* de la capa con 2359808 parámetros

9. *conv6* es de dimensión $10 \times 3 \times 1024$, siendo el valor de 1024 la cantidad de *kernels* de la capa con 4719616 parámetros

```
Model: "FlowNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 96, 320, 64)       18880
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 96, 320, 64)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 48, 160, 128)      204928
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 48, 160, 128)      0
_________________________________________________________________
conv3 (Conv2D)               (None, 24, 80, 256)       819456
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 24, 80, 256)       0
_________________________________________________________________
conv3_1 (Conv2D)             (None, 24, 80, 256)       590080
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 24, 80, 256)       0
_________________________________________________________________
conv4 (Conv2D)               (None, 12, 40, 512)       1180160
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 12, 40, 512)       0
_________________________________________________________________
conv4_1 (Conv2D)             (None, 12, 40, 512)       2359808
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 12, 40, 512)       0
_________________________________________________________________
conv5 (Conv2D)               (None, 6, 20, 512)        2359808
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 6, 20, 512)        0
_________________________________________________________________
conv5_1 (Conv2D)             (None, 6, 20, 512)        2359808
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 6, 20, 512)        0
_________________________________________________________________
conv6 (Conv2D)               (None, 3, 10, 1024)       4719616
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 3, 10, 1024)       0
=================================================================
Total params: 14,612,544
Trainable params: 0
Non-trainable params: 14,612,544
_________________________________________________________________
```

Figura 41. *Modelo secuencial FlowNet*

Cabe mencionar que el modelo de *FlowNet* es procesada en la tarjeta gráfica por las limitaciones en recursos que cuenta el equipo utilizado para el desarrollo de la presente comparativa. Como la capa *FlowNet* no va a ser modificada durante el entrenamiento de las arquitecturas a analizar es almacenada, para obtener la inferencia del flujo óptico que sirve como entrada para las siguientes capas de las diferentes arquitecturas.

6.2.2. Implementación de DeepVO

A continuación, se procede a crear el modelo *DeepVO* que utiliza los valores obtenidos de *FlowNet* como entrada a las siguientes capas, los valores se describen en la siguiente Tabla

9. En la columna dimensiones hace referencia a las dimensiones de salida de las capas que cuenta el modelo.

Tabla 9. Implementación DeepVO

Capas		**Dimensiones**
Imágenes apiladas		640x192x3
		640x192x3
Capa convolucional FlowNet	Conv 1	320x96x64
	Conv 2	160x48x128
	Conv 3	80x24x256
	Conv 3_1	80x24x256
	Conv 4	40x12x512
	Conv 4_1	40x12x512
	Conv 5	20x6x512
	Conv 5_1	20x6x512
	Conv 6	10x3x1024
Capa LSTM	LSTM1	1000
	LSTM2	1000
Capa completamente conectada		6
Resultado - Pose		6

Para interconectar la variable de salida del modelo *FlowNet* con la siguiente capa *LSTM* de 1000 estados es necesario convertir la matriz $10 \times 3 \times 1024$ a un vector de 1×30720, posterior se utiliza una capa *LSTM* de 1000 estados y por último una capa *Dense* de 6 neuronas para obtener las estimaciones de los valores de posición y orientación relativa entre el par de imágenes de entrada. Se utiliza *Dropout* de 0.5 como regularización de las capas *LSTM,* como se muestra en la Figura *42.*

La arquitectura *DeepVO* cuenta con un total de 149506550 parámetros, de los cuales 14612544 pertenecen al modelo de *FlowNet* que no son modificados durante el entrenamiento y los 134894006 son los que se modificaran en el entrenamiento de la arquitectura que corresponde a las dos capas *LSTM* y a la capa *Dense*.

```
Model: "DeepVO"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
FlowNet (Sequential)         (None, 3, 10, 1024)       14612544
_________________________________________________________________
reshape (Reshape)            (None, 1, 30720)          0
_________________________________________________________________
lstm (LSTM)                  (None, 1, 1000)           126884000
_________________________________________________________________
lstm_1 (LSTM)                (None, 1000)              8004000
_________________________________________________________________
dropout (Dropout)            (None, 1000)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 6006
=================================================================
Total params: 149,506,550
Trainable params: 134,894,006
Non-trainable params: 14,612,544
_________________________________________________________________
```

Figura 42. *Implementación DeepVO*

6.2.3. Implementación MagicVO

MagicVO contiene en las primeras capas el modelo de *FlowNet*, siendo la estimación del flujo óptico la entrada de las capas *Bi-LSTM* y *Dense*, descrita en la Tabla *10*

Tabla 10. Implementación MagicVO

Capas		**Dimensiones**
Imágenes apiladas		640x192x3
		640x192x3
Capa convolucional FlowNet	Conv 1	320x96x64
	Conv 2	160x48x128
	Conv 3	80x24x256
	Conv 3_1	80x24x256
	Conv 4	40x12x512
	Conv 4_1	40x12x512
	Conv 5	20x6x512
	Conv 5_1	20x6x512
	Conv 6	10x3x1024
Capa Bi-LSTM	LSTM1	1000
	LSTM2	1000
Capa completamente conectada		256
		6
Resultado - Pose		6

Para interconectar la variable de salida del modelo *FlowNet* con la siguiente capa *Bi-LSTM* compuesta por dos capas *LSTM* de 1000 estados cada una, por lo cual es necesario convertir

la matriz $10 \times 3 \times 1024$ a un vector de 1×30720, a continuación, se ubican dos capas adicionales *Dense* de 256 con unidades de activación de *Leaky ReLU* y de 6 neuronas con unidades de activación *Linear,* se utiliza *Dropout* de 0.5 como regularización para evitar el *overfitting*. *Dropout* se ubica entre las capas *LSTM* y de las capas *Dense* para obtener las estimaciones de los valores de posición y orientación relativa entre el par de imágenes de entrada como se muestra en la Figura *43*.

```
Model: "MagicVO"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
FlowNet (Sequential)         (None, 3, 10, 1024)       14612544
_________________________________________________________________
reshape (Reshape)            (None, 1, 30720)          0
_________________________________________________________________
bidirectional (Bidirectional (None, 2000)              253768000
_________________________________________________________________
dropout (Dropout)            (None, 2000)              0
_________________________________________________________________
dense (Dense)                (None, 256)               512256
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 1542
=================================================================
Total params: 268,894,342
Trainable params: 254,281,798
Non-trainable params: 14,612,544
_________________________________________________________________
```

Figura 43. *Implementación MagicVO*

La arquitectura del modelo *MagicVO* tiene un total de 268894342 parámetros, como se mencionó anteriormente los parámetros del modelo de FlowNet no son modificados durante el entrenamiento siendo un total de 254281798 los parámetros que se modificaran para la estimación de posición y orientación, de los cuales se encuentran 253768000 parámetros en la capa *Bi-LSTM*, 512256 en la capa *Dense* de 256 neuronas y 1542 en la ultima capa de 6 neuronas.

6.2.4. Implementación PoseConvGRU

La arquitectura *PoseConGRU* cuenta como las anteriores arquitecturas al modelo *FlowNet* como primera etapa para la extracción de características, esta arquitectura cuenta con la

peculiaridad de utilizar una capa *MaxPooling* después del modelo *FlowNet* para reducir el tamaño de características de entrada para las siguientes capas que son la capa *GRU* y *Dense* Tabla *11*.

Para interconectar la variable de salida del modelo *FlowNet* con la siguiente capa *GRU* compuesta por 3 estados, se emplea la capa *MaxPooling* para reducir la matriz de $10 \times 3 \times 1024$ a $5 \times 1 \times 1024$ haciendo que sobresalgan los valores máximos de la matriz de entrada, posteriormente se convierta la matriz a un vector de 1×5120. Se emplean varias capas *Dense* de 4096, 1024, 128 con unidades de activación de *Leaky ReLU* y de 6 neuronas con unidades de activación *Linear* siendo un total de 4389811 parámetros que serán modificados durante el entrenamiento, se utiliza *Dropout* de 0.5 como regularización de las capas *Dense* para obtener las estimaciones de los valores de posición y orientación relativa entre el par de imágenes de entrada evitando el *overfitting* como se muestra en la Figura *44*.

Tabla 11. implementación PoseConvGRU

Capas		**Dimensiones**
Imágenes apiladas		640x192x3
		640x192x3
Capa convolucional FlowNet	Conv 1	320x96x64
	Conv 2	160x48x128
	Conv 3	80x24x256
	Conv 3_1	80x24x256
	Conv 4	40x12x512
	Conv 4_1	40x12x512
	Conv 5	20x6x512
	Conv 5_1	20x6x512
	Conv 6	10x3x1024
	MaxPooling	5x1x1024
Capa GRU	GRU	3
Capa completamente conectada		4096
		1024
		128
		6
Resultado - Pose		6

```
Model: "PoseConvGRU"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
FlowNet (Sequential)         (None, 3, 10, 1024)       14612544
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 1, 5, 1024)        0
_________________________________________________________________
reshape (Reshape)            (None, 1, 5120)           0
_________________________________________________________________
gru (GRU)                    (None, 3)                 46125
_________________________________________________________________
dropout (Dropout)            (None, 3)                 0
_________________________________________________________________
dense (Dense)                (None, 4096)              16384
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              4195328
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 1024)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               131200
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 774
=================================================================
Total params: 19,002,355
Trainable params: 4,389,811
Non-trainable params: 14,612,544
_________________________________________________________________
```

Figura 44. *Implementación PoseConvGRU*

6.3. Entrenamiento de las arquitecturas

Como se mencionó en el capítulo 5, se emplearán las secuencias 00, 02, 04, 06, 08 y 10 para el entrenamiento de las arquitecturas y la secuencia 01 para evaluación durante el entrenamiento durante 200 épocas, se muestra a continuación el progreso del entrenamiento de las arquitecturas para converger a 0 con la función de pérdida Figura *45*.

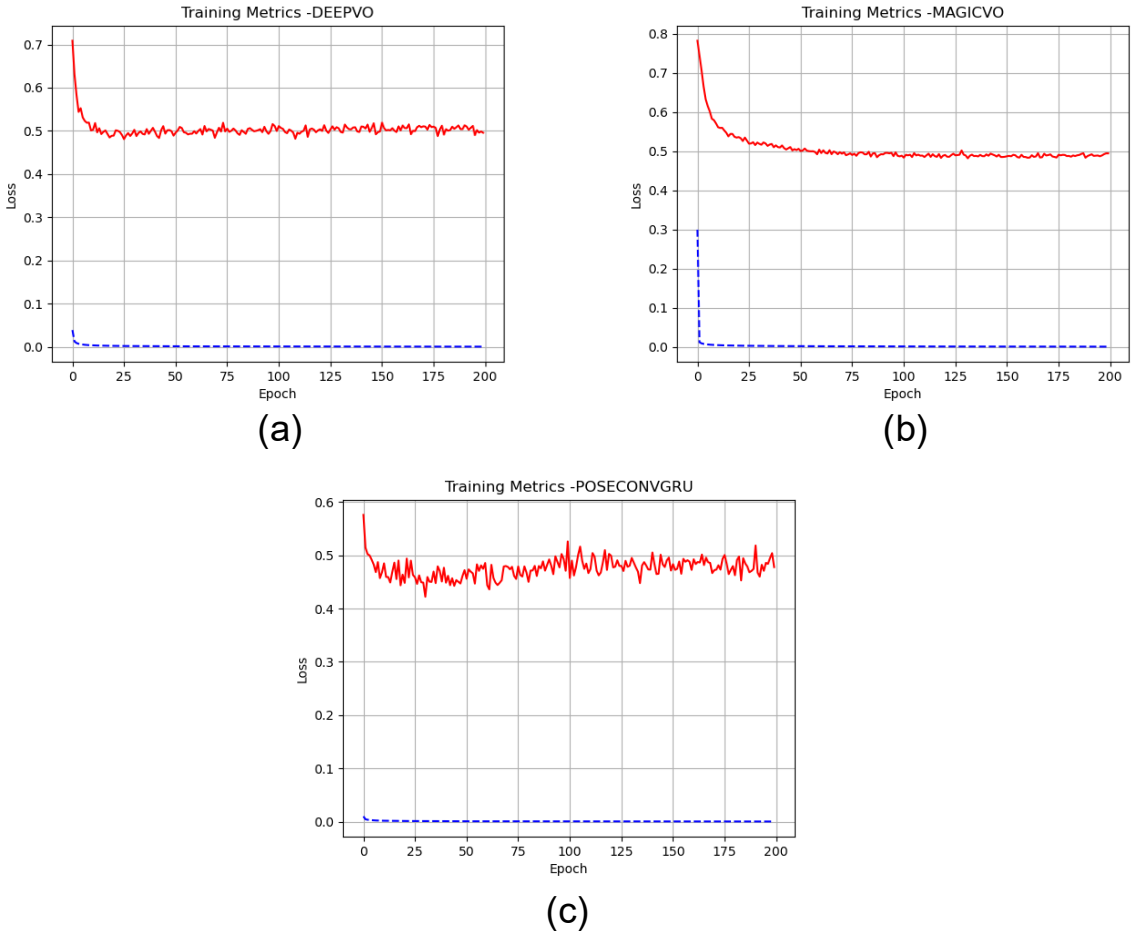

Figura 45. *Perdida durante el entrenamiento*

a) Pérdida de la arquitectura *DeepVO*: el entrenamiento de la arquitectura converge rápidamente a 0 de acuerdo con la función de pérdida, donde se observa que el valor mínimo que alcanza durante el entrenamiento es de 0.0007523670 con el conjunto de datos de entrenamiento, mientras que durante la evaluación con la secuencia 01 no se observa una mejora y el valor oscila en 0.5, el tiempo de entrenamiento por época es aproximadamente de 13 minutos siendo un total de aproximadamente 43 horas en cumplir las 200 épocas.

b) Pérdida de la arquitectura *MagicVO*: el entrenamiento de la arquitectura converge lentamente a 0, donde el valor mínimo que alcanza durante el entrenamiento es de 0.0009443219 con el conjunto de datos de entrenamiento, mientras que con el conjunto de datos de evaluación oscila en 0.5. Siendo el modelo que cuenta con una gran cantidad de parámetros de entrenamiento se observó que el tiempo de entrenamiento por época es aproximadamente de 20 minutos siendo un total de 67 horas en cumplir las 200 épocas.

c) Pérdida de la arquitectura *PoseConvGRU*: durante el entrenamiento de la arquitectura se observa que converge rápidamente a 0, siendo el valor mínimo alcanzado de 0.0006612031 con el conjunto de datos de entrenamiento mientras que con los datos de evaluación oscila intermitentemente en 0.5, el tiempo de entrenamiento por época es de aproximadamente 6 minutos con un total aproximado de 20 horas para cumplir las 200 épocas.

6.4. Parámetros y entrenamiento de las arquitecturas

A continuación, se muestra los parámetros que serán modificados durante el entrenamiento de las diferentes arquitecturas por lo cual no se toma en cuenta los parámetros del modelo *FlowNet* ya que el modelo es el encargado de permitir la extracción del flujo óptimo de los pares de imágenes Tabla *12*.

Tabla 12. Parámetros de las diferentes arquitecturas

Arquitectura	Parámetros que se modifican durante el entrenamiento	Total, de parámetros	Perdida al finalizar el entrenamiento	Tiempo de entrenamiento [minutos / época]
DeepVO	134894006	149506550	0.0007523670	13
MagicVO	254281798	268894342	0.0009443219	20
PoseConvGRU	4389811	19002355	0.0006612031	6

Como se pueda apreciar en todas las arquitecturas se cuenta con 14612544 parámetros que pertenecen al modelo del *FlowNet:*

- La arquitectura *DeepVO* cuenta con 134894006 que se modificaran durante el entrenamiento, que representan el 90.23% del total de parámetros que cuenta la arquitectura, siendo la segunda con gran número de parámetros que serán modificados en el entrenamiento. Además, en la etapa de entrenamiento durante las 200 épocas tardo un total de 13 minuntos por época alcanzando el valor mínimo de pérdida de 0.0007523670 siendo el segundo modelo con mayor número de parámetros y el segundo con mayor cantidad de tiempo en finalizar el entrenamiento de la red neuronal.

- La arquitectura *MagicVO* cuenta con 254281798 parámetros que serán modificados durante el entrenamiento de los cuales representa el 94.57% de los 268894342 parámetros que cuenta la arquitectura, siendo la arquitectura con el mayor número

de parámetros a ser modificados por el entrenamiento, por lo que tiene un tiempo de procesamiento de 20 minutos por época alcanzando un valor mínimo de pérdida de 0.0009443219.

- La arquitectura *PoseConvGRU* presenta 43612544 parámetros que se modificaran en el entrenamiento de la arquitectura, siendo el 23% del total de los 19002355 parámetros que cuenta toda la arquitectura, el tiempo de entrenamiento es reducido, aproximadamente 6 minutos por época alcanzando un valor mínimo de pérdida de 0.0006612031. Representando la arquitectura con menor numero de parámetros de las anteriores arquitecturas y con menor tiempo de entrenamiento.

En comparación la arquitectura *PoseConvGRU* representa el 1.73% del total de parámetros que se modificarán durante el entrenamiento de la arquitectura *MagicVO* y 3.25% del total de parámetros que se modificarán en el entrenamiento que cuenta la arquitectura DeepVO. Por ende, se observa que la arquitectura *PoseConvGRU* es la arquitectura más liviana en termino de cantidad de parámetros modificables, con menor tiempo en el entrenamiento y el que logra a alcanzar el menor valor de pérdida de los demás modelos.

6.5. Evaluación de las arquitecturas

Utilizando las secuencias presentadas en la sección 5.3.3 en la arquitectura *DeepVO*, logra tener un desempeño aceptable pero no en su totalidad, ya que la inferencia de los valores de posición absoluta no coincide durante todo el recorrido de la secuencia como se muestra en Figura 46.

- En la estimación de posición de la secuencia 03, se logra observar que, durante el primer trayecto desde el punto de inicio hasta la primera curva a la derecha coincide con la ruta verdadera, posteriormente a medida que el recorrido avanza la estimación se aleja pronunciadamente en hasta finalizar el recorrido con aproximadamente 70 metros alejada de la posición verdadera.

- En la estimación de la secuencia 05, se logra observar que, desde el punto inicial hasta la primera curva a la derecha, el valor real coincide con la estimación. Posteriormente ingresa en trayectos que generan bucles ya que recorren tramos del recorrido que se repiten, existe ligeros desplazamientos en los bucles, pero al llegar a la última recta se observa un desplazamiento considerable de aproximadamente 100 metros a los valores reales.

- En la estimación de la secuencia 07, se observa que, desde el punto inicial hasta finalizar el recorrido existe un desplazamiento que va desde los 5 metros hasta los 10 metros, pero tiene un desplazamiento menor de 10 metros al finalizar el recorrido.

De igual forma se emplearon las secuencias para estimar las posiciones relativas con la arquitectura *MagicVO*, donde se observa una gran diferencia en comparación con los valores de verdad de las secuencias como se muestra en la Figura 47.

- En la estimación de posición de la secuencia 03, se logra observar que, durante el primer trayecto desde el punto de inicio hasta la primera curva a la derecha coincide con la ruta verdadera, posteriormente a medida que el recorrido avanza la estimación se aleja pronunciadamente hasta finalizar el recorrido con aproximadamente 100 metros alejado de la posición verdadera.

- En la estimación de la secuencia 05, se logra observar que, desde el punto inicial no tarda mucho en que la estimación de la posición se vaya alejando a la posición verdadera. En los trayectos que generan bucles no existe coincidencia, en la ultima recta existe un desplazamiento que sobrepasa a los 200 metros.

- En la estimación de la secuencia 07, se observa que, desde el punto inicial hasta finalizar el recorrido existe un desplazamiento que va desde los 20 metros hasta los 70 metros, al finalizar el recorrido es donde existe un mayor desplazamiento de la posición verdadera.

Se emplearon las secuencias anteriormente mencionadas con la arquitectura *PoseConvGRU*, donde se observa que tiene un mayor desempeño que las anteriores arquitecturas, los valores obtenidos de la inferencia de la arquitectura se asemejan a los valores de verdad de las secuencias evaluadas Figura *48*.

- En la estimación de posición de la secuencia 03, se logra observar que, durante el primer trayecto desde el punto de inicio hasta la primera curva a la derecha coincide con la ruta verdadera, posteriormente a medida que el recorrido avanza la estimación se aleja, pero no como los anteriores, aquí existe un desplazamiento de aproximadamente 50 metros de la posición verdadera.

- En la estimación de la secuencia 05, se logra observar que, desde el punto inicial hasta terminar el recorrido de la secuencia no existe un desplazamiento pronunciado que va desde los 5 metros hasta los 10 metros. En los bucles existe un desplazamiento menor a los 10 metros.

- En la estimación de la secuencia 07, se observa que, desde el punto inicial no tarda mucho a alejarse del valor real, de modo que existe un desplazamiento que va desde los 8 metros hasta los 30 metros.

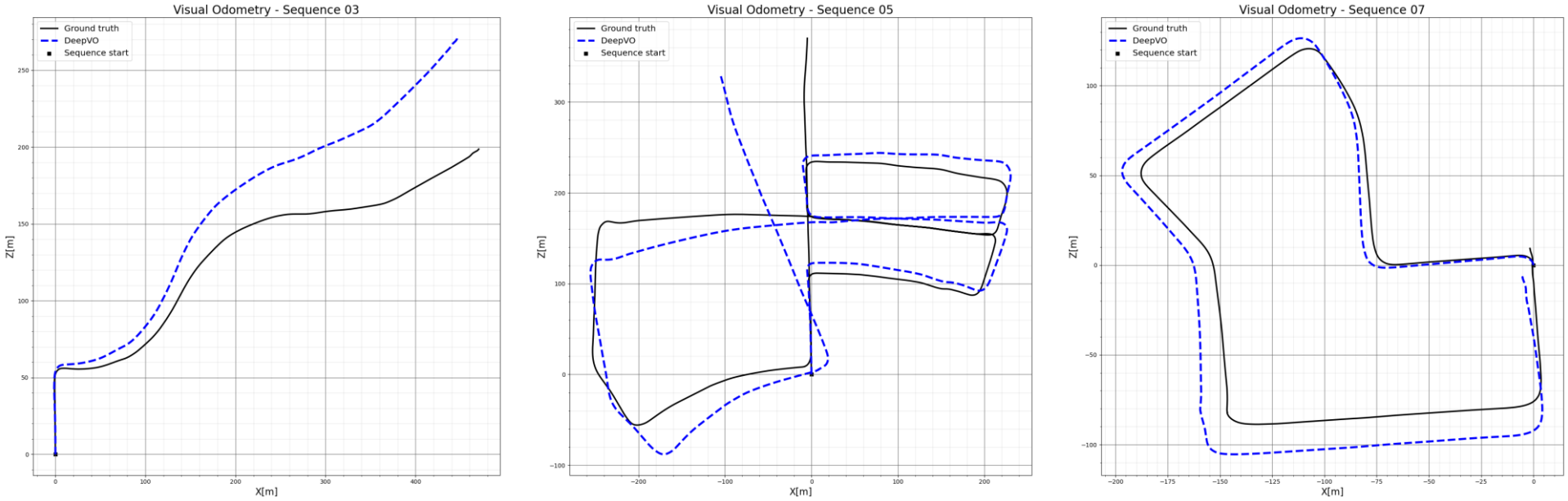

Figura 46. Evaluación de las arquitecturas DeepVO con las secuencias 03, 05 y 07

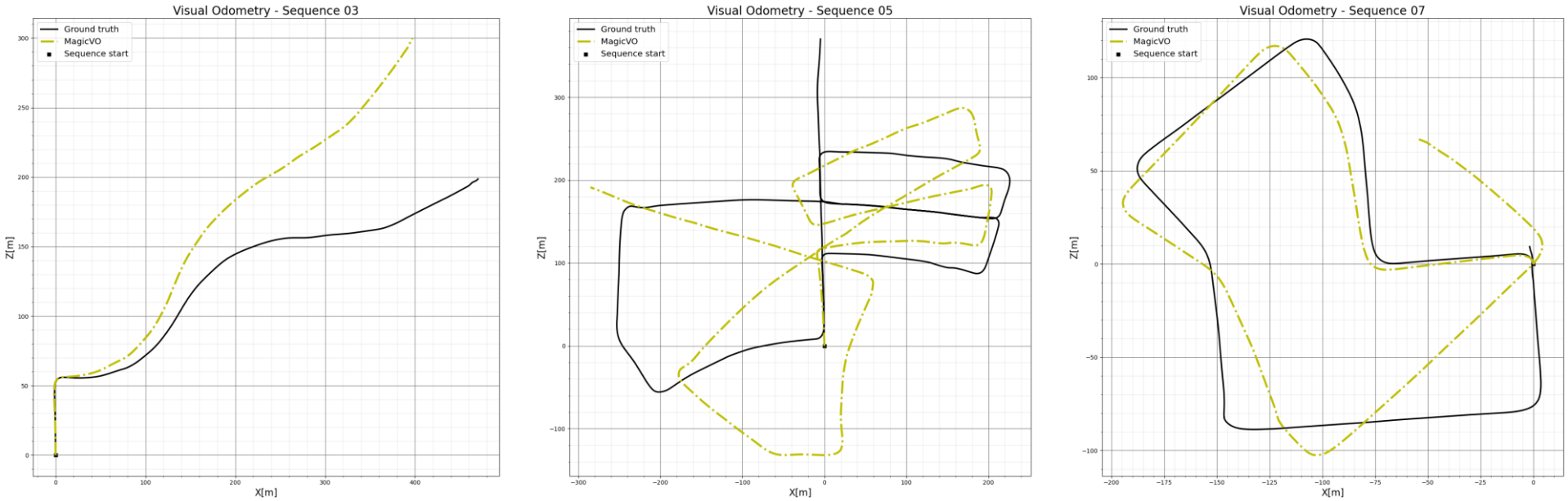

Figura 47. Evaluación de la arquitectura MagicVO con las secuencias 03, 05 y 07

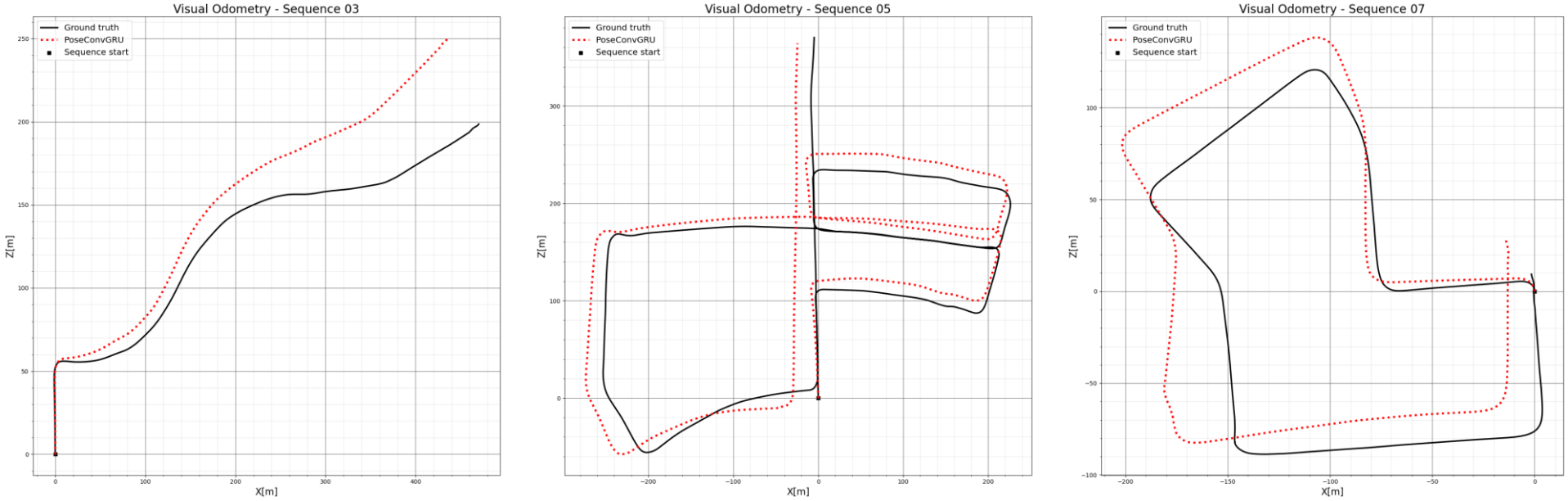

Figura 48. *Evaluación de la arquitectura PoseConGRU con las secuencias 03, 05 y 07*

A continuación, en la Figura *49* se observa que la arquitectura que logra inferir valores que se acerquen a los valores reales es la arquitectura *PoseConvGRU* (línea punteada de color rojo) en la secuencia 03, siendo la arquitectura *MagicVO* (línea con punto de color verde) la que genera estimaciones muy alejadas a los valores de verdad.

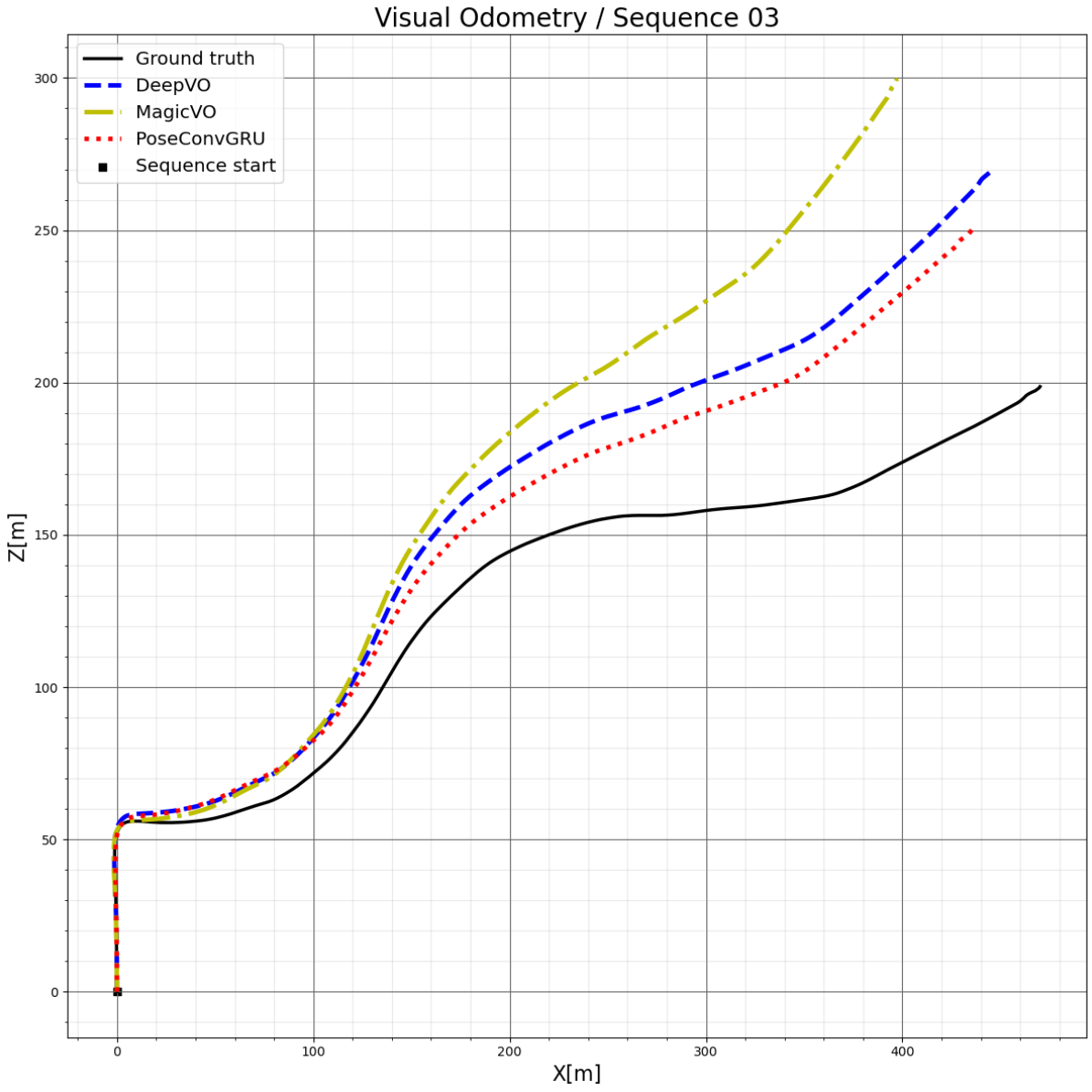

Figura 49. *Comparativa de arquitecturas secuencia 03*

Se observa que al utilizar la secuencia 05 para inferir los valores de las posiciones relativas, la arquitectura *PoseConvGRU* (línea punteada de color rojo) logra acercarse a los valores de verdad de la secuencia, superando en gran medida a las demás arquitecturas, siendo la que tiene un gran desplazamiento la inferencia de la arquitectura *MagicVO* (línea con punto de color verde) como se observa en la Figura *50*.

