Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias en Ciencias de la Computación

Corrección y seguimiento de la posición en ambientes virtuales utilizando aprendizaje profundo

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Maestro en Ciencias

Presenta:

Jesús Mauricio Jiménez Solorio

Ensenada, Baja California, México

2022

Tesis defendida por

Jesús Mauricio Jiménez Solorio

y aprobada por el siguiente Comité

Dr. Ubaldo Ruiz López

Codirector de tesis

Dr. Israel Becerra Durán

Codirector de tesis

Dr. Irvin Hussein López Nava

Dr. Miguel Ángel Alonso Arévalo

Dr. Pedro Gilberto López Mariscal

Coordinador del Posgrado en Ciencias de la Computación

Dr. Pedro Negrete Regagnon

Director de Estudios de Posgrado

Resumen de la tesis que presenta Jesús Mauricio Jiménez Solorio como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Corrección y seguimiento de la posición en ambientes virtuales utilizando aprendizaje profundo

Resumen aprobado por:

Dr. Ubaldo Ruiz López
Codirector de tesis

Dr. Israel Becerra Durán
Codirector de tesis

En esta tesis, se aborda el problema de la corrección del error de deriva en sensores inerciales utilizados para rastrear la posición en ambientes virtuales. A diferencia de los trabajos en el estado del arte, que hacen uso de sensores exteroceptivos para afrontar el problema, en esta tesis se presenta una solución basada en aprendizaje profundo que recibe exclusivamente mediciones de sensores propioceptivos, la cual es novedosa en el contexto de realidad virtual. Las contribuciones principales de este trabajo son: 1) la creación de una base de datos con lecturas tomas a partir de una unidad de medición inercial, localizada en un casco de realidad virtual, mientras los usuarios realizan diferentes actividades, y 2) el diseño de una red neuronal profunda basada en capas LSTM bidireccionales para la predicción de trayectorias en ambientes virtuales. Los resultados obtenidos con el método propuesto presentan una mejoría en la reconstrucción de la posición del casco de realidad virtual con respecto a la literatura, en particular, en la coordenada correspondiente a la altura.

Palabras clave: aprendizaje profundo, realidad virtual, error de deriva, sensor inercial

Abstract of the thesis presented by Jesús Mauricio Jiménez Solorio as a partial requirement to obtain the Master of Science degree in Computer Science.

Correction and monitoring of position in virtual environments using deep learning

Abstract approved by:

Dr. Ubaldo Ruiz López	Dr. Israel Becerra Durán
Thesis Co-Director	Thesis Co-Director

This thesis addresses the problem of correcting the drift error produced by inertial sensors used for tracking the position in virtual environments. Unlike works in the state-of-the-art, which use exteroceptive sensors to address the problem, this work presents a novel solution in the context of virtual reality based on deep learning and using data exclusively from proprioceptive sensors. The main contributions of this work are: 1) generating a database with readings taken from an inertial measurement unit located on a virtual reality headset while the users perform different activities and 2) designing a deep neural network based on bidirectional LSTM layers to predict trajectories in virtual environments. The proposed approach shows an improvement in the reconstruction of the HMD position with respect to the literature, in particular, in the coordinate corresponding to height.

Keywords: deep learning, virtual reality, drift error, inertial sensor

Dedicatoria

A mi familia y mi novia que siempre estuvieron a mi lado.

Agradecimientos

Al Centro de Investigación Científica y de Educación Superior de Ensenada por brindarme todo lo necesario para la elaboración de este trabajo.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría/doctorado. No. de becario: 995329

A mi comité de tesis por toda su ayuda y apoyo.

Tabla de contenido

Página

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de figuras (continuación)

Figura Página

Lista de figuras (continuación)

Figura Página

Lista de figuras (continuación)

Figura Página

Lista de tablas

Tabla Página

Lista de tablas (continuación)

Glosario

Posición Sitio espacial que ocupa un objeto. Puede ser determinado por coordenadas cartesianas (X, Y, Z).

Orientación Dirección física relativa de una persona u objeto en relación con su entorno.

Pose Una posición y orientación particular en la que se encuentra una persona u objeto.

Trayectoria Lugar geométrico descrito por las posiciones sucesivas que describe un objeto al desplazarse de un punto a otro.

Aceleración lineal Magnitud vectorial que señala el cambio de la velocidad por unidad de tiempo. Sus unidades son metros sobre segundo al cuadrado (m/s^2). A lo largo de este trabajo se le llamará únicamente aceleración.

Desplazamiento Magnitud vectorial que determina el cambio de la posición de una persona u objeto desde un punto inicial a un punto final.

HMD Head Mounted Display

IMU Inertial Measurement Unit

Capítulo 1. Introducción

Hoy en día, el uso de gráficos por computadora se ha convertido en un aspecto esencial para la mayoría de las plataformas modernas. Es difícil imaginarnos a un arquitecto, ingeniero o diseñador sin una estación gráfica de trabajo. En los últimos años la tecnología en el desarrollo de microprocesadores ha aumentado rápidamente, lo que ha ocasionado que los sistemas de cómputo sean más veloces y estén equipados con tarjetas gráficas más potentes. Además, debido a la vertiginosa expansión de la tecnología en el desarrollo de gráficos por computadora, los precios de estos equipos han disminuido considerablemente, por lo que en la actualidad estos sistemas son accesibles para los usuarios promedio. La fascinación con esta nueva realidad de los gráficos por computadora empieza en los videojuegos, donde se permite al usuario ver los alrededores de un mundo en otra dimensión y experimentar cosas que no son accesibles en la vida real (Mazuryk y Gervautz, 1996; Basu, 2019). Sin embargo, la emoción de los usuarios no termina aquí, más allá de solo ver imágenes a través de un monitor, desean tener una relación total con el ambiente que observan y poder interactuar con objetos como si de la vida real se tratara. La tecnología que hace esto posible es llamada realidad virtual (o VR por sus siglas en inglés) y en los últimos años ha tenido un gran impacto en la mayoría de los ámbitos de la vida moderna.

El concepto de lo que hoy llamamos realidad virtual nació en el año 1963 (Magnusson, 2019). Fue entonces cuando Ivan Sutherland introdujo Sketchpad, un programa que permitía al usuario interactuar con una interfaz gráfica conectada a una computadora, mediante un bolígrafo especial y una pantalla para dibujo asistido. Eventualmente, en 1965, Sutherland y su estudiante, Bob Sproull, crearon el primer casco con gráficos por computadora interactivos, un dispositivo con pantallas (Head-mounted Display o HMD) que se colocaba sobre la cabeza (figura 1), en él se podía interactuar con un mundo virtual en una simulación llamada The Sword of Damocles (Mazuryk y Gervautz, 1996). Este sistema generaba imágenes binoculares que eran representadas apropiadamente para la posición y orientación de la cabeza en movimiento. Esta fue la primera vez en la historia de los gráficos por computadora que las personas pudieron ver un mundo virtual generado por computadora (Basu, 2019).

Desde la introducción de Sketchpad por Ivan Sutherland se han formulado diversas definiciones para la realidad virtual, por ejemplo, Brooks Jr *et al.* (1992) definieron a

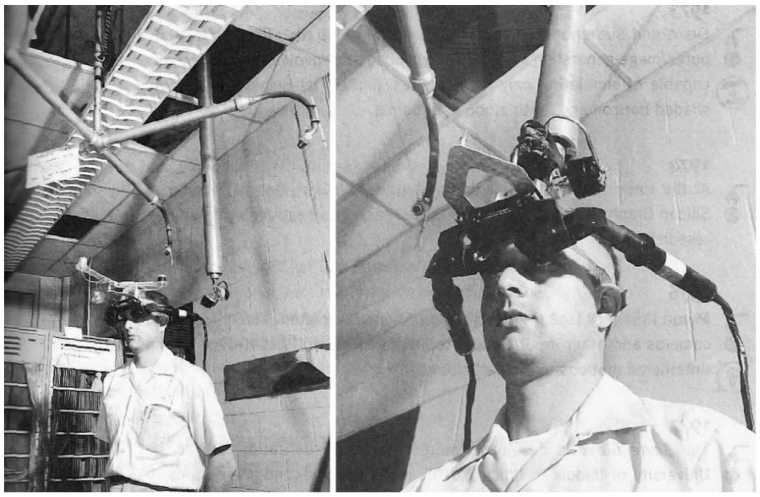


Figura 1. HMD de Ivan Sutherland. La pantalla tenía un contrapeso suspendido en forma de brazo mecánico y usaba transductores ultrasónicos para rastrear el movimiento de la cabeza (Basu, 2019).

la realidad virtual como “gráficos interactivos en tiempo real con modelos 3D, combinados con una tecnología de visualización que brinda al usuario la inmersión en el mundo modelo y la manipulación directa”; Earnshaw (2014) describieron a la realidad virtual como “la ilusión de participar en un entorno sintético en lugar de la observación externa de dicho entorno. La realidad virtual se basa en pantallas estereoscópicas 3D de seguimiento de cabeza, seguimiento de manos/cuerpo y sonido binaural[1]. La realidad virtual es una experiencia inmersiva y multisensorial”; Cruz-Neira *et al.* (1993) definieron la realidad virtual como “aquel sistema que proporciona una perspectiva de seguimiento de la cabeza centrada en el espectador en tiempo real con un gran ángulo de visión, control interactivo y pantalla binocular”; LaValle (2016) define la realidad virtual como “inducir un comportamiento objetivo a un organismo utilizando estimulación artificial, mientras el organismo tiene poca o nula percepción de la interfaz”. Como se puede notar, estas definiciones, aunque diferentes, resaltan tres características comunes de los sistemas de realidad virtual: inmersión, percepción de estar presente en un ambiente, e interacción con ese ambiente. Específicamente, la inmersión se refiere a la cantidad de sentidos estimulados, las interacciones y la similitud de la realidad de los estímulos utilizados para simular entornos. Esta característica puede depender de las propiedades del sistema tecnológico empleado para aislar al usuario de la realidad (Cipresso *et al.*, 2018).

Actualmente, la tecnología de la realidad virtual está evolucionando rápidamente.

[1]Es aquel que trata crear una sensación auditiva 3D

Con el avance de la tecnología en gráficos por computadora y en el desarrollo de sensores cada vez más pequeños, se han logrado construir mundos virtuales con un sinfín de aplicaciones, desde entretenimiento, como lo son películas y videojuegos, hasta arte y educación, como visitas virtuales a museos.

1.1. Planteamiento del problema

Como podemos imaginarnos, poder representar los movimientos del usuario en un mundo virtual de manera precisa es de gran importancia. En algunas aplicaciones médicas se hacen simulaciones de órganos humanos donde los cirujanos pueden practicar antes de realizar una operación, o en sistemas de entretenimiento, donde el usuario es capaz de moverse y controlar sus manos, en ocasiones necesitará interactuar con el mundo virtual. Por lo que encontrar la posición del usuario y representarla en un mundo virtual es un gran reto que deben resolver los sistemas de realidad virtual.

Los HMD actuales emplean dos tipos de sensores para rastrear el movimiento del usuario: sensores propioceptivos y exteroceptivos. Los sensores exteroceptivos son aquellos que utilizan el ambiente y sus características para poder procesar información, por ejemplo, cámaras o sensores infrarrojos que necesitan detectar el movimiento de luces led. Mientras que los sensores propioceptivos son aquellos que no necesitan ninguna información exterior o del ambiente para poder recolectar información. Entre los sensores propioceptivos de los HMD actuales se encuentran acelerómetros y giroscopios, usados para estimar el desplazamiento y orientación del casco. Sin embargo, las lecturas de estos sensores no son exactas, y con el paso del tiempo se vuelven imprecisas, por lo que es necesario utilizar sensores exteroceptivos para corregir y estimar de manera precisa la posición y orientación del casco.

Uno de los errores producido por el uso de sensores propioceptivos (acelerómetros y giroscopios en el caso de la realidad virtual) se denomina error de deriva; este error es generado debido a que la calibración de los sensores no es perfecta y a factores intrínsecos de los propios sensores. Para estimar la orientación del casco y representarla en el mundo virtual es necesario integrar las mediciones del giroscopio en función del tiempo, pero al hacerlo, acumulamos el error de deriva y este crecerá linealmente con el tiempo, lo mismo ocurre cuando integramos los resultados del acelerómetro para

obtener la velocidad. Para obtener la posición del casco es necesario hacer una doble integración de los resultados del acelerómetro, pero esto causará que el error de deriva aumente cuadráticamente con el tiempo, es por ello que para minimizar dicho error es necesario el uso de sensores exteroceptivos. No obstante, considerar sensores exteroceptivos no es una tarea sencilla, ya que para lograr corregir el error de deriva es necesario emplear algoritmos complejos y costosos computacionalmente. En algunos casos los sensores exteroceptivos deben ser colocados en puntos estratégicos de la habitación donde se hará uso del sistema de realidad virtual, en consecuencia, el sistema VR no es portátil. En otros casos, los sensores exteroceptivos son colocados sobre el casco de realidad virtual, por lo que el sistema es portable, pero esto aumenta el peso del casco. Encontrar un método alternativo para la corrección de la posición que no utilice sensores exteroceptivos podría minimizar los costos de los sistemas VR y, tal vez, obtener una mejor corrección que con los métodos actuales.

Por lo tanto, surge la pregunta: ¿existe alguna forma de minimizar el error de deriva sin utilizar sensores exteroceptivos? En el capítulo 3, se mencionan algunos trabajos encontrados en la literatura que buscan corregir la posición y orientación estimadas a partir de lecturas inerciales.

1.2. Hipótesis

Es posible construir un sistema de rastreo de posición para un HMD (Head Mounted Display) utilizando únicamente sensores propioceptivos con un margen de error similar al de los sistemas que hacen uso de sensores exteroceptivos.

1.3. Objetivos

1.3.1. Objetivo general

Diseñar, implementar, y evaluar un algoritmo (por ejemplo, una red neuronal) que utilice como datos de entrada las lecturas inerciales generadas por un usuario utilizando un casco de realidad virtual y arroje como resultado la trayectoria del casco.

1.3.2. Objetivos específicos

- Programar ambientes virtuales para la generación de bases de datos donde los participantes realicen los movimientos más usuales cuando utilizan un HMD.
- Crear una base de datos de lecturas inerciales obtenidas de un IMU localizado en un casco de realidad virtual.
- Desarrollar diferentes redes neuronales para predecir la posición del casco de realidad virtual.
- Evaluación cuantitativa y cualitativa de los resultados obtenidos con cada una de las redes neuronales propuestas.

En el siguiente capítulo se hablará sobre los primeros cascos de realidad virtual, su funcionamiento y aplicación. También se explicará cómo es que los primeros prototipos de cascos VR comerciales corregían el error de deriva y se detallará más a fondo el funcionamiento del casco Oculus Rift S, que es el que se empleó durante el desarrollo de este trabajo. En el capítulo 3 se describirán los trabajos relacionados encontrados en la literatura que también tratan de corregir el error de deriva de sensores inerciales, utilizando sensores adicionales y mediante técnicas de aprendizaje profundo. Se presentarán también, algunas de las bases de datos de lecturas inerciales que se utilizan actualmente para trabajar con redes neuronales. En el capítulo 4 se detalla la base de datos creada como parte de este trabajo de tesis y que será utilizada para entrenar los diferentes modelos de redes. Se describirán las características de la base de datos, los tipos de datos que contiene y el proceso que se llevó a cabo para recolectar los datos inerciales. En el capítulo 5 se describe la metodología que se siguió para realizar los diferentes experimentos realizados de este trabajo. En el capítulo 6 se presentan los resultados de los experimentos y, finalmente, en el capítulo 7, se abordan las conclusiones que se obtienen después de analizar los resultados y el trabajo futuro que podría desarrollarse.

Capítulo 2. Marco téorico

En este capítulo se describen algunos de los cascos de realidad virtual que se han desarrollado, el sistema de rastreo que utilizan y la implementación de sensores exteroceptivos para corregir su error de deriva. También se menciona otro método para corregir la posición y orientación de sensores inerciales, utilizando aprendizaje profundo.