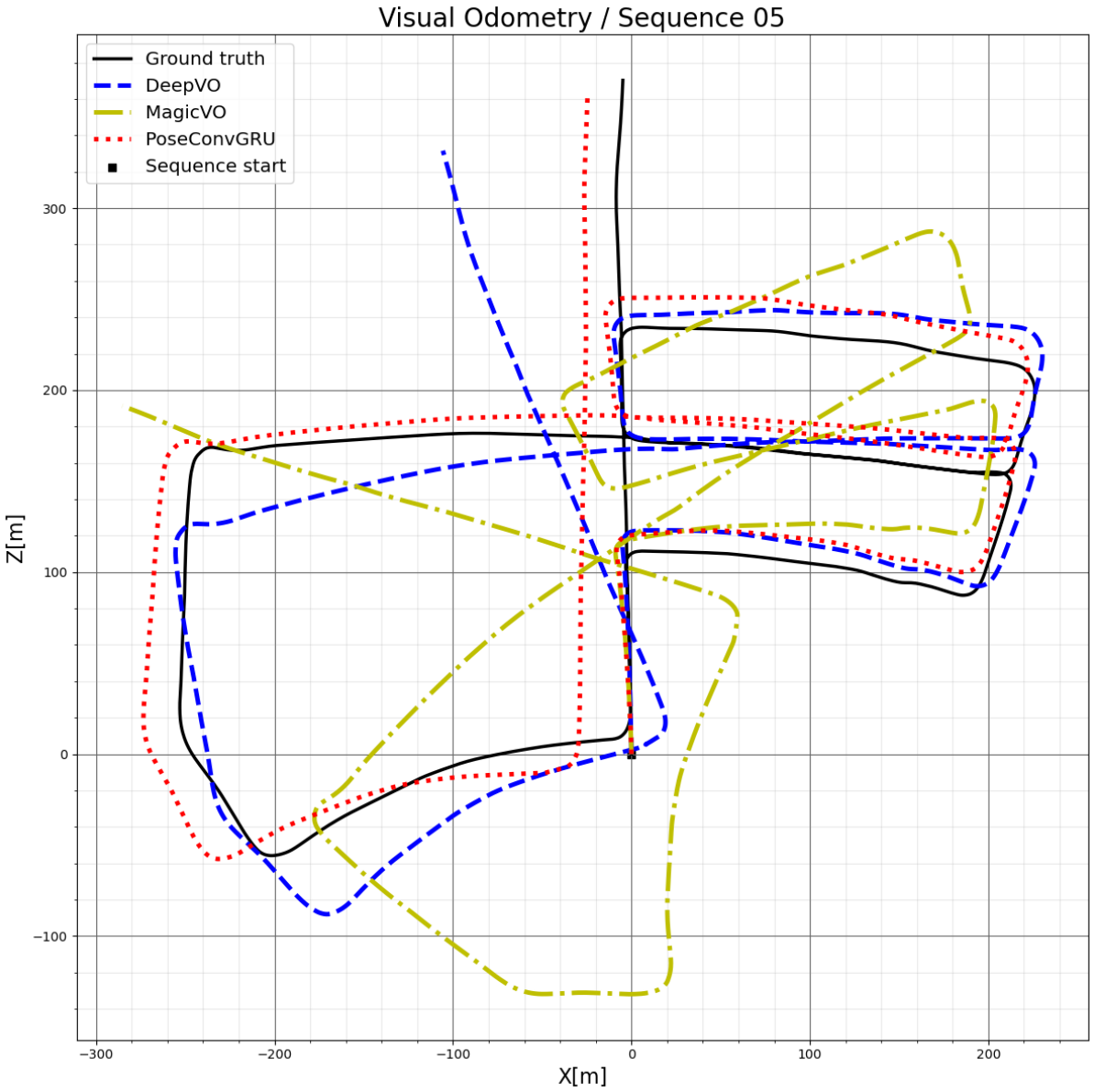

Figura 50. *Comparativa de arquitecturas secuencia 05*

Adicional se utilizó la secuencia 07 para estimar las posiciones relativas con las diferentes arquitecturas, siendo los valores de la arquitectura *DeepVO* (línea entrecortada de color azul) que logra acercarse a los valores de verdad de la secuencia, al igual que las anteriores secuencias se observa que la arquitectura *MagicVO* (línea con punto de color verde) es la que más alejada se encuentra en estimar los valores esperados Figura *51*.

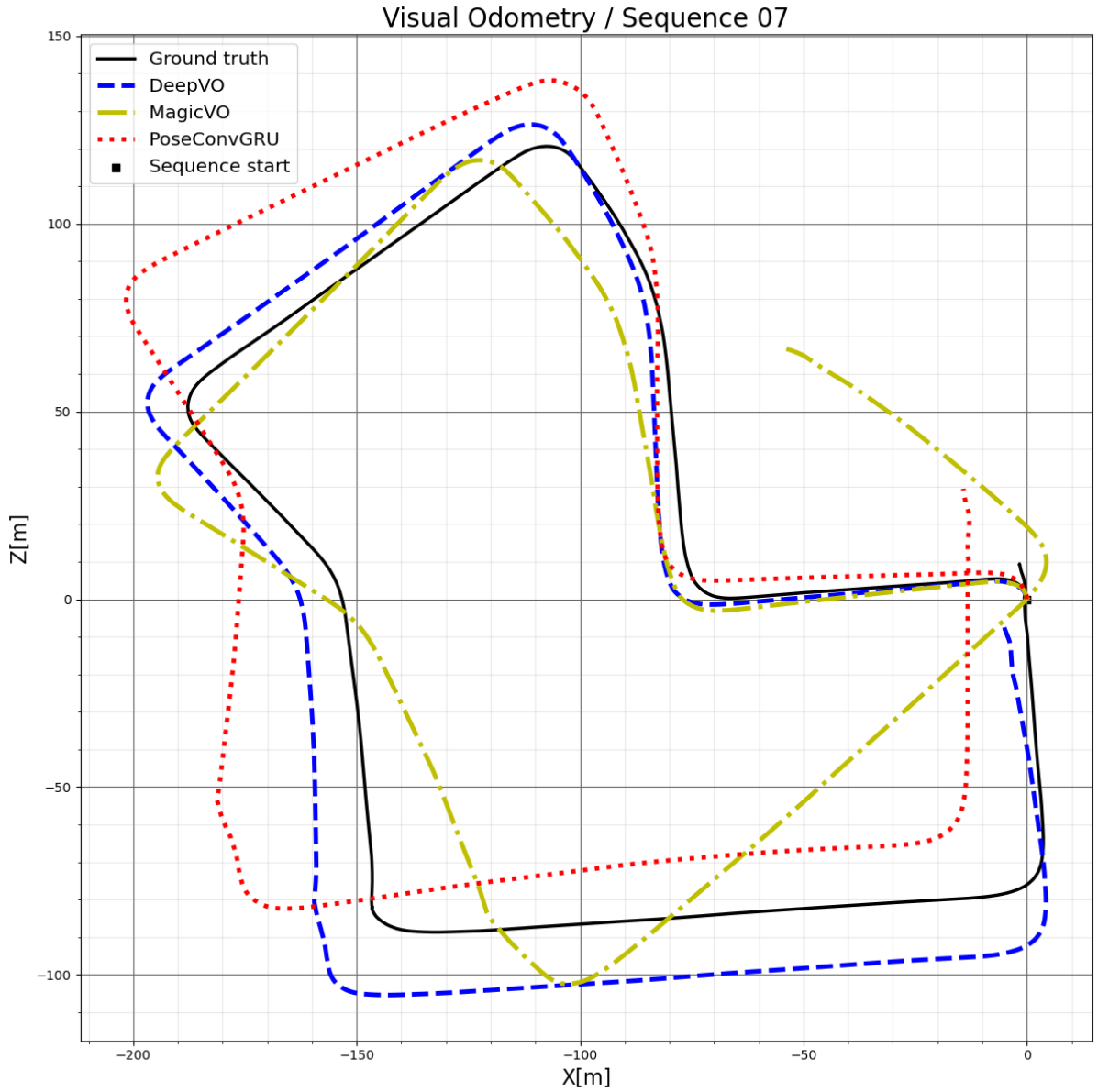

Figura 51. *Comparativa de arquitecturas secuencia 07*

6.6. Tiempo de inferencia de las arquitecturas

Como se mencionó en la sección 6.2 se empleó al modelo pre entrenado *FlowNet* para reducir el tiempo de entrenamiento de las arquitecturas, cabe mencionar que las estimaciones se realizaron en mini lotes de 8 pares de imágenes en cada iteración.

Por limitaciones del equipo donde fue desarrollado el presente trabajo se dividió el procesamiento de parámetros del modelo *FlowNet* en la tarjeta gráfica y el resto de las capas en el *CPU*, ya que existía un exceso de parámetros que la tarjeta gráfica logra procesar, de esta forma se logra optimizar recursos tanto en el procesamiento de las capas convolucionales

como la capas *LSTM*, *GRU* y *Dense*. A continuación, se describe el tiempo de inferencia de las demás capas que siguen al modelo *FlowNet.*

El tiempo de inferencia de las arquitecturas se muestra en la Tabla *13*. Donde por cada par de imágenes las arquitecturas logran estimar valores en un tiempo en orden de los milisegundos

Tabla 13. Tiempo de inferencia de las arquitecturas

Arquitectura	Tiempo de inferencia [s]
DeepVO	~ 0.1489
MagicVO	~ 0.1708
PoseConvGRU	~ 0.0936

Como se puede observar la arquitectura que logra estimar las posiciones en un tiempo relativamente rápido es el modelo *PoseConvGRU* con ~ 0.0936 segundos en inferir los valores, seguida de *DeepVO* con ~ 0.1489 segundos y finalmente el modelo *MagicVO* con ~ 0.1708 segundos, haciendo que la arquitectura *MagicVO* sea la que más tiempo se tarda en procesar los mini lotes de las secuencias.

7. Discusión y análisis de resultados

Como se ha mostrado en las secciones anteriores, el modelo *FlowNet* ha sido empleado en los modelos a analizar ya que presenta la propiedad de extraer las características de flujo óptico, de este modelo parte las demás arquitecturas. Es así como se empieza a realizar el análisis de las capas consecuentes del modelo *FlowNet* que son las que se modifican durante el entrenamiento.

Como primer punto se analiza las capas que fueron utilizadas en cada una de las arquitecturas y los valores de perdida al finalizar el entrenamiento:

- DeepVO: se encuentra estructurada por *FlowNet*, posteriormente se cuenta con dos capas *LSTM* de 1000 estados ocultos cada una. Estas capas *LSTM* son empleadas para que la arquitectura contenga unidades de memoria a largo plazo hacia el futuro, lo que permite que la arquitectura pueda recordar valores futuros con los que fue entrenada, la primera capa *LSTM* se observa un total de 126884000 parámetros, seguida con otra capa *LSTM* de 8004000 parámetros por lo que hace que el entrenamiento e inferencia de la arquitectura tenga un tiempo prolongado de procesamiento. Además, estas capas *LSTM* cuentan con regularizaciones *Dropuout* de 0.5, permitiendo que, durante el entrenamiento, el modelo evite el *oferfitting* ya que en cada iteración alterna el entrenamiento de los estados ocultos en un 50%, con lo que en cierto modo extiende la convergencia del modelo a 0. Posterior se encuentra la última capa *Dense* que cuenta con 6006 parámetros con unidad de activación Linear ya que se desea la inferencia de valores que se asemejen a un problema de regresión. Se observó que los parámetros de la arquitectura generan 0.0007523670 de valor de pérdida al finalizar el entrenamiento.

- MagicVO: se encuentra estructurada por FlowNet en la primera etapa del modelo, posterior se encuentra una capa *Bi-LSTM* de 2000 estados ocultos, las capas *Bi-LSTM* se forman al conectar dos capas *LSTM* una que sea capaz de contener memoria a largo hacia el futuro y otra que contenga memoria a largo plazo hacia el pasado, lo que permite que el modelo sea capaz de recordar valores futuros o pasados con los que fue entrenado, esta capa cuenta con 253768000 parámetros lo que hace que la arquitectura sea muy extensa y difícil de procesar debido a la gran cantidad de parámetros que cuenta solo esta capa, posteriormente se cuenta con capas densas de 256 neuronas con unidad de activación *Leaky ReLU* y 6 neuronas con unidad de activación *Linear,* adicional se cuentan con capas . Estas dos últimas capas cuentan

con 512256 y 1542 parámetros cada una para ser modificados durante el entrenamiento, siendo la arquitectura con mayor número de parámetros para ser entrenado y que el tiempo de entrenamiento es superior al resto, esto se debe a la gran cantidad de estados ocultos que cuenta en la capa *Bi-LSTM* y que este proceso no puede ser paralelizado como sucede con las capas *CNN*. Se observo que la pérdida al finalizar el entrenamiento de la arquitectura fue de 0.0009443219

- PoseConvGRU: emplea en las primeras capas *FlowNet* para la estimación del flujo óptico de las secuencias de imágenes, posterior se emplea una capa *MaxPooling* que permite reducir el número de parámetros y extraer aún más las características de la inferencia del modelo *FlowNet*, posterior se encuentra la capa *GRU* de 46125 parámetros, esta capa es similar a las *LSTM* con la diferencia que combina en celdas los estados ocultos tanto en los parámetros *forget gate* e *input gate,* haciendo que sea una capa con mejores características que las *LSTM* lo que permite que la arquitectura pueda mantener información a lo largo del tiempo sin necesidad de que los valores sean reiniciados lo que con lleva a que no se eliminen valores que pueden ser relevantes en la inferencia. Posteriormente se cuentan con capas *Dense* de 4096, 1024, 128 y 6 neuronas con unidades de activación *Leaky ReLU* a excepción de la última capa que ocupa la unidad de activación *Linear.* Esta arquitectura tiene la peculiaridad que cuenta con un número reducido de parámetros en comparación a las anteriores dos arquitecturas analizadas. En si es un modelo que con poca cantidad de parámetros puede inferir de mejor forma los valores de posición relativas. Se observó que los parámetros de la arquitectura generan 0.0006612031 de valor de pérdida al finalizar el entrenamiento.

De acuerdo con lo detallado anteriormente se observa que las arquitecturas cuentan con una gran cantidad de parámetros que son modificados durante el entrenamiento, lo que conlleva un mayor tiempo de entrenamiento, tiempo de inferencia y precisión en la estimación. Al finalizar el entrenamiento la arquitectura que cuenta con un valor mínimo de pérdida es la arquitectura *PoseConvGRU* con 0.0006612031, esto se debe a que cuenta con un menor número de parámetros en las capas *GRU* donde la red neuronal puede aprender las relaciones secuenciales entre imágenes, presentando un mayor desempeño al finalizar el entrenamiento de la arquitectura.

Como siguiente punto se analiza el error absoluto presente en la evaluación del modelo con las secuencias 03, 05 y 07. Como se observó en la anterior sección, no existe un modelo por el cual logre coincidir perfectamente con los valores de verdad, por lo que se ha graficado el error presente en la inferencia de las posiciones absolutas durante el recorrido de las

secuencias. A continuación, se presenta el error presente en el recorrido de la secuencia 03 tanto en los valores de posición como de orientación.

Como se observa en la Figura 52, a medida que el recorrido avanza el error se propaga en la estimación de la posición, esto se debe en gran medida que las transformaciones lineales que suceden para pasar de posición relativa a posición absoluta conllevan la utilización de los ángulos presentes en la inferencia de las arquitecturas.

- La estimación de posición en el eje *X* la arquitectura *DeepVO* presenta un error al finalizar el recorrido de 24 metros, a diferencia de la arquitectura MagicVO que presenta un error de 74 metros al finalizar el recorrido.

- La estimación de la posición en el eje *Y* la arquitectura *DeepVO* genera un error cercano a 40 metros al finalizar el recorrido, a diferencia de la arquitectura PoseConvGRU que genera un error aproximado de 120 metros.

- Para la estimación de la posición en el eje *Z* se observa que los valores estimados son mayores a los valores verdaderos, por ende, los valores son negativos. Como se observa la arquitectura *PoseConvGRU* tiene un menor error al finalizar el recorrido que es de aproximadamente 50 metros, a diferencia de la arquitectura *MagicVO* que genera un error por encima de los 100 metros.

Siendo la arquitectura que cuenta con una mayor cantidad de error es la de MagicVO, seguida de PoseConvGRU y la que tiene menor error en posición es la arquitectura DeepVO en las posiciones en el eje X e Y, a diferencia que la posiciones en el eje Z donde el que tiene menor error es la de PoseConvGRU.

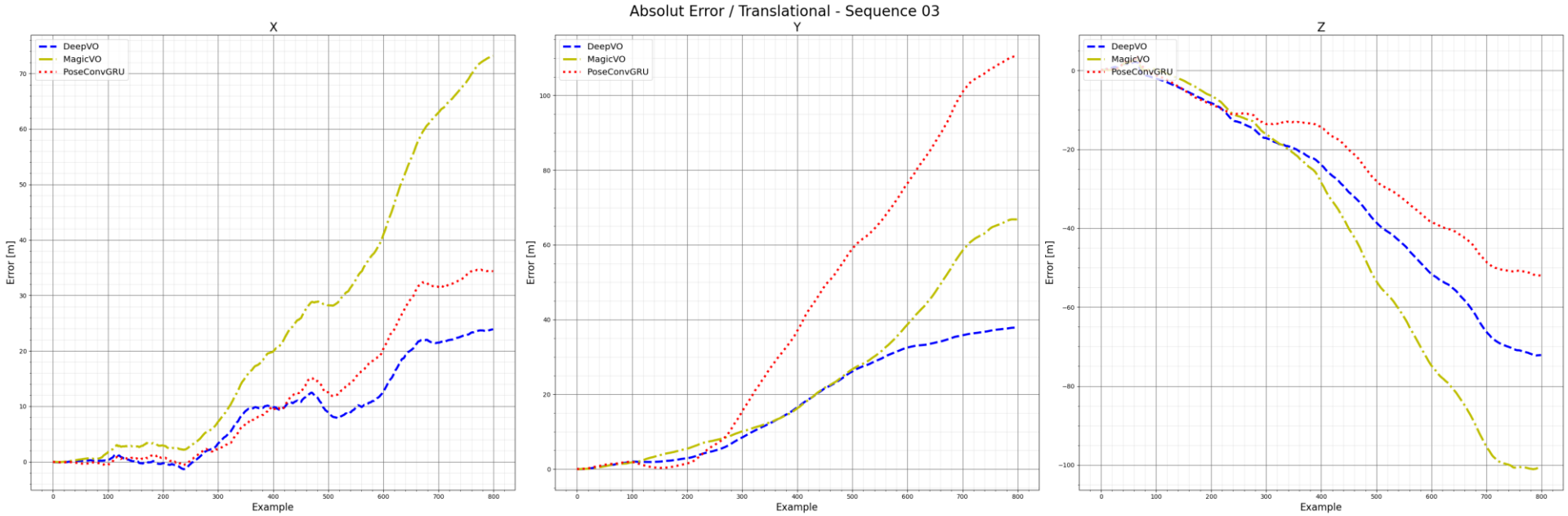

Figura 52. *Error absoluto de las arquitecturas secuencia 03 – Posición*

Posteriormente se observa el error absoluto presente en los valores de inferencia de la rotación de la secuencia 03 en la Figura 53:

- Para la estimación de rotación con respecto al eje *X* (*Roll*), se observa que durante el recorrido de la secuencia el error no se va acumulando, de manera que existen picos en los pares de imágenes 100 a 160 y de 400 a 500. Por lo que el error de la estimación del ángulo intenta acercarse a $0\,rad$, siendo la arquitectura MagicVO la que produce picos más pronunciados de aproximadamente $2\,rad$ seguida de la arquitectura *DeepVO* y *PoseConvGRU*. Al finalizar el recorrido la que menor error genera es la arquitectura *DeepVO* ya que genera un error cercano a 0 rad a diferencia de la arquitectura *MagicVO* y *PoseConvGRU* que generan un error de $0.5\,rad$.

- Para la estimación de rotación con respecto al eje *Y* (*Pitch*), se observa que el error va aumentando a medida que la secuencia avanza, llegando a ser el error de aproximadamente *0.5 rad* el valor mayor que genera la arquitectura *MagicVO,* a diferencia de las arquitecturas *DeepVO* y *PoseConvGRU* que presentan un error de *0.18* aproximadamente.

- Para la estimación de rotación con respecto al eje *Z* (*Yaw*), se observa un comportamiento parecido al error de estimación del eje *X,* ya que existen picos de error casi en las mismas zonas 100 a 160 y de 400 a 500. Siendo el error máximo de 2 rad que presenta la estimación de la arquitectura *MagicVO.* Al finalizar el recorrido las arquitecturas que tienen un valor cercano a 0 rad de error son las arquitecturas *DeepVO, y MagicVO.* La arquitectura *PoseConvGRU* genera un error de $0.2\,rad$ aproximadamente al finalizar el recorrido.

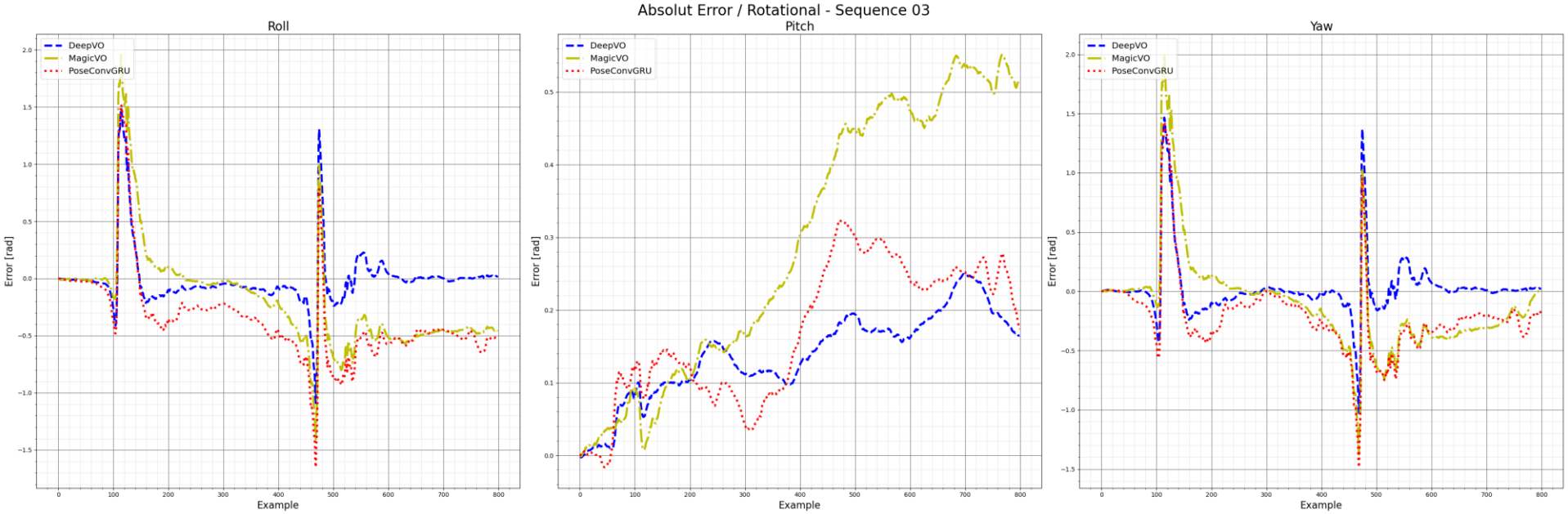

Figura 53. *Error absoluto de las arquitecturas secuencia 03 – Orientación*

Se presenta la evaluación de las arquitecturas con la secuencia 05 en la Figura 54, donde se observa:

- Para la estimación de la posición en el eje *X,* el error de las arquitecturas *DeepVO* y *PoseConvGRU* oscila de 0 a 20 metros durante el recorrido, pero al finalizar el recorrido la arquitectura *PoseConvGRU* genera un error de 10 metros aproximadamente que es menor a las demás arquitecturas. La arquitectura *MagicVO* genera un error de 290 metros aproximadamente y la arquitectura *DeepVO* que genera un error de 100 metros al finalizar la secuencia.

- Para la estimación de la posición en el eje *Y*, se observa que existe oscilaciones lo cual dependería en gran medida en el recorrido, cuenta con varias curvas que cierran circuitos (bucles). La estimación de la arquitectura *DeepVO* es una de las arquitecturas que genera mayores picos de error siendo el mayor de 100 metros aproximadamente. La arquitectura *MagicVO* tiene un error reducido a los 50 metros en los primeros 1000 ejemplos, pero a partir de los 1600 ejemplos el error crece mayor mente hasta el final del recorrido de 120 metros. A diferencia de la arquitectura PoseConvGRU que es menor de 50 metros.

- Para la estimación de la posición en el eje *Z*, se observa que la arquitectura *MagicVO* es la que genera la mayor oscilación de errores que van hasta 200 metros, esto puede ser causado por el número de épocas de entrenamiento. La arquitectura *DeepVO* genera errores menores a los 50 metros, siendo la arquitectura *PoseConvGRU* la arquitectura que genera un error por debajo de los 20 metros. Debido a la posición de la cámara, las gráficas dependen en gran medida a las estimaciones de los ejes *X* y *Z.*

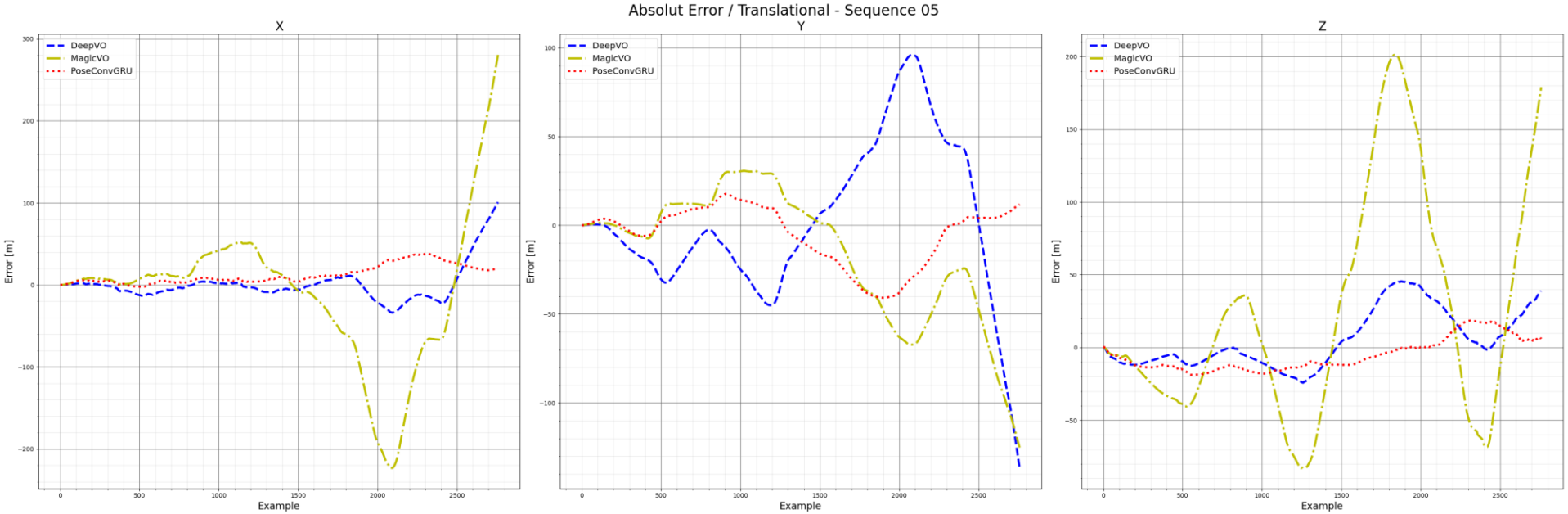

Figura 54. *Error absoluto de las arquitecturas secuencia 05 – Posición*

Como se puede apreciar en Figura *55* el error absoluto de estimación de orientación varía independientemente de la arquitectura, pero se logra observar que:

- Para la estimación de la orientación en el eje *X* (*Roll*) la arquitectura *DeepVO* es la que genera la mayor cantidad de picos de error hasta los $6\,rad$ pero al finalizar la secuencia logra reducir el error a menos de $1\,rad$. La arquitectura *MagicVO* genera errores que llegan hasta los $2\,rad$ a diferencia de 3 picos que llegan hasta los $6\,rad$ pero la arquitectura *PoseConvGRU* tiene el mismo comportamiento que la arquitectura MagicVO a diferencia que al final de la secuencia llega a tener un valor que casi es $0\,rad$.

- Para la estimación de la orientación en el eje *Y* (*Pitch*) la arquitectura *MagicVO* existe oscilaciones que llegan hasta los $1.8\,rad$ aproximadamente a diferencia de la arquitectura *DeepVO* genera errores menores a $0.5\,rad$. Siendo la arquitectura *PoseConvGRU* la arquitectura que genera un error reducido que oscila entre los $0.25\,rad$.

- Para la estimación de la orientación en el eje *Z* (*Yaw*) independiente de las arquitecturas los errores oscilan hasta los $6\,rad$ pero la que tiene mayores fluctuaciones es la arquitectura *DeepVO*. Además, al finalizar la secuencia el error es de $1\,rad$ aproximadamente.

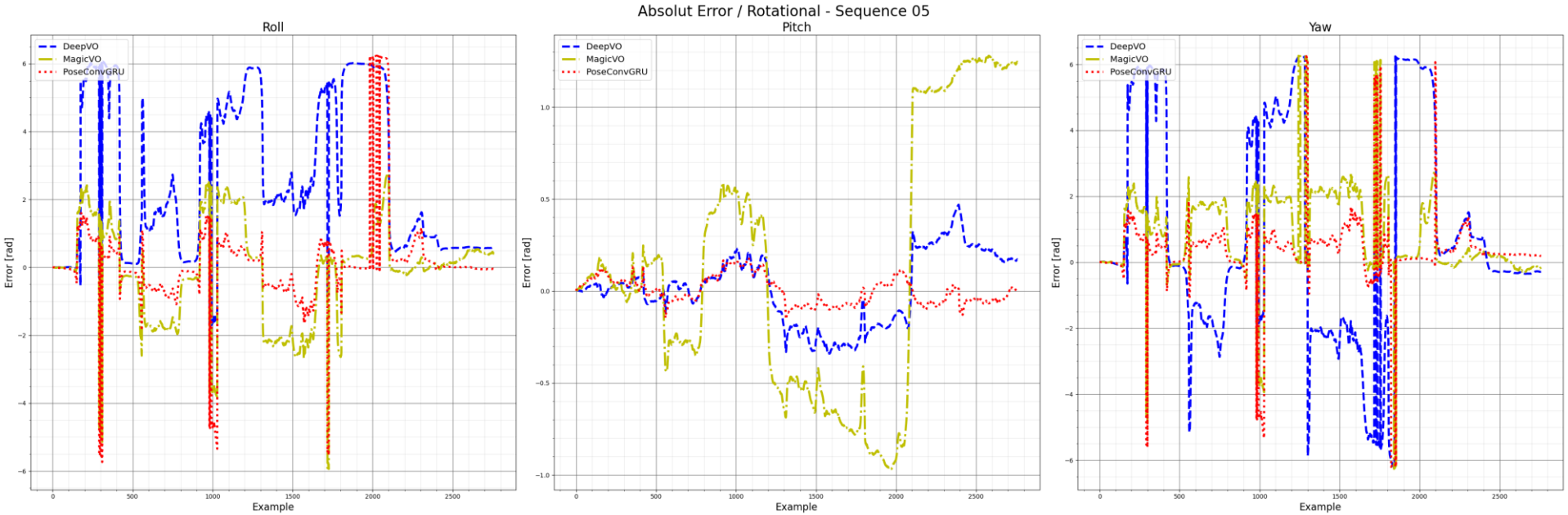

Figura 55. *Error absoluto de las arquitecturas secuencia 05 – Orientación*

Como ultima secuencia de evaluación de las arquitecturas, se utilizó la secuencia 07, donde se observa Figura 56 que:

- Para la estimación de la posición en el eje *X,* el error la arquitectura *DeepVO* es la que genera un menor error que llega hasta los 15 metros aproximadamente, a diferencia que la arquitectura *PoseConvGRU* que llega hasta los 35 metros aproximadamente. Pero la arquitectura *MagicVO* se observa que es la arquitectura que genera un mayor error que llega hasta los 50 metros.

- Para la estimación de la posición en el eje *Y*, se observa que existe oscilaciones siendo la arquitectura *DeepVO* la cual genera un error mayor de hasta 60 metros al finalizar la secuencia, de forma que el error a medida que avanza la secuencia el error va creciendo. A diferencia de las arquitecturas *MagicVO* y *PoseConvGRU* que presenta errores que van hasta los 20 metros siendo la arquitectura *MagicVO* la que cuenta con menor error de 5 metros aproximadamente al finalizar la secuencia.

- Para la estimación de la posición en el eje *Z*, se observa que la arquitectura *MagicVO* es la que genera el mayor error de hasta 80 metros. Las arquitecturas *DeepVO* y *PoseConvGRU* generan errores menores a los 20 metros siendo la arquitectura *PoseConvGRU* la que genera valores mayores al valor real.

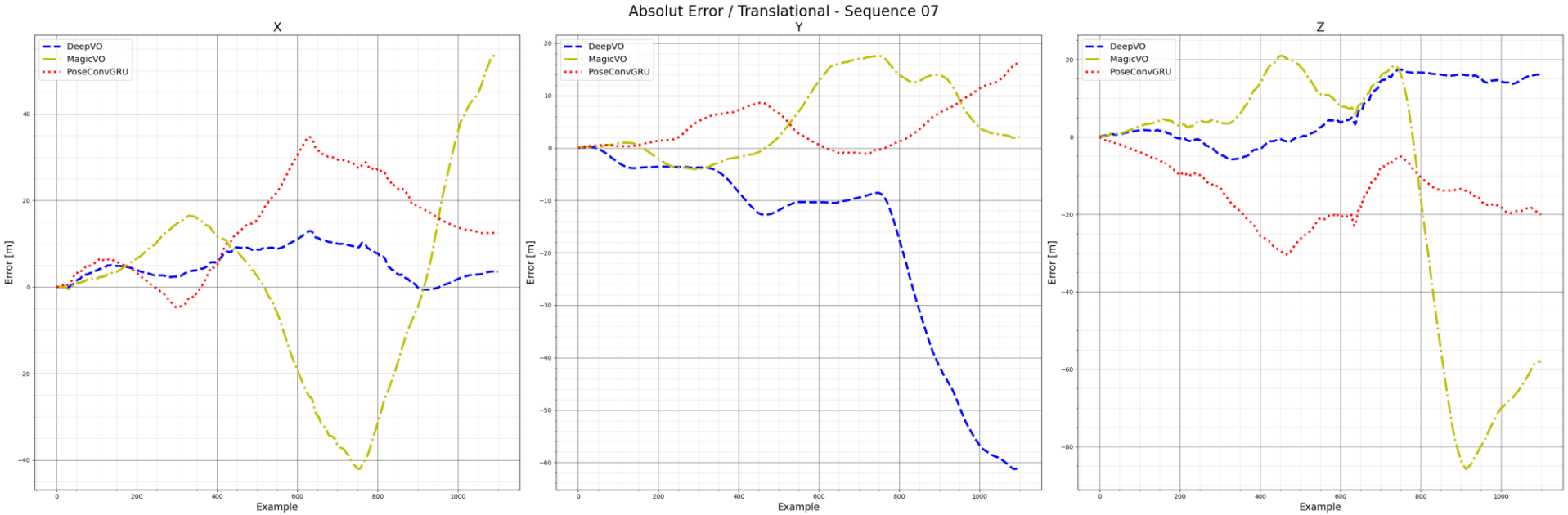

Figura 56. *Error absoluto de las arquitecturas secuencia 07 – Posición*

A continuación, en la Figura 57 se muestra el cálculo del error absoluto de la orientación donde se observa que el error absoluto varía por secciones.

- Para la estimación de la orientación en el eje *X* (*Roll*) la arquitectura *MagicVO* es la que genera la mayor cantidad de picos de error de hasta los $6\,rad$. La arquitectura *DeepVO* genera errores que llegan hasta los $6\,rad$ de igual forma la arquitectura *PoseConvGRU*. Pero al finalizar la secuencia las arquitecturas logran reducir el error a menos de $1\,rad$.

- Para la estimación de la orientación en el eje *Y* (*Pitch*) la arquitectura *MagicVO* existe oscilaciones que llegan hasta los $0.8\,rad$ aproximadamente a diferencia de la arquitectura *DeepVO* genera errores menores a $0.1\,rad$. Siendo la arquitectura *PoseConvGRU* la arquitectura que genera un error reducido que oscila entre los $0.25\,rad$.

- Para la estimación de la orientación en el eje *Z* (*Yaw*) la arquitectura *PoseConvGRU* es la arquitectura que genera picos de $6\,rad$, pero las arquitecturas *DeepVO* y *PoseConvGRU* tienen comportamientos similares. A diferencia que entre los ejemplos 1 hasta 100 y entre los 700 a 750 existe un error que va hasta los $6\,rad$.

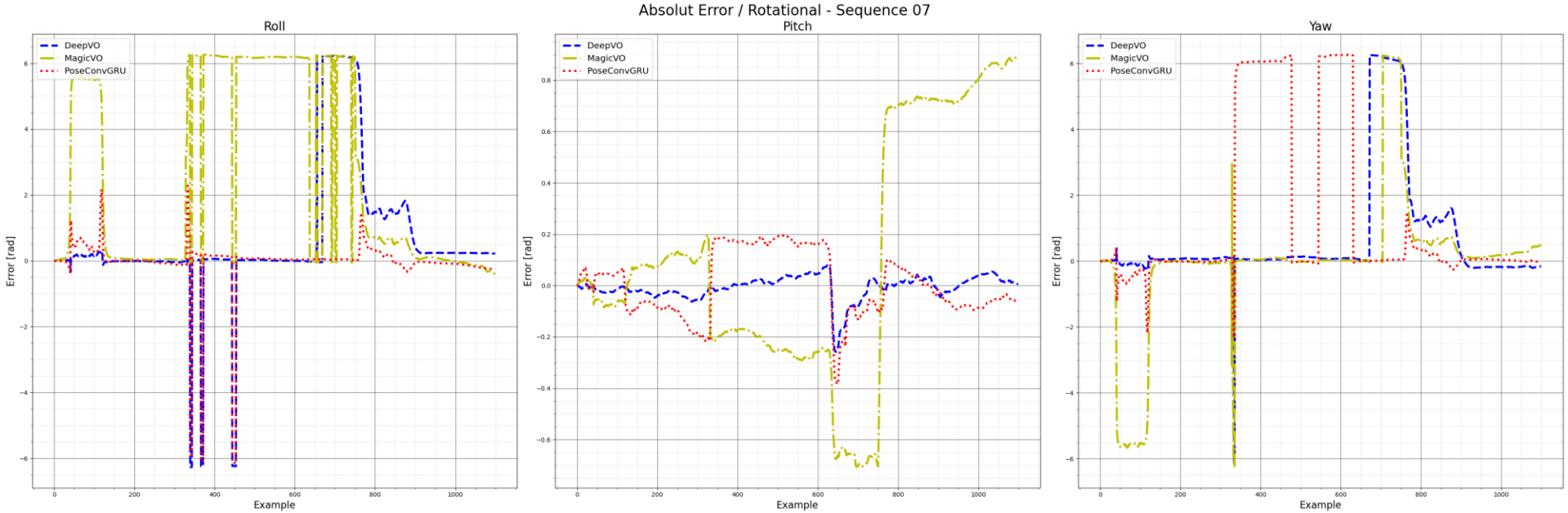

Figura 57. *Error absoluto de las arquitecturas secuencia 07 – Orientación*

A continuación, se muestra una tabla donde se presenta la raíz cuadrada el error cuadrático medio para las secuencias 03, 05 y 07.

Tabla 14. RMSE de las secuencias 03, 05 y 07

Secuencia	Arquitecturas					
	DeepVO		MagicVO		PoseConvGRU	
	Posición	**Orientación**	**Posición**	**Orientación**	**Posición**	**Orientación**
03	26.656219	0.235882	41.357521	0.427208	38.397789	0.399201
05	30.697763	2.809909	71.800529	1.326388	15.675267	1.144061
07	16.534847	1.619257	25.371986	2.540988	14.470999	1.698396
Promedio	24.62961	1.555016	46.17668	1.421528	22.84802	1.080553

Como se puede apreciar en la anterior tabla, tanto la arquitectura *DeepVO* y *PoseConvGRU* presentan un buen desempeño en las secuencias de evaluación, siendo la arquitectura *DeepVO* en la secuencia 03 la que genera menor error tanto en la estimación de posición como en la orientación. En las demás secuencias se observa que *PoseConvGRU* genera un error bajo en comparación con las demás arquitecturas tanto en la estimación de posición como en la de orientación.

En resumen en la Tabla *14*, el cálculo del promedio del *RMSE* de la estimación de posiciones como orientaciones demuestra que la arquitectura *PoseConvGRU* presenta un mejor desempeño que las demás arquitecturas con un *RSME* de 22.84802 en posición y 1.080553 en orientación, la arquitectura que se encuentra siguiente en rendimiento es *DeepVO* con *RSME* de 24.62961 en posición y 1.555016 en orientación, siendo el modelo que presenta un menor rendimiento en estimación es *MagicVO* con *RSME* de 46.17668 en posición y 1.421528 en orientación. Esto puede ser corroborado con las gráficas presentes en la sección 6.2.

8. Conclusiones y trabajo futuro

A continuación, se presenta las conclusiones que han generado el desarrollo del presenta trabajo comparativo de arquitecturas. Se habían planteado una serie de objetivos que se recogen del capítulo 4.

8.1. Conclusiones

Respecto al objetivo específico 1 que se refiere a **Identificar y describir las arquitecturas de aprendizaje profundo disponibles para odometría visual**. Para lo cual se ha analizado en el capítulo 3 de cómo ha ido avanzando los métodos de estimación de posición y orientación a través de métodos analíticos hasta llegar a la utilización de redes neuronales profundas. Se ha realizado un análisis en el capítulo 5.1, donde se observa la capacidad que tienen las redes neuronales recurrentes *LSTM* y su variante *GRU* para las tareas de procesamiento de secuencias de imágenes.

Como es el caso de los modelos seleccionados para el desarrollo del presente trabajo como *DeepVO*, *MagicVO* y *PoseConvGRU*, la utilización *CNN* presenta excelentes resultados en tareas de visión por computadora ya que a través de sus capas convolucionales permite que la red sea capaz de extraer características de las imágenes y más aún con el desarrollo tecnológico que se ha presentado en los últimos años.

Los avances tecnológicos han permitido que los procesos de entrenamiento de *CNN* sean paralelizados con la utilización de tarjetas gráficas. Las *GPU* de *NVIDIA* presentan núcleos *CUDA,* que son aprovechados por los desarrolladores para generar modelos más complejos al añadir una gran cantidad de capas convolucionales. Además, se ha recolectado información del funcionamiento de las redes neuronales *LSTM* y *GRU* tanto en su capacidad de almacenar memoria a largo plazo tanto para valores futuros o pasados.

Respecto al objetivo específico 2 que se refiere a **Identificar los conjuntos de datos disponibles para el entrenamiento y prueba de la odometría visual**. Se ha observado en el capítulo 5.2 que existen algunos conjuntos de datos para odometría visual, pero algunos cuentan con los datos puros de los sensores. Lo cual perjudicaría en el desempeño de las arquitecturas, por lo que el conjunto de datos *KITTI* (Geiger et al., 2012) cuenta con datos preprocesador con extensas secuencias de imágenes de gran calidad, se presentan 11 secuencias con valores de verdad y 10 secuencias de imágenes sin valores de verdad.

Respecto al objetivo específico 3 el cual se refiere a **Definir las métricas de éxito para la odometría visual**. Se ha desarrollado a detalle en el capítulo 5.3, donde se ha definido el dimensionamiento de las imágenes, las secuencias de entrenamiento y evaluación para los diferentes modelos, pérdida durante el entrenamiento de las arquitecturas y los diferentes cálculos de error como el error absoluto y el *RMSE*. Estas métricas se han utilizado en las investigaciones desarrolladas por los autores de las arquitecturas siendo útiles para el presente trabajo comparativo.

En el capítulo 6.2 se ha realizado el desarrollo respecto al objetivo específico 4 que hace referencia a **Implementar las arquitecturas seleccionadas para odometría visual**. Para el análisis se optó por seleccionar las arquitecturas *DeepVO* (Sen et al., 2017), donde se observó que presenta un excelente resultado para la estimación de odometría visual en comparación con los modelos VISO2_M y VISO2_S (Kitt et al., 2010), este modelo consistía en la primera etapa al modelo *FlowNet* seguido de capas *LSTM* y *Dense*. La siguiente arquitectura seleccionada fue MagicVO (Jiao et al., 2018) que obtuvo un mejor rendimiento que *DeepVO* y *ORB_SLAM2* (Mur-Artal & Tardos, 2017), esta arquitectura cuenta con *FlowNet* como primera etapa seguida de capas *Bi-LSTM* y *Dense*. La última arquitectura elegida fue *PoseConvGRU* (Zhai et al., 2019) que cuenta con capas *GRU* y *Dense*.

Una vez elegidos las arquitecturas, conjuntos de datos y las métricas a analizar, se ha implementado las arquitecturas. Se utilizó *TensorFlow* para el desarrollo de las redes neuronales y carga de datos. Esta librería presenta documentación libre y una comunidad activa que permite el desarrollo intuitivo y sin necesidad de crear funciones desde cero. Además, fue muy útil a la hora de compartir recursos durante el entrenamiento de las arquitecturas ya que permitió intercambiar información desde la *GPU* a la *CPU* con pocas líneas de código.

Cabe mencionar que *TensorFlow* permitió la carga dinámica de datos ya se trabajó con imágenes de dimensiones de 640×192, ya que era ineficiente el mantener todas las secuencias en la memoria *RAM*, si no se redimensionaban los valores del conjunto de datos causaban el desbordamiento de memoria.

Además, se ha empleado el modelo pre entrenado *FlowNet* para la estimación de flujo óptico de un par de imágenes secuenciales. La utilización de transferencia de aprendizaje ayudó a que las arquitecturas no sean entrenadas desde cero, sino solo las capas que se encuentran seguidas de las capas convolucionales.

En el capítulo 6.4 se hace referencia al desarrollo del objetivo específico 5 que se refiere a **Calcular y comparar el tiempo de entrenamiento entre las diferentes arquitecturas para**

un mismo conjunto de datos. Donde se demostró que la utilización de capas *LSTM* con varios estados ocultos hacían que el entrenamiento de las redes neuronales sea lento. Ya que no se pueden paralelizar como es el caso de las *CNN*. Adicional se observó que el modelo que tuvo un entrenamiento relativamente corto en las 200 épocas seleccionadas fue *PoseConvGRU*. Por contar con una cantidad reducida de parámetros en comparación de *MagicVO* que es el modelo con mayor número de parámetros a entrenar.

MagicVO tardó 67 horas en cumplir las 200 épocas de entrenamiento, esto quizá se deba a las limitaciones presentadas en el equipo. Además, la arquitectura *PoseConvGRU* fue la que logró alcanzar un valor mínimo de 0.0006612031 en su valor de perdida, siendo el menor valor en comparación de las otras arquitecturas.

De acuerdo con el objetivo específico 6 que se refiere a **Evaluar las diferentes arquitecturas con diferentes conjuntos de datos**. Se realizó el análisis en las secciones 6.5 y 6.6, donde se observó la capacidad de las arquitecturas para estimar los valores de posición y orientación. Se realizaron gráficas de los recorridos de las secuencias 03, 05 y 07 del conjunto de datos *KITTI*. En las gráficas se observa cómo la estimación de la posición y orientación genera un error, el cual se va propagando en el transcurso del recorrido. siendo la arquitectura PoseConvGRU logra estimar valores mas cercanos a las posiciones reales.

Además, se realizó el tiempo que tarda las diferentes arquitecturas en estimar las posiciones y orientaciones. Siendo la arquitectura *PoseConvGRU* la que menor tiempo tarde en realizar la estimación, con un tiempo aproximado de 0.0936 segundos. En comparación de las demás arquitecturas *DeepVO* y *MagicVO* con ~ 0.1489 y ~ 0.1708 respectivamente.

Referente al objetivo específico 7 que se refiere a **Comparar el resultado de estimación de la odometría visual de las diferentes arquitecturas**. En el capítulo 7 se observó los diferentes valores del error absoluto que presentaron las diferentes arquitecturas con respecto a los valores de verdad, donde el error de las posiciones se va propagando a través que avanza la secuencia de imágenes, esto se debe a que las posiciones son consecuentes de las estimaciones pasadas y por ende repercute en el cálculo de la siguiente posición ya que interviene tanto la posición anterior como la orientación.

Por ende, se ha determinado que la mejor arquitectura para estimación de la odometría visual es la arquitectura *PoseConvGRU* ya que es un modelo que no emplea una gran cantidad de parámetros o capas muy cargadas de parámetros para lograr lo esperado con un *RMSE* de 24.62961 en posición y 1.555016 en rotación, si bien las demás arquitecturas pueden mejorar con un reentrenamiento por un tiempo más prolongado o ser reentrenadas con otros conjuntos

de datos, con los parámetros de entrenamiento seleccionados no ha surtido mucho efecto en la mejora de la estimación de los valores de posición y orientación.

Como se observó, el emplear una gran cantidad de capas en las redes neuronales no siempre es lo adecuado, ya que en cierto modo repercute en el desempeño y genera un mayor esfuerzo computacional, esto puede ser comprobado en la etapa de entrenamiento de la arquitectura *MagicVO*, que tardo casi tres días en completar las 200 épocas y aun así no lograr superar la precisión de las otras dos arquitecturas.

Y por último de acuerdo con el cumplimiento de los diferentes objetivos específicos se logró llegar a cumplir el objetivo general de **realizar un estudio comparativo de las diferentes arquitecturas de aprendizaje profundo para la estimación de la odometría visual**. Se ha demostrado que durante el tiempo que tarda la arquitectura *PoseConvGRU* en estimar las posiciones relativas es menor que el resto de las arquitecturas, haciéndole la arquitectura no solo con mejor capacidad de estimar los valores de posición y orientación sino también en que se puede utilizar quizá en aplicaciones en tiempo real con la limitación de ser utilizada en computadoras de escritorio.

Adicional, al igual que los autores de las arquitecturas estudiadas, se pretende que estos modelos de redes neuronales profundas sean de soporte para los métodos analíticos existentes, ya que han presentado métodos capaces de producir excelentes resultados dando por solucionado el problema de odometría visual. Por lo que existen métodos o modelos que no generen mucha carga computacional y sean lo suficientemente precisos para emplearlos en sistemas con poca cantidad de recursos de procesamiento.

8.2. Líneas de trabajo futuro

- Una vez desarrollada la comparativa el código implementado se encuentra de manera libre con sus respectivos parámetros pre entrenados en el repositorio *GitHub*[2]. Pueden ser utilizados para continuar con el entrenamiento o evaluación de las arquitecturas. Así como su utilización en diferentes aplicaciones de robótica donde se desee estimar la posición del robot a partir de secuencias de imágenes.

- Si bien se ha utilizado un computador de escritorio para lograr la inferencia de los modelos. Se espera que en un futuro con los avances tecnológicos se pueda implementar en los sistemas embebidos de *NVIDIA JetsonNano* ya que cuenta con

[2] Repositorio https://github.com/EduardoTayupanta/VisualOdometry

núcleos *CUDA* en menor cantidad que las tarjetas gráficas. Si bien hoy en día se puede compilar *TensorFlow* aun no puede lograr procesar un gran número de parámetros.

9. Bibliografía

Agrawal, P., Carreira, J., & Malik, J. (2015). *Learning to See by Moving*.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2018). Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-January*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., & Asari, V. K. (2018). *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. http://arxiv.org/abs/1803.01164

Amidi, A., & Amidi, S. (2017). *CS 230 - Recurrent Neural Networks Cheatsheet*. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Anwar, A. (2019). *Difference between AlexNet, VGGNet, ResNet, and Inception | by Aqeel Anwar | Towards Data Science*. https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

Borenstein, J, Everett, H. R., Feng, L., Lee, S. W., Byrne, R. H., Borenstein, J., & Feng, L. (1996). *Where am I? Sensors and Methods for Mobile Robot Positioning Prepared by the University of Michigan For the Oak Ridge National Lab (ORNL) D&D Program and the United States Department of Energy's Robotics Technology Development Program Within the Environmental Restoration, Decontamination and Dismantlement Project*.

Borenstein, Johann, & Feng, L. (1995). *UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots*.

Brunetti, A., Buongiorno, D., Trotta, G. F., & Bevilacqua, V. (2018). Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. *Neurocomputing*, *300*, 17–33. https://doi.org/10.1016/j.neucom.2018.01.092

Chen, C., Lu, X., Wahlstrom, J., Markham, A., & Trigoni, N. (2019). Deep Neural Network Based Inertial Odometry Using Low-cost Inertial Measurement Units. *IEEE Transactions on Mobile Computing*, 1–1. https://doi.org/10.1109/tmc.2019.2960780

Chen, C., Zhao, P., Lu, C. X., Wang, W., Markham, A., & Trigoni, N. (2018). *OxIOD: The Dataset for Deep Inertial Odometry*. http://arxiv.org/abs/1809.07491

Chung, C., Patel, S., Lee, R., Fu, L., Reilly, S., Ho, T., Lionetti, J., George, M. D., & Taylor, P.

(2018). Vggnet. *American Journal of Health-System Pharmacy*, *75*(6), 398–406. https://doi.org/10.2146/ajhp170251

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. http://arxiv.org/abs/1412.3555

Costante, G., Mancini, M., Valigi, P., & Ciarfuglia, T. A. (2016). Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation. *IEEE Robotics and Automation Letters*, *1*(1), 18–25. https://doi.org/10.1109/LRA.2015.2505717

Dawson-Howe, K. M., & Vernon, D. (1994). Simple pinhole camera calibration. *International Journal of Imaging Systems and Technology*, *5*(1), 1–6. https://doi.org/10.1002/ima.1850050102

Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazırbas¸, H., Golkov, V., Van Der Smagt, P., Cremers, D., & Brox, T. (2015). *FlowNet: Learning Optical Flow with Convolutional Networks*.

Du, F., & Brady, M. (1993). Self-calibration of the intrinsic parameters of cameras for active vision systems. *IEEE Computer Vision and Pattern Recognition*, 477–482. https://doi.org/10.1109/cvpr.1993.341087

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robotics and Automation Magazine*, *13*(2), 99–108. https://doi.org/10.1109/MRA.2006.1638022

Engel, J. (2013). *Semi-Dense Visual Odometry for a Monocular Camera * 2013 IEEE International Conference on Computer Vision*. 1449–1456. https://doi.org/10.1109/ICCV.2013.183

Engineering, D. of M. and P. (2011). *ASL Datasets*. https://projects.asl.ethz.ch/datasets/doku.php?id=home

Engineering, D. of M. and P. (2020). *Homepage - D-MAVT – Department of Mechanical and Process Engineering | ETH Zurich*. https://mavt.ethz.ch/

Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T., & Philbrick, K. (2017). Toolkits and Libraries for Deep Learning. In *Journal of Digital Imaging* (Vol. 30, Issue 4, pp. 400–405). Springer New York LLC. https://doi.org/10.1007/s10278-017-9965-6

Forster, C., Pizzoli, M., & Scaramuzza, D. (2014). SVO: Fast semi-direct monocular visual

odometry. *Proceedings - IEEE International Conference on Robotics and Automation*, 15–22. https://doi.org/10.1109/ICRA.2014.6906584

Fu-chao, W., Fu-chao, W., Guang-hui, W., & Zhan-yi, H. (2003). *A Linear Approach for Determining Intrinsic Parameters and Pose of Cameras from Rectangles*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.3475

Geiger, A., Urtasun, P. L., & Raquel, U. (2012). *The KITTI Vision Benchmark Suite*. http://www.cvlibs.net/datasets/kitti/

Google. (2020). *Descending into ML: Training and Loss | Machine Learning Crash Course*. https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

Grandón-Pastén, N., Aracena-Pizarro, D., & Tozzi, C. L. (2007). RECONSTRUCCIÓN DE OBJETO 3D A PARTIR DE IMÁGENES CALIBRADAS. *Ingeniare. Revista Chilena de Ingeniería*, *15*(2), 158–168. https://doi.org/10.4067/S0718-33052007000200006

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5–6), 602–610. https://doi.org/10.1016/j.neunet.2005.06.042

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. *International Conference in Machine Learning*, *4*(7540), 14. http://arxiv.org/abs/1502.04623

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, *187*, 27–48. https://doi.org/10.1016/j.neucom.2015.09.116

Handa, A., Bloesch, M., Pătrăucean, V., Stent, S., McCormac, J., & Davison, A. (2016). Gvnn: Neural network library for geometric computer vision. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9915 LNCS*, 67–82. https://doi.org/10.1007/978-3-319-49409-8_9

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hecht-Nielsen, R. (1989). Neurocomputer Applications. In *Neural Computers* (pp. 445–453).

Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-83740-1_45

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hochreiter, S., Younger, A. S., & Conwell, P. R. (2001). Learning to learn using gradient descent. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2130*, 87–94. https://doi.org/10.1007/3-540-44668-0_13

Howard, A. (2008). Real-time stereo visual odometry for autonomous ground vehicles. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 3946–3952. https://doi.org/10.1109/IROS.2008.4651147

Huang, S., Wang, Z., Dissanayake, G., & Frese, U. (2009). *Iterated SLSJF: A Sparse Local Submap Joining Algorithm with Improved Consistency*. https://pdfs.semanticscholar.org/6e41/e7840bf378709098b624ed299d637c6fc63e.pdf

Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., & Brox, T. (2017). *FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks*. http://lmb.

InvenSense. (2016). *ICM-20600*. www.invensense.com

Jeong, W. Y., & Lee, K. M. (2006). Visual SLAM with line and corner features. *IEEE International Conference on Intelligent Robots and Systems*, 2570–2575. https://doi.org/10.1109/IROS.2006.281708

Jiao, J., Jiao, J., Mo, Y., Liu, W., & Deng, Z. (2018). *MagicVO: End-to-End Monocular Visual Odometry through Deep Bi-directional Recurrent Convolutional Neural Network*. http://arxiv.org/abs/1811.10964

Jordan, J. (2017). *Convolutional neural networks.* https://www.jeremyjordan.me/convolutional-neural-networks/

Kitt, B., Geiger, A., & Lategahn, H. (2010). *Visual Odometry based on Stereo Image Sequences with RANSAC-based Outlier Rejection Scheme*.

Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*. https://doi.org/10.1109/ISMAR.2007.4538852

Konda, K., & Memisevic, R. (2005). *Learning Visual Odometry with a Convolutional Network*. https://doi.org/10.5220/0005299304860490

Kreuzig, R., Ochs, M., & Mester, R. (2019). *DistanceNet: Estimating Traveled Distance from Monocular Images using a Recurrent Convolutional Neural Network*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. http://code.google.com/p/cuda-convnet/

Li, M., Hong, B., Cai, Z., & Luo, R. (2008). *Novel Rao-Blackwellized Particle Filter for Mobile Robot SLAM Using Monocular Vision*.