2.1. Cascos de realidad virtual

Ivan Sutherland, en 1963, describió su sistema como una ventana a través de la cual un usuario percibe el mundo virtual como si se viera, sintiera y sonara real, y en la que el usuario pudiera actuar de manera realista (Cipresso *et al.*, 2018). A partir de este momento la tecnología de la realidad virtual empezó a desarrollarse y a ser estudiada con más profundidad, sin embargo, las limitantes tecnológicas en aquel entonces eran una barrera muy grande a superar para los científicos de todo el mundo. Uno de los proyectos que más sobresalió en 1982, fue cuando Thomas Furness desarrolló un simulador de vuelo avanzado llamado VCASS (Visually Coupled Airborne Systems Simulator) como parte de su investigación en las fuerzas aéreas, donde los pilotos utilizaban un HMD para realizar simulaciones de combates aéreos (Mazuryk y Gervautz, 1996). El primer HMD comercial, llamado EyePhones, fue desarrollado por VPL en los 80s. Estos HMD usaban pantallas LCD para producir una imagen, sin embargo, tenían una resolución extremadamente baja para los estándares actuales (360x240 píxeles), por lo que la imagen virtual daba la impresión de estar distorsionada. Otros defectos de este sistema fueron su gran peso (2.4 kg) y su alto precio que lo hacia accesible solo para una pequeña parte de la población (37,000 dólares actuales) (Burdea, 2006).

En los años 90s, salieron a la venta los primeros cascos de realidad virtual para el público en general, enfocados en videojuegos. Sin embargo, las experiencias no fueron lo suficientemente convincentes o cómodas para atraer el interés de las masas. Entre los problemas que mostraban estos cascos se encuentra la fatiga muscular que sentía el usuario después de utilizarlos por algún tiempo, ya que eran bastante pesados. Además, entre otros factores causantes de su baja popularidad, se encuentran aquellos ocasionados por las limitaciones tecnológicas que tenían los sensores utilizados en ese

momento.

Fue hasta el año 2016 que Palmer Luckey lanzó al mercado un casco de realidad virtual llamado Oculus Rift. Este dispositivo fue popular debido a que su precio era accesible para el público (400 dólares) y, a diferencia de los primeros cascos, era considerablemente ligero, su sistema de rastreo de movimiento funcionaba bastante bien y la calidad gráfica era muy buena. Meses después, HTC y la compañía Valve anunciaron el sistema de realidad virtual HTC Vive, dando así inicio a la producción y venta en masa de los HMD (LaValle, 2016).

2.2. Sistema de rastreo de movimiento

El primer casco Oculus Rift cuenta con dos versiones preliminares, el Development Kit 1 (DK1) y el Development Kit 2 (DK2). El primero, anunciado en 2012, tenía una resolución de 1280 x 800 (640 x 800 por cada ojo), pesaba un total de 380 gramos y únicamente realizaba rastreo rotacional del movimiento de la cabeza. Utilizaba un sensor especial desarrollado por Hillcrest Labs para realizar un rastreo de movimiento con 3 grados de libertad, enfocados solamente en las rotaciones (roll, pitch y yaw). El DK2 fue anunciado en el 2014, tenía una resolución de 1920 x 1080 (960 x 1080 por cada ojo), pesaba 440 gramos y era capaz de realizar un rastreo del movimiento rotacional y traslacional del usuario. La versión final del casco Oculus Rift fue lanzada al publico el 28 de marzo del 2016, contaba con una pantalla OLED y una resolución de 1080 x 1200 por cada ojo, con una frecuencia de actualización de 90 Hz (Nafees, 2016). Al igual que el DK2, podía realizar rastreo de movimiento y orientación de la cabeza y también representar los movimientos traslacionales del usuario en el mundo virtual. Para esto, contaba con un acelerómetro, un giroscopio, un magnetómetro y una cámara externa encargada de rastrear una constelación de LEDs infrarrojos situados en el casco y los controles del usuario (Desai *et al.*, 2014).

Otro HMD de gran importancia es el casco Vive de HTC y Valve, lanzado al mercado en abril del 2016. Este casco contaba con una pantalla OLED, una resolución de 2160 x 1200 (1080 x 1200 por cada ojo) y una frecuencia de actualización de 90 Hz. Entre sus sensores se encuentran un acelerómetro y un giroscopio, pero a diferencia del casco Oculus Rift, el casco Vive no cuenta con un magnetómetro ni con una cámara externa

(Borrego *et al.*, 2018). El sistema de rastreo del casco Vive era llamado *inside-out principle*, no requería cámaras externas, en su lugar se utilizaban dos láseres emisores llamados faros (figura 2). Estos dos faros mandaban alternadamente barridos de láser infrarrojos horizontales y verticales, que abarcaban 120 grados en cada dirección. En la superficie del casco Vive y sus controles se encuentran fotodiodos que indican cuando son golpeados por los láseres de los faros. La diferencia temporal en la que los fotodiodos detectan la señal permite recuperar la posición y orientación del HMD (Niehorster *et al.*, 2017).

En 2019, salió a la venta un modelo mejorado del casco Oculus Rift, llamado Oculus Rift S. Este nuevo sistema tiene una resolución de 2560x1440, pesa aproximadamente 400 gramos, su pantalla tiene una frecuencia de actualización de 80 Hz y puede rastrear de manera precisa los movimientos del usuario; la rotación de la cabeza, la traslación del cuerpo y la rotación-translación de las manos. En este sistema de rastreo se utilizan cinco cámaras internas montadas sobre el casco en sustitución de la cámara externa de la versión anterior. Para el rastreo del movimiento de la cabeza se emplean un giroscopio y un acelerómetro localizados en el casco. Las mediciones del acelerómetro y giroscopio se integran para obtener el desplazamiento y orientación del usuario. Sin embargo, con el paso del tiempo, estas mediciones se vuelven imprecisas debido a un error error de deriva, por lo que es necesario corregir la estimación de la posición y es aquí donde son utilizadas las cinco cámaras integradas del casco. En la siguiente sección se describe con más detalle el error de deriva y cómo se se ataca en el casco Oculus Rift S.

2.3. Error de deriva

Entre los sensores más importantes para rastrear el movimiento se encuentran los giroscopios y los acelerómetros. No obstante, existe un componente de ruido en las mediciones de estos sensores, causados por errores de calibración u otros factores intrínsecos del propio sensor, por lo que las mediciones registradas por los sensores difieren con las reales. Cuando se trata de calcular la orientación a partir de los datos del giroscopio se tienen que integrar los valores obtenidos, sin embargo, el hacer esto provoca que el error incremente linealmente con el tiempo, a la acumulación de este error se le conoce como error de deriva. Lo mismo ocurre cuando se integran las me-

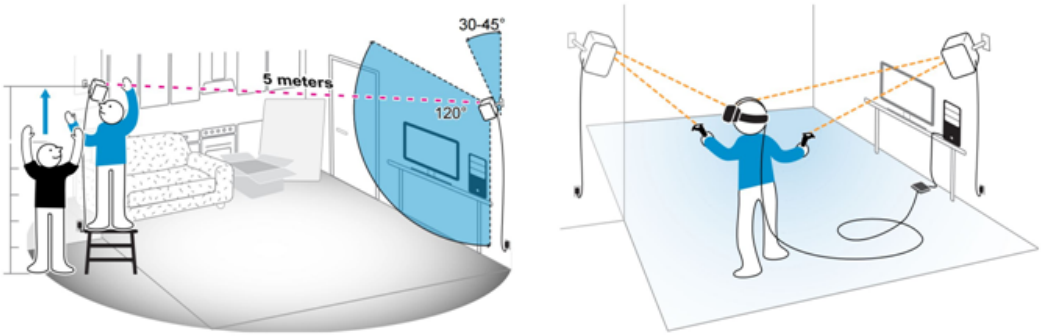


Figura 2. Diagrama de sensores exteroceptivos del casco HTC Vive obteniendo información del movimiento del usuario mediante detección de señales infrarrojas (Sensoryx, 2019).

diciones del acelerómetro para tratar de calcular velocidades, el error de deriva crece linealmente con el tiempo. Además, si se quiere calcular la posición a partir del acelerómetro, se debe integrar dos veces, lo cual provocará que el error de deriva crezca cuadráticamente con el tiempo, por lo que la estimación de la posición se vuelve imprecisa rápidamente (LaValle, 2016). Es por esto que es indispensable contar con un método que permita solucionar este problema.

Para tratar de corregir este error en los sensores de los HMD se utilizan sensores exteroceptivos. En el caso del casco Oculus Rift se utiliza una cámara externa que normalmente se posiciona frente al usuario, para el casco Vive se utilizan dos faros emisores de láseres infrarrojos que detectan fotodiodos situados en el casco. El casco Oculus Rift S, al igual que los dos anteriores, también utiliza sensores exteroceptivos, sin embargo, estos no son externos, sino que están situados sobre el casco (figura 3). El casco Oculus Rift S utiliza cinco cámaras colocadas sobre el casco, estas se encargan de tomar fotografías del ambiente que rodea al usuario, y a partir de ellas se genera un mapa 3D de la habitación, agregando marcas en puntos estratégicos que son observados continuamente y se utilizan como puntos de referencia para rastrear la posición del casco y los controles. Las marcas generadas en el mapa se actualizan constantemente, por lo que el sistema del casco Oculus Rift S debe rastrear la posición del usuario y al mismo tiempo crear un mapa de su entorno, este es un problema muy conocido en el área de la robótica llamado SLAM, por sus siglas en inglés (Simultaneous Localization And Mapping) (Hesch *et al.*, 2019).

Algo importante a recalcar, es que aunque únicamente se ha hablado sobre la co-

Figura 3. a) Casco Oculus Rift con una cámara externa que necesita ser posicionada en un punto fijo de la habitación. b) Casco Oculus Rift S con cinco cámaras integradas. En ambos casos las cámaras cumplen la misma función, corregir el error de deriva generado por los sensores propioceptivos utilizados para rastrear la posición del usuario (Borrego *et al.*, 2018) (Oculus, 2019).

rrección del error de deriva en ambientes virtuales, también existe un muy amplio estudio sobre su corrección en otras áreas, por ejemplo, corrección del error de deriva en sensores inerciales de teléfonos móviles (Chen *et al.*, 2018a), utilizados para obtener la trayectoria del usuario, o en sensores inerciales localizados en automóviles, para conducción autónoma (Clark *et al.*, 2017). También, en el área de la química, existen sensores inerciales para medir muestras químicas (Yan y Zhang, 2016), en los cuales también existe un error de deriva. Para los problemas donde se requiere una corrección de deriva se han implementado soluciones basadas en aprendizaje profundo, herramienta que ha proporcionado soluciones prometedoras en las áreas ya mencionadas. Debido a lo anterior, en esta tesis, se propondrá un algoritmo basado en aprendizaje profundo para el rastreo de la posición del HMD. En el capítulo 4 se presenta un descripción más detallada sobre la implementación de algoritmos utilizando aprendizaje profundo, sin embargo, para poder comprenderlos es necesario conocer el funcionamiento de una de las redes neuronales más importantes cuando se trabaja con series de tiempo.

2.4. Redes neuronales

Una característica de las redes neuronales, como las densas o convolucionales, es que no tienen memoria. Cada entrada es procesada independientemente, sin algún estado o relación entre las entradas anteriores. Con dichas redes, para procesar una secuencia o una serie temporal de datos, se debe mostrar la secuencia completa a la red a la vez, es decir, convirtiéndola en un único punto de datos. La información se mueve en una única dirección, de los nodos de entrada, a través de las capas ocultas

hacia la capa de salida. No hay ciclos o bucles en estas redes (Poznyak *et al.*, 2018).

En contraste, una red neuronal recurrente (RNN) procesa secuencias iterando a través de los elementos de la secuencia y manteniendo un estado que contiene información relativa de lo que ha visto hasta entonces. En efecto, una RNN es un tipo de red neuronal que tiene un ciclo interno (figura 4). El estado de una RNN se reinicia entre dos secuencias independientes (como dos reseñas de películas), así que cada secuencia se sigue considerando como un solo punto de datos (una sola entrada a la red). Lo que cambia es que este punto no se procesa en un solo paso, en su lugar, el ciclo interno de la red procesa cada elemento de la secuencia.

El mayor problema con las RNN, es que, aunque teóricamente sean capaces de retener información en el tiempo t de entradas vistas hace mucho tiempo atrás, en la práctica, esta dependencia a largo plazo es imposible de aprender, esto debido al problema de desvanecimiento de gradiente, que afecta también a las redes neuronales no recurrentes cuando tienen muchas capas. Para solucionar este problema se proponen las capas LSTM.

Las capas LSTM son una variante a las capas simples RNN, añaden una forma de llevar información a lo largo de las entradas. La información de una secuencia puede ser transportada a una entrada posterior, intacta y utilizada cuando se requiera. Esto es esencialmente lo que hacen las capas LSTM: guardan información para usarla más adelante, evitando así que las señales más antiguas desaparezcan gradualmente durante el procesamiento, haciendo que su desempeño sea mejor que el de las RNN.

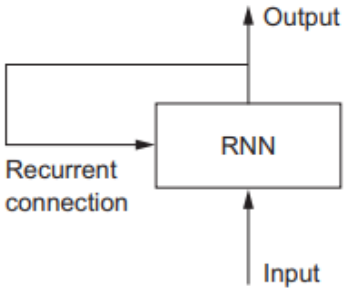


Figura 4. Diagrama de un ciclo en una red neuronal recurrente (Vázquez, 2018).

Capítulo 3. Trabajo previo

Detectar los movimientos del usuario y verlos reflejados en un mundo virtual fue uno de los retos más importantes que fue necesario resolver para llegar a lo que actualmente se conoce como cascos de realidad virtual. Este problema se complica debido a que los sensores propioceptivos que se utilizan para rastrear el movimiento no son perfectos, y llevan con ellos un error de deriva, el cual aumenta conforme pasa el tiempo y hace que el rastreo de movimiento sea impreciso. Para solucionar esto se han implementado diferentes metodologías, entre las que resaltan dos: el uso de sensores adicionales para corregir la posición del usuario, y la implementación de redes neuronales. Sin embargo, el uso de sensores adicionales implica un aumento en el costo del sistema y, además, su implementación no es sencilla. En este capítulo, se analizan diferentes métodos que se han ido desarrollando para tratar de corregir el error de deriva que se genera en sensores utilizados para rastrear el movimiento.

3.1. Rastreo por sensores añadidos

En el trabajo realizado por LaValle *et al.* (2014) se utilizan dos sensores para tratar de corregir el error de deriva ocasionado cuando se trata de estimar la orientación del usuario. Los autores dividieron el error de deriva en dos componentes, el error de inclinación (tilt error) y el error de dirección (yaw error). El primero de ellos corresponde a todos los componentes del error de deriva, a excepción de la rotación alrededor del eje vertical en el mundo físico, y el segundo, corresponde a la rotación alrededor del eje vertical (paralelo al vector de la gravedad). Como se puede notar, la unión de estos dos componentes forma en su totalidad el error de deriva en la orientación estimada (figura 5).

El error de inclinación se corrige utilizando un acelerómetro, el cual debe ser ajustado para que sus mediciones correspondan únicamente a las aceleraciones del objeto, ya que además de medir dicha aceleración, también mide la aceleración producida por la gravedad. El error de dirección se corrige empleando un magnetómetro, el cual se utiliza como si fuese una especie de “brújula”; se aprovecha que este sensor mide un vector que siempre apunta al norte. Sin embargo, en la práctica, la solución no es tan sencilla como hacer uso únicamente de estos dos sensores. El magnetómetro

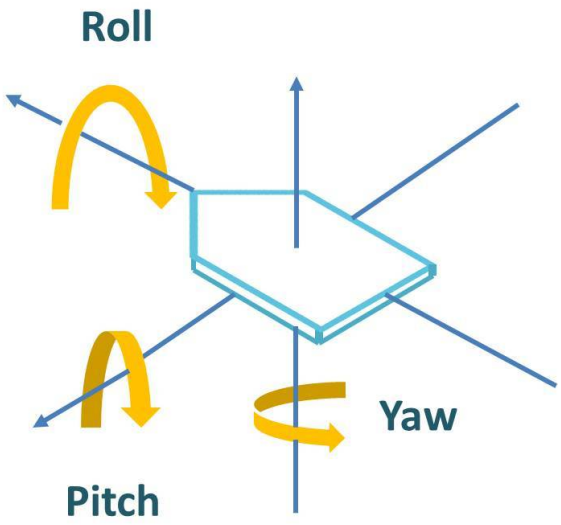


Figura 5. Representación de los 3 ejes rotacionales. El error de inclinación es la combinación del error de las coordenadas roll (alabeo) y pitch (elevación), mientras que el error de dirección es únicamente el error de la coordenada yaw (guiñada) (Shahid, 2017).

puede causar problemas cuando interfiere con otros campos magnéticos, por ejemplo, las placas de los circuitos en el casco generalmente contienen materiales ferrosos que generan campos magnéticos e interfieren con las mediciones del magnetómetro. Existen dos tipos de sesgos que pueden surgir, uno causado por los componentes cerca del sensor, que se ve como un desplazamiento constante en los resultados, y otro es el sesgo que surge por los campos magnéticos del exterior (metales en las paredes, edificios de los alrededores, etc.).