Li, R., Wang, S., Long, Z., & Gu, D. (2018). UnDeepVO: Monocular Visual Odometry Through Unsupervised Deep Learning. *Proceedings - IEEE International Conference on Robotics and Automation*, 7286–7291. https://doi.org/10.1109/ICRA.2018.8461251

Maddern, W., Pascoe, G., Gadd, M., Barnes, D., Yeomans, B., & Newman, P. (2015). *Real-time Kinematic Ground Truth for the Oxford RobotCar Dataset*. https://www.novatel.com/products/span-gnss-inertial-systems/

Maddern, W., Pascoe, G., Linegar, C., & Newman, P. (2017). 1 year, 1000 km: The Oxford RobotCar dataset. *The International Journal of Robotics Research*, *36*(1), 3–15. https://doi.org/10.1177/0278364916679498

Martinez-Cantin, R., & Castellanos, J. A. (2005). Unscented SLAM for large-scale outdoor environments. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 328–333. https://doi.org/10.1109/IROS.2005.1545002

Muller, P. M. (2016). *Optical Flow and Deep Learning Based Approach to Visual Optical Flow and Deep Learning Based Approach to Visual Odometry Odometry*. https://scholarworks.rit.edu/theses

Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, *31*(5), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671

Mur-Artal, R., & Tardos, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, *33*(5), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103

Newcombe, R. A., Lovegrove, S. J., & Davison, A. J. (2011). DTAM: Dense tracking and

mapping in real-time. *Proceedings of the IEEE International Conference on Computer Vision*, 2320–2327. https://doi.org/10.1109/ICCV.2011.6126513

Nistér, D., Naroditsky, O., & Bergen, J. (2004). Visual odometry. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *1*. https://doi.org/10.1109/cvpr.2004.1315094

NovAtel. (2017). *SPAN CPT7 Performance Specifications*. https://docs.novatel.com/OEM7/Content/Technical_Specs_Receiver/SPANCPT7_Performance_Specs.htm

Olah, C. (2015). *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

OpenCV. (2017). *OpenCV: Optical Flow*. https://docs.opencv.org/3.4.0/d7/d8b/tutorial_py_lucas_kanade.html

OXTS. (2013). *RT3000 v3 - GNSS-aided inertial navigation system for automotive testing*. https://www.oxts.com/products/rt3000/

Perez, J., Serrano, C., Acha, B., Serrano, T., & Linares, B. (2013). *Red neuronal convolucional rápida sin fotogramas para reconocimientos de dígitos*. http://digital.csic.es/handle/10261/84753

Qin, T., Li, P., & Shen, S. (2018). VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, *34*(4), 1004–1020. https://doi.org/10.1109/TRO.2018.2853729

Saffiotti, A. (1997). The uses of fuzzy logic in autonomous robot navigation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, *1*(4), 180–197. https://doi.org/10.1007/s005000050020

Sak, H., Senior, A., & Beaufays, F. (2014). *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. http://arxiv.org/abs/1402.1128

Scaramuzza, D., & Fraundorfer, F. (2011). Tutorial: Visual odometry. *IEEE Robotics and Automation Magazine*, *18*(4), 80–92. https://doi.org/10.1109/MRA.2011.943233

Sen, Ronald, Hongkai, & Niki. (2017). DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. *Proceedings - IEEE International*

Conference on Robotics and Automation, 2043–2050. https://doi.org/10.1109/ICRA.2017.7989236

Singh, B., Marks, T. K., Jones, M., Tuzel, O., & Shao, M. (2016). *A Multi-Stream Bi-Directional Recurrent Neural Network for Fine-Grained Action Detection*.

Steffen, L., Alec, G., Thomas, F., & Kevin, M. (2017). *Visual Odometry using Convolutional Neural Networks*. https://www.researchgate.net/publication/322525765_Visual_Odometry_using_Convolutional_Neural_Networks

Strobl, K. H., & Hirzinger, G. (2011). More accurate pinhole camera calibration with imperfect planar target. *Proceedings of the IEEE International Conference on Computer Vision*, 1068–1075. https://doi.org/10.1109/ICCVW.2011.6130369

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June-2015*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Venegas, J. (2009). *Encoders*. http://support.automationdirect.com/docs/absolute_encoders.pdf

Wang, R., Pizer, S. M., & Frahm, J.-M. (2019). *Recurrent Neural Network for (Un-)supervised Learning of Monocular Video Visual Odometry and Depth*.

Watson, J. D. (2020). *AFIT Scholar AFIT Scholar Improved Ground-Based Monocular Visual Odometry Estimation Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks using Inertially-Aided Convolutional Neural Networks*. https://scholar.afit.edu/etd/3622

Wu, N. (2017). *Tensorflow detection model zoo*. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Zhai, G., Liu, L., Zhang, L., & Liu, Y. (2019). PoseConvGRU: A Monocular Approach for Visual Ego-motion Estimation by Learning. *Pattern Recognition*, *102*. http://arxiv.org/abs/1906.08095

Zhang, Y. (2006). Towards piecewise-linear primal neural networks for optimization and

redundant robotics. *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, ICNSC'06*, 374–379. https://doi.org/10.1109/icnsc.2006.1673175

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(11), 1330–1334. https://doi.org/10.1109/34.888718

Anexos

Anexo. Artículo Análisis comparativo de arquitecturas de aprendizaje profundo para odometría visual

Análisis comparativo de arquitecturas de aprendizaje profundo para odometría visual

Eduardo Tayupanta Zúñiga

Universidad Internacional de la Rioja, Logroño (España)

Septiembre 2020

Resumen

Una de las principales líneas de investigación en las áreas de la robótica, es el desarrollo de técnicas que permitan que los robots sean capaces de desplazarse autónomamente en diferentes entornos. Con el fin de que el robot obtenga los valores de posición y orientación se han empleado técnicas de odometría visual, que permiten la estimación de posición y orientación a través de secuencias de imágenes. En el presente estudio se ha realizado la implementación, entrenamiento y comparativa de rendimiento de tres diferentes arquitecturas de odometría visual que emplean modelos de redes neuronales *CNN* y su combinación con *LSTM* o *GRU*. Se ha demostrado que la utilización del aprendizaje profundo en la odometría visual puede llegar a estimar valores cercanos a los valores reales de las secuencias de imágenes, por consiguiente, los resultados son competitivos con los métodos analíticos desarrollados para la estimación de posición y orientación.

Palabras Clave

Aprendizaje profundo, GRU, LSTM, SLAM.

I. Introducción

La navegación autónoma de un agente ha sido un área de investigación en la robótica, la cual busca que el robot sea capaz de desplazarse en entornos del mundo real. Se han desarrollado diferentes métodos desde modelos analíticos hasta modelos de aprendizaje automático para la estimación de posición y orientación del robot. Los estudios de odometría visual utilizando modelos de aprendizaje profundo permiten la estimación de posición y orientación a partir de una secuencia de imágenes. Si bien existen modelos analíticos que pueden solventar el problema de la odometría visual, se observa que los modelos de aprendizaje profundo pueden tener resultados competitivos sin tener una gran cantidad de parámetros para su procesamiento, el cual depende en gran medida de las geometrías que presentan las cámaras con las que fueron capturadas las secuencias de imágenes.

Los modelos analíticos necesitan las propiedades intrínsecas de las cámaras, así como cierto tipo de condiciones al capturar las imágenes, tales como: la resolución, la cantidad de luz y la variación en contraste, entre otras. Por consiguiente, los modelos de redes neuronales han demostrado tener la capacidad de extraer características durante su entrenamiento, de esta manera resultan atractivos para la estimación de posición y orientación a partir de secuencias de imágenes [1].

Las redes neuronales convolucionales fueron desarrolladas a finales de los 90, pero en los últimos años han tenido un gran éxito en tareas de procesamiento de imágenes, ya que pueden extraer características concretas de estas. Su desarrollo es más complejo al utilizar una gran cantidad de capas convolucionales en cascada, de esta forma, en cada capa convolucional la red neuronal es capaz de aprender cierto tipo de características [61].

Las redes neuronales recurrentes pueden extraer información a través de series temporales, es decir, permiten tratar la dimensión del tiempo. Estos modelos fueron desarrollados en la década de los 80, en sus inicios, eran difíciles de tratar ya que necesitaban un gran esfuerzo computacional. Con los avances tecnológicos durante los últimos años se han vuelto más accesibles y su popularidad ha ido creciendo [9].

Las redes neuronales *LSTM* (Long Short Term Memory) son una extensión de las redes neuronales recurrentes, logran ampliar la memoria para aprender a través de sucesos que han transcurrido durante el tiempo. Este tipo de modelos mantienen información que puede ser modificada dependiendo de la importancia en la que es asignada la información [64]. Otra variante de las redes neuronales recurrentes son las *GRU* (*Gated Recurrent Unit*), que aparecieron en el año 2014 y conservan el mismo principio de las *LSTM*, con la diferencia de que son computacionalmente más eficientes que las *LSTM* [12].

Una vez analizados los estudios previos para solventar el problema de odometría visual, se presenta el estudio comparativo de tres tipos de arquitecturas que emplean *CNN*, *LSTM* y *GRU*. Cada arquitectura fue implementada en *Python* con la librería TensorFlow, debido a que no se cuenta con el código fuente desarrollado por los investigadores. El lenguaje de programación *Python* se caracteriza por ser versátil y práctico. Además, la librería *TensorFlow* permite implementar arquitecturas de redes neuronales sin necesidad de utilizar numerosas líneas de código.

II. Estado del Arte

Las principales tendencias de investigación en la robótica es el desarrollo de técnicas para navegación autónoma en ambientes del mundo real [63]. Para que un robot sea completamente

autónomo debe ser capaz de movilizarse en un entorno desconocido al mismo tiempo que construye un mapa incremental y estima su posición Figura 1, la aplicación de técnicas de localización y mapeo simultáneo (*SLAM Simultaneous Localization and Mapping*) proporcionan los medios para que un robot móvil sea realmente autónomo [17].

Fig. 1. ORB-SLAM 2 [56].

Las mejoras de rendimiento de aplicaciones desarrolladas por visión por computadora, la combinación hardware especializado como cámaras estéreo, cámaras *RGB-D* y sensores capaces de medir distancia desde el emisor al objeto, han hecho posible la implementación de procesos en tiempo real proporcionando estimaciones de profundidad de cada píxel [36].

Métodos basados en características dispersas que se enfoca en la visión estereotípica, presentan un gran inconveniente en la precisión al no incluir la transición de tiempo en cada cuadro. Para superar el inconveniente, se desarrolló una alternativa llamada visual *SLAM* que optimiza con precisión de manera continua el mapa de características al aplicar un filtro de Kalman extendido [39], pero debido a la complejidad computacional y a la dependencia de sincronización de cuadros emparejados genera errores en la estimación de movimiento [51].

El SLAM monocular mejora la incertidumbre al emplear un filtro de partículas, debido a su alta complejidad computacional lo que genera inconvenientes en aplicaciones en tiempo real [47] Los modelos basados en *PTAM* (seguimiento y mapeo paralelo) mejora notablemente el tiempo real, el modelo utiliza la extracción de fotogramas clave [43].

Los métodos en base a unión de sub-mapas locales han mejorado la agilidad de un sistema *SLAM*, se emplean filtros de información dispersos capaces de reducir el coste computacional para la construcción de mapas globales, el método es resistente al desenfoque causado por el movimiento rápido de la captura de las imágenes [37]. Un último enfoque desarrolla *SLAM* a base de cámara monoculares, los autores centran su investigación en mapeo, localización y cierre de bucles. Utiliza funciones *ORB* y describe un nuevo concepto Essential Graph que acelera el proceso de correcciones de cierre de bucle (detección con corrección). Puede ser ejecutado en *CPU* teniendo un gran desempeño en tiempo real [53].

A continuación, se presentan diferentes métodos para el cálculo de la odometría visual como el estudio *Real-time stereo visual odometry for autonomous ground vehicles* [35] que describe un algoritmo de odometría visual a partir de pares consecutivos de imágenes estéreo, no realiza suposiciones del movimiento de la cámara y opera con imágenes de disparidad densa, el error en la estimación de posición es del 0.25% del total de la distancia recorrida utilizando imágenes de 512×384 de dimensión.

Para el método de extracción de características geométricas de las imágenes monoculares, se aplica un riguroso procesamiento matemático, para así obtener la posición a partir de la odometría visual [40]. Sin embargo, las cámaras presentan propiedades internas llamados parámetros intrínsecos que son valores que derivan de la geometría epipolar [16].

Los métodos directos para odometría visual, no emplean cálculos para determinar descriptores o puntos clave en las imágenes siendo su principal objetivo el estimar movimiento a partir de la iluminación que presenta el píxel, los métodos suponen la invarianza de la imagen en escala de grises lo que provee mayor robustez y precisión.

Métodos basados en cámara monoculares realizan cálculos para estimar la postura de la cámara utilizando un mapa de profundidad semidenso, se puede ejecutar en tiempo real en una *CPU*, pero genera incertidumbre de posición al no emplear detección de cierre de bucle [18]. Otro método se basa en la predicción probabilística de la profundidad de cada píxel es sensible a cambios de iluminación, pero reduce la incertidumbre en la estimación de la posición debido a que no se basa en la uniformidad de intensidad del píxel [55].

El método de odometría visual monocular semidirecto elimina la costosa tarea de extraer características y coincidencias entre imágenes para estimación de movimiento, extrae bloques en la imagen mejorando la solidez, implementa métodos probabilísticos de mapeo para la estimación de puntos 3D y el alto procesamiento de 55 cuadros por segundo para un sistema a bordo y a 300 cuadros por segundo para una computadora de escritorio, a demostrado ser robusto en escenarios con alta frecuencia y poca textura repetitiva [22].

Al ser un método rápido logra tener éxito cuando la cámara se mueve despacio ya que la búsqueda del gradiente puede dar como resultado que el algoritmo falle, de modo que el método directo es sensible al cambio de iluminación. Los problemas de los métodos convencionales para estimar la posición y orientación a partir de imágenes consecutivas como: extracción de características de la imagen, alto costo computacional, presentan sensibilidad a la iluminación, se requiere un alto grado de texturas y un elevado número de cuadros por segundo [40].

Actualmente el aprendizaje profundo ha sido ampliamente desarrollado en el área de visión por computadora, la aplicación de redes neuronales convolucionales ha generado varias aplicaciones con cámaras monoculares como, por ejemplo: la clasificación de imágenes, detección de objetos, reconocimiento facial, entre otras. A pesar del gran éxito que ha tenido el uso del aprendizaje profundo con imágenes, son pocas las alternativas desarrolladas para solventar los inconvenientes presentados con los métodos tradicionales de la odometría visual [68].

Aprendizaje profundo

Un subconjunto de algoritmos de aprendizaje automático es el aprendizaje profundo. El aprendizaje profundo son algoritmos que permiten procesar grandes conjuntos de datos para resolver de manera eficiente los problemas tradicionales de inteligencia artificial. Las recientes mejoras en hardware han fortalecido el estudio de diferentes arquitecturas de redes neuronales, permitiendo un menor coste computacional y reduciendo el tiempo de procesamiento. Han tenido éxito en transferencia de aprendizaje, estimación de profundidad, odometría visual, visión por computadora, sistemas de control, robótica móvil, entre otras [29].

El aprendizaje profundo ha sido ampliamente utilizado en el

área de visión por computadora ya que emplean redes neuronales convolucionales para la detección y extracción de patrones en imágenes [8], gracias al cómputo paralelo ha fortalecido la aplicación de redes convolucionales y las redes neuronales recurrentes motivando soluciones en tiempo real para problemas de la robótica móvil [76].

Red neuronal convolucional

Las redes neuronales convolucionales han demostrado ser capaces de solventar varios retos de visión por computadora, ya que son capaces de superar a las máquinas de aprendizaje automático. Una red neuronal convolucional es un tipo de red neuronal artificial para el procesamiento de imágenes ya que puede procesar datos por píxel.

El término convolución se refiere a la operación matricial de aplicar filtros que recorren la imagen [68].

La Figura 2 muestra una capa convolucional, se puede visualizar la capa como pequeñas cuadriculas llamados *kernels* que recorren la imagen en búsqueda de patrones, si el píxel coincide con los patrones del *kernel* el resultado es un valor positivo y si no coincide de vuelve un valor cercano a cero [2]. Un parámetro principal para las capas convolucionales es el uso de *stride*, que se refiere al salto del *kernel* entre píxeles.

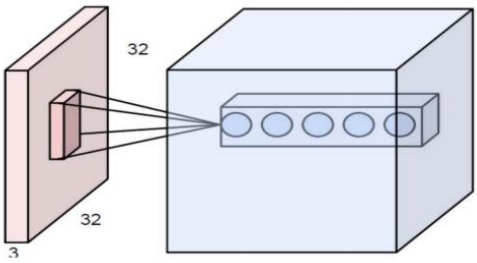

Fig. 2. Operación de convolución en CNN [2].

Las CNN están constituidas por múltiples capas convolucionales, capas no lineales, capas de agrupación y capas completamente conectadas [3], que han tenido mucho éxito para la extracción de características, reconocimiento de objetos sin tomar en cuenta las condiciones de iluminación gracias a las mejoras de velocidad de procesamiento ya que el cómputo se puede paralelizar gracias a la utilización de hardware especializado [46]

En la actualidad existen arquitecturas de redes neuronales convolucionales complejas que han sido entrenadas con una gran cantidad de datos para aplicaciones específicas, se puede utilizar modelos pre entrenados para el reconocimiento de objetos como *ResNet* [31], *SENet* , etc., o como los modelos de detección como *SDD Mobilenet, SDD Inception*, entre otras [74]. Ya que la odometría visual está más relacionada con la geometría, arquitecturas de *CNN* como *VGGNet* [12] y *GoogleNet* [70] no pueden ser empleadas para este propósito.

Red neuronal recurrente

Las redes neuronales recurrentes *RNN* han sido desarrolladas desde la década de 1990, su diseño permite el aprendizaje de patrones de tiempo variable o secuenciales, con un sistema de retroalimentación con conexiones de bucle cerrado, permite que las salidas previas sean utilizadas como entradas mientras se mantengan los estados ocultos como se muestra en la Figura 3 [32].

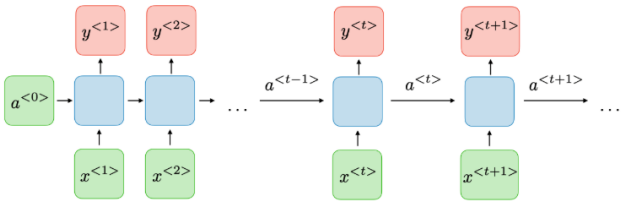

Fig. 3. Arquitectura de una RNN [6].

Donde para cada paso en el tiempo t existe una activación $a^{<t>}$ que produce una salida $y^{<t>}$, se expresa esta transición de estados de acuerdo con las siguientes ecuaciones.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Donde $W_{aa}, W_{ax}, W_{ya}, b_a$ y b_y se refiere a los coeficientes que son compartidos temporalmente y g_1, g_2 se refiere a las funciones de activación dentro de cada estado [64].

La aplicación de redes neuronales recurrentes ha demostrado superar significativamente a modelos como las redes neuronales convolucionales tradicionales. Se presenta mejoras en precisión, la entrada al modelo puede ser de cualquier longitud, el tamaño del modelo no depende de la longitud de la entrada, para las estimaciones cuenta con información histórica lo que permite que los pesos sean compartidos a lo largo del tiempo. Pero aún existe problemas en la implementación por su gran coste computacional, no se puede acceder a información que ha sido procesada hace mucho tiempo [66] aplicaciones como la generación de imágenes emplea *RNN* presentando un marco de auto codificación secuencias para construir iterativamente imágenes complejas [28].

LSTM

La arquitectura *LSTM* [33] fue desarrollada para mejorar el error existente en las *RNN*. Se demostró que los retrasos presentes en la arquitectura eran inaccesibles debido a que el error al retro propagarse decae o explota exponencialmente [34].

Las capas *LSTM* es encuentra formada por múltiples bloques de memoria conectados. Estos bloques contienen varias celdas de memoria conectada de forma recurrente y tres unidades multiplicativas denominadas *input*, *output* y *forget gate* que permiten operaciones continuas de lectura, escritura y reinicio de la celda de memoria, de forma que la input de cada celda es multiplicada por la unidad de activación de la puerta de entrada, el output es multiplicada por la puerta de salida y los valores correspondientes a las celdas anteriores son multiplicadas por la *forget gate* [27].

Las *LSTM* presentan la misma estructura interna de las *RNN*, pero con la variación de que el módulo de repetición presenta 4 capas que interactúan. De las cuales 3 permiten el control del estado, como se muestra en la Figura 4.

Por ejemplo, dada una entrada x_k en el tiempo k, con el estado oculto h_{k-1} en la celda de memoria c_{k-1} de la celda *LSTM* anterior, la *LSTM* se actualiza según la siguiente formula:

$$i_k = \sigma(W_{xi}x_k + W_{hi}h_{k-1} + b_i)$$

$$f_k = \sigma\left(W_{xf}x_k + W_{hf}h_{k-1} + b_f\right)$$

$$g_k = \tanh\left(W_{xg}x_k + W_{hg}h_{k-1} + b_g\right)$$

$$c_k = f_k \odot c_{k-1} + i_k \odot g_k$$

$$o_k = \sigma(W_{xo}x_k + W_{ho}h_{k-1} + b_o)$$

$$h_k = o_k \odot \tanh(c_k)$$

Donde $\odot$ se refiere al producto por elemento de cada vector, σ es la unidad de activación $sigmoid$, $tanh$ es la tangente hiperbólica no lineal, W se refiere a las matrices de peso correspondiente, b se refiere al *bias* y los términos i_k, f_k, g_k, c_k y o_k se refiere a la puerta *input, forget gate, memory cell* y *output gate* para un tiempo k respectivamente [66].

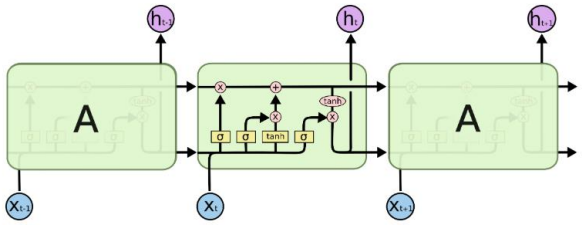

Fig. 4. Estructura interna LSTM [60].

Bi-LSTM

Investigadores denotan que las *RNN* pueden memorizar a corto plazo, lo que permite conectar información que ha sido retenida en memoria para el proceso actual [40].

Siendo la información x_t en el tiempo t, la *RNN* se actualiza mediante la siguiente fórmula:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

Donde:

- h_t se refiere a la variable de estado de la capa en el tiempo t.
- y_t se refiere a la variable de resultado.
- W_{hh} y W_{xh} se refiere a las matrices de peso de la capa.
- b_h y b_y se refiere a vectores de polarización de la capa.
- f se refiere a la función de activación.

Las *RNN* generan problemas para el descenso de gradiente y su anulación en el proceso de propagación inversa, para solventar el inconveniente de las *RNN* se aplica el modelo *LSTM*. Para ello se añade al modelo *RNN* compuertas adicionales, pero los modelos *LSTM* solo pueden contener una función de memoria para la secuencia hacia adelante Figura 5.

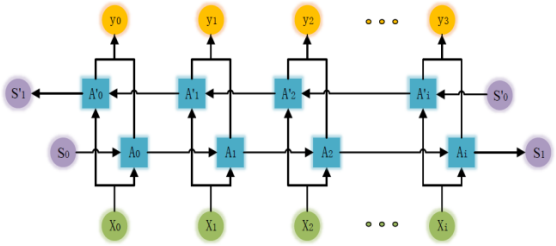

Fig. 5. Estructura interna Bi-LSTM [42].

Función de pérdida

Para lograr la predicción de la posición, el cálculo de la probabilidad condicional para el problema de odometría visual, investigaciones presentan el cálculo de la siguiente forma. Siendo $y_t = (y_1, y_2, \dots, y_t)$ la pose a predecir y $x_t = (x_1,\ x_2, \dots, x_t)$ las secuencias de cuadros monoculares, el cálculo a realizar es el siguiente:

$$p(Y_t|X_t) = p(y_1, y_2, \dots, y_n|x_1, x_2, \dots, x_n)$$

Los investigadores proponen el cálculo del valor θ^* optimo, maximizando $p(Y_t|X_t)$, de este modo se pretende realizar la predicción de la rotación φ_k en ángulos de Euler y la traslación p_k. Donde para cada par (p_k, φ_k) corresponde un (p'_k, φ'_k) [40]. La función de pérdida se denota como:

$$\theta^* = \arg_\theta min \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{3}\|p_k - p'_k\|_2^2 + w\|\varphi - \varphi'_k\|_2^2$$

Donde, $\|.\|$ se refiere a la norma, k se refiere al número de estados en cada posición, como se cuenta con tres ángulos de Euler, k va de 1 a 3, N se refiere al número de secuencias de imágenes por lote, w se refiere al peso, para equilibrar la posición y traslación y θ^* el valor se obtiene al aplicar descenso de gradiente de forma iterativa.

Algoritmos de odometría visual con aprendizaje profundo

Con el auge de las redes neuronales profundas ha generado métodos para resolver los problemas de odometría visual, no requieren un gran cálculo geométrico como los métodos descritos anteriormente, sino más bien modelos descriptivos y concisos.

La aplicación de modelos basados en redes neuronales profundas provee resultados en la predicción de cambios de dirección y la velocidad de la cámara. Modelos emplean el aprendizaje automático para predecir movimiento y profundidad, a partir de un conjunto de datos.

Investigadores emplearon redes neuronales convolucionales para aprender representaciones ópticas de pares de imágenes consecutivas, el modelo fue capaz de procesar imágenes con desenfoque debido al movimiento entre imágenes y a cambios bruscos de iluminación, la utilización de conjuntos de datos *RGB-D* como se muestra en la Figura 6. Para implementar una red neuronal capaz de estimar la profundidad, transformaciones globales, transformación de píxel, generando una biblioteca de software *GVNN* (visión geométrica con red neuronal).

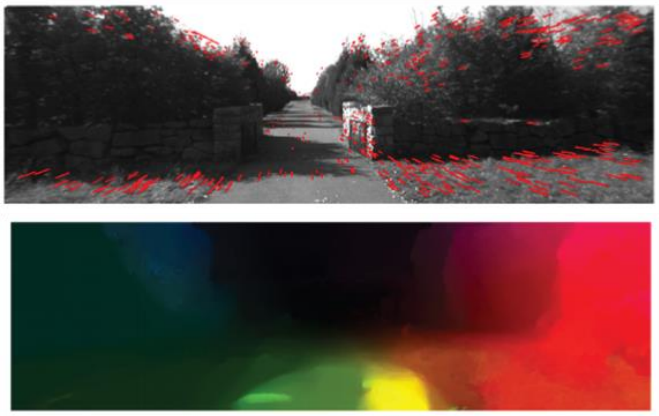

Fig. 6. Campo de flujo óptico denso [15].

En comparación con los algoritmos tradicionales para el cálculo de estimación de pose a través de imágenes monoculares, los algoritmos de aprendizaje profundo han demostrado que pueden ser eficientes para la extracción de información de movimiento entre secuencias de cuadros, reemplazando la complejidad de las fórmulas matemáticas y el esfuerzo computacional para lograr extraer características para buscar

confidencias en las imágenes siguientes.

Siendo el aprendizaje profundo un método simple e intuitivo para solventar los problemas de odometría visual, recientes investigaciones han demostrado que al ignorar las relaciones presentes entre las secuencias de cuadros conllevan a la pérdida de precisión, por lo que la aplicación de redes neuronales recurrentes en combinación con redes neuronales convolucionales con unidades de memoria a largo plazo unidireccional. Los resultados obtenidos demostraron que tuvieron problemas al emplear secuencias de imágenes diferentes al conjunto de entrenamiento, ya que el modelo no se generalizo.

FlowNet

Modelos de redes neuronales han propuesto la solución de odometría visual a partir del flujo óptico, creando interpolación y extrapolación entre los cuadros de imágenes. La arquitectura se observa en la Tabla I con la implementación de redes neuronales convolucionales para predecir campos de flujo a partir de una secuencia de imágenes de entrada.

TABLA I

ESTRUCTURA FLOWNET

CAPA	KERNEL	PADDING	STRIDE	CANALES
Conv1	7×7	3	2	64
Conv2	5×5	2	2	128
Conv3	5×5	2	2	256
Conv3_1	3×3	1	1	256
Conv4	3×3	1	2	512
Conv4_1	3×3	1	1	512
Conv5	3×3	1	2	512
Conv5_1	3×3	1	1	512
Conv6	3×3	1	2	1024

Para ello se ha empleado redes neuronales convolucionales, para la predicción densa de pixeles el modelo contiene una parte de expansión para refinar inteligentemente el flujo para alta resolución. La red se refiere a contraer y expandir los partes que se entrenan como un todo utilizando retro propagación. La arquitectura muestra que la entrada a la red se concatenan las imágenes una seguida de la otra formando así seis canales, que corresponde al par de imágenes en formato *RGB*, lo que permite que la red sea capaz de extraer la información de movimiento por sí misma.

Para la obtención de valores verdaderos se ha identificado como una tarea difícil ya que las verdaderas correspondencias del mundo real entre pixeles no se han podido determinar fácilmente ya que se observa un pequeño número de conjuntos de datos disponibles para ser empleado para esta tarea.

Enfoques para el cálculo de la odometría visual a través de cámaras monoculares, es la utilización de redes neuronales convolucionales *CNN* en combinación con redes recurrentes *LSTM* o con redes bidireccionales con unidades de memoria a largo plazo *Bi-LSTM*, que genera una pose de escala absoluta con 6 grados de libertad a partir de secuencia de imágenes monoculares continuas.

Conjuntos de datos

En general se parte de conjuntos de datos, que se encuentran a disposición de otros investigadores para sus investigaciones, así se pueden dividir en conjunto de datos de entrenamiento y validación, para lo cual se presentan las siguientes alternativas utilizadas por diferentes autores [37] como:

La página web *KITTI* [28] provee conjuntos de datos para visión estéreo, flujo óptico, odometría visual, detección de objetos, etc. El conjunto de datos *KITTI Visual Odometry* provee 22 secuencias estéreo, datos adicionales del conjunto de datos es que la tasa de almacenamiento de las imágenes es a 10 fps, la velocidad con la que iba el automóvil para recorrer el escenario es de 90 km/h.

El Departamento de Ingeniería Mecánica y de Procesos de la Escuela Politécnica Federal de Zúrich [19], provee conjuntos de datos *ASL*, donde su principal objetivo es el de facilitar datos para la comunidad robótica para desarrollo, comparación de resultados y evaluación. Las imágenes son capturadas por un dron donde las imágenes contienen una severa inquietud por el movimiento de la cámara.

Otro conjunto de datos disponible gratuitamente es del departamento de *Computer Science* de la Universidad de Oxford [11], donde se pretende explotar los datos inerciales de navegación peatonal para estimación de movimiento y ubicación. El conjunto de datos presenta mediciones en base a *IMU* (Unidad de medición Inercial) de diferentes escalas de dispositivos móviles con diferentes recursos, proporciona directrices para el uso de datos en bruto de *IMU*.

III. OBJETIVOS Y METODOLOGÍA

De acuerdo con el análisis presentado en los anteriores apartados se ha observado que la utilización del aprendizaje profundo ha logrado ser una alternativa para el problema de odometría visual. Por consiguiente, el presente trabajo tiene como objetivo de realizar un estudio comparativo de las diferentes arquitecturas de aprendizaje profundo para la estimación de la odometría visual.

Para lograr alcanzar el objetivo, se han planteado los siguientes objetivos específicos:

1. Identificar y describir las arquitecturas de aprendizaje profundo disponibles para odometría visual.
2. Identificar los conjuntos de datos disponibles para el entrenamiento y prueba de la odometría visual.
3. Definir las métricas de éxito para la odometría visual.
4. Implementar las arquitecturas seleccionadas para odometría visual.
5. Calcular y comparar el tiempo de entrenamiento entre las diferentes arquitecturas para un mismo conjunto de datos.
6. Evaluar las diferentes arquitecturas con diferentes conjuntos de datos.
7. Comparar el resultado de estimación de la odometría visual de las diferentes arquitecturas.

Para llevar acabo los objetivos del presente estudio, se ha separado por fases:

1. Se realizará un estudio profundo de las arquitecturas de odometría visual. En esta fase se desarrollará una búsqueda bibliográfica enfocada a modelos de aprendizaje profundo que pueden contribuir en la estimación de posición a partir de una secuencia de imágenes. Adicional, se identificarán los diferentes parámetros como sus funciones de perdida, funciones de activación y optimizadores empleados en cada arquitectura.

2. Se analizarán los conjuntos de datos existentes para para odometría visual con el fin de escoger el óptimo para este estudio.
3. En esta fase se enfocará en los siguientes puntos:
   3.1. Desarrollo y en la implementación de las arquitecturas en código Python.
   3.2. Optimización de carga de los conjuntos de datos seleccionados.
   3.3. Optimización de procesamiento de datos para el computador empleado para el presente estudio.
4. En esta última fase se realizarán los siguientes puntos:
   4.1. Ejecución de las arquitecturas de odometría visual con el conjunto de datos seleccionado.
   4.2. Se desarrollará la comparativa de los diferentes modelos en tiempo de ejecución, así como precisión en la estimación de la posición.
   4.3. Una vez desarrollada la comparativa se determinará el modelo óptimo para la estimación de posición a partir de secuencia de imágenes a color.

IV. CONTRIBUCIÓN

Se presenta una comparativa de los modelos *DeepVO*, *MagicVO* y *PoseConvGRU* tanto en el número de parámetros que cuenta cada arquitectura, estimación de posición y orientación en los conjuntos de datos de prueba, tiempo de inferencia de la arquitectura, etc.

Para el entrenamiento de las diferentes arquitecturas se ha elegido el conjunto de datos *KITTI Visual Odometry* [28] como se discutió en el apartado anterior.

Se presentan 11 secuencias de imágenes con su respectivo valor de verdad, cada secuencia de imágenes cuenta con dos conjuntos de imágenes izquierda y derecha, ya que se ha utilizado cámaras estero con dimensión de 1241×376 pixeles.

Como se mencionó en el anterior apartado, cada secuencia representa diferentes recorridos, los cuales se encuentran con entornos donde se observan, objetos estáticos como dinámicos durante el recorrido. Por lo cual se ha utilizado el conjunto de imágenes de la cámara izquierda de las secuencias 00, 02, 04, 06, 08 y 10 por ser secuencias relativamente extensas. Con un total de 15846 imágenes en total para el entrenamiento y las secuencias 03, 05 y 07 se han utilizado para evaluación.

Se ha implementado las arquitecturas *DeepVO*, *MagicVO* y *PoseConvGRU* en *Python*, se utilizó la librería de alto rendimiento de *TensorFlow* con el submódulo de *Keras* para la creación de redes neuronales profundas porque cuenta con una gran variedad de documentación y soporte para implementar diferentes modelos de redes neuronales.

Para el conjunto de datos seleccionado se ha creado código para que el pipeline de carga de datos sea dinámico al momento de realizar el entrenamiento, esto se debe a que almacenar todas las secuencias en memoria *RAM* es ineficiente y satura los recursos del computador.

Para ello se ha utilizado la librería *tf.data* de *TensorFlow* que permite la carga dinámica de datos y la selección de imágenes aleatorias durante el entrenamiento de redes neuronales profundas. De forma que se carga un mini lote de 8 pares de imágenes en memoria para que sean procesadas, durante ese tiempo se carga el siguiente mini lote en memoria y así sucesivamente hasta completar todas las secuencias de los datos de entrenamiento. El entrenamiento de cada arquitectura se realizó durante 200 épocas para realizar una comparativa a la par con el mismo conjunto de datos.

Para acelerar el entrenamiento se ha utilizado el modelo pre entrenado de *FlowNet*, ya que es la primera capa en la cual se lleva el procesamiento con redes neuronales convolucionales. El modelo pre entrenado *FlowNet* tiene la peculiaridad de que fue entrenado con unidades de activación *Leaky ReLU* con el coeficiente alpha de 0.1, a diferencia de los desarrollados en las arquitecturas a *DeepVO*, *MagicVO* y *PoseCnvGRU* que emplean funciones de activación *ReLU* en la capa de convolución.

Adicional se ha modificado las unidades de activación en las capas con redes neuronales completamente conectadas a *Leaky ReLU* para ayudar a la convergencia de los modelos para la estimación de odometría visual. También se ha modificado la última capa que cuenta con la unidad de activación Linear para generar los valores de posición y rotación relativa.

DeepVO

La aplicación *DeepVO* emplea redes neuronales convolucionales en combinación con redes neuronales recurrentes *RCNN*, donde no solo el modelo es capaz de aprender representaciones de características efectivas para el problema de odometría visual, sino que también es capaz de modelar las dinámicas secuenciales con la utilización de las redes neuronales recurrentes.

La Tabla II presenta un redimensionamiento de las imágenes a $1280 \times 384 \times 3$, se aplica el par de imágenes a la entrada del modelo, la capa *CNN* se encarga de producir características de odometría visual, la capa convolucional consta de 9 capas convolucionales donde se toma imágenes en *RGB* como entrada para la red, posteriormente se aplana la salida matricial a un vector que será introducido en la capa *RNN* para producir un aprendizaje secuencial.

TABLA II
ESTRUCTURA DEEPVO

CAPA		DIMENSIONES
Imágenes apiladas		$640 \times 192 \times 3$ $640 \times 192 \times 3$
Capa convolucional FlowNet	Conv1	$320 \times 96 \times 64$
	Conv2	$160 \times 48 \times 128$
	Conv3	$80 \times 24 \times 256$
	Conv3_1	$80 \times 24 \times 256$
	Conv4	$40 \times 12 \times 512$
	Conv4_1	$40 \times 12 \times 512$
	Conv5	$20 \times 6 \times 512$
	Conv5_1	$20 \times 6 \times 512$
	Conv6	$10 \times 3 \times 1024$
Capa LSTM	LSTM1	1000
	LSTM2	1000
Capa completamente conectada		6

Se pretende que el modelo sea capaz de extraer características simultaneas y secuenciales a través de la combinación de las capas *CNN* y *RNN* ya que la estimación de la posición actual puede ser beneficiada de información representada en anteriores pares de imágenes, por lo cual se emplea dos capas *LSTM* de

1000 estados ocultos cada una. La cara *RNN* genera estimaciones en cada paso basado en lo obtenido en las capas *CNN* anteriores la estructura del modelo.

MagicVO

El modelo presenta un sistema que no requiere parámetros intrínsecos de la cámara, el cálculo de la posición es absoluto, la combinación de redes convolucionales con redes bidireccionales con unidades de memoria a largo plazo que permite a más de la propiedad de aprender características del par de imágenes aprende la relación entre secuencia de imágenes antes y después del cuadro actual, de este modo genera un modelo con alta precisión.

En la Tabla III presenta la arquitectura para el preprocesamiento de secuencias de imágenes, se realiza un redimensionamiento de escala a $640 \times 192 \times 3$. Se superpone el tercer canal dos cuadros adyacentes al actual, se realiza normalización de los valores de los pixeles, como primera parte se presenta una *CNN* seguida de la capa *Bi-LSTM* y finalmente una capa completamente conectada.

Se presenta capas ocultas de *RNN* con *Bi-LSTM*, así la correlación existente entre cuadros de imagen puede ser utilizada para estimar la estimación de poses, el modelo presenta 1000 nodos *LSTM* hacia adelante y 1000 hacia atrás.

TABLA III

ESTRUCTURA MAGICVO

CAPA		DIMENSIONES
Imágenes apiladas		$640 \times 192 \times 3$ $640 \times 192 \times 3$
Capa convolucional FlowNet	Conv1	$320 \times 96 \times 64$
	Conv2	$160 \times 48 \times 128$
	Conv3	$80 \times 24 \times 256$
	Conv3_1	$80 \times 24 \times 256$
	Conv4	$40 \times 12 \times 512$
	Conv4_1	$40 \times 12 \times 512$
	Conv5	$20 \times 6 \times 512$
	Conv5_1	$20 \times 6 \times 512$
	Conv6	$10 \times 3 \times 1024$
Capa Bi-LSTM	LSTM1	1000
	LSTM2	1000
Capa completamente conectada		256
		6

Posteriormente se presenta dos capas completamente conectadas para integrar la información procedente de las capas *Bi-LTSM* para estimar con precisión la pose final con 6 grados de libertad. Siendo el resultado los ángulos de Euler y los puntos de traslación.

PoseConvGRU

Presenta una novedosa red neuronal convolucional recurrente con la utilización de arquitecturas *GRU*, el modelo es capaz de codificar de un par de imágenes secuenciales funciones de movimiento a corto plazo, así el modelo propaga información en el tiempo.

La entrada de la *CNN* son las imágenes *RGB* de un par de imágenes con redimensionadas a $1280 \times 384 \times 3$ como se muestra en la Tabla IV, estos pares de imágenes son sometidos a 10 capas convolucionales siendo la última capa de *Max Pooling*. El uso de una *Max Pooling* en la última capa *CNN* reduce de manera significativa los parámetros de salida de *FlowNet* aumentando la complejidad de entrenamiento de la red neuronal.

TABLA IV

ESTRUCTURA POSECONVGRU

CAPA		DIMENSIONES
Imágenes apiladas		$640 \times 192 \times 3$ $640 \times 192 \times 3$
Capa convolucional FlowNet	Conv1	$320 \times 96 \times 64$
	Conv2	$160 \times 48 \times 128$
	Conv3	$80 \times 24 \times 256$
	Conv3_1	$80 \times 24 \times 256$
	Conv4	$40 \times 12 \times 512$
	Conv4_1	$40 \times 12 \times 512$
	Conv5	$20 \times 6 \times 512$
	Conv5_1	$20 \times 6 \times 512$
	Conv6	$10 \times 3 \times 1024$
Capa GRU	GRU	3
Capa completamente conectada		4096
		1024
		128
		6

El módulo *ConvGRU* es capaz de almacenar a largo plazo una representación visual de la secuencia de imágenes, ya que permite que la red neuronal aprenda las relaciones intrínsecas de poses, se emplea la arquitectura *GRU* para que la red pueda recordar estados de sucesos históricos como las relaciones geométricas provenientes de las secuencias anteriores del par de imágenes, se emplea la arquitectura *GRU* para reducir la cantidad de parámetros ya que presenta un rendimiento similar a la arquitectura *LSTM*.

Parámetros de las arquitecturas

Se muestra los parámetros que serán modificados durante el entrenamiento de las diferentes arquitecturas por lo cual no se toma en cuenta los parámetros del modelo *FlowNet* ya que el modelo es el encargado de permitir la extracción del flujo óptimo de los pares de imágenes Tabla V.

TABLA V

PARÁMETROS DE LAR ARQUITECTURAS

ARQUITECTURA	PARÁMETROS
DeepVO	149506550
MAGICVO	268894342
PoseConvGRU	19002355

V. RESULTADOS

Se ha observado el conjunto de datos *KITTI* es el ideal para el entrenamiento y prueba de las diferentes arquitecturas. Se empleó un equipo con *CPU* i7-9700KF de 3.60GHz, 16 *GB* de memoria RAM, tarjeta gráfica *NVIDIA GTX* 1660 super de 6 *GB*, como el equipo cuenta con tarjeta gráfica *NVIDIA* se instaló *CUDA Toolkit* 10.2 para obtener soporte para *GPU* en TensorFlow. De esta forma se optimiza tiempos de ejecución de los modelos por la paralelización que presenta la arquitectura de la tarjeta gráfica.

Dimensiones de imágenes

Debido a las limitaciones del equipo se decidió reducir el tamaño de las imágenes de 1241×376 a 640×192, de esta forma se reduce el tiempo de entrenamiento y no satura al equipo durante la ejecución del entrenamiento de las arquitecturas. Sin embargo, debido a la reducción en la dimensión de las imágenes podría causar perdidas de información en el entrenamiento de las arquitecturas porque las relaciones entre pixeles serían inconsistentes.

Secuencias de entrenamiento

Se emplean las secuencias 00, 02, 04, 06, 08 y 10 cuentan con 4541, 4661, 271, 1101, 4071 y 1201 pares imágenes, las trayectorias reales.

Se ha escogido estas secuencias para el entrenamiento por presentar un mayor número de imágenes para los diferentes recorridos, la secuencia 02 es la que cuenta con el mayor número de datos de entrenamiento. Además de presentar en los recorridos cambios de dirección hacia adelante. Para la gráfica se ha tomado los ejes *X* y *Z* por la disposición de los ejes la cámara de video.

Secuencias de prueba

Para la evaluación de las diferentes arquitecturas se emplean las secuencias 03, 05 y 07 cuenta con 801, 2761 y 1101 pares imágenes respectivamente.

Se ha escogido estas secuencias para la evaluación de los modelos, por presentar un menor número de imágenes para los diferentes recorridos. Además de presentar en los recorridos cambios de dirección hacia adelante. Para las gráficas se ha tomado los ejes *X* y *Z* por la disposición de la cámara de video.

Entrenamiento de las arquitecturas

se emplearán las secuencias 00, 02, 04, 06, 08 y 10 para el entrenamiento de las arquitecturas y la secuencia 01 para evaluación durante el entrenamiento durante 200 épocas, se muestra a continuación el progreso del entrenamiento de las arquitecturas para converger a 0 con la función de pérdida como se muestra en la Figura 7.

a) Pérdida de la arquitectura *DeepVO*: el entrenamiento de la arquitectura converge rápidamente a 0 de acuerdo con la función de pérdida, donde se observa que el valor mínimo que alcanza durante el entrenamiento es de 0.0007523670 con el conjunto de datos de entrenamiento, mientras que durante la evaluación con la secuencia 01 no se observa una mejora y el valor oscila en 0.5, el tiempo de entrenamiento por época es aproximadamente de 13 minutos siendo un total de aproximadamente 43 horas en cumplir las 200 épocas.

b) Pérdida de la arquitectura *MagicVO*: el entrenamiento de la arquitectura converge lentamente a 0, donde el valor mínimo que alcanza durante el entrenamiento es de 0.0009443219 con el conjunto de datos de entrenamiento, mientras que con el conjunto de datos de evaluación oscila en 0.5. Siendo el modelo que cuenta con una gran cantidad de parámetros de entrenamiento se observó que el tiempo de entrenamiento por época es aproximadamente de 20 minutos siendo un total de 67 horas en cumplir las 200 épocas.

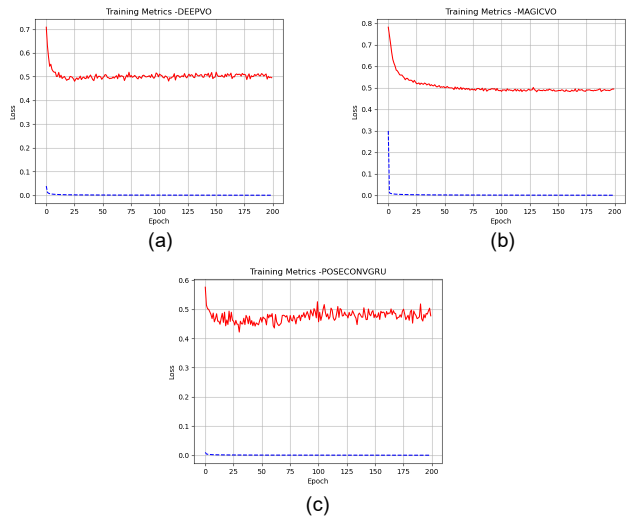

Fig. 7. Pérdida durante el entrenamiento. a) DeepVO, b) MagicVO, c) PoseConvGRU

c) Pérdida de la arquitectura *PoseConvGRU*: durante el entrenamiento de la arquitectura se observa que converge rápidamente a 0, siendo el valor mínimo alcanzado de 0.0006612031 con el conjunto de datos de entrenamiento mientras que con los datos de evaluación oscila intermitentemente en 0.5, el tiempo de entrenamiento por época es de aproximadamente 6 minutos con un total aproximado de 20 horas para cumplir las 200 épocas.

Evaluación de las arquitecturas

A continuación, se muestra grafica donde se logra observar que la arquitectura que logra inferir valores que se acerquen a los valores de verdad es la arquitectura *PoseConvGRU* (línea punteada de color rojo) en la secuencia 03, siendo la arquitectura *MagicVO* (línea con punto de color verde) la que genera estimaciones muy alejadas a los valores de verdad Figura 8.

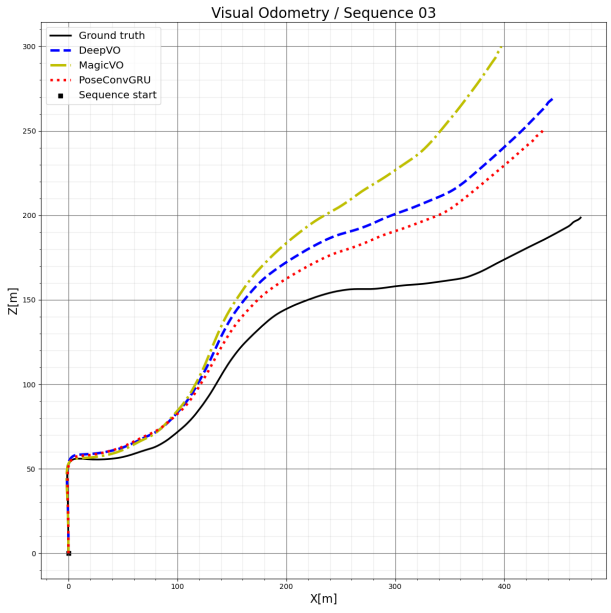

Fig. 8. Comparativa de arquitecturas con la secuencia 03

Se observa que al utilizar la secuencia 05 para inferir los valores de las posiciones relativas, la arquitectura *PoseConvGRU* (línea punteada de color rojo) logra acercarse a los valores de verdad de la secuencia, superando en gran medida a las demás arquitecturas, siendo la que tiene un gran desplazamiento la inferencia de la arquitectura *MagicVO* (línea con punto de color verde) como se observa en la Figura 9.

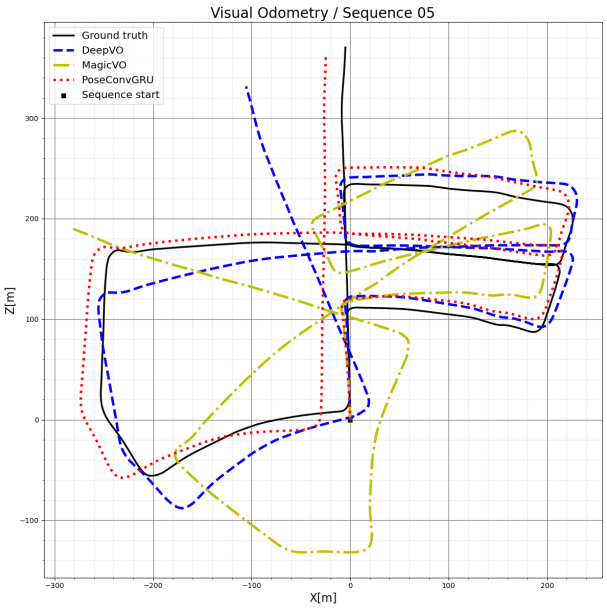

Fig. 9. Comparativa de arquitecturas con la secuencia 05

Adicional se utilizó la secuencia 07 para estimar las posiciones relativas con las diferentes arquitecturas, siendo los valores de la arquitectura *DeepVO* (línea entrecortada de color azul) que logra acercarse a los valores de verdad de la secuencia, al igual que las anteriores secuencias se observa que la arquitectura *MagicVO* (línea con punto de color verde) es la que más alejada se encuentra en estimar los valores esperados como se muestra en la Figura 10.

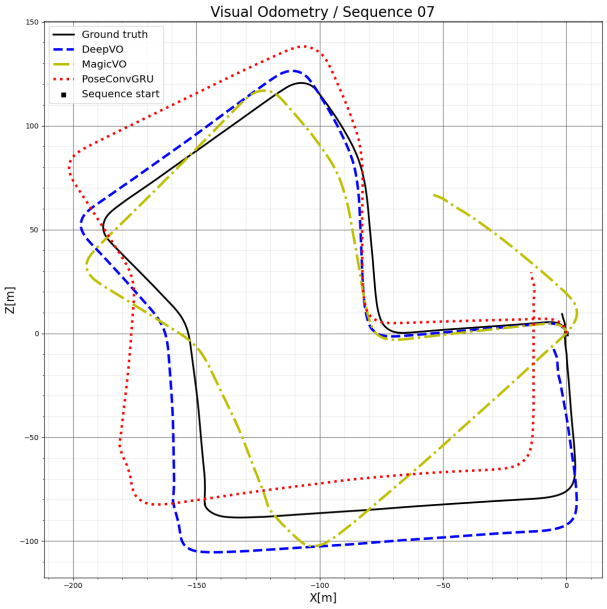

Fig. 10. Comparativa de arquitecturas con la secuencia 07

Tiempo de inferencia de las arquitecturas

Se empleó al modelo pre entrenado *FlowNet* para reducir el tiempo de entrenamiento de las arquitecturas, cabe mencionar que las estimaciones se realizaron en mini lotes de 8 pares de imágenes en cada iteración.

Por limitaciones del equipo donde fue desarrollado el presente trabajo se dividió el procesamiento de parámetros del modelo *FlowNet* en la tarjeta gráfica y el resto de las capas en el *CPU*, ya que existía un exceso de parámetros que la tarjeta gráfica logra procesar, de esta forma se logra optimizar recursos tanto en el procesamiento de las capas convolucionales como la capas *LSTM*, *GRU* y *Dense*. A continuación, se describe el tiempo de inferencia de las demás capas que siguen al modelo *FlowNet*.

TABLA VI

TIEMPO DE INFERENCIA DE LAS ARQUITECTURAS

ARQUITECTURA	TIEMPO DE INFERENCIA [S]
DeepVO	~0.1489
MAGICVO	~0.1708
PoseConvGRU	~0.0936

El tiempo de inferencia de las arquitecturas se muestra en la Tabla VI. Donde por cada par de imágenes las arquitecturas logran estimar valores en un tiempo en orden de los milisegundos.

Como se puede observar la arquitectura que logra estimar las posiciones en un tiempo relativamente rápido es el modelo *PoseConvGRU* con ~0.0936 segundos en inferir los valores, seguida de *DeepVO* con ~0.1489 segundos y finalmente el modelo *MagicVO* con ~0.1708 segundos, haciendo que la arquitectura *MagicVO* sea la que más tiempo se tarda en procesar los mini lotes de las secuencias.

VI. DISCUSIÓN

Como primer punto se analiza las capas que fueron utilizadas en cada una de las arquitecturas y los valores de perdida al finalizar el entrenamiento:

DeepVO: se encuentra estructurada por *FlowNet*, posteriormente se cuenta con dos capas *LSTM* de 1000 estados ocultos cada una. Estas capas *LSTM* son empleadas para que la arquitectura contenga unidades de memoria a largo plazo hacia el futuro, lo que permite que la arquitectura pueda recordar valores futuros con los que fue entrenada, la primera capa *LSTM* se observa un total de 126884000 parámetros, seguida con otra capa *LSTM* de 8004000 parámetros por lo que hace que el entrenamiento e inferencia de la arquitectura tenga un tiempo prolongado de procesamiento.

MagicVO: se encuentra estructurada por *FlowNet* en la primera etapa del modelo, posterior se encuentra una capa *Bi-LSTM* de 2000 estados ocultos, las capas *Bi-LSTM* se forman al conectar dos capas *LSTM* una que sea capaz de contener memoria a largo hacia el futuro y otra que contenga memoria a largo plazo hacia el pasado, lo que permite que el modelo sea capaz de recordar valores futuros o pasados con los que fue entrenado, esta capa cuenta con 253768000 parámetros lo que hace que la arquitectura sea muy extensa y difícil de procesar debido a la gran cantidad de parámetros que cuenta solo esta capa, posteriormente se cuenta con capas densas de 256 neuronas con unidad de activación *Leaky ReLU* y 6 neuronas con unidad de activación *Linear*, adicional se cuentan con capas

PoseConvGRU: emplea en las primeras capas *FlowNet* para la estimación del flujo óptico de las secuencias de imágenes,

posterior se emplea una capa *MaxPooling* que permite reducir el número de parámetros y extraer aún más las características de la inferencia del modelo FlowNet, posterior se encuentra la capa *GRU* de 46125 parámetros, esta capa es similar a las *LSTM* con la diferencia que combina en celdas los estados ocultos tanto en los parámetros *forget gate* e *input gate*, haciendo que sea una capa con mejores características que las *LSTM* lo que permite que la arquitectura pueda mantener información a lo largo del tiempo sin necesidad de que los valores sean reiniciados lo que con lleva a que no se eliminen valores que pueden ser relevantes en la inferencia.

De acuerdo con lo detallado anteriormente se observa que las arquitecturas cuentan con una gran cantidad de parámetros que son modificados durante el entrenamiento, lo que conlleva un mayor tiempo de entrenamiento, tiempo de inferencia y precisión en la estimación. Al finalizar el entrenamiento la arquitectura que cuenta con un valor mínimo de pérdida es la arquitectura *PoseConvGRU* con 0.0006612031, esto se debe a que cuenta con un menor número de parámetros en las capas GRU donde la red neuronal puede aprender las relaciones secuenciales entre imágenes, presentando un mayor desempeño al finalizar el entrenamiento de la arquitectura.

Error absoluto

Como siguiente punto se analiza el error absoluto presente en la evaluación del modelo con las secuencias 03, 05 y 07. Como se observó en la anterior sección, no existe un modelo por el cual logre coincidir perfectamente con los valores de verdad, por lo que se ha graficado el error presente en la inferencia de las posiciones absolutas durante el recorrido de las secuencias. A continuación, se presenta el error presente en el recorrido de la secuencia 03 tanto en los valores de posición como de orientación.

Como se observa en la Figura 11, a medida que el recorrido avanza el error se propaga en la estimación de la posición, esto se debe en gran medida que las transformaciones lineales que suceden para pasar de posición relativa a posición absoluta conllevan la utilización de los ángulos presentes en la inferencia de las arquitecturas.

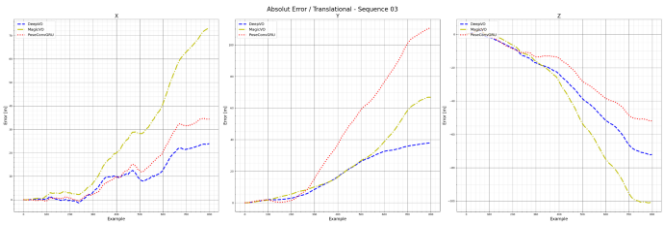

Fig. 11. Error absoluto de las arquitecturas secuencia 03 – Posición

La estimación de posición en el eje X la arquitectura *DeepVO* presenta un error al finalizar el recorrido de 24 metros, a diferencia de la arquitectura *MagicVO* que presenta un error de 74 metros al finalizar el recorrido.

- La estimación de la posición en el eje *Y* la arquitectura *DeepVO* genera un error cercano a 40 metros al finalizar el recorrido, a diferencia de la arquitectura *PoseConvGRU* que genera un error aproximado de 120 metros.
- Para la estimación de la posición en el eje Z se observa que los valores estimados son mayores a los valores verdaderos, por ende, los valores son negativos. Como se observa la arquitectura *PoseConvGRU* tiene un menor error al finalizar el recorrido que es de aproximadamente 50 metros.

Siendo la arquitectura que cuenta con una mayor cantidad de error es la de *MagicVO*, seguida de *PoseConvGRU* y la que tiene menor error en posición es la arquitectura *DeepVO* en las posiciones en el eje X e Y, a diferencia que la posiciones en el eje Z donde el que tiene menor error es la de *PoseConvGRU*.

Posteriormente se observa el error absoluto presente en los valores de inferencia de la rotación de la secuencia 03 en la Figura 12:

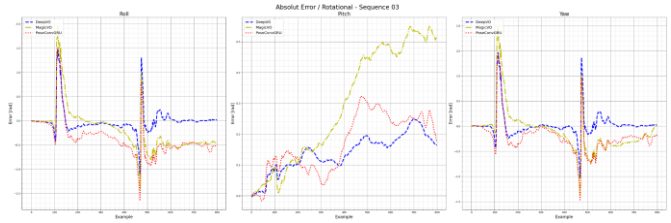

Fig. 12. Error absoluto de las arquitecturas secuencia 03 – Orientación

- Para la estimación de rotación con respecto al eje X (Roll), se observa que durante el recorrido de la secuencia el error no se va acumulando, de manera que existen picos en los pares de imágenes 100 a 160 y de 400 a 500. Por lo que el error de la estimación del ángulo intenta acercarse a 0 rad, siendo la arquitectura *MagicVO* la que produce picos más pronunciados de aproximadamente 2 rad seguida de la arquitectura *DeepVO* y *PoseConvGRU*.
- Para la estimación de rotación con respecto al eje Y (Pitch), se observa que el error va aumentando a medida que la secuencia avanza, llegando a ser el error de aproximadamente 0.5 rad el valor mayor que genera la arquitectura *MagicVO*.
- Para la estimación de rotación con respecto al eje Z (Yaw), se observa un comportamiento parecido al error de estimación del eje X, ya que existen picos de error casi en las mismas zonas 100 a 160 y de 400 a 500. Siendo el error máximo de 2 rad que presenta la estimación de la arquitectura *MagicVO*.

Se presenta la evaluación de las arquitecturas con la secuencia 05 en la Figura 13, donde se observa:

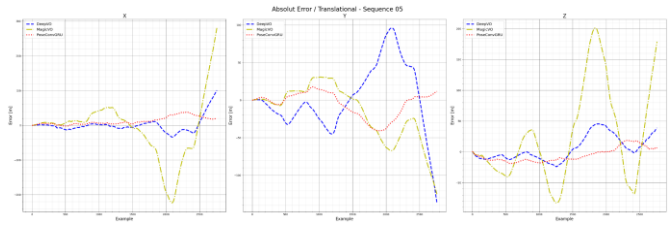

Fig. 13. Error absoluto de las arquitecturas secuencia 05 – Posición

- Para la estimación de la posición en el eje X, el error de las arquitecturas *DeepVO* y *PoseConvGRU* oscila de 0 a 20 metros durante el recorrido, pero al finalizar el recorrido la arquitectura *PoseConvGRU* genera un error de 10 metros aproximadamente que es menor a las demás arquitecturas.
- Para la estimación de la posición en el eje Y, se observa que existe oscilaciones lo cual dependería en gran medida en el recorrido, cuenta con varias curvas que cierran circuitos (bucles). La estimación de la arquitectura *DeepVO* es una de las arquitecturas que genera mayores picos de error siendo el mayor de 100 metros aproximadamente.
- Para la estimación de la posición en el eje Z, se observa que la arquitectura *MagicVO* es la que genera la mayor

oscilación de errores que van hasta 200 metros, esto puede ser causado por el número de épocas de entrenamiento.