Para corregir el error de deriva cuando se trata de estimar la posición, se utiliza un método de restricciones cinemáticas, donde es posible modelar los movimientos de una manera sencilla, por ejemplo, si se sabe que el usuario estará sentado en una silla, se puede representar su torso y cabeza como barras acopladas. A partir de la aceleración de la cabeza, se puede deducir la velocidad lineal tanto de la cabeza como del torso, pero como el resultado puede contener error, necesita ser actualizado para ajustarse al modelo cinemático propuesto. Los resultados obtenidos aplicando este método son bastante precisos, el error en la posición del objeto después de un minuto es de 1.1 metros aproximadamente, mientras que en otros métodos, por ejemplo, utilizar la doble integración de las lecturas registradas por una unidad de medición inercial, o IMU, por sus siglas en inglés (Inertial Measurement Unit), el error puede ser de hasta 500 metros después de un minuto (LaValle *et al.*, 2014).

Otra manera de corregir el error de deriva es utilizando sensores exteroceptivos, como si fueran un par especial emisor-detector, donde el emisor envía una señal y el detector la recibe. Calculando el tiempo transcurrido entre el origen de la señal y la recepción de la misma, se puede estimar la posición del detector conociendo la velocidad a la cual se propaga la señal. Un sistema muy parecido a este es el que se empleó para el casco HTC Vive en el 2016. Se utilizó un sistema de faros y el Escáner Minnesota de 1989. El seguimiento se puede realizar sin problemas siempre que una de las señales del emisor (faros) pueda llegar a los sensores (dados al usuario en forma de controles). Se requiere un faro adicional para ayudar a la detección. Se utiliza una señal sincronizada cableada o inalámbrica para coordinar el orden de sincronización entre los dos faros, de modo que cada faro se turne para barrer los ejes X y Y simultáneamente para evitar interferencias. A diferencia de otros métodos de seguimiento, el faro no se comunica con computadoras ni recibe señales del dispositivo rastreado para fines de rastreo. Las señales de los faros son recibidas por los sensores ópticos de fotodiodos en los objetos a rastrear. En la práctica, el objeto rastreado utiliza más de quince receptores ópticos, para garantizar que al menos cinco sensores ópticos hayan recibido señales de los faros en un momento determinado, que es el requisito mínimo básico, esto debido a que con cinco detectores que hayan recibido una señal, o inclusive cuatro, es posible calcular la posición de los receptores a partir de una técnica llamada trilateración. La señal óptica se convierte en una marca de tiempo digital que representa el tiempo en el que el fotodiodo detectó la señal en ese sensor en particular. Sin embargo, la señal no contiene ninguna información orientativa. El software requiere la información geométrica de cada sensor. Un algoritmo calcula el ángulo relativo de cada sensor desde el faro y calcula la traslación y la rotación del objeto a seguir. La posición y orientación del objeto rastreado se calculan triangulando las posiciones relativas de los sensores ópticos y comparándolas con la posición geométrica de los sensores conocidos (Ng *et al.*, 2017).

3.2. Aprendizaje profundo para corrección de deriva

En el área de aprendizaje profundo también se han adaptado diferentes métodos para tratar de solucionar este problema. Yan y Zhang (2016) propusieron emplear una red neuronal artificial (autoencoder) para disminuir el error en las mediciones de

“narices electrónicas”, las cuales son instrumentos utilizados para medir muestras de gas. El error que se asocia a estos dispositivos es causado debido a un error de deriva que va aumentando gradualmente con el tiempo, como lo es también en el caso de giroscopios y acelerómetros.

Clark *et al.* (2017) desarrollaron VINet, una red neuronal que combina odometría inercial y odometría visual para el problema de conducción autónoma (figura 6). Una parte de la red se encarga de procesar las imágenes captadas por cámaras fotográficas y otra parte se encarga de procesar la información de un IMU localizado en el asiento del pasajero de un automóvil. Concatenando estos resultados la red es capaz de estimar la posición y orientación del automóvil.

Wu y Li (2020) proponen un algoritmo novedoso para los sensores de redes inalámbricas, ya que el error de deriva de estos sensores afecta la confiabilidad de los sistemas de monitoreo. El algoritmo se basa en técnicas de aprendizaje de máquina y en el filtro de Kalman para rastrear y calibrar el error de deriva de los sensores.

Yao *et al.* (2017) propusieron DeepSense, un framework basado en aprendizaje profundo que aborda directamente los problemas de calibración y ruidos mencionados anteriormente para las unidades de medición inercial, tales como acelerómetros y giroscopios. Esto se logra entrenando la red neuronal con datos de diversos IMUs, y así extraer relaciones locales de cada IMU para modelar interacciones globales.

Yan *et al.* (2020) propusieron un nuevo método llamado RoNIN (Robust Neuronal Inertial Navigation), aplicado para rastrear la trayectoria de usuarios utilizando un teléfono móvil (figura 7). Este método puede manejar diferentes orientaciones y posiciones del teléfono celular ya que se utiliza un marco de referencia especial que ajusta las lecturas del IMU y las hace independientes de la orientación. Para comprobar que esta idea es correcta, se desarrollaron tres arquitecturas diferentes, mostrando buenos resultados en estas.

Liu *et al.* (2020) proponen un framework basado en el filtro de Kalman para la estimación de estados usando únicamente un IMU (figura 8). Este artículo muestra una red que regresa las estimaciones de desplazamiento 3D y su incertidumbre, lo que brinda la capacidad de fusionar la medición del estado relativo con un EKF (Extended Kalman Filter) para resolver los sesgos de pose, velocidad y del sensor. La red puede producir

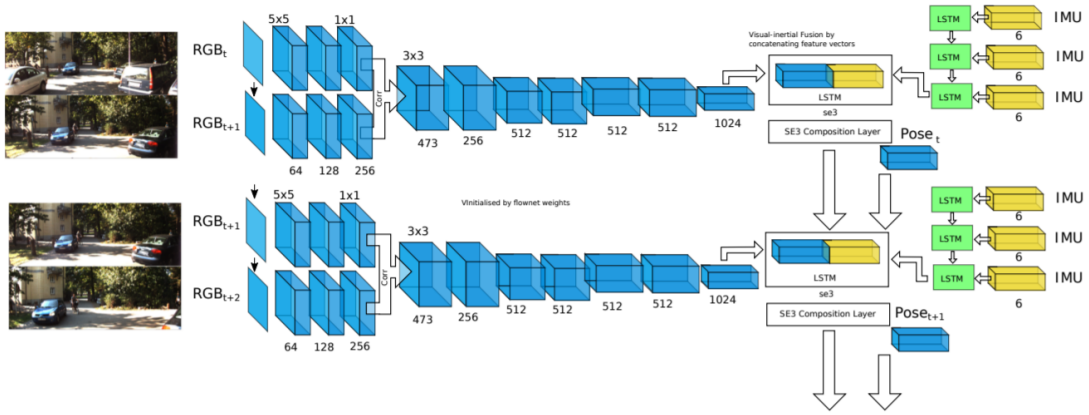


Figura 6. Diagrama de la red VINet (Clark *et al.*, 2017).

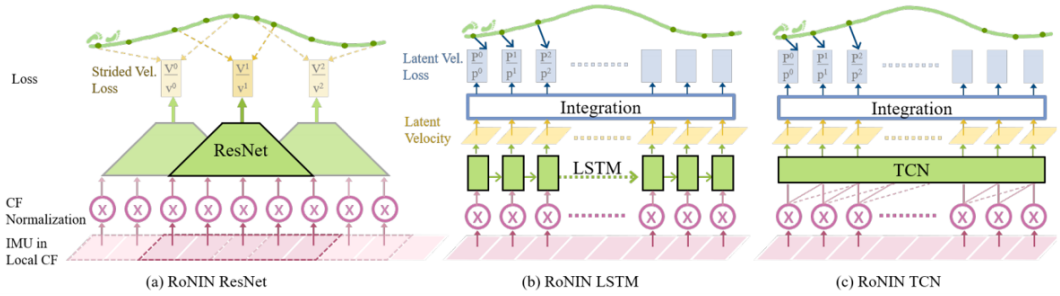


Figura 7. Las tres arquitecturas probadas con el metodo RoNIN: a) RoNIN ResNet, b) RoNIN LSTM, c) RoNIN TCN (Temporal Convolutional Network) (Yan *et al.*, 2020).

mediciones e incertidumbres estadísticamente consistentes para ser utilizadas como el paso de actualización en el filtro de Kalman.

Lim *et al.* (2019) mostraron RONet, una red LSTM bidireccional, es decir, una red LSTM que puede leer las entradas tanto de derecha a izquierda como de izquierda a derecha. Lo anterior permite a la red favorecerse de lecturas futuras y obtener así una mejor predicción, sin embargo, al requerir lecturas futuras existe un pequeño retraso entre las entradas de los datos y la predicción realizada (figura 9). La estructura de la red está compuesta por tres capas LSTM bidireccionales acopladas, siendo la entrada de la primer capa las lecturas de un IMU. La red fue utilizada para tratar de predecir en tiempo real la posición de un robot móvil.

Esfahani *et al.* (2020) propusieron una red neuronal de triple canal, llamada AbolDeepIO, para solucionar el problema del error de deriva en unidades de medición inercial, donde cada uno de los canales recibe entradas diferentes. El primero utiliza

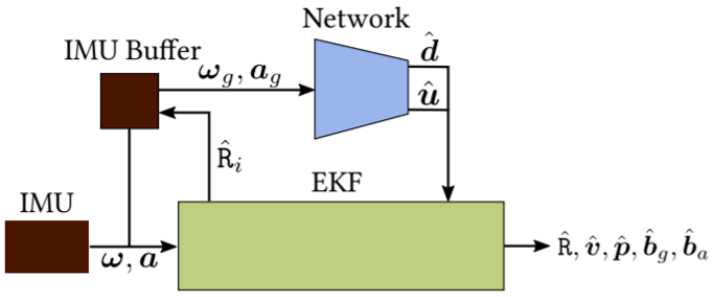


Figura 8. Diagrama del framework utilizado. Consiste en dos componentes importantes, la red y el filtro de Kalman. El *IMU buffer* proporciona segmentos de mediciones del IMU y la red genera el desplazamiento y la incertidumbre, que se utilizan como actualización de la medición en el filtro. El filtro estima la rotación, la velocidad, la posición y los sesgos del IMU (Liu *et al.*, 2020).

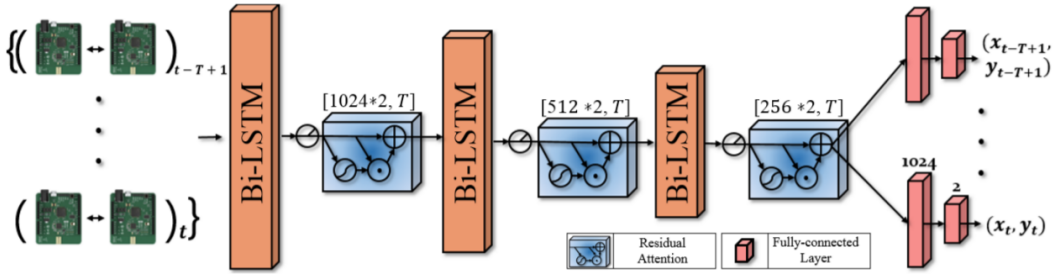


Figura 9. Diagrama de la red RONet. La capa LSTM bidireccional recibe como entrada las lecturas inerciales de un IMU y se procesan a lo largo de la red para predecir una posición (x, y) (Lim *et al.*, 2019).

las mediciones hechas por el acelerómetro, el segundo recibe las mediciones hechas por el giroscopio, y el tercero utiliza una entrada extra que no había sido considerada en otros trabajos, el tiempo de lectura entre mediciones hechas por el acelerómetro (figura 10).

Sun *et al.* (2021) desarrollaron el método llamado IDOL, para estimar la orientación 3D y la posición 2D de un IMU localizado en un teléfono celular (figura 11). El sistema completo consiste de dos partes, una se encarga de calcular la orientación del teléfono y la segunda de la traslación. La red es entrenada con datos del acelerómetro, giroscopio y magnetómetro del teléfono celular.

Asraf *et al.* (2021) propusieron un método llamado PDRNet, para predecir la trayectoria de usuarios utilizando un teléfono celular mientras caminan. Este problema conocido como *Pedestrian Dead Reckoning* usualmente se resuelve empleando datos empíricos biomecánicos, por ejemplo, la distancia que recorre una persona en cada paso o el tiempo entre pasos. Lo anterior ocasiona que la predicción dependa de la

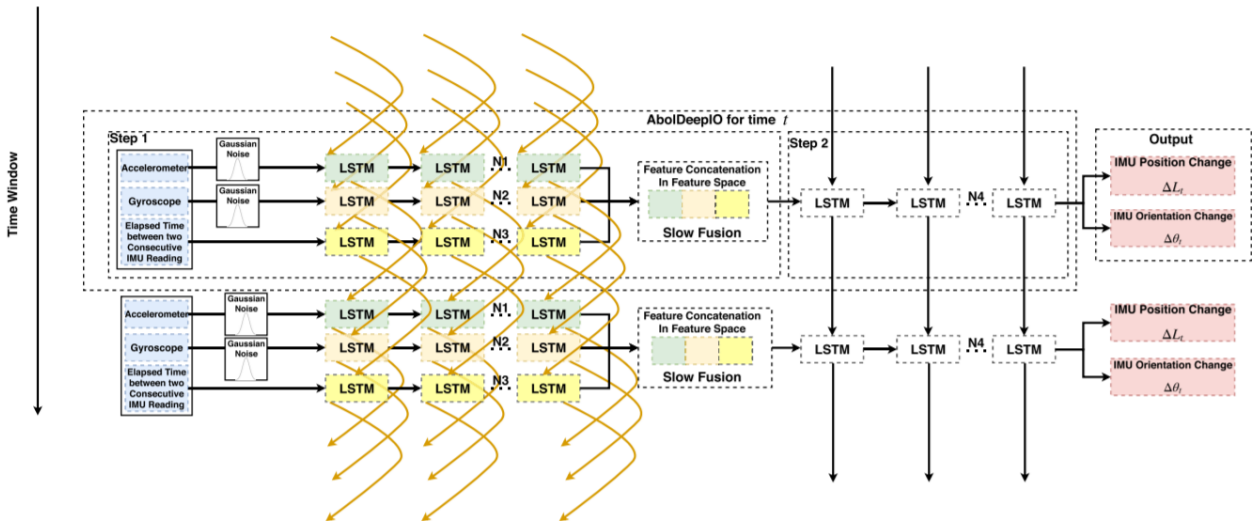


Figura 10. Diagrama de la red AbolDeepIO, donde se puede apreciar que las entradas de aceleración, velocidad angular y tiempo de muestreo se procesan de manera independiente por capas LSTM (Esfahani *et al.*, 2020).

forma de caminar de cada persona, es por ello que Asraf *et al.* (2021) buscan entrenar una red neuronal donde no sea necesario definir estos datos biomecánicos.

Chen *et al.* (2018a) propusieron una red neuronal recurrente llamada IONet, que puede generar trayectorias precisas a partir de una secuencia de datos extraídos de sensores inerciales (figura 12). Su método divide la información recolectada por los sensores en ventanas de muestreo, las cuales son independientes unas con otras, y obtiene la posición y orientación para cada una de ellas. De esta manera, el error de deriva que se va generando conforme pasa el tiempo, no va acumulándose entre ventanas, haciendo que los resultados obtenidos por el algoritmo sean más precisos. Sin embargo, los resultados de este trabajo únicamente funcionan para trayectorias en dos dimensiones.

Además de las redes neuronales mencionadas anteriormente, se encontró una que fue de gran importancia en el desarrollo de este trabajo. La red propuesta por Lima *et al.* (2019), en comparación con los modelos anteriores, es capaz de realizar odometría para seis grados de libertad. Toma como entrada una secuencia de lecturas realizadas por un giroscopio y un acelerómetro, y devuelve una posición relativa entre dos secuencias. Realizando esta operación consecutivamente, se puede estimar una posición 3D a partir de una posición y orientación inicial. En el capítulo 5 se hablará más detalladamente sobre esta red.

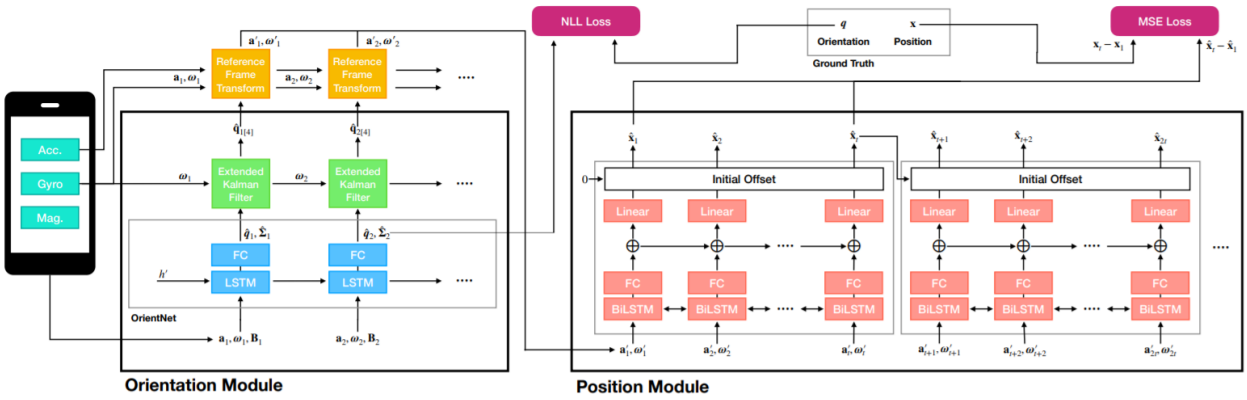


Figura 11. Diagrama del sistema IDOL. La primera parte de la red, encargada de predecir la orientación, esta compuesta de una red LSTM, un filtro de Kalman y una transformación a un nuevo marco de referencia, mientras que la segunda parte, encargada de predecir la traslación, es una red conformada de capas LSTM bidireccionales (Sun *et al.*, 2021).

A partir de la revisión de la literatura anterior, en este trabajo surgió la pregunta sobre si es posible aplicar un método de aprendizaje profundo para calcular y representar de manera precisa, en un mundo virtual, la trayectoria de un usuario que lleva puesto un casco de realidad virtual. Lo anterior, sin la necesidad de tener que utilizar sensores exteroceptivos para corregir el error de deriva. Los capítulos cinco y seis de este trabajo se enfocaron en tratar de dar una respuesta a dicha pregunta.

3.3. Bases de datos en la literatura

Como podemos imaginarnos, para entrenar un red neuronal se requiere una gran cantidad de datos del problema a solucionar. De los trabajos mencionados anteriormente, la gran mayoría utiliza datos de un IMU localizado en un teléfono celular. En algunos de esos trabajos los autores han empleado bases de datos creadas y subidas a la red por terceras personas, mientras que para otros han generado su propia base de datos. Sin embargo, la creación de una base de datos no es tarea sencilla, ya que se requiere una considerable cantidad de tiempo para poder recolectar los datos necesarios.

Por ejemplo, hablando únicamente sobre sistemas de corrección de deriva, encontramos el problema de navegación de peatones, donde se trata de obtener la trayectoria de personas mientras sostienen un teléfono celular. Para este problema en particular existen diversas bases de datos en la red, estructuradas en diversos mo-

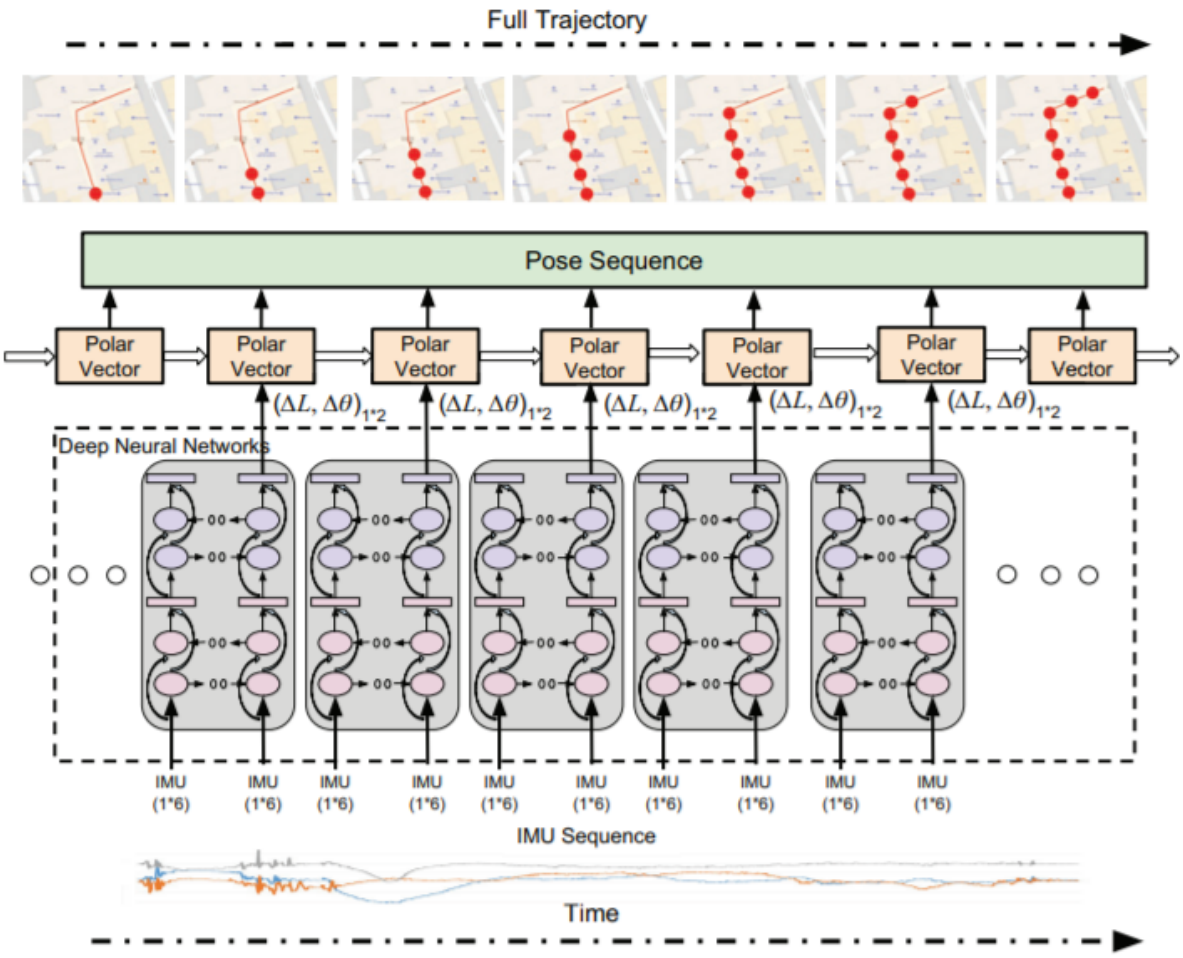


Figura 12. Diagrama de la red IONet. Las entradas de la red son ventanas de 200 lecturas inerciales formadas por la aceleración y velocidad angular. Las capas LSTM bidireccionales se encargan de predecir un vector polar que contiene el desplazamiento y el cambio de orientación en cada ventana, concatenando los resultados de cada ventana se puede estimar la posición del usuario (Chen *et al.*, 2018a).

vimientos y posiciones del teléfono, como, por ejemplo, datos donde el teléfono es cargado en una bolsa de mano, en un bolsillo del pantalón, o en la mano mandando mensajes de texto mientras se camina. También se dividen en movimientos donde la persona esté corriendo, caminando normalmente o caminando lento. Algunas de estas bases de datos contienen información de lecturas inerciales que corresponden a recorridos de entre 4.5 kilómetros (Reina *et al.*, 2018) y 45 kilómetros (Chen *et al.*, 2020).

En el problema de conducción autónoma también se tiene que tratar el error de deriva generado en sensores inerciales, por lo que es necesario obtener datos de un IMU localizado en un automóvil. Para este caso se encuentra la base de datos de Mad-

dern *et al.* (2017) que contiene más de 1000 kilómetros de información recolectada a lo largo de un año, y para conducción autónoma con odómetro visual, se encuentra la base de datos de Geiger *et al.* (2012) que contiene imágenes de un recorrido de 40 kilómetros aproximadamente.

De lo anterior, podemos concluir que contar con una base de datos existente para el problema que se busca resolver en esta tesis facilitaría nuestro trabajo. Sin embargo, al comienzo del mismo no se había documentado ninguna base de datos en la literatura que contenga información inercial de un IMU localizado en un casco de realidad virtual, por lo que fue necesario crearla. Al final del desarrollo de esta tesis, en agosto del 2021, se publicó una base de datos con lecturas inerciales de un casco de realidad virtual (Kovalenko *et al.*, 2021), sin embargo, esta base únicamente contaba con 30 minutos de lecturas, lo cual esta por debajo de lo utilizado en trabajos basados en aprendizaje profundo para la estimación de trayectorias a partir de datos inerciales.

Capítulo 4. Base de datos

Para entrenar una red neuronal, es necesario contar con una gran cantidad de datos. Normalmente, cuando se requiere utilizar una red neuronal para alguna tarea es posible encontrar en la red diversas bases de datos relacionadas con el trabajo por hacer. Por ejemplo, existen bases de datos de imágenes, palabras, series de tiempo, etc. En nuestro caso, necesitamos entrenar una red neuronal que reciba como entrada datos inerciales obtenidos por un IMU colocado en un casco de realidad virtual y realice una predicción de la posición del usuario. Sin embargo, una base de datos que contenga estas características no se ha documentado en la red, por lo que fue necesario crear nuestra propia base de datos.

4.1. Características de los datos

En este trabajo, se busca entrenar una red que reciba como entrada datos inerciales obtenidos de un IMU localizado en un HMD y prediga la posición de dicho IMU. Para crear el conjunto de entrenamiento se necesitan los datos de entrada y de salida. Los datos de entrada son lecturas inerciales representadas por un vector $(a_x.a_y, a_z)$ de la aceleración y un vector $(\omega_x, \omega_y, \omega_z)$ de la velocidad angular, por lo que el vector de entrada para nuestra red será de seis dimensiones. Ahora, la red debe ser capaz de tomar estos datos inerciales y convertirlos a una trayectoria, por lo que es necesario obtener los valores de salida con los que se comparará la red para cuantificar qué tan precisa es la predicción. Estos datos de comparación serán la trayectoria real seguida por el usuario y se obtendrán a partir de la estimación corregida por el casco de realidad virtual mediante un software de computadora. Los datos de comparación, serán los puntos (x, y, z) que representan la posición real del usuario en cada instante de tiempo.

Como este trabajo se centra en predecir la trayectoria de una persona utilizando un casco de realidad virtual, se busca que los datos inerciales de la base de datos representen los movimientos más usuales que se realizan mientras se utiliza un HMD. Para identificar estos movimientos se hizo una investigación sobre los principales usos de los cascos de realidad virtual y se observó que se emplean mayormente para videojuegos. A partir de lo anterior, se buscaron cuáles son las aplicaciones más populares para

observar los movimientos que realizan las personas cuando utilizan dichas aplicaciones. Entre los juegos más populares para realidad virtual se encuentran Beat Saber, donde el usuario necesita moverse de izquierda a derecha y agacharse. Otro juego importante es Half-Life: Alix, donde a diferencia del anterior, el desplazamiento del usuario es opcional y se centra mayormente en el rastreo de las manos y en la orientación de la cabeza. De esta investigación se decidió que los movimientos que contendrá la base de datos serán obtenidos de actividades hechas por el usuario donde necesite moverse constantemente. Como se requiere que la base de datos contenga una gran variedad de movimientos, se decidió separar los datos en categorías, donde cada categoría contiene un tipo distinto de movimiento. Esto, además, es una ventaja al momento de entrenar una red neuronal, ya que se busca que los datos utilizados tengan similitud para así reducir el error de la predicción. Las categorías en las que se separó son:

- Movimientos siguiendo una trayectoria en forma de cuadrado.
- Movimientos de traslación libre y agachándose.
- Únicamente movimientos rotacionales de la cabeza, sin trasladarse.
- Movimientos rotacionales de la cabeza con traslación.
- Movimientos traslaciones de izquierda a derecha únicamente.
- Movimientos de izquierda a derecha y agachándose, sin desplazarse del punto inicial.

Una vez diseñada la estructura de la base de datos nos vimos en la tarea de investigar cómo acceder a los sensores el casco de realidad virtual para empezar a obtener los datos requeridos.

4.2. Recolección de los datos inerciales

La obtención de los datos inerciales del IMU del casco de realidad virtual fue una tarea compleja. El primer reto enfrentado fue que no se conocía cómo acceder a los sensores del casco; encontramos que mediante dos frameworks era posible acceder

a funciones internas del casco que normalmente no están disponibles para el usuario. Los frameworks que permiten esto son Unreal Engine y Unity. Estos dos programas son motores gráficos utilizados para la creación de videojuegos, tanto para desarrolladores independientes como profesionales. Ambos programas permiten la creación de aplicaciones de realidad virtual y es posible programarlas para que permitan el acceso a los datos del IMU del casco y poder guardar así las lecturas realizadas. El siguiente reto a superar fue desarrollar aplicaciones de realidad virtual donde el usuario simulara los movimientos requeridos para nuestra base de datos. Por lo que fue necesario aprender a manipular Unreal Engine y Unity, para la creación de aplicaciones en realidad virtual. Cuando los usuarios utilizaban estas aplicaciones realizaban los movimientos requeridos y al mismo tiempo se iban almacenando las lecturas del IMU.

Se comenzó desarrollando aplicaciones simples de realidad virtual en ambos motores gráficos, donde únicamente se podía observar los alrededores sin desplazarse. El entorno de programación de Unreal Engine utiliza un esquema de programación basado en la conexión de componentes (blueprints) para ejecutar las acciones de la aplicación, mientras que Unity se basa en C# para programar todas las acciones. Una vez familiarizados con el entorno de programación, se procedió a crear aplicaciones más sofisticadas donde el usuario fuera capaz de trasladarse e interactuar con el ambiente. Una vez creado el ambiente procedimos a recolectar los datos inerciales del casco. Para poder acceder a las lecturas de los sensores se deben utilizar librerías más complejas, sin embargo, como son procedimientos que usualmente no se suelen utilizar al momento de crear aplicaciones, no están detalladas y la documentación de ellas es escasa. En ambos motores gráficos se encontraron funciones que permiten acceder a los sensores del casco, sin embargo, debido a su simplicidad de manejo y la documentación disponible, decidimos utilizar Unity como nuestro motor gráfico para desarrollar las aplicaciones que nos servirán para obtener los datos inerciales requeridos.

Utilizando las funciones que permiten acceder a los sensores del casco es posible obtener los siguientes datos:

- Aceleración traslacional.
- Aceleración angular.

- Velocidad angular.
- Posición.
- Orientación (cuaternión).
- Velocidad traslacional.