Como se puede apreciar en la Figura 14 el error absoluto de estimación de orientación varía independientemente de la arquitectura, pero se logra observar que:

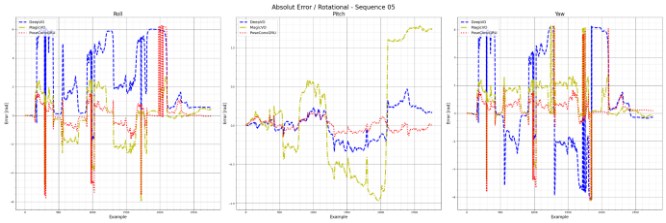

Fig. 14. Error absoluto de las arquitecturas secuencia 05 – Orientación

- Para la estimación de la orientación en el eje X (Roll) la arquitectura *DeepVO* es la que genera la mayor cantidad de picos de error hasta los 6 rad pero al finalizar la secuencia logra reducir el error a menos de 1 rad. La arquitectura *MagicVO* genera errores que llegan hasta los 2 rad a diferencia de 3 picos que llegan hasta los 6 rad pero la arquitectura *PoseConvGRU* tiene el mismo comportamiento que la arquitectura *MagicVO* a diferencia que al final de la secuencia llega a tener un valor que casi es 0 rad.
- Para la estimación de la orientación en el eje Y (Pitch) la arquitectura *MagicVO* existe oscilaciones que llegan hasta los 1.8 rad aproximadamente a diferencia de la arquitectura *DeepVO* genera errores menores a 0.5 rad.
- Para la estimación de la orientación en el eje Z (Yaw) independiente de las arquitecturas los errores oscilan hasta los 6 rad pero la que tiene mayores fluctuaciones es la arquitectura *DeepVO*.

Como ultima secuencia de evaluación de las arquitecturas, se utilizó la secuencia 07, donde se observa Figura 15 que:

Fig. 15. Error absoluto de las arquitecturas secuencia 07 – Posición

- Para la estimación de la posición en el eje X, el error la arquitectura *DeepVO* es la que genera un menor error que llega hasta los 15 metros aproximadamente, a diferencia que la arquitectura *PoseConvGRU* que llega hasta los 35 metros aproximadamente.
- Para la estimación de la posición en el eje Y, se observa que existe oscilaciones siendo la arquitectura *DeepVO* la cual genera un error mayor de hasta 60 metros al finalizar la secuencia, de forma que el error a medida que avanza la secuencia el error va creciendo.
- Para la estimación de la posición en el eje Z, se observa que la arquitectura *MagicVO* es la que genera el mayor error de hasta 80 metros. Las arquitecturas *DeepVO* y *PoseConvGRU* generan errores menores a los 20 metros siendo la arquitectura *PoseConvGRU* la que genera valores mayores al valor real.

A continuación, se muestra el cálculo del error absoluto de la orientación donde se observa que el error absoluto varía por secciones.

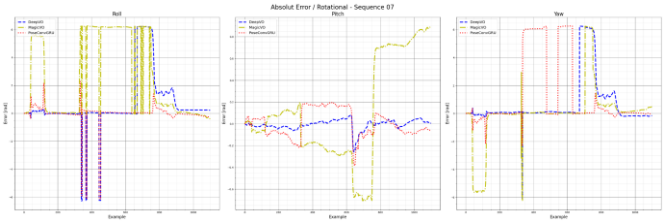

Fig. 16. Error absoluto de las arquitecturas secuencia 07 – Orientación

- Para la estimación de la orientación en el eje X (Roll) la arquitectura *MagicVO* es la que genera la mayor cantidad de picos de error de hasta los 6 rad. La arquitectura *DeepVO* genera errores que llegan hasta los 6 rad de igual forma la arquitectura *PoseConvGRU*. Pero al finalizar la secuencia las arquitecturas logran reducir el error a menos de 1 rad.
- Para la estimación de la orientación en el eje Y (Pitch) la arquitectura *MagicVO* existe oscilaciones que llegan hasta los 0.8 rad aproximadamente a diferencia de la arquitectura *DeepVO* genera errores menores a 0.1 rad.
- Para la estimación de la orientación en el eje Z (Yaw) la arquitectura *PoseConvGRU* es la arquitectura que genera picos de 6 rad, pero las arquitecturas

RMSE

Como se puede apreciar en la anterior Tabla VII, tanto la arquitectura *DeepVO* y *PoseConvGRU* presentan un buen desempeño en las secuencias de evaluación, siendo la arquitectura *DeepVO* en la secuencia 03 la que genera menor error tanto en la estimación de posición como en la orientación. En las demás secuencias se observa que *PoseConvGRU* genera un error bajo en comparación con las demás arquitecturas tanto en la estimación de posición como en la de orientación.

Tanto la arquitectura DeepVO y PoseConvGRU presentan un buen desempeño en las secuencias de evaluación, siendo la arquitectura DeepVO en la secuencia 03 la que genera menor error tanto en la estimación de posición como en la orientación.

TABLA VII

RMSE DE LAS SECUENCIAS 03, 05 Y 07

SECU	DEEPVO		MAGICVO		POSECONVGRU	
	T	R	T	R	T	R
03	26.65	0.23	41.35	0.42	38.39	0.39
05	30.69	2.80	71.80	1.32	15.67	1.14
07	16.53	1.61	25.37	2.54	14.47	1.69
MEAN	24.62	1.55	46.17	1.42	22.84	1.08

T: RMSE de traslación en m con el conjunto de datos KITTI

R: RMSE de rotación en rad con el conjunto de datos KITTI

VII. CONCLUSIONES

Respecto al primer objetivo específico Identificar y describir las arquitecturas de aprendizaje profundo disponibles para odometría visual. Se ha observado que la capacidad que tienen las redes neuronales recurrentes *LSTM* y su variante *GRU* en tareas de procesamiento de secuencias de imágenes ha logrado ser útiles para el problema de odometría visual. Para lo cual se ha analizado en el estado del arte como ha ido avanzando los métodos de estimación de posición y orientación a través de métodos analíticos hasta llegar a la utilización de redes neuronales profundas.

Como es el caso de los modelos seleccionados para el desarrollo del presente trabajo como *DeepVO*, *MagicVO* y *PoseConvGRU*, la utilización *CNN* presenta excelentes resultados en tareas de visión por computadora ya que a través de sus capas convolucionales permite que la red sea capaz de extraer características de las imágenes y más aún con el desarrollo tecnológico que se ha presentado en los últimos años.

Los avances tecnológicos han permitido que los procesos de entrenamiento de *CNN* sean paralelizados con la utilización de tarjetas gráficas que presentan núcleos *CUDA*, estos avances fueron aprovechados por los desarrolladores para generar modelos más complejos al añadir una gran cantidad de capas convolucionales. Además, se ha recolectado información del funcionamiento de las redes neuronales *LSTM* y *GRU* tanto en su capacidad de almacenar memoria a largo plazo tanto para valores futuros o pasados.

Se ha empleado modelos que han sido pre entrenados para tareas específicas ya sea el caso de *FlowNet* que permite la estimación de flujo óptico de un par de imágenes secuenciales, donde este par de imágenes se encuentra concatenado uno tras el otro formando una matriz de 6 canales. La utilización de transferencia de aprendizaje ayudo a que las arquitecturas no sean entrenadas desde cero es decir con todas las capas, sino solo las capas que se encuentran seguidas de las capas convolucionales.

Respecto al objetivo específico Identificar los conjuntos de datos disponibles para el entrenamiento y prueba de la odometría visual. Se ha observado el conjunto de datos *KITTI* [24] cuenta con datos extensos con imágenes de gran calidad, se presentan 11 secuencias con valores de verdad y 10 secuencias de imágenes sin valores de verdad.

Respecto al objetivo específico Definir las métricas de éxito para la odometría visual. Se ha definido el dimensionamiento de las imágenes, las secuencias de entrenamiento y evaluación para los diferentes modelos, pérdida durante el entrenamiento de las arquitecturas y los diferentes cálculos de error como el error absoluto y el *RMSE*. Que son métricas que se han utilizado en los trabajos utilizados en este trabajo comparativo.

Se ha realizado el desarrollo respecto al objetivo específico Implementar las arquitecturas seleccionadas para odometría visual. Para el análisis se optó por seleccionar las arquitecturas *DeepVO* [66], donde se observó que presenta un excelente resultado para la estimación de odometría visual en comparación con los modelos *VISO2_M* y *VISO2_S* [42], este modelo consistía en la primera etapa al modelo *FlowNet* seguido de capas *LSTM* y *Dense*. La siguiente arquitectura seleccionada fue *MagicVO* [40] que obtuvo un mejor rendimiento que *DeepVO* y *ORB_SLAM2* [53], esta arquitectura cuenta con FlowNet como primera etapa seguida de capas *Bi-LSTM* y *Dense*. La última arquitectura elegida fue *PoseConvGRU* [75] que cuenta con capas *GRU* y *Dense*.

Una vez elegidos las arquitecturas y conjuntos de datos se ha implementado las arquitecturas, se utilizó *TensorFlow* para el desarrollo de las redes neuronales y carga de datos. Se escogió esta librería por contener documentación y una comunidad activa, que permite el desarrollo intuitivo y sin necesidad de crear funciones desde cero. Además, fue muy útil a la hora de compartir recursos durante el entrenamiento de las arquitecturas ya que permitió intercambiar información desde la *GPU* a la *CPU* con pocas líneas de código, cabe mencionar que *TensorFlow* permitió la carga dinámica de datos ya que como se trabajó con imágenes de dimensiones de 640×192 que era ineficiente el mantener todas las secuencias en la memoria *RAM* ya que los valores en conjunto causaban el desbordamiento de memoria.

Se hace referencia al desarrollo del objetivo específico Calcular y comparar el tiempo de entrenamiento entre las diferentes arquitecturas para un mismo conjunto de datos. Donde se demostró que la utilización de capas *LSTM* con varios estados ocultos hacían que el entrenamiento de las redes neuronales sea lento ya que no se pueden paralelizar como es el caso de las *CNN*. Adicional se observó que el modelo que tuvo un entrenamiento relativamente corto en las 200 épocas seleccionadas fue *PoseConvGRU*, ya que cuenta con una cantidad reducida de parámetros en comparación de *MagicVO* que es el modelo con mayor número de parámetros a entrenar. *MagicVO* tardó 67 horas en cumplir las 200 épocas de entrenamiento, esto quizá se deba a las limitaciones presentadas en el equipo en el cual se desarrolló el entrenamiento. Además, la arquitectura *PoseConvGRU* fue la que logro alcanzar un valor mínimo de 0.0006612031 en su valor de perdida, siendo la arquitectura en alcanzar el menor valor de las otras arquitecturas.

De acuerdo con el objetivo específico Evaluar las diferentes arquitecturas con diferentes conjuntos de datos. Se observaron los diferentes valores del error absoluto que presentaron las diferentes arquitecturas con respecto a los valores de verdad, donde el error de las posiciones se va propagando a través que avanza la secuencia de imágenes, esto se debe a que las posiciones son consecuentes de las estimaciones pasadas y por ende repercute en el cálculo de la siguiente posición ya que interviene tanto la posición anterior como la orientación.

Referente al objetivo específico Comparar el resultado de estimación de la odometría visual de las diferentes arquitecturas. Se ha determinado que la mejor arquitectura para estimación de la odometría visual es la arquitectura *PoseConvGRU* ya que es un modelo que no emplea una gran cantidad de parámetros o capas muy cargadas de parámetros para lograr lo esperado con un *RMSE* de 24.62961 en posición y 1.555016 en rotación, si bien las demás arquitecturas pueden mejorar con un reentrenamiento por un tiempo más prolongado o ser reentrenadas con otros conjuntos de datos, con los parámetros de entrenamiento seleccionados no ha surtido mucho efecto en la mejora de la estimación de los valores de posición y orientación.

Como se observó el emplear una gran cantidad de capas en las redes neuronales no siempre es lo adecuado, ya que en cierto modo repercute en el desempeño y genera un mayor esfuerzo computacional, esto puede ser comprobado en la etapa de entrenamiento de la arquitectura *MagicVO*, que tardo casi tres días en completar las 200 épocas y aun así no lograr superar la precisión de las otras dos arquitecturas.

Y por último se observó que de acuerdo con el cumplimiento de los diferentes objetivos específicos se logró llegar a cumplir el objetivo general de realizar un estudio comparativo de las diferentes arquitecturas de aprendizaje profundo para la estimación de la odometría visual. Se ha demostrado que durante el tiempo que tarda la arquitectura *PoseConvGRU* en estimar las posiciones relativas es menor que el resto de las arquitecturas,

haciéndole la arquitectura no solo con mejor capacidad de estimar los valores de posición y orientación sino también en que se puede utilizar quizá en aplicaciones en tiempo real con la limitación de ser utilizada en computadoras de escritorio.

Adicional, al igual que los autores de las arquitecturas estudiadas, se pretende que estos modelos de redes neuronales profundas sean de soporte para los métodos analíticos existentes, ya que han presentado métodos capaces de producir excelentes resultados dando por solucionado el problema de odometría visual. Por lo que existen métodos o modelos que no generen mucha carga computacional y sean lo suficientemente precisos para emplearlos en sistemas con poca cantidad de recursos de procesamiento.

Una vez desarrollada la comparativa el código implementado se encuentra de manera libre con sus respectivos parámetros pre entrenados en el repositorio *GitHub*. Pueden ser utilizados para continuar con el entrenamiento o evaluación de las arquitecturas. Así como su utilización en diferentes aplicaciones de robótica donde se desee estimar la posición del robot a partir de secuencias de imágenes.

Si bien se ha utilizado computador de escritorio para lograr la inferencia de los modelos se espera que en un futuro con los avances tecnológicos se pueda implementar en los sistemas embebidos de *NVIDIA JetsonNano* ya que cuenta con núcleos *CUDA* en menor cantidad que las tarjetas gráficas, si bien hoy en día se puede compilar TensorFlow con soporte a *GPU* en el sistema embebidos aun no puede lograr procesar un gran número de parámetros.

Referencias

[1] Agrawal, P., Carreira, J., & Malik, J. (2015). *Learning to See by Moving*.

[2] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2018). Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-January*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

[3] Agrawal, P., Carreira, J., & Malik, J. (2015). *Learning to See by Moving*.

[4] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2018). Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-January*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

[5] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., & Asari, V. K. (2018). *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. http://arxiv.org/abs/1803.01164

[6] Amidi, A., & Amidi, S. (2017). *CS 230 - Recurrent Neural Networks Cheatsheet*. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

[7] Anwar, A. (2019). *Difference between AlexNet, VGGNet, ResNet, and Inception | by Aqeel Anwar | Towards Data Science*. https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

[8] Borenstein, J, Everett, H. R., Feng, L., Lee, S. W., Byrne, R. H., Borenstein, J., & Feng, L. (1996). Where am I? Sensors and Methods for Mobile Robot Positioning Prepared by the University of Michigan For the Oak Ridge National Lab (ORNL) D&D Program and the United States Department of Energy's Robotics Technology Development Program Within the Environmental Restoration, Decontamination and Dismantlement Project.

[9] Borenstein, Johann, & Feng, L. (1995). UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots.

[10] Brunetti, A., Buongiorno, D., Trotta, G. F., & Bevilacqua, V. (2018). Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. *Neurocomputing*, *300*, 17–33. https://doi.org/10.1016/j.neucom.2018.01.092

[11] Chen, C., Lu, X., Wahlstrom, J., Markham, A., & Trigoni, N. (2019). Deep Neural Network Based Inertial Odometry Using Low-cost Inertial Measurement Units. *IEEE Transactions on Mobile Computing*, 1–1. https://doi.org/10.1109/tmc.2019.2960780

[12] Chen, C., Zhao, P., Lu, C. X., Wang, W., Markham, A., & Trigoni, N. (2018). *OxIOD: The Dataset for Deep Inertial Odometry*. http://arxiv.org/abs/1809.07491

[13] Chung, C., Patel, S., Lee, R., Fu, L., Reilly, S., Ho, T., Lionetti, J., George, M. D., & Taylor, P. (2018). Vggnet. *American Journal of Health-System Pharmacy*, *75*(6), 398–406. https://doi.org/10.2146/ajhp170251

[14] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. http://arxiv.org/abs/1412.3555

[15] Costante, G., Mancini, M., Valigi, P., & Ciarfuglia, T. A. (2016). Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation. *IEEE Robotics and Automation Letters*, *1*(1), 18–25. https://doi.org/10.1109/LRA.2015.2505717

[16] Dawson-Howe, K. M., & Vernon, D. (1994). Simple pinhole camera calibration. *International Journal of Imaging Systems and Technology*, *5*(1), 1–6. https://doi.org/10.1002/ima.1850050102

[17] Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazırbaş¸, H., Golkov, V., Van Der Smagt, P., Cremers, D., & Brox, T. (2015). *FlowNet: Learning Optical Flow with Convolutional Networks*.

[18] Du, F., & Brady, M. (1993). Self-calibration of the intrinsic parameters of cameras for active vision systems. *IEEE Computer Vision and Pattern Recognition*, 477–482. https://doi.org/10.1109/cvpr.1993.341087

[19] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robotics and Automation Magazine*, *13*(2), 99–108. https://doi.org/10.1109/MRA.2006.1638022

[20] Engel, J. (2013). Semi-Dense Visual Odometry for a Monocular Camera * 2013 IEEE International Conference on Computer Vision. 1449–1456. https://doi.org/10.1109/ICCV.2013.183

[21] Engineering, D. of M. and P. (2011). *ASL Datasets*. https://projects.asl.ethz.ch/datasets/doku.php?id=home

[22] Engineering, D. of M. and P. (2020). Homepage - D-MAVT – Department of Mechanical and Process Engineering | ETH Zurich. https://mavt.ethz.ch/

[23] Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T., & Philbrick, K. (2017). Toolkits and Libraries for Deep Learning. In *Journal of Digital Imaging* (Vol. 30, Issue 4, pp. 400–405). Springer New York LLC. https://doi.org/10.1007/s10278-017-9965-6

[24] Forster, C., Pizzoli, M., & Scaramuzza, D. (2014). SVO: Fast semi-direct monocular visual odometry. *Proceedings - IEEE International Conference on Robotics and Automation*, 15–22. https://doi.org/10.1109/ICRA.2014.6906584

[25] Fu-chao, W., Fu-chao, W., Guang-hui, W., & Zhan-yi, H. (2003). *A Linear Approach for Determining Intrinsic Parameters and Pose of Cameras from Rectangles*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.3475

[26] Geiger, A., Urtasun, P. L., & Raquel, U. (2012). *The KITTI Vision Benchmark Suite*. http://www.cvlibs.net/datasets/kitti/

[27] Google. (2020). *Descending into ML: Training and Loss | Machine Learning Crash Course*. https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

[28] Grandón-Pastén, N., Aracena-Pizarro, D., & Tozzi, C. L. (2007). RECONSTRUCCIÓN DE OBJETO 3D A PARTIR DE IMÁGENES CALIBRADAS. *Ingeniare. Revista Chilena de Ingeniería*, *15*(2), 158–168. https://doi.org/10.4067/S0718-33052007000200006

[29] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5–6), 602–610. https://doi.org/10.1016/j.neunet.2005.06.042

[30] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. *International Conference in Machine Learning*, *4*(7540), 14. http://arxiv.org/abs/1502.04623

[31] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, *187*, 27–48. https://doi.org/10.1016/j.neucom.2015.09.116

[32] Handa, A., Bloesch, M., Pătrăucean, V., Stent, S., McCormac, J., &

Davison, A. (2016). Gvnn: Neural network library for geometric computer vision. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9915 LNCS*, 67–82. https://doi.org/10.1007/978-3-319-49409-8_9

[33] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. https://doi.org/10.1109/CVPR.2016.90

[34] Hecht-Nielsen, R. (1989). Neurocomputer Applications. In *Neural Computers* (pp. 445–453). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-83740-1_45

[35] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[36] Hochreiter, S., Younger, A. S., & Conwell, P. R. (2001). Learning to learn using gradient descent. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2130, 87–94. https://doi.org/10.1007/3-540-44668-0_13

[37] Howard, A. (2008). Real-time stereo visual odometry for autonomous ground vehicles. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 3946–3952. https://doi.org/10.1109/IROS.2008.4651147

[38] Huang, S., Wang, Z., Dissanayake, G., & Frese, U. (2009). *Iterated SLSJF: A Sparse Local Submap Joining Algorithm with Improved Consistency*. https://pdfs.semanticscholar.org/6e41/e7840bf378709098b624ed299d637c6fc63e.pdf

[39] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., & Brox, T. (2017). *FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks*. http://lmb.

[40] InvenSense. (2016). *ICM-20600*. www.invensense.com

[41] Jeong, W. Y., & Lee, K. M. (2006). Visual SLAM with line and corner features. *IEEE International Conference on Intelligent Robots and Systems*, 2570–2575. https://doi.org/10.1109/IROS.2006.281708

[42] Jiao, J., Jiao, J., Mo, Y., Liu, W., & Deng, Z. (2018). MagicVO: End-to-End Monocular Visual Odometry through Deep Bi-directional Recurrent Convolutional Neural Network. http://arxiv.org/abs/1811.10964

[43] Jordan, J. (2017). *Convolutional neural networks*. https://www.jeremyjordan.me/convolutional-neural-networks/

[44] Kitt, B., Geiger, A., & Lategahn, H. (2010). Visual Odometry based on Stereo Image Sequences with RANSAC-based Outlier Rejection Scheme.

[45] Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*. https://doi.org/10.1109/ISMAR.2007.4538852

[46] Konda, K., & Memisevic, R. (2005). *Learning Visual Odometry with a Convolutional Network*. https://doi.org/10.5220/0005299304860490

[47] Kreuzig, R., Ochs, M., & Mester, R. (2019). DistanceNet: Estimating Traveled Distance from Monocular Images using a Recurrent Convolutional Neural Network.

[48] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. http://code.google.com/p/cuda-convnet/

[49] Li, M., Hong, B., Cai, Z., & Luo, R. (2008). Novel Rao-Blackwellized Particle Filter for Mobile Robot SLAM Using Monocular Vision.

[50] Li, R., Wang, S., Long, Z., & Gu, D. (2018). UnDeepVO: Monocular Visual Odometry Through Unsupervised Deep Learning. *Proceedings - IEEE International Conference on Robotics and Automation*, 7286–7291. https://doi.org/10.1109/ICRA.2018.8461251

[51] Maddern, W., Pascoe, G., Gadd, M., Barnes, D., Yeomans, B., & Newman, P. (2015). *Real-time Kinematic Ground Truth for the Oxford RobotCar Dataset*. https://www.novatel.com/products/span-gnss-inertial-systems/

[52] Maddern, W., Pascoe, G., Linegar, C., & Newman, P. (2017). 1 year, 1000 km: The Oxford RobotCar dataset. *The International Journal of Robotics Research*, *36*(1), 3–15. https://doi.org/10.1177/0278364916679498

[53] Martinez-Cantin, R., & Castellanos, J. A. (2005). Unscented SLAM for large-scale outdoor environments. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 328–333. https://doi.org/10.1109/IROS.2005.1545002

[54] Muller, P. M. (2016). Optical Flow and Deep Learning Based Approach to Visual Optical Flow and Deep Learning Based Approach to Visual Odometry Odometry. https://scholarworks.rit.edu/theses

[55] Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, *31*(5), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671

[56] Mur-Artal, R., & Tardos, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, *33*(5), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103

[57] Newcombe, R. A., Lovegrove, S. J., & Davison, A. J. (2011). DTAM: Dense tracking and mapping in real-time. *Proceedings of the IEEE International Conference on Computer Vision*, 2320–2327. https://doi.org/10.1109/ICCV.2011.6126513

[58] Nistér, D., Naroditsky, O., & Bergen, J. (2004). Visual odometry. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *1*. https://doi.org/10.1109/cvpr.2004.1315094

[59] NovAtel. (2017). *SPAN CPT7 Performance Specifications*. https://docs.novatel.com/OEM7/Content/Technical_Specs_Receiver/SPANCPT7_Performance_Specs.htm

[60] Olah, C. (2015). *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[61] OpenCV. (2017). *OpenCV: Optical Flow*. https://docs.opencv.org/3.4.0/d7/d8b/tutorial_py_lucas_kanade.html

[62] OXTS. (2013). RT3000 v3 - GNSS-aided inertial navigation system for automotive testing. https://www.oxts.com/products/rt3000/

[63] Perez, J., Serrano, C., Acha, B., Serrano, T., & Linares, B. (2013). *Red neuronal convolucional rápida sin fotogramas para reconocimientos de dígitos*. http://digital.csic.es/handle/10261/84753

[64] Qin, T., Li, P., & Shen, S. (2018). VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, *34*(4), 1004–1020. https://doi.org/10.1109/TRO.2018.2853729

[65] Saffiotti, A. (1997). The uses of fuzzy logic in autonomous robot navigation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, *1*(4), 180–197. https://doi.org/10.1007/s005000050020

[66] Sak, H., Senior, A., & Beaufays, F. (2014). Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. http://arxiv.org/abs/1402.1128

[67] Scaramuzza, D., & Fraundorfer, F. (2011). Tutorial: Visual odometry. *IEEE Robotics and Automation Magazine*, *18*(4), 80–92. https://doi.org/10.1109/MRA.2011.943233

[68] Sen, Ronald, Hongkai, & Niki. (2017). DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. *Proceedings - IEEE International Conference on Robotics and Automation*, 2043–2050. https://doi.org/10.1109/ICRA.2017.7989236

[69] Singh, B., Marks, T. K., Jones, M., Tuzel, O., & Shao, M. (2016). A Multi-Stream Bi-Directional Recurrent Neural Network for Fine-Grained Action Detection.

[70] Steffen, L., Alec, G., Thomas, F., & Kevin, M. (2017). *Visual Odometry using Convolutional Neural Networks*. https://www.researchgate.net/publication/322525765_Visual_Odometry_using_Convolutional_Neural_Networks

[71] Strobl, K. H., & Hirzinger, G. (2011). More accurate pinhole camera calibration with imperfect planar target. *Proceedings of the IEEE International Conference on Computer Vision*, 1068–1075. https://doi.org/10.1109/ICCVW.2011.6130369

[72] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-*

June-2015, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[73] Venegas, J. (2009). *Encoders*. http://support.automationdirect.com/docs/absolute_encoders.pdf

[74] Wang, R., Pizer, S. M., & Frahm, J.-M. (2019). Recurrent Neural Network for (Un-)supervised Learning of Monocular Video Visual Odometry and Depth.

[75] Watson, J. D. (2020). AFIT Scholar AFIT Scholar Improved Ground-Based Monocular Visual Odometry Estimation Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks using Inertially-Aided Convolutional Neural Networks. https://scholar.afit.edu/etd/3622

[76] Wu, N. (2017). *Tensorflow detection model zoo*. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

[77] Zhai, G., Liu, L., Zhang, L., & Liu, Y. (2019). PoseConvGRU: A Monocular Approach for Visual Ego-motion Estimation by Learning. *Pattern Recognition*, *102*. http://arxiv.org/abs/1906.08095

[78] Zhang, Y. (2006). Towards piecewise-linear primal neural networks for optimization and redundant robotics. *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, ICNSC'06*, 374–379. https://doi.org/10.1109/icnsc.2006.1673175

[79] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(11), 1330–1334. https://doi.org/10.1109/34.888718

Comparative analysis of Deep learning architectures for visual odometry

Eduardo Tayupanta Zúñiga

Universidad Internacional de la Rioja, Logroño (España)

September 2020

Abstract

One of the main lines of research in the areas of robotics is the development of techniques that allow robots to be able to move autonomously in different environments. For the robot to obtain position and orientation values, visual odometry techniques have been used, which allow the estimation of position and orientation through image sequences. In the present study, the implementation, training and performance comparison of three different visual odometry architectures that use CNN neural network models and their combination with LSTM or GRU have been carried out. It has been shown that the use of deep learning in visual odometry can estimate values close to the real values of the image sequences, therefore, the results are competitive with the analytical methods developed for the estimation of position and orientation.

Keywords

Deep learning, GRU, LSTM, SLAM

I. Introduction

The autonomous navigation of an agent has been an area of research in robotics, which seeks to make the robot capable of moving in real world environments. Different methods have been developed from analytical models to machine learning models for estimating robot position and orientation. Visual odometry studies using deep learning models allow the estimation of position and orientation from a sequence of images. Although there are analytical models that can solve the problem of visual odometry, it is observed that deep learning models can have competitive results without having a large number of parameters for their processing, which depends largely on the geometries presented by the cameras with which the image sequences were captured.

Analytical models need the intrinsic properties of cameras, as well as certain types of conditions when capturing images, such as: resolution, amount of light and variation in contrast, among others. Consequently, neural network models have been shown to have the ability to extract features during their training, thus making them attractive for position and orientation estimation from image sequences [1].

Convolutional neural networks were developed in the late 1990s, but in recent years they have had great success in image processing tasks, as they can extract specific characteristics from them. Its development is more complex when using many convolutional layers in cascade, in this way, in each convolutional layer the neural network is capable of learning certain types of characteristics [61].

Recurrent neural networks can extract information through time series, that is, they allow us to treat the dimension of time. These models were developed in the 80s, in the beginning, they were difficult to deal with since they required a great computational effort. With technological advances in recent years they have become more accessible and their popularity has been growing [9].

LSTM (Long Short-Term Memory) neural networks are an extension of recurrent neural networks; they manage to expand memory to learn through events that have passed over time. This type of model maintains information that can be modified depending on the importance to which the information is assigned [64]. Another variant of recurrent neural networks are the GRU (Gated Recurrent Unit), which appeared in 2014 and retain the same principle as the LSTM, with the difference that they are computationally more efficient than the LSTM [12].

Once the previous studies to solve the problem of visual odometry have been analyzed, the comparative study of three types of architectures that use CNN, LSTM and GRU is presented. Each architecture was implemented in Python with the TensorFlow library, since there is no source code developed by the researchers. The Python programming language is characterized by being versatile and practical. Furthermore, the TensorFlow library allows you to implement neural network architectures without the need to use numerous lines of code.

II. State of the Art

The main research trends in robotics is the development of techniques for autonomous navigation in real world environments [63]. For a robot to be completely autonomous, it must be able to move in an unknown environment while building an incremental map and estimating its position Figure 1, the application of simultaneous localization and mapping techniques (SLAM Simultaneous Localization and Mapping) provide the means for a mobile robot to be truly autonomous [17].

The performance improvements of applications developed by computer vision, the combination of specialized hardware such as stereo cameras, RGB-D cameras and sensors capable of measuring distance from the emitter to the object, have made possible the implementation of processes in real time providing depth estimates. of each pixel [36].

Methods based on dispersed characteristics that focus on the stereotypical vision, present a great disadvantage in precision by not including the time transition in each frame. To overcome the drawback, an alternative called visual SLAM was developed that continuously optimizes the feature map by applying an extended Kalman filter [39], but due to computational complexity and the dependence of paired frame synchronization, it generates errors in motion estimation [51].

Fig. 1. ORB-SLAM 2 [56].

Monocular SLAM improves uncertainty when using a particle filter, due to its high computational complexity, which generates inconveniences in real-time applications [47] PTAM-based models (parallel tracking and mapping) significantly improves real-time, the model uses keyframe extraction [43].

The methods based on the union of local sub-maps have improved the agility of a SLAM system, dispersed information filters are used capable of reducing the computational cost for the construction of global maps, the method is resistant to blurring caused by fast movement. of capturing the images [37]. A last approach develops SLAM based on monocular cameras, the authors focus their research on mapping, localization and closure of loops. It uses ORB functions and describes a new Essential Graph concept that speeds up the loop closure correction process (detection with correction). It can be executed in CPU having a great performance in real time [53].