La posición real del usuario es obtenida directamente del casco. Esta posición real o “ground truth” es calculada por el HMD resolviendo el problema de SLAM utilizando las cinco cámaras que porta el casco, por lo que es una trayectoria con alta confiabilidad. Según el trabajo de Jost *et al.* (2019); **?**). Para nuestro trabajo únicamente necesitamos la aceleración, velocidad angular y posición, sin embargo, para que nuestra base de datos pudiera ser utilizada para otros fines se decidió almacenar todos los datos posibles a los que tenemos acceso.

Una vez que se logró acceder a los sensores del HMD, se programó una función que guardara los datos en un archivo delimitado por comas (csv). Haciendo un análisis de los datos se observó que pertenecen a un marco de referencia global, lo cual representa una ventaja ya que en la mayoría de los trabajos previos analizados, los datos obtenidos por los IMU corresponden a un marco de referencia local, por lo que se tenía que hacer un preprocesado de los mismos antes de poderlos utilizar. La tasa de actualización de los datos enviados por el casco es de 80 Hz, a pesar de que el IMU instalado en el mismo tiene una frecuencia de 1000 Hz (Desai *et al.*, 2014). Esto se debe a que las librerías disponibles en Unity solo son capaces de obtener las lecturas inerciales con la misma frecuencia de actualización de la pantalla del casco, y al ser una pantalla de 80 Hz, los datos se envían a esta frecuencia. Creemos que debido a que el casco Oculus Rift S es un proyecto comercial no se le permite al usuario tener control total del funcionamiento de los sensores.

4.3. Aplicaciones desarrolladas

Una vez que se logró acceder a las lecturas de los sensores, se procedió a crear las diferentes aplicaciones que nos ayudarían a capturar los datos requeridos.

4.3.1. Aplicación #1

Para obtener los datos inerciales de una trayectoria fija, se diseñó una aplicación donde el usuario podía ver sus pies sobre un camino a seguir, este camino tenia forma de cuadrado con un tamaño de $2x2$ metros (figura 13(a)). Cuando los usuarios participaban en este juego, todos iniciaban en el mismo punto del cuadrado, pero la dirección del movimiento era elegido por ellos.

4.3.2. Aplicación #2

Para que el usuario realizara movimientos caminando y agachándose simultáneamente, se creó un ambiente donde las personas pasan por debajo de tres obstáculos, estos obstáculos estaban colocados a diferentes alturas, el primero a una altura de 1.75 metros, el segundo a 1.65 metros y el tercero a 1.45 metros (figura 13(b)). De esta manera los usuarios podían desplazarse libremente por el ambiente y se veían forzados a agacharse.

4.3.3. Aplicación #3

Para obtener datos donde el usuario hiciera únicamente rotaciones de la cabeza sin trasladarse, se diseño una aplicación donde el objetivo era voltear a ver una luz encendida (figura 13(c)). La habitación que rodeaba al usuario estaba oscura y contaba con varias luces apagadas, únicamente había una luz encendida, y cuando el usuario la localizaba, presionaba un botón con el mando para apagarla y encender otra al azar. De esta manera el usuario tenia que voltear constante a sus alrededores sin necesidad de trasladarse de su punto inicial.

4.3.4. Aplicación #4

Para combinar los movimientos de rotación de la cabeza y traslaciones, se creó una aplicación donde los usuarios recorrían libremente un ambiente (figura 13(d)). Aquí podían voltear a ver los alrededores y desplazarse desde su punto inicial para explorar

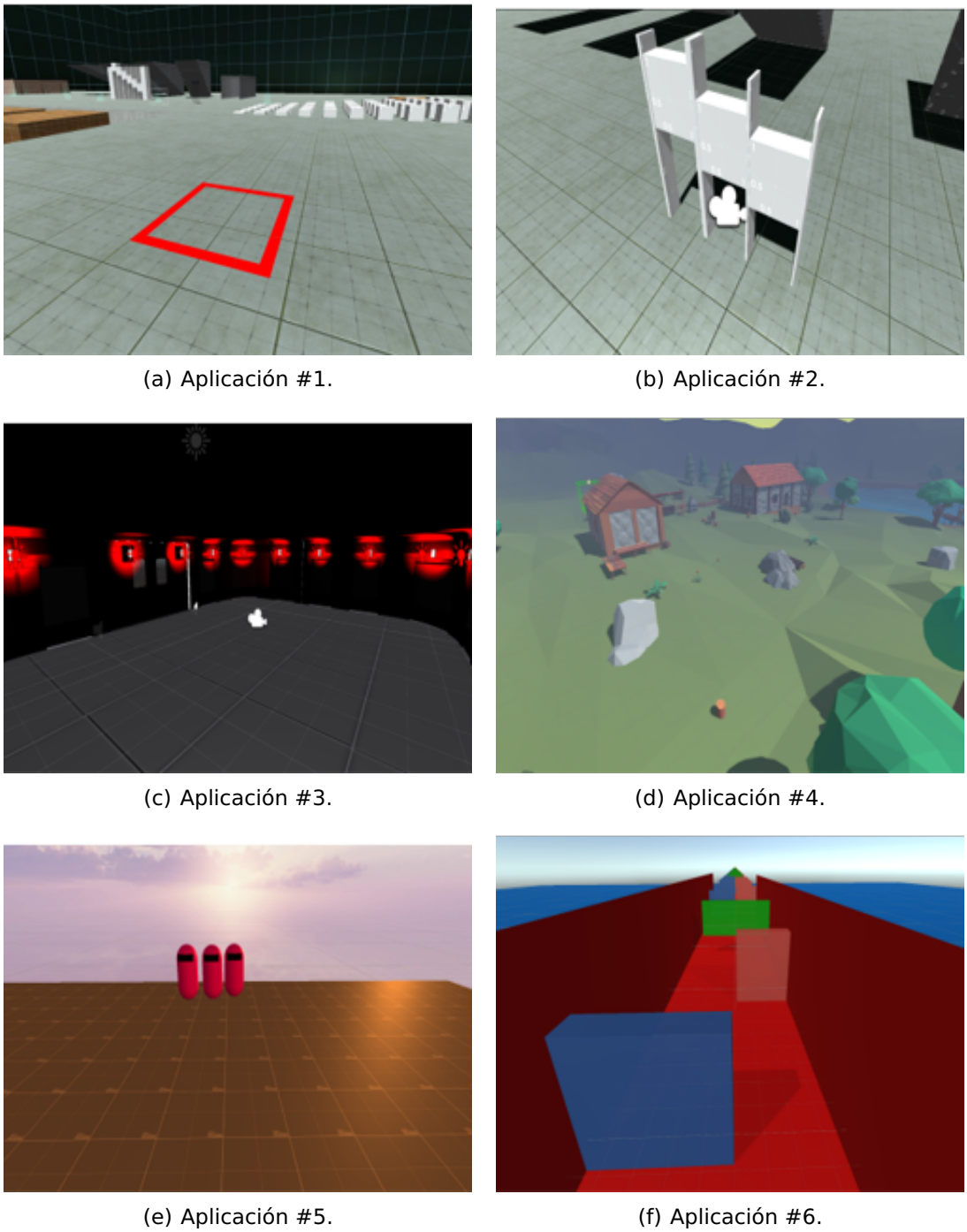
el ambiente. También se contaba con una opción para teletransportarse a distancias más lejanas.

4.3.5. Aplicación #5

Para lograr que el usuario se desplazara de izquierda a derecha se diseñó una aplicación donde el participante es el objetivo de tres enemigos que le disparan constantemente, el jugador debía moverse rápidamente a los lados tratando de esquivar los proyectiles (figura 13(e)). Cada uno de los tres enemigos disparaban proyectiles a un ritmo constante, uno lanzaba el proyectil cada 6.5 segundos, el segundo cada 4.5 segundos y el tercero cada 3 segundos.

4.3.6. Aplicación #6

Finalmente, en la ultima aplicación era necesario que el usuario realizara movimientos de izquierda a derecha y agachándose pero sin trasladarse de su posición original. Para esto, se creó una aplicación donde el usuario tenia que esquivar diferentes barreras que se acercaban hacia él (figura 13(f)). Para esquivar las barreras únicamente era necesario mover la parte superior del torso, por lo que los participantes no se desplazaban de su punto inicial.



(a) Aplicación #1. (b) Aplicación #2.

(c) Aplicación #3. (d) Aplicación #4.

(e) Aplicación #5. (f) Aplicación #6.

Figura 13. Aplicaciones creadas para la recolección de datos inerciales.

Una vez creados los ambientes, se procedió a recolectar los datos de cada aplicación. Se reunió un total de 15 personas, con edades comprendidas entre los 25 y 60 años. Todos los participantes afirmaron tener una buena salud motriz, por lo que no tenían dificultades para la realización de los movimientos requeridos en cada aplicación. Cada persona participó en las actividades de las aplicaciones por cinco minutos, y al

tener un total de seis aplicaciones, cada persona utilizaba el casco por un total de 30 minutos. Este tiempo de uso es adecuado para personas que nunca han experimentado con un casco de realidad virtual, ya que después de cierto tiempo puede causar mareos. Después de que todos los participantes hicieran uso de las aplicaciones, se logró obtener un total de 7.5 horas de lecturas inerciales.

Después de recolectar los datos inerciales, se procedió a realizar pruebas con diferentes redes neuronales. En el siguiente capítulo se mostrarán las arquitecturas utilizadas y su funcionamiento.

Capítulo 5. Metodología para rastreo de posición

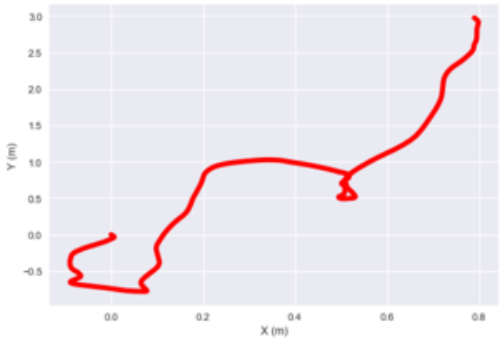
En este capítulo, se muestra el efecto que tiene el error de deriva cuando se trata de estimar la posición a partir de los datos inerciales de un acelerómetro. También se describe a detalle la red creada por Lima *et al.* (2019), 6DOF IO (6 Degree Of Freedom Inertial Odometry) la cual fue de gran importancia en el desarrollo de este trabajo. Finalmente, se discuten las modificaciones realizadas a esta red y el por qué se decidió hacerlo.

5.1. Visualización del error de deriva

Para comenzar este trabajo, primero fue necesario entender el error de deriva y observar cómo es que se genera. Para ello, se recolectaron datos inerciales con un teléfono celular Samsung S7 mientras se recorría una trayectoria dada. El IMU era capaz de realizar mediciones de la aceleración, velocidad, velocidad angular, aceleración angular, campo magnético y orientación. De estos datos nos interesa la aceleración, ya que al integrarla dos veces obtendremos una estimación de la posición del IMU. Para tener una referencia de la trayectoria esperada, se trazaron dos caminos en el piso los cuales debía seguir durante 30 segundos el usuario que portaba el teléfono. Durante ese tiempo el IMU recolectaba la información. La primera trayectoria tenía forma de rectángulo con dimensiones de $3.5x2.5m$, y la otra, un poco más compleja, una forma similar al número “2”.

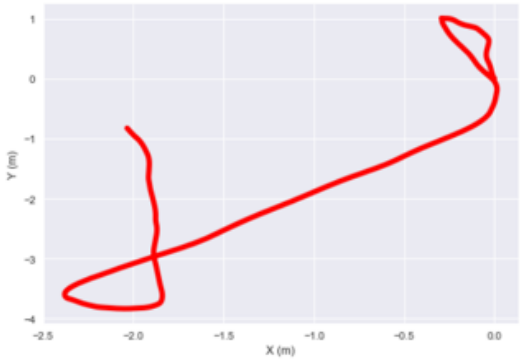
Como se puede apreciar en la figura 14, en teoría, al integrar dos veces la aceleración es posible estimar una posición, sin embargo, el error entre la trayectoria real y la trayectoria estimada se hace notar rápidamente. Realizando pruebas con los datos se pudo observar que el método de la doble integración funciona bien para escalas de tiempo muy pequeñas, entre los 5 y 15 segundos, después de ese tiempo el error de deriva comienza a crecer rápidamente y se pierde toda conexión con la trayectoria real. A partir de los resultados anteriores es posible notar cómo el error de deriva juega un papel importante cuando se trata de estimar la posición y el por qué es necesario encontrar una manera de reducirlo.

(a) Trayectoria en forma de cuadrado.



(b) Trayectoria estimada de la figura (*a*) después de integrar dos veces la aceleración.

(c) Trayectoria en forma de “2”.



(d) Trayectoria estimada de la figura (*c*) después de integrar dos veces la aceleración.

Figura 14. Recorridos de prueba para comprobar el efecto que tiene el error de deriva después de graficar 30 segundos de datos inerciales.

5.2. Odometría para 6 grados de libertad

La red propuesta por Lima *et al.* (2019), llamada 6DOF IO, fue un trabajo clave en el desarrollo de esta tesis. La red realiza odometría con 6 grados de libertad, tomando como entrada únicamente la aceleración lineal y velocidad angular del IMU (figura 15). Al igual que en la red Ionet de Chen *et al.* (2018a), mencionada anteriormente, la entrada es una ventana de 200 datos inerciales, donde cada dato contiene un vector 3D de velocidad angular $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$ y un vector 3D de aceleración $\mathbf{a} = (a_x, a_y, a_z)$. La información obtenida con el giroscopio y el acelerómetro es procesada por separado por dos capas convolucionales 1D, con 128 neuronas y un kernel de tamaño 11. Después de estas dos capas, se utiliza una capa max pooling de tamaño tres. La salida de estas capas es concatenada y utilizada para alimentar dos capas bidireccionales

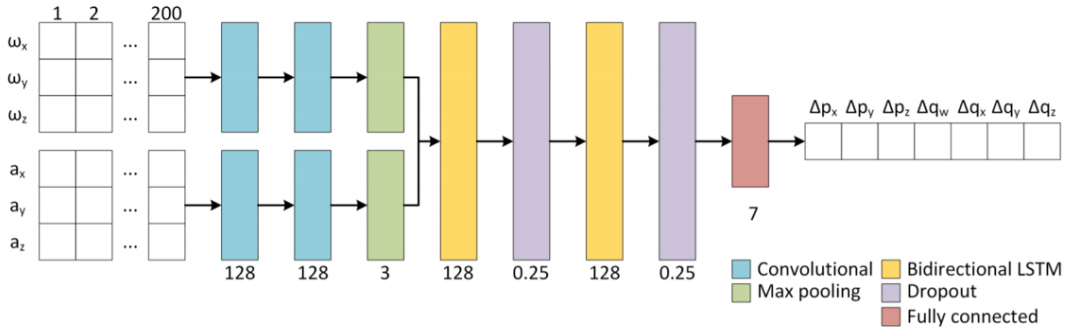


Figura 15. Arquitectura de la red utilizada (Lima *et al.*, 2019).

LSTM con 128 neuronas, por lo que 100 lecturas anteriores y 100 lecturas posteriores del IMU tienen una influencia en la posición calculada. Después de cada capa LSTM se pasa la información por una capa dropout al 25 % para evitar el sobre-entrenamiento. Finalmente, se utiliza una capa completamente conectada que genera como salida la posición relativa, compuesta por un vector de 7 dimensiones, donde los primeros 3 componentes son los cambios en la posición $\Delta\mathbf{P} = (\Delta p_x, \Delta p_y, \Delta p_z)$ y los últimos 4 son el cambio en la orientación $\Delta\mathbf{Q} = (\Delta q_w, \Delta q_x, \Delta q_y, \Delta q_z)$.

Las ventanas consecutivas de lecturas del IMU tienen un paso de 10 lecturas, es decir, la primer ventana contendrá de las lecturas #1 a la #200, la segunda ventana de la #20 a la #210, y así consecutivamente, como se aprecia en la figura 16. En este caso, se calcula una nueva posición cada 10 lecturas. Dada una ventana de 200 lecturas, la posición relativa calculada ocurre entre los lecturas #95 y #105. Esto permite una composición de posiciones para estimar una trayectoria con un ligero retraso de acuerdo con el tiempo de muestreo. También permite que las capas LSTM se beneficien de las lecturas pasadas y futuras para mejorar la predicción. Es posible evitar el retraso que se genera por utilizar lecturas futuras, pero eso implicaría una pérdida en la precisión de la red ya que se tendrían que reemplazar las capas LSTM bidireccionales por capas LSTM normales. La red 6DOF IO originalmente fue entrenada con la base de datos OxIOD, creada por Chen *et al.* (2018b), que es la misma base de datos utilizada para entrenar la red IONet. Esta base de datos contiene un total de 42.5 km de recorrido en un tiempo de 14.7 horas (Chen *et al.*, 2020).

Existen diferentes enfoques para representar una pose con 6 grados de libertad. Uno es extender el sistema de coordenadas polar (propuesto en IONet) a un sistema

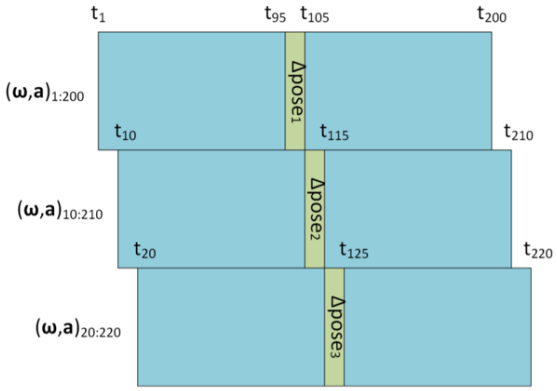


Figura 16. Esquema de la organización de los datos estructurados en forma de ventanas, en esta imagen se muestran tres ventanas consecutivas de tamaño 200 con lecturas superpuestas. (Lima *et al.*, 2019).

de coordenadas esférico. En el sistema de coordenadas esférico, la pose se representa como un cambio en la distancia recorrida Δl, el cambio en la inclinación $\Delta\Theta$ y el cambio de dirección $\Delta\Psi$. A partir de una posición previa $(x_{(t-1)}, y_{(t-1)}, z_{(t-1)})$, una inclinación previa $\Theta_{(t-1)}$ y una dirección previa $\Psi_{(t-1)}$, la posición actual (x_t, y_t, z_t) después de un cambio $(\Delta l, \Delta\Theta, \Delta\Psi)$ está dada por

$$x_t = x_{t-1} + \Delta l \cdot \sin(\theta_{t-1} + \Delta\Theta) \cdot \cos(\Psi_{t-1} + \Delta\Psi), \quad (1)$$

$$y_t = y_{t-1} + \Delta l \cdot \sin(\theta_{t-1} + \Delta\Theta) \cdot \sin(\Psi_{t-1} + \Delta\Psi), \quad (2)$$

$$z_t = z_{t-1} + \Delta l \cdot \cos(\theta_{t-1} + \Delta\Theta), \quad (3)$$

sin embargo, un inconveniente de este método es que la orientación solo será consistente cuando se produzca un movimiento hacia adelante. Por ejemplo, si el usuario se mueve hacia atrás o hacia los lados sin cambio en la orientación, esto se interpretará como un movimiento hacia adelante junto con un cambio de orientación en la dirección del movimiento. Por lo que la red 6DOF IO consideró otro enfoque donde se usa un vector 3D de posición Δp y un cuaternión unitario Δq que representa la orientación. Esta representación maneja correctamente la orientación cuando se hacen movimientos en cualquier dirección sin necesidad de girar. Desde una posición previa $p_{(t-1)}$ y una orientación $q_{(t-1)}$, la posición actual p_t y orientación q_t después de un cambio en la pose $(\Delta p, \Delta q)$ está dada por:

$$p_t = p_{t-1} + R(q_{t-1})\Delta p, \tag{4}$$

$$q_t = q_{t-1} \otimes \Delta q, \tag{5}$$

donde $R(q)$ es la matriz de rotación para q y $\otimes$ es el producto Hamiltoniano. Utilizando este método se podrá ser capaz de reconstruir una trayectoria 3D.

Para el entrenamiento, es necesario cotejar las trayectorias predichas por la red con las verdaderas, por lo que se requiere definir una función de pérdida que pueda compararlas. Un método sencillo para estimar la distancia entre la posición real y la posición predicha por la red es calculando su error cuadrático medio (MSE), con la función de pérdida $\mathcal{L}_{MSE}$. Sin embargo, MSE es una función que mide únicamente distancias algebraicas y, en este caso, se necesita una función de distancia geométrica capaz de calcular el error en la posición y en la orientación. Por lo que considerar una representación de la pose con 6 grados de libertad que utiliza cuaterniones, en el trabajo de Lima *et al.* (2019) se definió una función de pérdida que se relaciona con la diferencia geométrica entre el valor real (p, q) y la posición predicha $(\hat{p}, \hat{q})$.

$$\mathcal{L}_{TMAE} = \|\hat{p} - p\|_1, \tag{6}$$

$$\mathcal{L}_{QME} = 2 \cdot \left\| imag(\hat{q} \otimes q^*) \right\|_1. \tag{7}$$

La forma más directa de calcular la pérdida para un problema de odometría con 6 grados de libertad es asumir una ponderación uniforme de las pérdidas para cada salida, como la rotación y posición. Sin embargo, el peso que se utilice para una salida afecta en gran medida los resultados de la otra ya que estas tienen diferente naturaleza y escala, por lo tanto, se necesita que cada salida sea tratada por separado. Para atacar este problema, Lima *et al.* (2019) desarrollaron un framework multi-tarea para encontrar la mejor ponderación de pérdidas para las n tareas (en este caso dos, estimación en el cambio de posición y orientación).

5.3. Modificaciones de la red

El trabajo de Lima *et al.* (2019), descrito previamente, muestra algunos de los resultados más precisos reportados en la literatura para predicción de trayectorias a partir de sensores inerciales, es por esto que se eligió como punto de partida. Como primer paso, se tomó la arquitectura original de la red 6DOF IO y se entrenó con la base de datos creada en este trabajo, de esta manera se pudo verificar que la arquitectura funciona correctamente y que los datos recolectados entrenaron satisfactoriamente la red. Sin embargo, el método propuesto por Lima *et al.* (2019) presenta un error muy notable en el eje cardinal paralelo a la gravedad (altura), el cual se manifiesta tanto con la base de datos que fue entrenado originalmente como con la nuestra (ver capítulo 6), por lo que creemos que existe algún componente en la red que puede ocasionarlo. Por esta razón se decidió hacer modificaciones a la red y entrenar modelos con diferentes características, por ejemplo, cambiar el tamaño de la ventana, el número de archivos, la función de pérdida, etc. A continuación, se presentan las diferentes modificaciones que se implementaron en la arquitectura de la red, la información que se utilizó como su entrada y variaciones en la función de pérdida. 3

5.3.1. Red original: $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{L}}, \Delta\hat{\mathbf{Q}})$

En forma de resumen, en la figura 17 se muestra un diagrama general del funcionamiento de la red 6DOF IO.

La red recibe como entrada la velocidad angular y aceleración $(\boldsymbol{\omega}, \mathbf{a})$ y como resultado predice un vector de posición local $\Delta\hat{\mathbf{P}}_{\mathbf{L}}$ y un vector cuaternión $\Delta\hat{\mathbf{Q}}$. En este caso, los vectores de posición pertenecen a un marco de referencia local respecto al IMU. Una vez entrenada la red, los vectores predichos deben ser convertidos a un marco de referencia global para poder ser comparados con la trayectoria real y graficarlos utilizando el enfoque de cuaterniones.

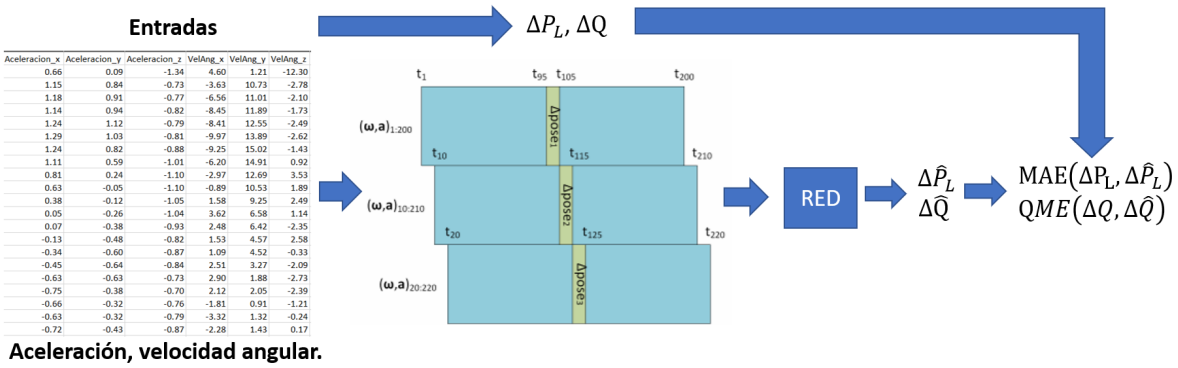


Figura 17. Esquema general de las entradas, salidas, y función de pérdida de la red 6DOF IO.

5.3.2. Red #2: $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_\mathbf{G})$

El error que se genera en la red 6DOF IO se hace muy notorio al poco tiempo de predecir una trayectoria, por lo que se decidió alterar la estructura de la red e investigar que es lo que lo causa. La red original utiliza cuaterniones para predecir la posición del casco, se planteó la posibilidad de que esta representación produce el error que se aprecia en el eje paralelo a la gravedad. En la figura 18 se muestra la primera modificación a la red.

Como se observa en la figura anterior, la red modificada recibe como entrada una secuencia de ventanas compuestas por la velocidad angular y aceleración lineal $(\boldsymbol{\omega}, \mathbf{a})$, predice un vector de posición desde un marco de referencia global $\Delta\hat{\mathbf{P}}_\mathbf{G}$ por ventana, y lo compara con el correspondiente vector real de posición $\Delta\mathbf{P}_\mathbf{G}$. Dado que en este caso no se requieren cuaterniones para predecir la trayectoria, la función de pérdida utilizada considera únicamente el error absoluto medio entre las dos trayectorias. Una vez obtenidos los vectores de posición $\Delta\mathbf{P}_\mathbf{G}$, dado que están desde un marco de referencia global, se grafica la trayectoria haciendo una composición de los desplazamientos, de acuerdo a la siguiente ecuación

$$\mathbf{P_t} = \mathbf{P_{t-1}} + \Delta\mathbf{P}, \tag{8}$$

donde $\mathbf{P_t}$ representa la posición del usuario en el tiempo t.

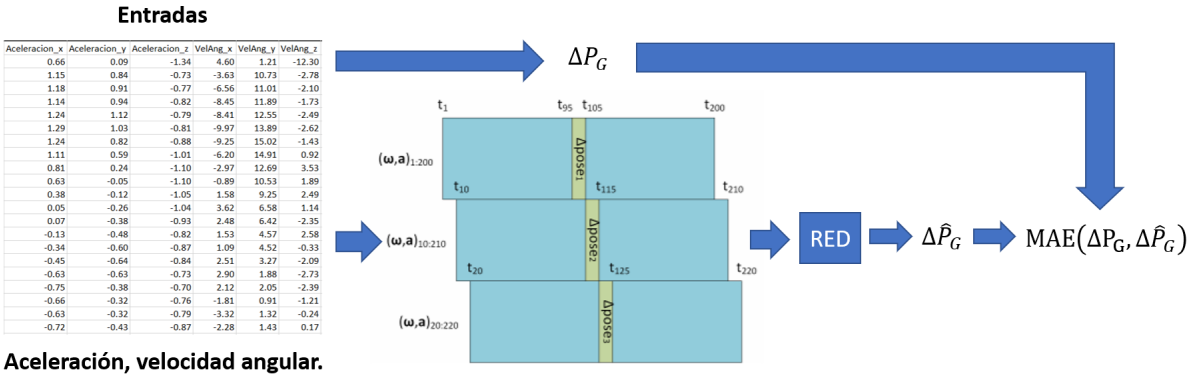


Figura 18. Esquema general de las entradas, salidas, y función de pérdida de la red #2, que no utiliza cuaterniones.

5.3.3. Red #3: $(\mathbf{a}) \rightarrow (\Delta \hat{\mathbf{P}}_\mathbf{G})$

Como segunda modificación a la red 6DOF IO, solo se consideró como única entrada la aceleración, como se aprecia en la figura 19, sin tomar en cuenta la velocidad angular. Esto porque se quería observar el efecto que las lecturas de la velocidad angular causaban a la predicción, la hipótesis fue que si bien podrían ayudar a la red a mejorar la estimación, también era posible que añadieran ruido y dificultaran el proceso. En esta versión tampoco se consideraron los cuaterniones para evaluar la posición.

Como se puede ver en la figura anterior, la entrada de la red es únicamente un vector de aceleración **a**, a diferencia de la red anterior donde también se consideraba la velocidad angular. La salida continúa siendo un vector de posición global $\Delta\mathbf{P_G}$ y se utiliza la función de pérdida MAE. Para graficar los resultados se utiliza la ecuación 8.

5.3.4. Red #4: $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta \hat{\mathbf{P}}_\mathbf{G}, \Delta \hat{\mathbf{Q}})$

Esta versión es muy parecida a la original, únicamente se omitió que los vectores de posición $\Delta\mathbf{P}$ fueran rotados a un nuevo marco de referencia por lo que permanecían globales, a diferencia de la red original, donde los vectores de posición debían ser rotados para poder representarlos correctamente utilizando cuaterniones. En esta versión se conservaron los cuaterniones para observar si ayudaban a la red a predecir de manera más precisa los vectores de posición, pero no se utilizaron para graficar los resultados, lo cual se hizo con la ecuación 8. En la figura 20 se muestra que la entrada

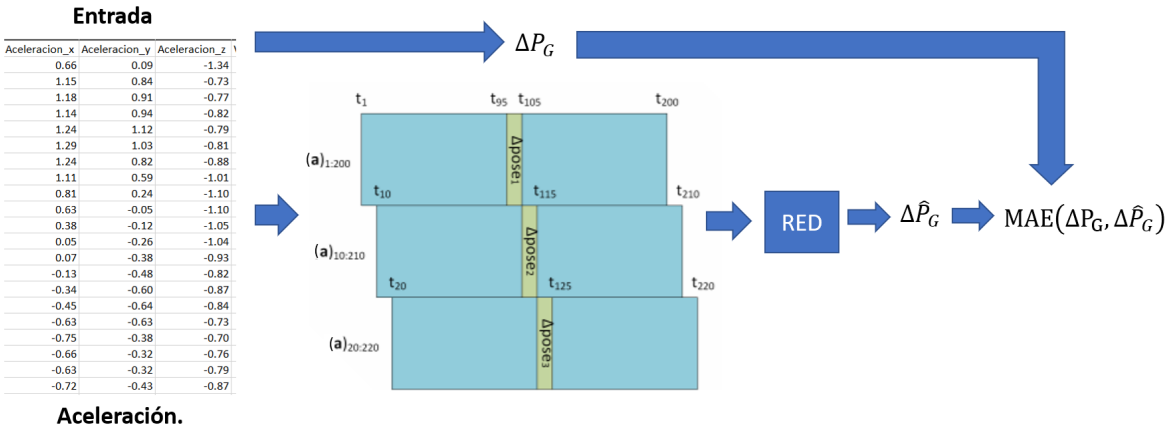


Figura 19. Esquema general de las entradas, salidas, y función de pérdida de la red #3, que utiliza únicamente la aceleración como entrada.