Next, different methods for the calculation of visual odometry are presented, such as the Real-time stereo visual odometry for autonomous ground vehicles study [35] that describes a visual odometry algorithm from consecutive pairs of stereo images, it does not make assumptions about the camera movement and operates with dense disparity images, the error in the position estimate is 0.25% of the total distance traveled using images of 512 × 384 dimension.

For the method of extracting geometric characteristics from monocular images, rigorous mathematical processing is applied, in order to obtain the position from visual odometry [40]. However, chambers have internal properties called intrinsic parameters, which are values derived from epipolar geometry [16].

The direct methods for visual odometry do not use calculations to determine descriptors or key points in the images, their main objective being to estimate movement from the illumination presented by the pixel, the methods assume the invariance of the grayscale image, which provides greater robustness and precision.

Monocular camera-based methods perform calculations to estimate camera posture using a semi-dense depth map, it can be executed in real time on a CPU, but it generates position uncertainty by not employing loop closure detection [18]. Another method is based on the probabilistic prediction of the depth of each pixel that is sensitive to changes in illumination, but reduces the uncertainty in the estimation of the position because it is not based on the uniformity of intensity of the pixel [55].

The semi-direct monocular visual odometry method eliminates the costly task of extracting features and coincidences between images for motion estimation, extracts blocks in the image improving robustness, implements probabilistic mapping methods for 3D point estimation, and high 55-frame processing per second for an on-board system and at 300 frames per second for a desktop computer, it has proven robust in scenarios with high frequency and little repetitive texture [22].

Being a fast method, it succeeds when the camera moves slowly since the search for the gradient can result in the algorithm failing, so that the direct method is sensitive to the change in lighting. The problems of conventional methods to estimate the position and orientation from consecutive images such as: extraction of image characteristics, high computational cost, have sensitivity to lighting, a high degree of textures are required and a high number of frames per second [40].

Currently, deep learning has been widely developed in the area of computer vision, the application of convolutional neural networks has generated several applications with monocular cameras such as: image classification, object detection, facial recognition, among others. Despite the great success that the use of deep learning with images has had, few alternatives have been developed to solve the disadvantages presented by traditional methods of visual odometry [68].

Deep learning

A subset of machine learning algorithms is deep learning. Deep learning are algorithms that allow large data sets to be processed to efficiently solve traditional artificial intelligence problems. Recent improvements in hardware have strengthened the study of different neural network architectures, allowing lower computational cost and reducing processing time. They have been successful in learning transfer, depth estimation, visual odometry, computer vision, control systems, mobile robotics, among others [29].

Deep learning has been widely used in the area of computer vision since they use convolutional neural networks for the detection and extraction of patterns in images [8], thanks to parallel computing it has strengthened the application of convolutional networks and recurrent neural networks, motivating real-time solutions for mobile robotics problems [76].

Convolutional neural network

Convolutional neural networks have proven capable of solving several computer visions challenges, as they are capable of outperforming machine learning machines. A convolutional neural network is a type of artificial neural network for image processing as it can process data per pixel.

The term convolution refers to the matrix operation of applying filters that traverse the image [68].

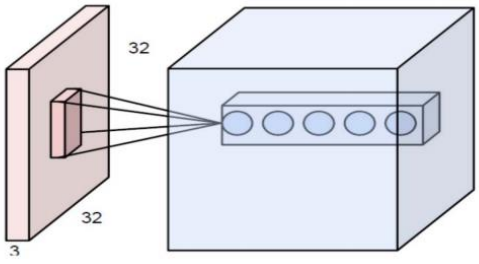

Fig. 2. Convolution operation on CNN [2].

Figure 2 shows a convolutional layer, the layer can be

visualized as small squares called kernels that run through the image in search of patterns, if the pixel matches the kernel patterns the result is a positive value and if it does not match a value returns close to zero [2]. A main parameter for convolutional layers is the use of stride, which refers to the kernel jumping between pixels.

CNNs are made up of multiple convolutional layers, non-linear layers, grouping layers, and fully connected layers [3], which have been very successful for feature extraction, object recognition without taking lighting conditions into account thanks to improvements of processing speed since the computation can be parallelized thanks to the use of specialized hardware [46]

At present there are complex convolutional neural network architectures that have been trained with a large amount of data for specific applications. Pre-trained models can be used for object recognition such as ResNet [31], SENet, etc., or as model's detection systems such as SDD Mobilenet, SDD Inception, among others [74]. Since visual odometry is more related to geometry, CNN architectures like VGGNet [12] and GoogleNet.

Recurrent neural network

RNN recurrent neural networks have been developed since the 1990s, their design allows the learning of sequential or variable time patterns, with a feedback system with closed loop connections, allowing the previous outputs to be used as inputs while they are maintained the hidden states as shown in Figure 3 [32].

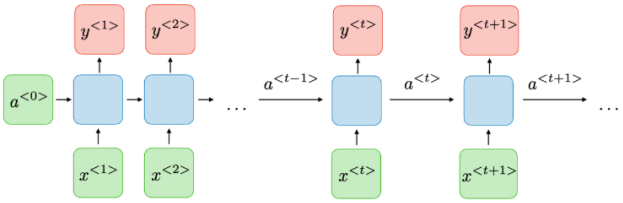

Fig. 3. Architecture of an RNN [6].

Where for each step-in time t there is an activation $a^{<t>}$ that produces an output $y^{<t>}$, this transition of states is expressed according to the following equations.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Where W_{aa}, W_{ax}, W_{ya}, b_a and b_y refer to the coefficients that are temporally shared and g_1, g_2 refers to the activation functions within each state [64].

The application of recurrent neural networks has been shown to significantly outperform models such as traditional convolutional neural networks. Improvements in precision are presented, the input to the model can be of any length, the size of the model does not depend on the length of the input, for the estimates it has historical information which allows the weights to be shared over time. But there are still problems in the implementation due to its high computational cost, it is not possible to access information that has been processed for a long time [66] applications such as the generation of images employ RNN presenting a framework of self-coding sequences to iteratively build complex images [28].

LSTM

The LSTM architecture [33] was developed to improve the existing error in RNNs. It was shown that the delays present in the architecture were inaccessible since the backward propagation error decays or explodes exponentially [34].

The LSTM layers are made up of multiple connected memory blocks. These blocks contain several memory cells connected in a recurring way and three multiplicative units called input, output and forget gate that allow continuous read, write and reset operations of the memory cell, so that the input of each cell is multiplied by the activation unit of the entrance gate, the output is multiplied by the exit gate and the values corresponding to the previous cells are multiplied by the forget gate [27].

LSTMs have the same internal structure as RNNs, but with the variation that the repeating module has 4 interacting layers. Of which 3 allow status control, as shown in Figure 4.

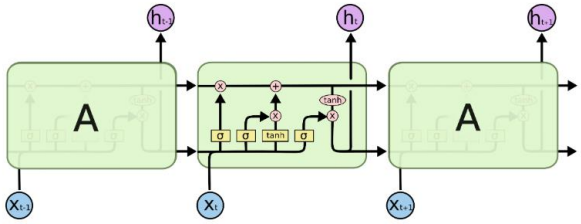

Fig. 4. LSTM internal structure [60].

For example, given an input x_k at time k, with the hidden state h_{k-1} in memory cell c_{k-1} of the previous LSTM cell, the LSTM is updated according to the following equations:

$$i_k = \sigma(W_{xi}x_k + W_{hi}h_{k-1} + b_i)$$

$$f_k = \sigma\left(W_{xf}x_k + W_{hf}h_{k-1} + b_f\right)$$

$$g_k = \tanh\left(W_{xg}x_k + W_{hg}h_{k-1} + b_g\right)$$

$$c_k = f_k \odot c_{k-1} + i_k \odot g_k$$

$$o_k = \sigma(W_{xo}x_k + W_{ho}h_{k-1} + b_o)$$

$$h_k = o_k \odot \tanh(c_k)$$

Where $\odot$ refers to the product per element of each vector, σ is the $sigmoid$ activation unit, $tanh$ is the non-linear hyperbolic tangent, W refers to the corresponding weight matrices, b refers to the bias and the terms i_k, f_k, g_k, c_k and o_k refer to the input gate, forget gate, memory cell and output gate for time k respectively [66].

Bi-LSTM

Researchers indicate that RNNs can memorize in the short term, which allows connecting information that has been retained in memory for the current process [40].

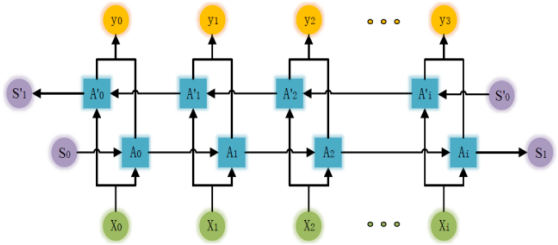

Fig. 5. Estructura interna Bi-LSTM [42].

Since the information is x_t at time t, the RNN is updated using the following equations:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

Where:

- h_t refers to the state variable of the layer at time t.
- y_t refers to the result variable.
- W_{hh} and W_{xh} refer to the weight matrices of the layer.
- b_h and b_y refer to polarization vectors of the layer.

- f refers to the activation function.

The RNNs generate problems for the descent of the gradient and its cancellation in the reverse propagation process, to solve the inconvenience of the RNN the LSTM model is applied. For this, additional gates are added to the RNN model, but the LSTM models can only contain a memory function for the forward sequence Figure 5.

Loss function

In order to achieve the position prediction, the conditional probability calculation for the visual odometry problem, investigations present the calculation as follows. Since $y_t = (y_1, y_2, \dots, y_t)$ the pose to be predicted and $x_t = (x_1,\ x_2, \dots, x_t)$ the sequences of monocular frames, the calculation to be performed is as follows:

$$p(Y_t|X_t) = p(y_1, y_2, \dots, y_n|x_1, x_2, \dots, x_n)$$

The researchers propose the calculation of the optimal θ^* value, maximizing $p(Y_t|X_t)$, in this way it is intended to make the prediction of the rotation φ_k in Euler angles and the translation p_k.. Where for each pair (p_k, φ_k) corresponds a (p'_k, φ'_k) [40]. The loss function is denoted as:

$$\theta^* = \arg_\theta min \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{3} \|p_k - p'_k\|_2^2 + w\|\varphi - \varphi'_k\|_2^2$$

Where, $\|.\|$ refers to the norm, k refers to the number of states in each position, as there are three Euler angles, k goes from 1 to 3, N refers to the number of image sequences per batch, w refers to weight, to balance the position and translation and θ^* the value is obtained by applying gradient descent iteratively.

Visual odometry algorithms with deep learning

With the rise of deep neural networks, methods have been generated to solve visual odometry problems, they do not require a large geometric calculation like the methods described above, but rather descriptive and concise models.

The application of models based on deep neural networks provides results in the prediction of direction changes and the speed of the camera. Models use machine learning to predict movement and depth from a data set.

Researchers used convolutional neural networks to learn optical representations of pairs of consecutive images, the model was able to process images with blur due to movement between images and sudden changes in lighting, using RGB-D data sets as shown in Figure 6. To implement a neural network capable of estimating depth, global transformations, pixel transformation, generating a GVNN software library (Geometric Vision with Neural Network).

Fig. 6. Dense optical flow field [15].

Compared to traditional algorithms for calculating pose estimation through monocular images, deep learning algorithms have shown that they can be efficient for extracting motion information between frame sequences, replacing the complexity of mathematical formulas and the computational effort to extract characteristics to look for confidences in the following images.

Being deep learning a simple and intuitive method to solve visual odometry problems, recent research has shown that ignoring the relationships between the sequences of frames leads to a loss of precision, so the application of recurrent neural networks in combination with convolutional neural networks with unidirectional long-term memory units. The results obtained showed that they had problems when using sequences of images different from the training set, since the model was not generalized.

FlowNet

Neural network models have proposed the solution of visual odometry based on the optical flow, creating interpolation and extrapolation between the image frames. The architecture is observed in Table I with the implementation of convolutional neural networks to predict flow fields from a sequence of input images.

TABLE I

FLOWHOT STRUCTURE

LAYER	KERNEL	PADDING	STRIDE	CHANNELS
Conv1	7×7	3	2	64
Conv2	5×5	2	2	128
Conv3	5×5	2	2	256
Conv3_1	3×3	1	1	256
Conv4	3×3	1	2	512
Conv4_1	3×3	1	1	512
Conv5	3×3	1	2	512
Conv5_1	3×3	1	1	512
Conv6	3×3	1	2	1024

Convolutional neural networks have been used for this, for the dense prediction of pixels the model contains an expansion part to intelligently refine the flow for high resolution. The network refers to contracting and expanding the parts that are trained using back propagation. The architecture shows that at the entrance to the network the images are concatenated one followed by the other, thus forming six channels, which corresponds to the pair of images in RGB format, which allows the network to be able to extract the movement information by itself. .

Obtaining true values has been identified as a difficult task since the true real-world correspondences between pixels have not been easily determined since a small number of data sets are available to be used for this task.

Approaches to the calculation of visual odometry through monocular cameras, is the use of convolutional neural networks CNN in combination with recurrent networks LSTM or with bidirectional networks with long-term memory units Bi-LSTM, which generates a pose of absolute scale with 6 degrees of freedom from sequence of continuous monocular images.

Data sets

In general, it starts from data sets, which are available to other researchers for their research, thus they can be divided into training and validation data sets, for which the following alternatives used by different authors are presented [37] as :

The KITTI [28] website provides data sets for stereo vision, optical flow, visual odometry, object detection, etc. The KITTI

Visual Odometry data set provides 22 stereo sequences, additional data from the data set is that the image storage rate is at 10 fps, the speed with which the car was going to travel the stage is 90 km / h .

The Department of Mechanical and Process Engineering of the Federal Polytechnic School of Zurich [19], provides ASL data sets, where its main objective is to provide data for the robotics community for development, comparison of results and evaluation. The images are captured by a drone where the images contain severe concern for camera shake.

Another set of freely available data is from the Department of Computer Science at the University of Oxford [11], where it is intended to exploit inertial pedestrian navigation data to estimate movement and location. The dataset presents measurements based on IMU (Inertial Measurement Unit) of different scales of mobile devices with different resources, provides guidelines for the use of raw IMU data.

III. Objectives and Methodology

According to the analysis presented in the previous sections, it has been observed that the use of deep learning has managed to be an alternative for the problem of visual odometry. Therefore, the present work aims to carry out a comparative study of the different deep learning architectures for the estimation of visual odometry.

In order to achieve the objective, the following specific objectives have been set:

1. Identify and describe the deep learning architectures available for visual odometry.

2. Identify the data sets available for visual odometry training and testing.

3. Define success metrics for visual odometry.

4. Implement the selected architectures for visual odometry.

5. Calculate and compare the training time between the different architectures for the same data set.

6. Evaluate different architectures with different data sets.

7. Compare the result of estimating the visual odometry of the different architectures.

To carry out the objectives of this study, it has been separated into phases:

1. A thorough study of visual odometry architectures will be carried out. In this phase, a bibliographic search will be developed focused on deep learning models that can contribute to the estimation of position from a sequence of images. Additionally, the different parameters such as their loss functions, activation functions and optimizers used in each architecture will be identified.

2. Existing data sets for visual odometry will be analyzed in order to choose the optimal one for this study.

3. In this phase you will focus on the following points:

3.1. Development and implementation of architectures in Python code.

3.2. Load optimization of selected data sets.

3.3. Data processing optimization for the computer used for the present study.

4. In this last phase the following points will be carried out:

4.1. Execution of the visual odometry architectures with the selected data set.

4.2. The comparison of the different models in execution time will be developed, as well as precision in the estimation of the position.

4.3. Once the comparison has been developed, the optimal model for position estimation from a sequence of color images will be determined.

IV. Contribution

A comparison of the DeepVO, MagicVO and PoseConvGRU models is presented both in the number of parameters that each architecture counts, estimation of position and orientation in the test data sets, architecture inference time, etc.

For the training of the different architectures, the KITTI Visual Odometry [28] data set has been chosen as discussed in the previous section.

11 sequences of images are presented with their respective truth value, each sequence of images has two sets of left and right images, since stereo cameras with a dimension of 1241 × 376 pixels have been used.

As mentioned in the previous section, each sequence represents different routes, which are found in environments where static and dynamic objects are observed during the route. Therefore, the set of images from the left camera of sequences 00, 02, 04, 06, 08 and 10 has been used as they are relatively long sequences. With a total of 15846 images in total for training and sequences 03, 05 and 07 have been used for evaluation.

The DeepVO, MagicVO and PoseConvGRU architectures have been implemented in Python, the TensorFlow high-performance library was used with the Keras submodule for the creation of deep neural networks because it has a wide variety of documentation and support to implement different network models neuronal.

For the selected data set, code has been created so that the data loading pipeline is dynamic at the time of training, this is because storing all the sequences in RAM is inefficient and saturates the computer's resources.

For this, the TensorFlow library tf.data has been used, which allows the dynamic loading of data and the selection of random images during the training of deep neural networks. So, a mini batch of 8 pairs of images is loaded into memory to be processed, during that time the next mini batch is loaded into memory and so on until all the training data sequences are completed. The training of each architecture was carried out for 200 epochs to make a comparison at par with the same data set.

To speed up the training, the pre-trained FlowNet model has been used, since it is the first layer in which the processing with convolutional neural networks is carried out. The pre-trained FlowNet model has the peculiarity that it was trained with Leaky ReLU activation units with the alpha coefficient of 0.1, unlike those developed in the DeepVO, MagicVO and PoseCnvGRU architectures that use ReLU activation functions in the convolution layer. .

Additionally, the activation units in the layers with neural networks completely connected to Leaky ReLU have been modified to help the convergence of the models for the estimation of visual odometry. The last layer that has the Linear activation unit has also been modified to generate the position and relative rotation values.

DeepVO

The DeepVO application uses convolutional neural networks in combination with recurrent neural networks RCNN, where not

only is the model capable of learning representations of effective characteristics for the visual odometry problem, but it is also capable of modeling sequential dynamics with the use of the Recurrent neural networks.

Table II presents a resizing of the images to 1280 × 384 × 3, the pair of images is applied to the input of the model, the CNN layer is responsible for producing visual odometry characteristics, the convolutional layer consists of 9 convolutional layers where takes RGB images as input to the network, then the matrix output is flattened to a vector that will be introduced in the RNN layer to produce sequential learning.

TABLE II

DEEPVO STRUCTURE

LAYER		DIMENSIONS
Stacked images		640 × 192 × 3 640 × 192 × 3
Convolutional layer FlowNet	Conv1	320 × 96 × 64
	Conv2	160 × 48 × 128
	Conv3	80 × 24 × 256
	Conv3_1	80 × 24 × 256
	Conv4	40 × 12 × 512
	Conv4_1	40 × 12 × 512
	Conv5	20 × 6 × 512
	Conv5_1	20 × 6 × 512
	Conv6	10 × 3 × 1024
Layer LSTM	LSTM1	1000
	LSTM2	1000
Fully connected layer		6

It is intended that the model is capable of extracting simultaneous and sequential characteristics through the combination of the CNN and RNN layers since the estimation of the current position can be benefited from information represented in previous pairs of images, for which two LSTM layers of 1000 hidden states each. The RNN face generates estimates at each step based on what was obtained in the CNN layers above the model structure.

MagicVO

The model presents a system that does not require intrinsic parameters of the camera, the calculation of the position is absolute, the combination of convolutional networks with bidirectional networks with long-term memory units that allows more of the property of learning characteristics of the pair of images learns the relationship between image sequences before and after the current frame, thus generating a model with high precision.

Table III presents the architecture for the preprocessing of image sequences; a scale resizing is performed to 640 × 192 × 3. The third channel is superimposed on two adjacent frames to the current one, normalization of the pixel values is carried out, as the first part a CNN is presented followed by the Bi-LSTM layer and finally a completely connected layer.

Hidden layers of RNN are presented with Bi-LSTM, so the correlation between image frames can be used to estimate the pose estimate, the model presents 1000 LSTM nodes forward and 1000 backward.

Subsequently, two fully connected layers are presented to integrate the information from the Bi-LTSM layers to accurately estimate the final pose with 6 degrees of freedom. The result being the Euler angles and the translation points.

TABLE III

MAGICVO STRUCTURE

LAYER		DIMENSIONS
Stacked images		640 × 192 × 3 640 × 192 × 3
Convolutional layer FlowNet	Conv1	320 × 96 × 64
	Conv2	160 × 48 × 128
	Conv3	80 × 24 × 256
	Conv3_1	80 × 24 × 256
	Conv4	40 × 12 × 512
	Conv4_1	40 × 12 × 512
	Conv5	20 × 6 × 512
	Conv5_1	20 × 6 × 512
	Conv6	10 × 3 × 1024
Layer Bi-LSTM	LSTM1	1000
	LSTM2	1000
Fully connected layer		256
		6

PoseConvGRU

It presents a novel recurrent convolutional neural network with the use of GRU architectures, the model is capable of coding short-term motion functions from a pair of sequential images, thus the model propagates information over time.

The CNN input is the RGB images of a pair of images with resized to 1280 × 384 × 3 as shown in Table IV, these pairs of images are subjected to 10 convolutional layers being the last layer of Max Pooling. The use of Max Pooling in the last CNN layer significantly reduces FlowNet output parameters increasing the complexity of training the neural network.

TABLE IV

POSECONVGRU STRUCTURE

LAYER		DIMENSIONS
Stacked images		640 × 192 × 3 640 × 192 × 3
Convolutional layer FlowNet	Conv1	320 × 96 × 64
	Conv2	160 × 48 × 128
	Conv3	80 × 24 × 256
	Conv3_1	80 × 24 × 256
	Conv4	40 × 12 × 512
	Conv4_1	40 × 12 × 512
	Conv5	20 × 6 × 512
	Conv5_1	20 × 6 × 512
	Conv6	10 × 3 × 1024
Layer GRU	GRU	3
Fully connected layer		4096
		1024
		128
		6

The ConvGRU module is capable of storing a long-term visual representation of the sequence of images, since it allows the neural network to learn the intrinsic relationships of poses, the

GRU architecture is used so that the network can remember states of historical events such as geometric relationships from the previous sequences of the image pair, the GRU architecture is used to reduce the number of parameters since it presents a similar performance to the LSTM architecture.

Architectural parameters

The parameters that will be modified during the training of the different architectures are shown, so the parameters of the FlowNet model are not taken into account since the model is in charge of allowing the extraction of the optimal flow of the pairs of images Table V.

TABLE V

ARCHITECTURES PARAMETERS

ARCHITECTURE	PARAMETERS
DeepVO	149506550
MAGICVO	268894342
PoseConvGRU	19002355

V. RESULTS

It has been observed that the KITTI data set is the ideal one for training and testing the different architectures. A computer with a 3.60GHz i7-9700KF CPU, 16GB of RAM, NVIDIA GTX 1660 super 6GB graphics card was used, as the computer has an NVIDIA graphics card, CUDA Toolkit 10.2 was installed to obtain GPU support in TensorFlow. In this way, execution times of the models are optimized due to the parallelization presented by the architecture of the graphics card.

Image Dimensions

Due to the limitations of the equipment, it was decided to reduce the size of the images from 1241 × 376 to 640 × 192, in this way the training time is reduced and the equipment is not saturated during the execution of the architecture training. However, due to the reduction in the dimensions of the images, it could cause loss of information in the training of the architectures because the relationships between pixels would be inconsistent.

Training sequences

The sequences 00, 02, 04, 06, 08 and 10 are used, they have 4541, 4661, 271, 1101, 4071 and 1201 image pairs, the real trajectories.

These sequences have been chosen for training because they present a greater number of images for the different routes, sequence 02 is the one with the largest number of training data. In addition to presenting changes of direction forward on the routes. For the graph, the X and Z axes have been taken due to the arrangement of the video camera axes.

Test sequences

For the evaluation of the different architectures, sequences 03, 05 and 07 are used, with 801, 2761 and 1101 image pairs respectively.

These sequences have been chosen for the evaluation of the models, because they present a smaller number of images for the different routes. In addition to presenting changes of direction forward on the routes. For the graphs, the X and Z axes have been taken due to the arrangement of the video camera.

Architectures training

Sequences 00, 02, 04, 06, 08 and 10 will be used for training the architectures and sequence 01 for evaluation during training for 200 epochs, the progress of the training of the architectures to converge to 0 with the loss function as shown in Figure 7.

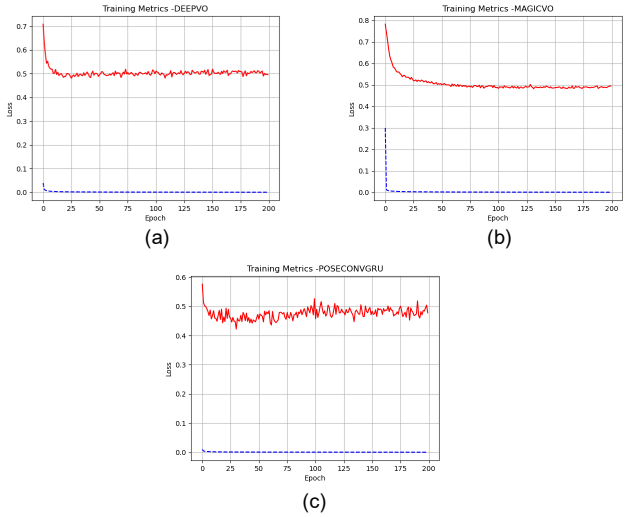

Fig. 7. Loss during training. a) DeepVO, b) MagicVO, c) PoseConvGRU

a) Loss of the DeepVO architecture: the architecture training quickly converges to 0 according to the loss function, where it is observed that the minimum value reached during training is 0.0007523670 with the training data set, while during training evaluation with sequence 01 no improvement is observed and the value oscillates in 0.5, the training time per epoch is approximately 13 minutes for a total of approximately 43 hours to complete 200 epochs.

b) Loss of MagicVO architecture: architecture training slowly converges to 0, where the minimum value it reaches during training is 0.0009443219 with the training data set, while with the evaluation data set it oscillates at 0.5. Being the model that has many training parameters, it was observed that the training time per epoch is approximately 20 minutes, for a total of 67 hours to complete the 200 epochs.

c) Loss of the PoseConvGRU architecture: during the architecture training, it is observed that it quickly converges to 0, the minimum value reached being 0.0006612031 with the training data set, while with the evaluation data it oscillates intermittently at 0.5, the training time per epoch is approximately 6 minutes with a total of approximately 20 hours to complete 200 epochs.

Architectural evaluation

Next, a graph is shown where it can be observed that the architecture that manages to infer values that are close to the truth values is the PoseConvGRU architecture (red dotted line) in sequence 03, being the MagicVO architecture (line with point of green color) which generates estimates very far from the truth values Figure 8.

It is observed that when using the 05 sequence to infer the values of the relative positions, the PoseConvGRU architecture (red dotted line) manages to get close to the truth values of the sequence, greatly surpassing the other architectures, being the one that The inference of the MagicVO architecture (line with a green dot) has a large displacement as seen in Figure 9.

In addition, sequence 07 was used to estimate the relative positions with the different architectures, being the values of the

DeepVO architecture (broken line in blue) that manages to approach the truth values of the sequence, as the previous sequences are observed that the MagicVO architecture (line with a green dot) is the one that is furthest from estimating the expected values as shown in Figure 10.

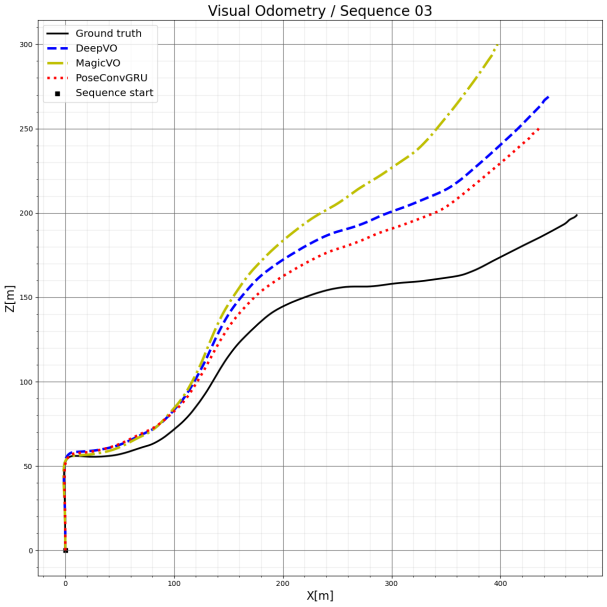

Fig. 8. Comparison of architectures with sequence 03

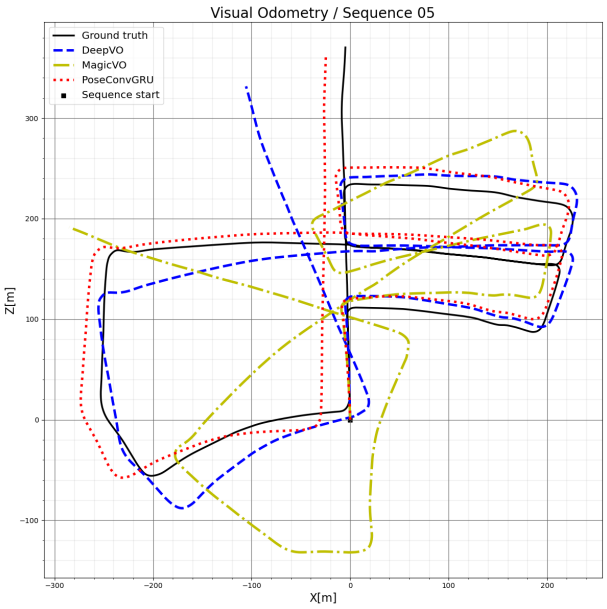

Fig. 9. Comparison of architectures with sequence 05

Inference time of architectures

The FlowNet pre-trained model was used to reduce the time to train the architectures, it is worth mentioning that the estimates were made in mini batches of 8 pairs of images in each iteration.

Due to limitations of the equipment where the present work was developed, the processing of parameters of the FlowNet model on the graphics card and the rest of the layers on the CPU was divided, since there was an excess of parameters that the graphics card was able to process, in this way It is possible to optimize resources both in the processing of the convolutional layers such as the LSTM, GRU and Dense layers. The following describes the inference time of the other layers that follow the FlowNet model.

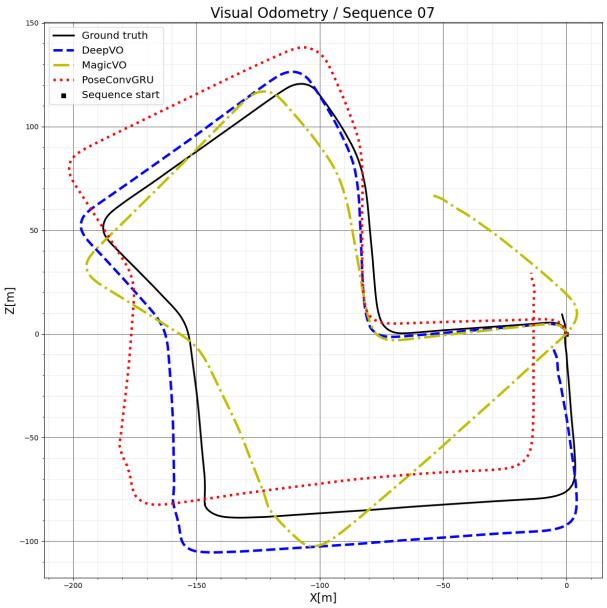

Fig. 10. Comparison of architectures with sequence 07

The inference time of the architectures is shown in Table VI. Where for each pair of images the architectures manage to estimate values in a time in order of milliseconds.