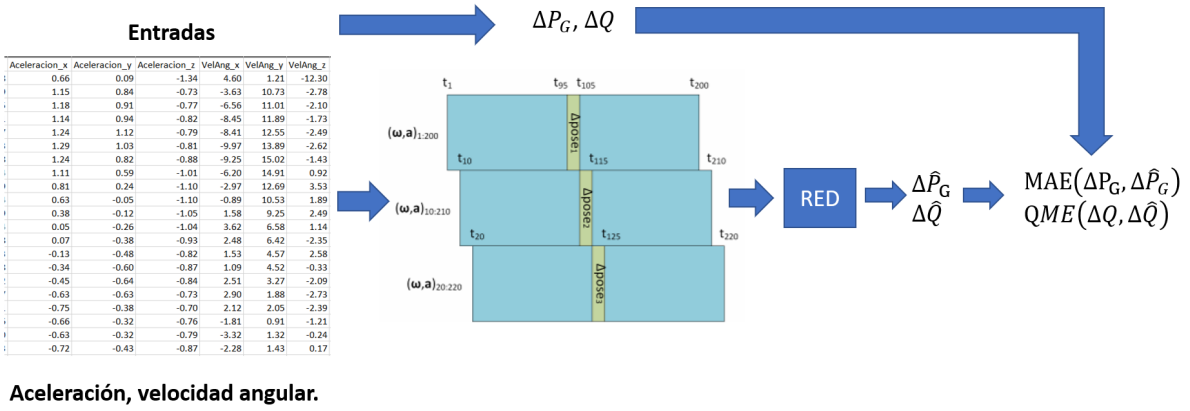


Figura 20. Esquema general de las entradas, salidas, y función de pérdida de la red #4, que estima vectores de posición globales.

de la red es un vector $(\boldsymbol{\omega}, \mathbf{a})$, pero, a diferencia de la figura 17, los vectores $\Delta\hat{\mathbf{P}}$ de salida están en un marco de referencia global.

5.3.5. Añadir capas LSTM bidireccionales

A partir de las modificaciones anteriores, se pudo notar que la red alcanzaba un punto en el cual la capacidad de predicción no mejoraba, incluso añadiendo más datos durante el entrenamiento (ver capítulo 6). Por lo tanto, se decidió construir una nueva red neuronal añadiendo capas adicionales, esto causa que el tiempo de entrenamiento aumente pero también se ve reflejado en que la red tenga una mejor predicción. Se añadieron dos capas LSTM bidireccionales adicionales, por lo que la red ahora es más

profunda y permite crear una representación más compleja de los datos utilizados.

5.3.6. Parámetros probados para cada versión

Originalmente, la red 6DOF IO fue entrenada con un tamaño de ventana de 200 lecturas inerciales considerando un total de 17 archivos de lecturas inerciales tomados de la base de datos OxIOD que corresponden a 55,000 ventanas y cada ventana tenía una separación de 10 lecturas entre ellas. Para nuestro trabajo, se probaron variaciones de estos parámetros para cada versión de la red propuesta en este capítulo. Lo anterior se hizo porque los datos obtenidos con el casco de realidad virtual corresponden a situaciones de movimiento diferentes a los realizados en la base de datos OxIOD. Nuestro objetivo era obtener la combinación de parámetros más adecuada para la red. En el siguiente capítulo se muestran los resultados para cada una de las versiones propuestas y cómo la elección de los parámetros impacta a la predicción de la trayectoria.

Capítulo 6. Resultados

En este capítulo se presentan los resultados obtenidos al predecir trayectorias utilizando las redes mencionadas en el capítulo anterior entrenadas con la base de datos generada en este trabajo. También se incluye una comparación entre las trayectorias obtenidas con las redes y el método de la doble integración. Además, se presentan comparaciones entre redes con diferentes configuraciones para conocer cuál de ellas se adapta mejor a nuestro trabajo.

6.1. Entrenamiento con la base de datos creada

Una vez recolectados los datos inerciales de las seis aplicaciones creadas, procedimos a entrenar las redes mencionadas en el capítulo 5. Ya que no se conocen los parámetros más adecuados para entrenar la red, se hicieron varias pruebas modificando la cantidad de lecturas inerciales por ventana, el número de archivos y la separación entre cada ventana. Se buscó verificar si a mayor cantidad de archivos, mejor era la predicción, por lo que se comenzó entrenando la red con pocos archivos. Posteriormente, se analizó la predicción de la red a medida que se incrementaban el número de archivos de entrenamiento. Además, para cada cantidad de archivos utilizada, se variaba el tamaño de la ventana, probando con los siguientes valores: 50, 100, 150, 200 y 250. Para tener una medida de qué tan buena es la predicción de la red, se calculó el error absoluto medio (Mean Absolute Error o MAE) y la raíz del error cuadrático medio (Root Mean Square Error o RMSE) entre la trayectoria real y la trayectoria predicha.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}, \tag{9}$$

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|. \tag{10}$$

En la tabla 1 se muestra el número de épocas utilizado para entrenar las redes, el cual depende del número de archivos empleados. Adicionalmente se incorporó una

Tabla 1. Número de épocas máximo y de paro utilizadas para las redes entrenadas. Si la red neuronal excedía la condición de paro sin mostrar mejoría, se detenía el entrenamiento.

# de archivos	Épocas	Condición de paro
6	400	50
10	800	100
14	1000	150

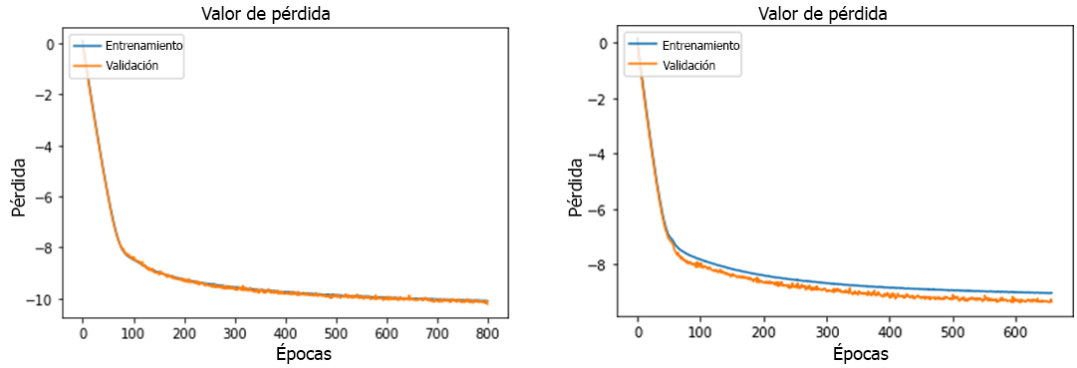
condición de paro que se encargaba de detener el entrenamiento cuando la red pasaba una cierta cantidad de épocas sin mejorar el resultado de la función de pérdida. Ambas formas de finalizar el entrenamiento se muestran en la figura 21.

6.1.1. Red original: $(\boldsymbol{\omega}, \boldsymbol{\alpha}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{L}}, \Delta\hat{\mathbf{Q}})$

La primera red neuronal entrenada utilizando la base de datos creada fue la red original 6DOF IO. Por simplicidad, se comenzó a entrenar la red con los datos inerciales de la aplicación #1, donde los usuarios tenían que caminar sobre un cuadrado. En este juego no había movimientos repentinos o complicados, por lo que las lecturas de los sensores inerciales no contienen demasiado ruido y suponemos puede ser interpretadas más fácilmente por la red. En la figura 22 se muestran los resultados obtenidos después de entrenar la red original 6DOF IO utilizando los primeros seis archivos de la aplicación #1 de nuestra base de datos.

Los resultados se presentan ordenados por el tamaño de la ventana utilizada, y como se puede apreciar, estos parecen variar en gran medida dependiendo la cantidad de lecturas inerciales que contiene cada ventana. Sin embargo, independientemente del tamaño de la ventana utilizada, se observa que hay un error en las coordenadas (X, Y, Z). En particular, el error en la coordenada Y, que corresponde a la altura, muestra un error muy alto en comparación con las otras coordenadas, en algunos casos alcanzando hasta el medio metro de diferencia. Este error también se observa en los resultados del trabajo de Lima *et al.* (2019), donde la coordenada asociada a la altura contiene un error mayor que el correspondiente a las otras coordenadas. Como se comentó en el capítulo anterior, para analizar si era posible reducir el error en los resultados, se entrenó este mismo modelo de la red pero ahora con 10 archivos de lecturas inerciales. Los resultados se muestran en la figura 23.

Como se aprecia en la figura 23, en algunos casos la predicción se asemeja a la



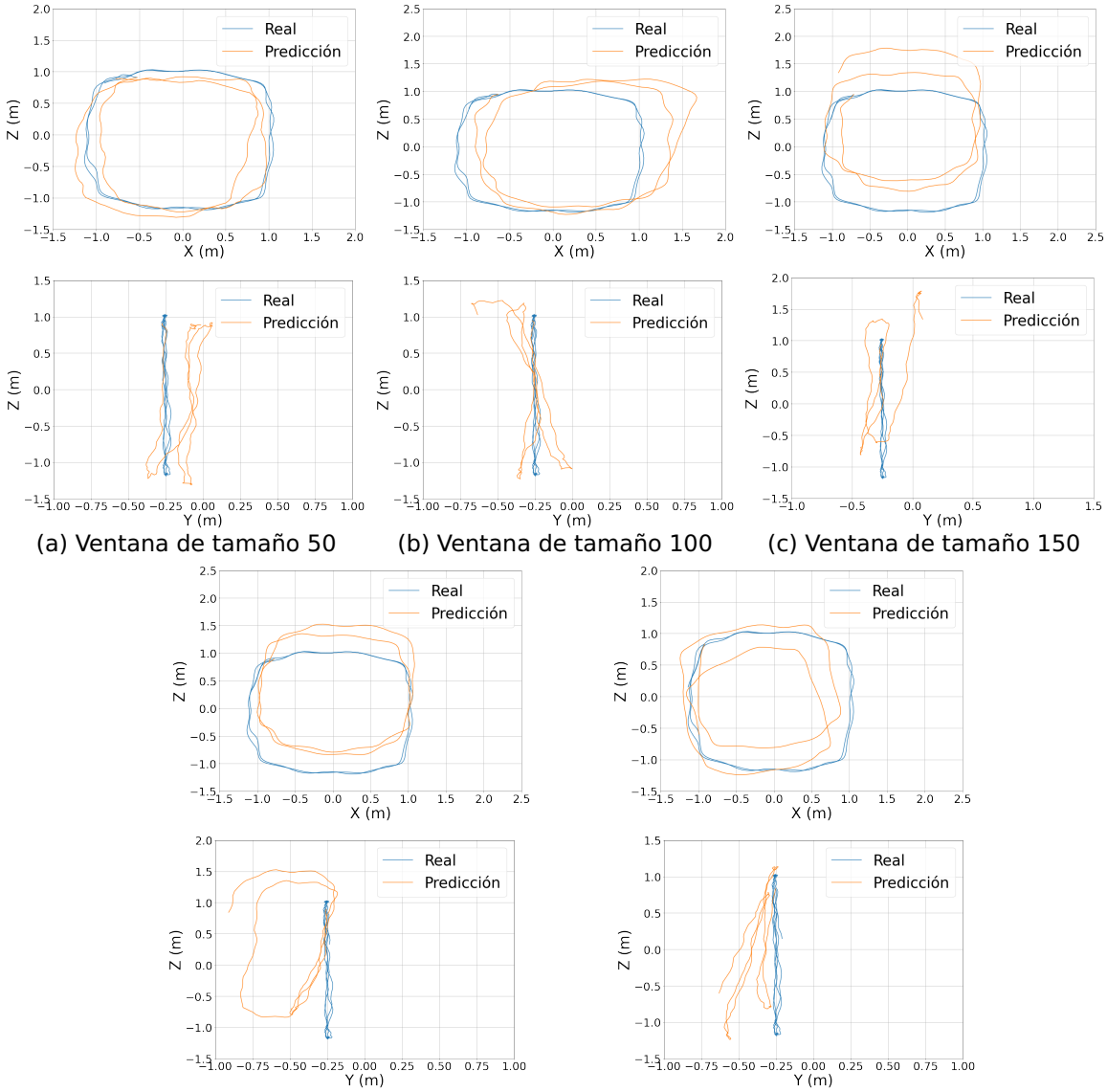
(a) Número máximo de épocas. (b) Condición de paro.

Figura 21. Valor de pérdida para una red entrenada con 10 archivos y 800 épocas. (a) Red cuyo entrenamiento finalizó por alcanzar el máximo de épocas. (b) Red que superó la cantidad de épocas permitidas sin mejorar su predicción.

trayectoria real en el plano (X, Z), por ejemplo, en las ventanas de tamaño 100 y 250, sin embargo, el error en el plano (Y, Z) no parece mostrar mejoría. Posteriormente, se entrenó la red con un total de 14 archivos, el máximo disponible para la aplicación. Los resultados se muestran en la figura 24.

Las predicciones de las ventanas de tamaño 50, 100, 150 y 200 siguen una trayectoria cuadrada, aunque está ligeramente separada de la trayectoria real, mientras que para la ventana de 250 la trayectoria cuadrada se pierde después de unos segundos. El error en el plano (Y, Z) no parece disminuir al considerar la máxima cantidad de datos disponibles.

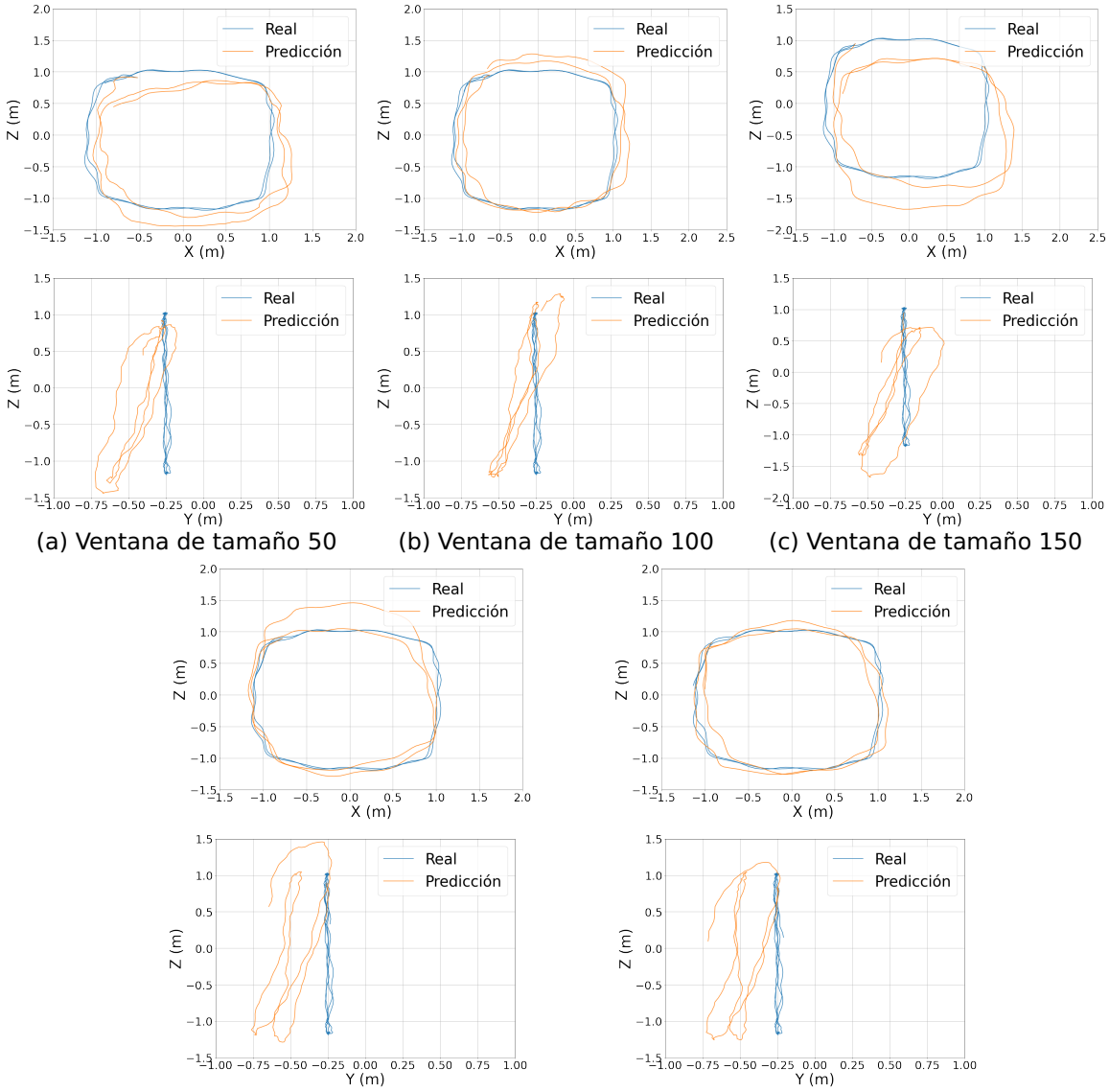
En la figura 25 se muestran los resultados del RMSE y MAE entre la trayectoria real y la predicción para todas las configuraciones presentadas. Como se puede observar, el error varia dependiendo el número de archivos y el tamaño de la ventana. Para las redes entrenadas con seis archivos, el menor error se dio para una ventana de 50 lecturas inerciales, mientras que para las redes entrenadas con 10 y 14 archivos, el menor error corresponde a una ventana de tamaño 100. Por otro lado, independientemente del número de archivos o tamaño de la ventana, se observa en las figuras 22, 23 y 24 que el error en el plano (Y, Z) es considerablemente mayor que el correspondiente al plano (X, Z), por lo que no parece ser un error generado por falta de datos. En su lugar, podría tratarse de algún elemento relacionado a la red neuronal, por lo que se procedió a crear una nueva versión de la red.