TABLA VI

TIEMPO DE INFERENCIA DE LAS ARQUITECTURAS

ARQUITECTURA	TIEMPO DE INFERENCIA [S]
DeepVO	~0.1489
MAGICVO	~0.1708
PoseConvGRU	~0.0936

As can be seen, the architecture that manages to estimate the positions in a relatively fast time is the PoseConvGRU model with ~ 0.0936 seconds to infer the values, followed by DeepVO with ~ 0.1489 seconds and finally the MagicVO model with ~ 0.1708 seconds, making the architecture MagicVO is the one that takes the longest to process the mini batches of the sequences.

VI. DISCUSSION

As a first point, the layers that were used in each of the architectures and the loss values at the end of the training are analyzed:

DeepVO: it is structured by FlowNet, later it has two LSTM layers of 1000 hidden states each. These LSTM layers are used so that the architecture contains long-term memory units in the future, which allows the architecture to remember future values with which it was trained, the first LSTM layer is observed a total of 126,884,000 parameters, followed by Another 8004000 parameter LSTM layer makes training and inference architecture time consuming.

MagicVO: it is structured by FlowNet in the first stage of the model, later there is a Bi-LSTM layer of 2000 hidden states, the Bi-LSTM layers are formed by connecting two LSTM layers, one that is capable of containing long memory towards the future and another that contains long-term memory to the past, which allows the model to be able to remember future or past values with which

it was trained, this layer has 253,768,000 parameters which makes the architecture very extensive and difficult to process due to the large number of parameters that only this layer has, later there are dense layers of 256 neurons with a Leaky ReLU activation unit and 6 neurons with a Linear activation unit, additionally there are layers

PoseConvGRU: uses FlowNet in the first layers for the estimation of the optical flow of the image sequences, later a MaxPooling layer is used that allows to reduce the number of parameters and further extract the characteristics of the inference of the FlowNet model, later is the GRU layer of 46125 parameters, this layer is similar to the LSTM with the difference that it combines in cells the hidden states in both the forget gate and input gate parameters, making it a layer with better characteristics than the LSTM, which allows the architecture It can maintain information over time without the need for the values to be reset, which means that values that may be relevant in the inference are not eliminated.

As detailed above, it is observed that the architectures have many parameters that are modified during training, which entails a longer training time, inference time and estimation precision. At the end of the training, the architecture that has a minimum loss value is the PoseConvGRU architecture with 0.0006612031, this is because it has a lower number of parameters in the GRU layers where the neural network can learn the sequential relationships between images, presenting higher performance at the end of architecture training.

Absolute error

As the next point, the absolute error present in the evaluation of the model with the sequences 03, 05 and 07 is analyzed. As observed in the previous section, there is no model by which it can perfectly coincide with the truth values, so the error present in the inference of the absolute positions during the sequence has been plotted. Next, the error present in the path of sequence 03 is presented in both the position and orientation values.

As can be seen in Figure 11, as the route progresses, the error propagates in the estimation of the position, this is largely due to the fact that the linear transformations that occur to go from relative position to absolute position involve the use of the angles present in the inference of architectures.

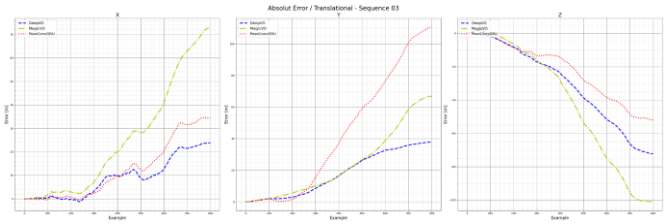

Fig. 11. Absolute error of the architecture sequence 03 - Position

- The position estimation in the X axis of the DeepVO architecture presents an error at the end of the 24-meter tour, unlike the MagicVO architecture that presents an error of 74 meters at the end of the tour.
- The estimation of the position in the Y axis of the DeepVO architecture generates an error of close to 40 meters at the end of the tour, unlike the PoseConvGRU architecture that generates an error of approximately 120 meters.
- To estimate the position in the Z axis, it is observed that the estimated values are greater than the true values, therefore, the values are negative. As can be seen, the PoseConvGRU architecture has a lower error at the end of the route, which is approximately 50 meters.

Being the architecture that has the highest amount of error is MagicVO, followed by PoseConvGRU and the one with the least error in position is the DeepVO architecture in the positions on the X and Y axis, as opposed to the positions on the Z axis. where the one with the least error is PoseConvGRU.

Subsequently, the absolute error present in the inference values of the rotation of sequence 03 in Figure 12 is observed:

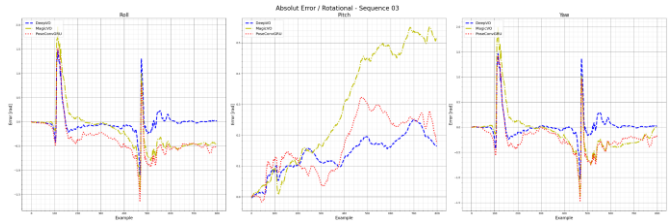

Fig. 12. Absolute error of sequence 03 architectures - Orientation

- For the estimation of rotation with respect to the X (Roll) axis, it is observed that during the sequence run the error does not accumulate, so that there are peaks in the pairs of images 100 to 160 and from 400 to 500. So, the angle estimation error tries to approach 0 rad, with the MagicVO architecture producing the steepest peaks of approximately 2 rad followed by the DeepVO and PoseConvGRU architecture.
- For the estimation of rotation with respect to the Y axis (Pitch), it is observed that the error increases as the sequence progresses, the error of approximately 0.5 rad being the largest value generated by the MagicVO architecture.
- For the estimation of rotation with respect to the Z axis (Yaw), a behavior similar to the estimation error of the X axis is observed, since there are error peaks almost in the same areas 100 to 160 and from 400 to 500. The error being maximum of 2 rad shown by the MagicVO architecture estimate.

The evaluation of the architectures with sequence 05 is presented in Figure 13, where it is observed:

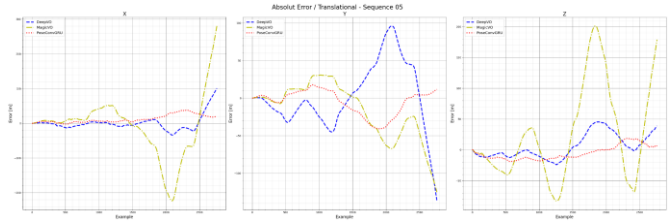

Fig. 13. Absolute error of the architectures sequence 05 - Position

- For the estimation of the position in the X axis, the error of the DeepVO and PoseConvGRU architectures ranges from 0 to 20 meters during the tour, but at the end of the tour the PoseConvGRU architecture generates an error of approximately 10 meters that is less than the others. architectures.
- For the estimation of the position in the Y axis, it is observed that there are oscillations which would depend to a great extent on the route, it has several curves that close circuits (loops). The estimation of the DeepVO architecture is one of the architectures that generates the highest error peaks, the highest being approximately 100 meters.
- For the estimation of the position in the Z axis, it is observed that the MagicVO architecture is the one that generates the greatest oscillation of errors that go up to 200 meters, this can be caused by the number of training epochs.

As can be seen in Figure 14, the absolute orientation estimation error varies independently of the architecture, but it can be observed that:

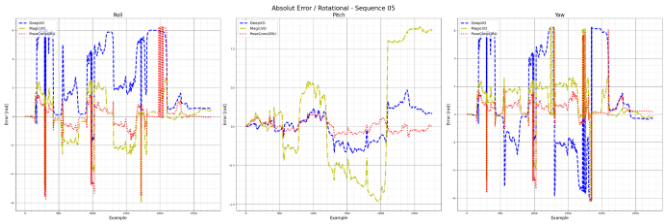

Fig. 14. Absolute error of the architectures sequence 05 - Orientation

- For the estimation of the orientation in the X axis (Roll), the DeepVO architecture is the one that generates the highest number of error peaks up to 6 rad, but at the end of the sequence it manages to reduce the error to less than 1 rad. The MagicVO architecture generates errors that go up to 2 rad, as opposed to 3 peaks that go up to 6 rad, but the PoseConvGRU architecture has the same behavior as the MagicVO architecture, unlike at the end of the sequence it reaches a value that is almost 0 rad.
- For the estimation of the orientation in the Y axis (Pitch), the MagicVO architecture exists oscillations that reach approximately 1.8 rad, unlike the DeepVO architecture, it generates errors less than 0.5 rad.
- For the estimation of the orientation in the Z axis (Yaw) independent of the architectures, the errors range up to 6 rad, but the one with the greatest fluctuations is the DeepVO architecture.

As the last evaluation sequence of the architectures, sequence 07 was used, where Figure 15 is observed that:

Fig. 15. Absolute error of the architectures sequence 07 - Position

- For the estimation of the position in the X axis, the error of the DeepVO architecture is the one that generates the least error that reaches approximately 15 meters, unlike the PoseConvGRU architecture that reaches approximately 35 meters.
- For the estimation of the position on the Y axis, it is observed that there are oscillations, the DeepVO architecture being which generates an error greater than 60 meters at the end of the sequence, so that the error as the sequence progresses the error grows .
- For the estimation of the position in the Z axis, it is observed that the MagicVO architecture is the one that generates the greatest error of up to 80 meters. The DeepVO and PoseConvGRU architectures generate errors smaller than 20 meters, the PoseConvGRU architecture being the one that generates values greater than the real value.

Next, the calculation of the absolute error of the orientation is shown where it is observed that the absolute error varies by sections.

- For the estimation of the orientation in the X axis (Roll), the MagicVO architecture is the one that generates the highest number of error peaks of up to 6 rad. The DeepVO architecture generates errors that reach up to 6 rad in the same way as the PoseConvGRU architecture. But at the end of the sequence, the architectures manage to reduce the error to less than 1 rad.

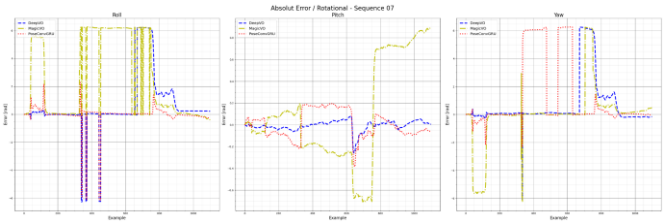

Fig. 16. Absolute error of sequence 07 architectures - Orientation

- For the estimation of the orientation in the Y axis (Pitch), the MagicVO architecture exists oscillations that reach approximately 0.8 rad, unlike the DeepVO architecture, it generates errors of less than 0.1 rad.
- For the estimation of the orientation in the Z axis (Yaw), the PoseConvGRU architecture is the architecture that generates peaks of 6 rad, but the architectures

RMSE

As can be seen in the previous Table VII, both the DeepVO and PoseConvGRU architecture present a good performance in the evaluation sequences, being the DeepVO architecture in sequence 03 the one that generates the least error in both position estimation and orientation. In the other sequences, it is observed that PoseConvGRU generates a low error compared to the other architectures, both in the estimation of position and in the orientation.

Both the DeepVO and PoseConvGRU architecture perform well in the evaluation sequences, with the DeepVO architecture in sequence 03 the one that generates the least error in both position estimation and orientation.

TABLE VII

RMSE OF SEQUENCES 03, 05 AND 07

SEQUENCES	DEEPVO		MAGICVO		POSECONVGRU	
	T	R	T	R	T	R
03	26.65	0.23	41.35	0.42	38.39	0.39
05	30.69	2.80	71.80	1.32	15.67	1.14
07	16.53	1.61	25.37	2.54	14.47	1.69
MEAN	24.62	1.55	46.17	1.42	22.84	1.08

T: RMSE of translation in m with the KITTI dataset

R: Rotation RMSE in rad with KITTI dataset

VII. CONCLUSION

Regarding the first specific objective Identify and describe the deep learning architectures available for visual odometry. It has been observed that the ability of the recurrent neural network's LSTM and its variant GRU in image sequence processing tasks has been found to be useful for the problem of visual odometry. For which it has been analyzed in the state of the art how position and orientation estimation methods have progressed through analytical methods until reaching the use of deep neural networks.

As is the case of the models selected for the development of this work, such as DeepVO, MagicVO and PoseConvGRU, the use of CNN presents excellent results in computer vision tasks since through its convolutional layers it allows the network to be

able to extract characteristics of images and even more so with the technological development that has occurred in recent years.

Technological advances have allowed CNN's training processes to be parallelized with the use of graphic cards that have CUDA cores, these advances were used by developers to generate more complex models by adding many convolutional layers. In addition, information has been collected on the operation of the LSTM and GRU neural networks both in their capacity to store long-term memory for both future and past values.

Models have been used that have been pre-trained for specific tasks, in the case of FlowNet, which allows the estimation of optical flow of a pair of sequential images, where this pair of images is concatenated one after the other, forming a 6-channel matrix. . The use of learning transfer helped the architectures not to be trained from scratch, that is, with all the layers, but only the layers that are followed by the convolutional layers.

Regarding the specific objective Identify the data sets available for training and testing of visual odometry. The KITTI dataset [24] has been observed with extensive data with high quality images, 11 sequences with truth values and 10 image sequences without truth values are presented.

Regarding the specific objective Define the success metrics for visual odometry. The dimensioning of the images, the training and evaluation sequences for the different models, loss during the training of the architectures and the different error calculations such as absolute error and RMSE have been defined. Which are metrics that have been used in the works used in this comparative work.

Development has been carried out with respect to the specific objective Implementing the selected architectures for visual odometry. For the analysis, it was decided to select the DeepVO architectures [66], where it was observed that it presents an excellent result for the estimation of visual odometry in comparison with the VISO2_M and VISO2_S [42] models, this model consisted in the first stage of the FlowNet model followed by LSTM and Dense layers. The next selected architecture was MagicVO [40] which obtained better performance than DeepVO and ORB_SLAM2 [53], this architecture has FlowNet as the first stage followed by Bi-LSTM and Dense layers. The last architecture chosen was PoseConvGRU [75] which has GRU and Dense layers.

Once the architectures and data sets were chosen, the architectures were implemented, TensorFlow was used for the development of neural networks and data loading. This library was chosen because it contains documentation and an active community, which allows intuitive development and without the need to create functions from scratch. In addition, it was very useful when it came to sharing resources during the training of the architectures since it allowed to exchange information from the GPU to the CPU with few lines of code, it is worth mentioning that TensorFlow allowed the dynamic loading of data since how it worked with images of dimensions 640 × 192 that it was inefficient to keep all the sequences in RAM since the values together caused memory overflow.

Reference is made to the development of the specific objective Calculate and compare the training time between the different architectures for the same data set. Where it was shown that the use of LSTM layers with several hidden states made the training of neural networks slow since they cannot be parallelized as is the case with CNNs. Additionally, it was observed that the model that had a relatively short training in the 200 selected epochs was PoseConvGRU, since it has a reduced number of parameters compared to MagicVO, which is the model with the highest number of parameters to train. MagicVO took 67 hours to complete the 200 training epochs, this may be due to the limitations presented in the equipment in which the training was developed. In addition, the PoseConvGRU architecture was the one that managed to reach a minimum value of 0.0006612031 in its loss value, with the architecture reaching the lowest value of the other architectures.

According to the specific objective Evaluate different architectures with different data sets. The different values of the absolute error presented by the different architectures with respect to the truth values were observed, where the error of the positions propagates through the sequence of images, this is due to the fact that the positions are consistent with the Past estimates and therefore affects the calculation of the next position since both the previous position and the orientation intervene.

Regarding the specific objective To compare the estimation result of the visual odometry of the different architectures. It has been determined that the best architecture for estimating visual odometry is the PoseConvGRU architecture since it is a model that does not use a large number of parameters or layers heavily loaded with parameters to achieve what is expected with an RMSE of 24.62961 in position and 1.555016 in Rotation, although the other architectures may improve with retraining for a longer time or be retrained with other data sets, with the selected training parameters it has not had much effect in improving the estimation of position and orientation values .

As it was observed, using a large number of layers in neural networks is not always appropriate, since in a way it affects performance and generates a greater computational effort, this can be verified in the MagicVO architecture training stage, that it took me almost three days to complete the 200 epochs and still not be able to surpass the precision of the other two architectures.

And finally, it was observed that according to the fulfillment of the different specific objectives, the general objective of carrying out a comparative study of the different deep learning architectures for the estimation of visual odometry was achieved. It has been shown that the time it takes for the PoseConvGRU architecture to estimate the relative positions is less than the rest of the architectures, making the architecture not only better able to estimate position and orientation values but also in which it can be used perhaps in real-time applications with the limitation of being used on desktop computers.

Additionally, like the authors of the architectures studied, it is intended that these deep neural network models are of support for existing analytical methods, since they have presented methods capable of producing excellent results, considering the problem of visual odometry solved. Therefore, there are methods or models that do not generate a lot of computational load and are accurate enough to be used in systems with little amount of processing resources.

Once the comparison has been developed, the implemented code is freely available with its respective pre-trained parameters in the GitHub repository. They can be used to continue training or evaluation of architectures. As well as its use in different robotics applications where it is desired to estimate the position of the robot from image sequences.

Although a desktop computer has been used to achieve the inference of the models, it is expected that in the future with technological advances it can be implemented in NVIDIA JetsonNano embedded systems since it has CUDA cores in fewer numbers than graphic cards, if Well today you can compile TensorFlow with GPU support in the embedded system, it still cannot process a large number of parameters.

REFERENCES

[1] Agrawal, P., Carreira, J., & Malik, J. (2015). *Learning to See by Moving.*

[2] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2018).

Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-January*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

[3] Agrawal, P., Carreira, J., & Malik, J. (2015). *Learning to See by Moving*.

[4] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2018). Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-January*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

[5] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., & Asari, V. K. (2018). *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. http://arxiv.org/abs/1803.01164

[6] Amidi, A., & Amidi, S. (2017). *CS 230 - Recurrent Neural Networks Cheatsheet*. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

[7] Anwar, A. (2019). *Difference between AlexNet, VGGNet, ResNet, and Inception | by Aqeel Anwar | Towards Data Science*. https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

[8] Borenstein, J, Everett, H. R., Feng, L., Lee, S. W., Byrne, R. H., Borenstein, J., & Feng, L. (1996). Where am I? Sensors and Methods for Mobile Robot Positioning Prepared by the University of Michigan For the Oak Ridge National Lab (ORNL) D&D Program and the United States Department of Energy's Robotics Technology Development Program Within the Environmental Restoration, Decontamination and Dismantlement Project.

[9] Borenstein, Johann, & Feng, L. (1995). UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots.

[10] Brunetti, A., Buongiorno, D., Trotta, G. F., & Bevilacqua, V. (2018). Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. *Neurocomputing*, *300*, 17–33. https://doi.org/10.1016/j.neucom.2018.01.092

[11] Chen, C., Lu, X., Wahlstrom, J., Markham, A., & Trigoni, N. (2019). Deep Neural Network Based Inertial Odometry Using Low-cost Inertial Measurement Units. *IEEE Transactions on Mobile Computing*, 1–1. https://doi.org/10.1109/tmc.2019.2960780

[12] Chen, C., Zhao, P., Lu, C. X., Wang, W., Markham, A., & Trigoni, N. (2018). *OxIOD: The Dataset for Deep Inertial Odometry*. http://arxiv.org/abs/1809.07491

[13] Chung, C., Patel, S., Lee, R., Fu, L., Reilly, S., Ho, T., Lionetti, J., George, M. D., & Taylor, P. (2018). Vggnet. *American Journal of Health-System Pharmacy*, *75*(6), 398–406. https://doi.org/10.2146/ajhp170251

[14] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. http://arxiv.org/abs/1412.3555

[15] Costante, G., Mancini, M., Valigi, P., & Ciarfuglia, T. A. (2016). Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation. *IEEE Robotics and Automation Letters*, *1*(1), 18–25. https://doi.org/10.1109/LRA.2015.2505717

[16] Dawson-Howe, K. M., & Vernon, D. (1994). Simple pinhole camera calibration. *International Journal of Imaging Systems and Technology*, *5*(1), 1–6. https://doi.org/10.1002/ima.1850050102

[17] Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazırbas¸, H., Golkov, V., Van Der Smagt, P., Cremers, D., & Brox, T. (2015). *FlowNet: Learning Optical Flow with Convolutional Networks*.

[18] Du, F., & Brady, M. (1993). Self-calibration of the intrinsic parameters of cameras for active vision systems. *IEEE Computer Vision and Pattern Recognition*, 477–482. https://doi.org/10.1109/cvpr.1993.341087

[19] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robotics and Automation Magazine*, *13*(2), 99–108. https://doi.org/10.1109/MRA.2006.1638022

[20] Engel, J. (2013). Semi-Dense Visual Odometry for a Monocular Camera * 2013 IEEE International Conference on Computer Vision. 1449–1456. https://doi.org/10.1109/ICCV.2013.183

[21] Engineering, D. of M. and P. (2011). *ASL Datasets*. https://projects.asl.ethz.ch/datasets/doku.php?id=home

[22] Engineering, D. of M. and P. (2020). Homepage - D-MAVT – Department of Mechanical and Process Engineering | ETH Zurich. https://mavt.ethz.ch/

[23] Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T., & Philbrick, K. (2017). Toolkits and Libraries for Deep Learning. In *Journal of Digital Imaging* (Vol. 30, Issue 4, pp. 400–405). Springer New York LLC. https://doi.org/10.1007/s10278-017-9965-6

[24] Forster, C., Pizzoli, M., & Scaramuzza, D. (2014). SVO: Fast semi-direct monocular visual odometry. *Proceedings - IEEE International Conference on Robotics and Automation*, 15–22. https://doi.org/10.1109/ICRA.2014.6906584

[25] Fu-chao, W., Fu-chao, W., Guang-hui, W., & Zhan-yi, H. (2003). *A Linear Approach for Determining Intrinsic Parameters and Pose of Cameras from Rectangles*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.3475

[26] Geiger, A., Urtasun, P. L., & Raquel, U. (2012). *The KITTI Vision Benchmark Suite*. http://www.cvlibs.net/datasets/kitti/

[27] Google. (2020). *Descending into ML: Training and Loss | Machine Learning Crash Course*. https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

[28] Grandón-Pastén, N., Aracena-Pizarro, D., & Tozzi, C. L. (2007). RECONSTRUCCIÓN DE OBJETO 3D A PARTIR DE IMÁGENES CALIBRADAS. *Ingeniare. Revista Chilena de Ingeniería*, *15*(2), 158–168. https://doi.org/10.4067/S0718-33052007000200006

[29] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5–6), 602–610. https://doi.org/10.1016/j.neunet.2005.06.042

[30] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. *International Conference in Machine Learning*, *4*(7540), 14. http://arxiv.org/abs/1502.04623

[31] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, *187*, 27–48. https://doi.org/10.1016/j.neucom.2015.09.116

[32] Handa, A., Bloesch, M., Pătrăucean, V., Stent, S., McCormac, J., & Davison, A. (2016). Gvnn: Neural network library for geometric computer vision. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9915 LNCS*, 67–82. https://doi.org/10.1007/978-3-319-49409-8_9

[33] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. https://doi.org/10.1109/CVPR.2016.90

[34] Hecht-Nielsen, R. (1989). Neurocomputer Applications. In *Neural Computers* (pp. 445–453). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-83740-1_45

[35] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[36] Hochreiter, S., Younger, A. S., & Conwell, P. R. (2001). Learning to learn using gradient descent. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2130, 87–94. https://doi.org/10.1007/3-540-44668-0_13

[37] Howard, A. (2008). Real-time stereo visual odometry for autonomous ground vehicles. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 3946–3952. https://doi.org/10.1109/IROS.2008.4651147

[38] Huang, S., Wang, Z., Dissanayake, G., & Frese, U. (2009). *Iterated SLSJF: A Sparse Local Submap Joining Algorithm with Improved Consistency*. https://pdfs.semanticscholar.org/6e41/e7840bf378709098b624ed299d637c6fc63e.pdf

[39] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., & Brox, T. (2017). *FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks*. http://lmb.

[40] InvenSense. (2016). *ICM-20600*. www.invensense.com

[41] Jeong, W. Y., & Lee, K. M. (2006). Visual SLAM with line and corner features. *IEEE International Conference on Intelligent Robots and Systems*, 2570–2575. https://doi.org/10.1109/IROS.2006.281708

[42] Jiao, J., Jiao, J., Mo, Y., Liu, W., & Deng, Z. (2018). MagicVO: End-to-End Monocular Visual Odometry through Deep Bi-

directional Recurrent Convolutional Neural Network. http://arxiv.org/abs/1811.10964

[43] Jordan, J. (2017). *Convolutional neural networks*. https://www.jeremyjordan.me/convolutional-neural-networks/

[44] Kitt, B., Geiger, A., & Lategahn, H. (2010). Visual Odometry based on Stereo Image Sequences with RANSAC-based Outlier Rejection Scheme.

[45] Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*. https://doi.org/10.1109/ISMAR.2007.4538852

[46] Konda, K., & Memisevic, R. (2005). *Learning Visual Odometry with a Convolutional Network*. https://doi.org/10.5220/0005299304860490

[47] Kreuzig, R., Ochs, M., & Mester, R. (2019). DistanceNet: Estimating Traveled Distance from Monocular Images using a Recurrent Convolutional Neural Network.

[48] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. http://code.google.com/p/cuda-convnet/

[49] Li, M., Hong, B., Cai, Z., & Luo, R. (2008). Novel Rao-Blackwellized Particle Filter for Mobile Robot SLAM Using Monocular Vision.

[50] Li, R., Wang, S., Long, Z., & Gu, D. (2018). UnDeepVO: Monocular Visual Odometry Through Unsupervised Deep Learning. *Proceedings - IEEE International Conference on Robotics and Automation*, 7286–7291. https://doi.org/10.1109/ICRA.2018.8461251

[51] Maddern, W., Pascoe, G., Gadd, M., Barnes, D., Yeomans, B., & Newman, P. (2015). *Real-time Kinematic Ground Truth for the Oxford RobotCar Dataset*. https://www.novatel.com/products/span-gnss-inertial-systems/

[52] Maddern, W., Pascoe, G., Linegar, C., & Newman, P. (2017). 1 year, 1000 km: The Oxford RobotCar dataset. *The International Journal of Robotics Research*, *36*(1), 3–15. https://doi.org/10.1177/0278364916679498

[53] Martinez-Cantin, R., & Castellanos, J. A. (2005). Unscented SLAM for large-scale outdoor environments. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 328–333. https://doi.org/10.1109/IROS.2005.1545002

[54] Muller, P. M. (2016). Optical Flow and Deep Learning Based Approach to Visual Optical Flow and Deep Learning Based Approach to Visual Odometry Odometry. https://scholarworks.rit.edu/theses

[55] Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, *31*(5), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671

[56] Mur-Artal, R., & Tardos, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, *33*(5), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103

[57] Newcombe, R. A., Lovegrove, S. J., & Davison, A. J. (2011). DTAM: Dense tracking and mapping in real-time. *Proceedings of the IEEE International Conference on Computer Vision*, 2320–2327. https://doi.org/10.1109/ICCV.2011.6126513

[58] Nistér, D., Naroditsky, O., & Bergen, J. (2004). Visual odometry. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *1*. https://doi.org/10.1109/cvpr.2004.1315094

[59] NovAtel. (2017). *SPAN CPT7 Performance Specifications*. https://docs.novatel.com/OEM7/Content/Technical_Specs_Receiver/SPANCPT7_Performance_Specs.htm

[60] Olah, C. (2015). *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[61] OpenCV. (2017). *OpenCV: Optical Flow*. https://docs.opencv.org/3.4.0/d7/d8b/tutorial_py_lucas_kanade.html

[62] OXTS. (2013). RT3000 v3 - GNSS-aided inertial navigation system for automotive testing. https://www.oxts.com/products/rt3000/

[63] Perez, J., Serrano, C., Acha, B., Serrano, T., & Linares, B. (2013). *Red neuronal convolucional rápida sin fotogramas para reconocimientos de dígitos*. http://digital.csic.es/handle/10261/84753

[64] Qin, T., Li, P., & Shen, S. (2018). VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, *34*(4), 1004–1020. https://doi.org/10.1109/TRO.2018.2853729

[65] Saffiotti, A. (1997). The uses of fuzzy logic in autonomous robot navigation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, *1*(4), 180–197. https://doi.org/10.1007/s005000050020

[66] Sak, H., Senior, A., & Beaufays, F. (2014). Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. http://arxiv.org/abs/1402.1128

[67] Scaramuzza, D., & Fraundorfer, F. (2011). Tutorial: Visual odometry. *IEEE Robotics and Automation Magazine*, *18*(4), 80–92. https://doi.org/10.1109/MRA.2011.943233

[68] Sen, Ronald, Hongkai, & Niki. (2017). DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. *Proceedings - IEEE International Conference on Robotics and Automation*, 2043–2050. https://doi.org/10.1109/ICRA.2017.7989236

[69] Singh, B., Marks, T. K., Jones, M., Tuzel, O., & Shao, M. (2016). A Multi-Stream Bi-Directional Recurrent Neural Network for Fine-Grained Action Detection.

[70] Steffen, L., Alec, G., Thomas, F., & Kevin, M. (2017). *Visual Odometry using Convolutional Neural Networks*. https://www.researchgate.net/publication/322525765_Visual_Odometry_using_Convolutional_Neural_Networks

[71] Strobl, K. H., & Hirzinger, G. (2011). More accurate pinhole camera calibration with imperfect planar target. *Proceedings of the IEEE International Conference on Computer Vision*, 1068–1075. https://doi.org/10.1109/ICCVW.2011.6130369

[72] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June-2015*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[73] Venegas, J. (2009). *Encoders*. http://support.automationdirect.com/docs/absolute_encoders.pdf

[74] Wang, R., Pizer, S. M., & Frahm, J.-M. (2019). Recurrent Neural Network for (Un-)supervised Learning of Monocular Video Visual Odometry and Depth.

[75] Watson, J. D. (2020). AFIT Scholar AFIT Scholar Improved Ground-Based Monocular Visual Odometry Estimation Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks using Inertially-Aided Convolutional Neural Networks. https://scholar.afit.edu/etd/3622

[76] Wu, N. (2017). *Tensorflow detection model zoo*. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

[77] Zhai, G., Liu, L., Zhang, L., & Liu, Y. (2019). PoseConvGRU: A Monocular Approach for Visual Ego-motion Estimation by Learning. *Pattern Recognition*, *102*. http://arxiv.org/abs/1906.08095

[78] Zhang, Y. (2006). Towards piecewise-linear primal neural networks for optimization and redundant robotics. *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, ICNSC'06*, 374–379. https://doi.org/10.1109/icnsc.2006.1673175

[79] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(11), 1330–1334. https://doi.org/10.1109/34.888718