(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

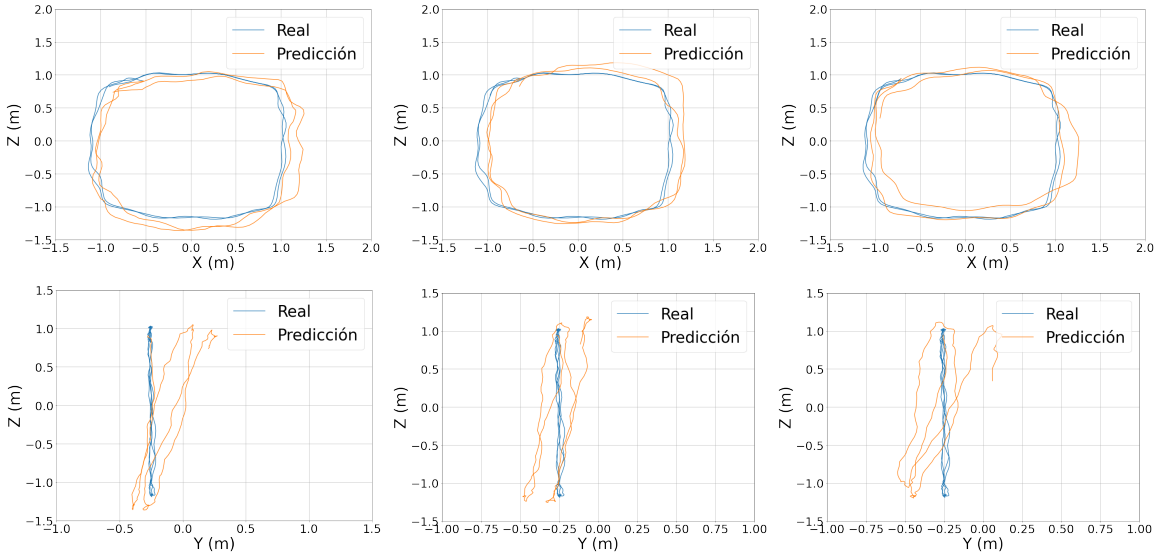
Figura 22. Resultados después de graficar 30 segundos de la trayectoria predicha por la red entrenada con seis archivos de la aplicación #1, ordenados según el tamaño de ventana. Podemos observar que en algunos casos la predicción en el plano (X, Z) trata de ajustarse al valor real de la trayectoria, pero comienza a separarse conforme pasa el tiempo. Además, se puede apreciar un error muy notable en el eje Y, que representa la altura a la que se encuentra el casco.



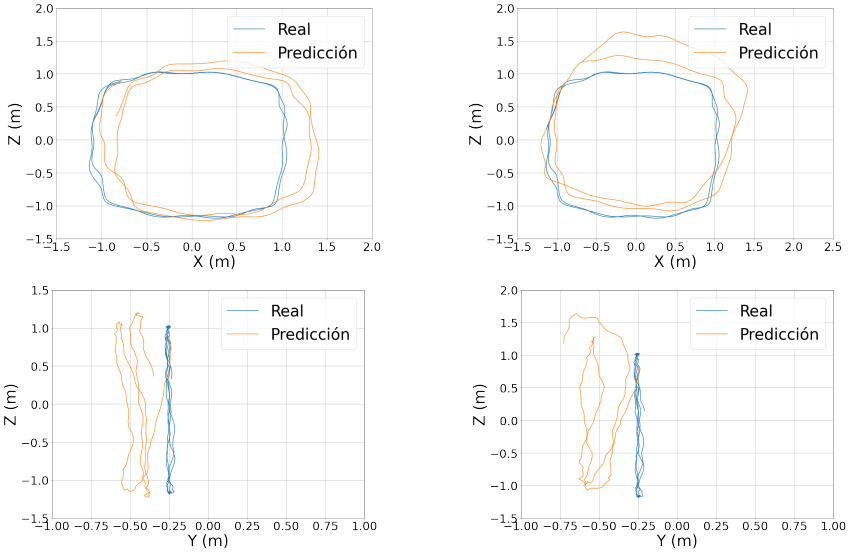
(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 23. Resultados después de graficar 30 segundos de la trayectoria predicha por la red entrenada con 10 archivos de la aplicación #1.



(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150



(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 24. Resultados después de graficar 30 segundos de la trayectoria predicha por la red entrenada con 14 archivos de la aplicación #1.

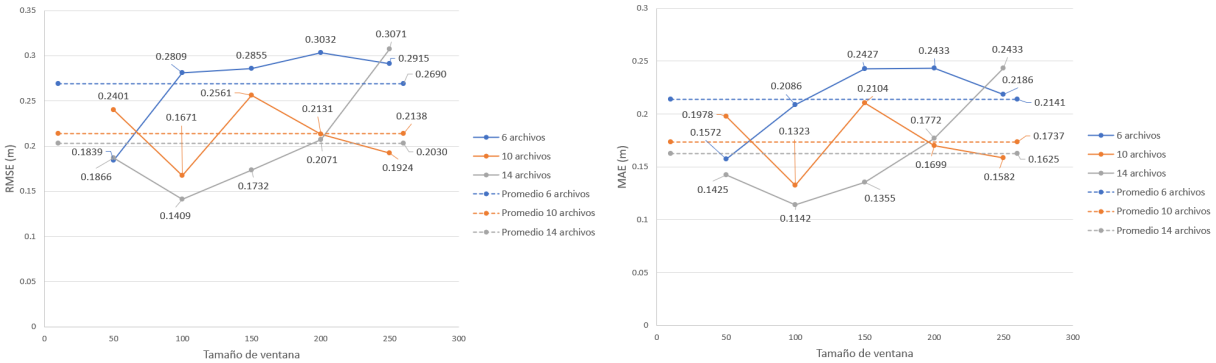
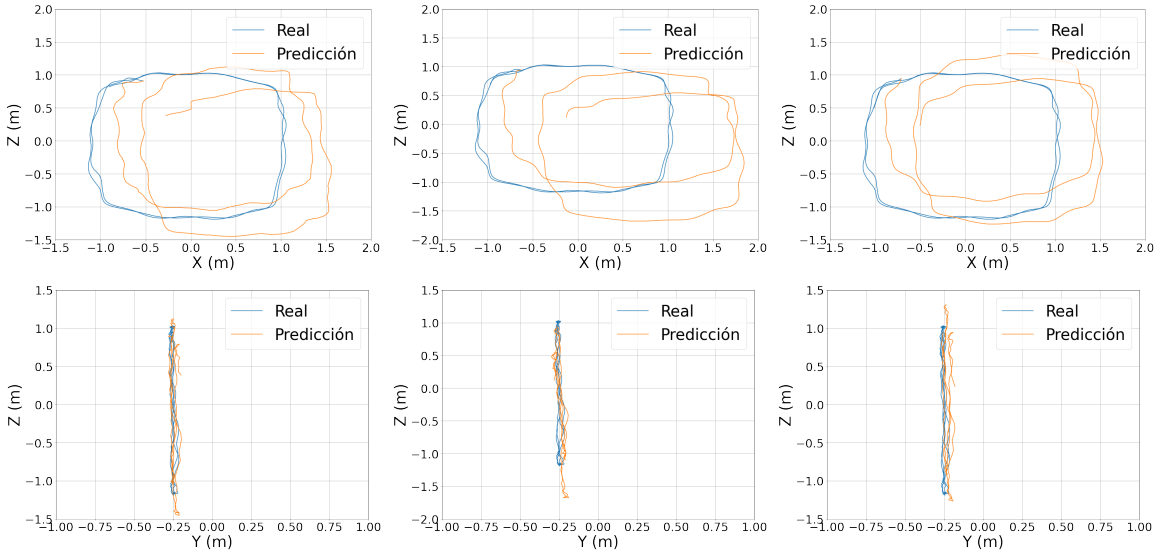


Figura 25. RMSE y MAE de los resultados obtenidos por la red original entrenada con la aplicación #1.

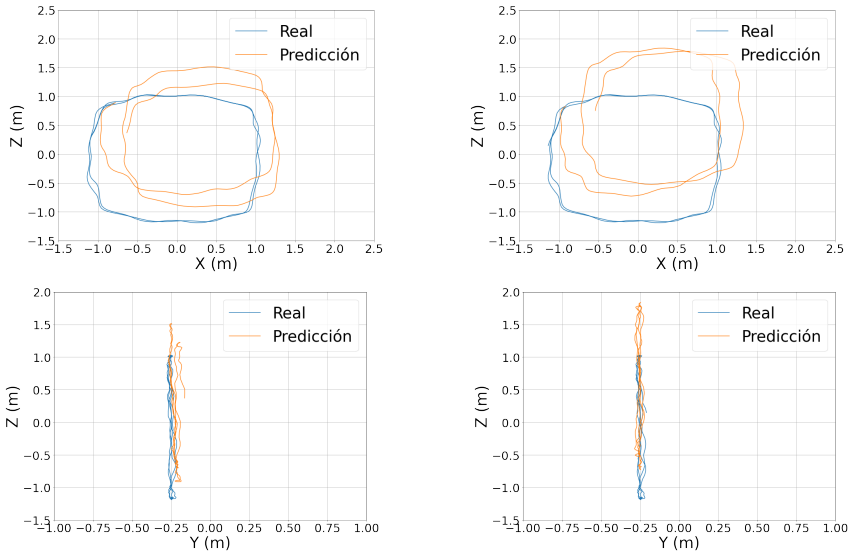
6.1.2. Red #2: $(\boldsymbol{\omega}, \boldsymbol{\alpha}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$

En esta versión se evitó el uso de cuaterniones, por lo que los vectores de posición $\Delta\mathbf{P}$ no eran rotados a un nuevo marco de referencia y quedaban representados desde el marco de referencia global. El vector de entrada es un vector de seis dimensiones compuesto por la aceleración lineal y velocidad angular registrada por el IMU, mientras que la salida es un vector de tres dimensiones compuesto por los desplazamientos $(\Delta x, \Delta y, \Delta z)$ desde un marco de referencia global. Al igual que en la red original, se entrenó primero con seis archivos, después con 10 y finalmente con 14, esto para ver el comportamiento que genera el incremento de datos. Lo mismo se hizo para el tamaño de ventana, para cada cantidad de archivos se entrenó un modelo con una ventana de 50, 100, 150, 200 y 250 lecturas inerciales. En la figura 26 se muestran los resultados de la red #2 entrenada con seis archivos de la aplicación #1.

Como se observa, podemos destacar que el error en el plano (Y, Z) se redujo considerablemente en comparación con los resultados obtenidos en la red original, lo que sugiere que los cuaterniones tienen un papel importante en el incremento de este error. No obstante, el error en el plano (X, Z) aumentó en comparación con la red original, pero no es claro si este aumento en el error es generado por la poca cantidad de archivos o por la falta de los cuaterniones. Por lo anterior, se entrenó la misma arquitectura de la red aumentando el número de archivos a 10. Los resultados se presentan en la figura 27.

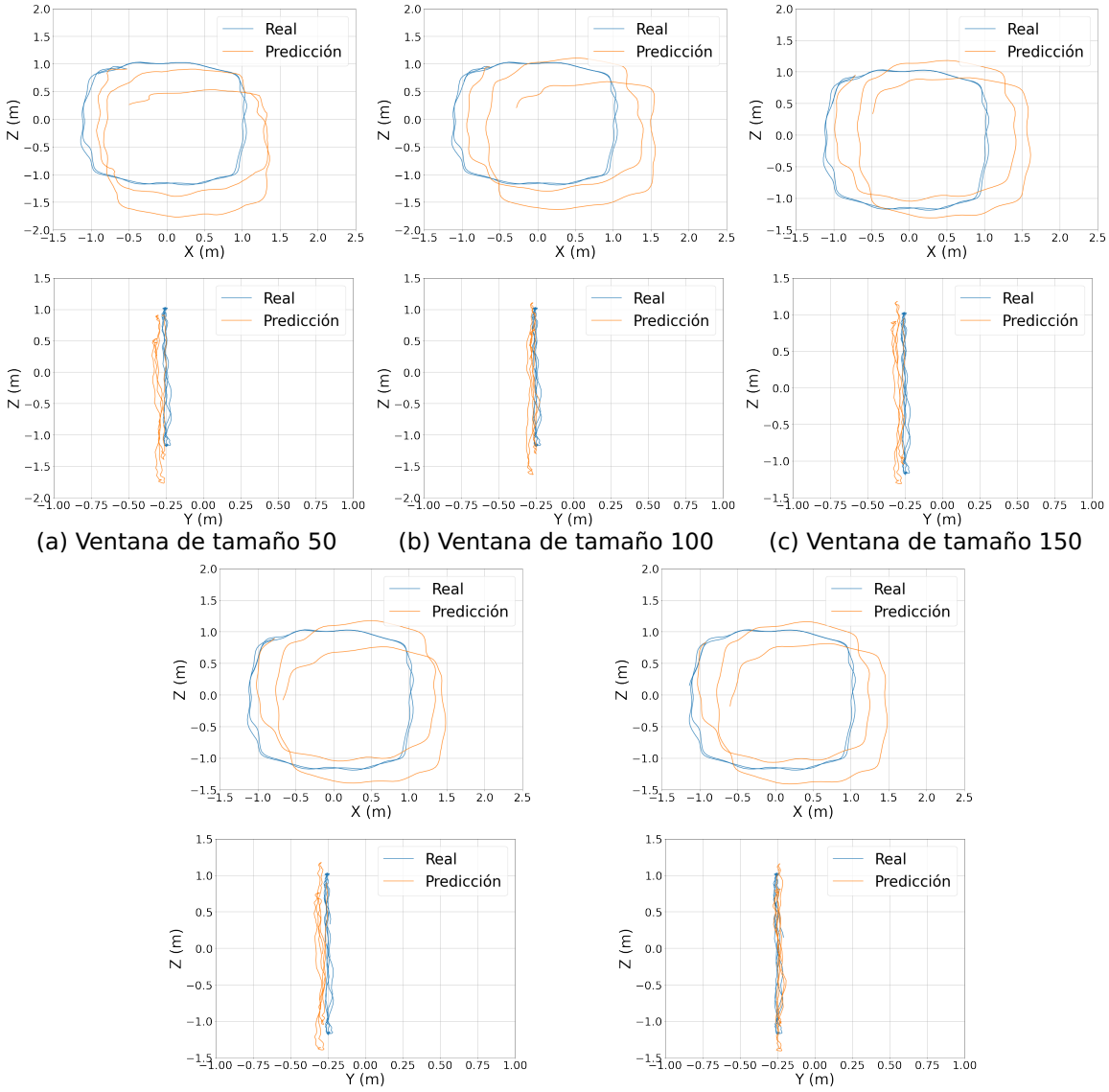


(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150



(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 26. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #2 entrenada con seis archivos de la aplicación #1.

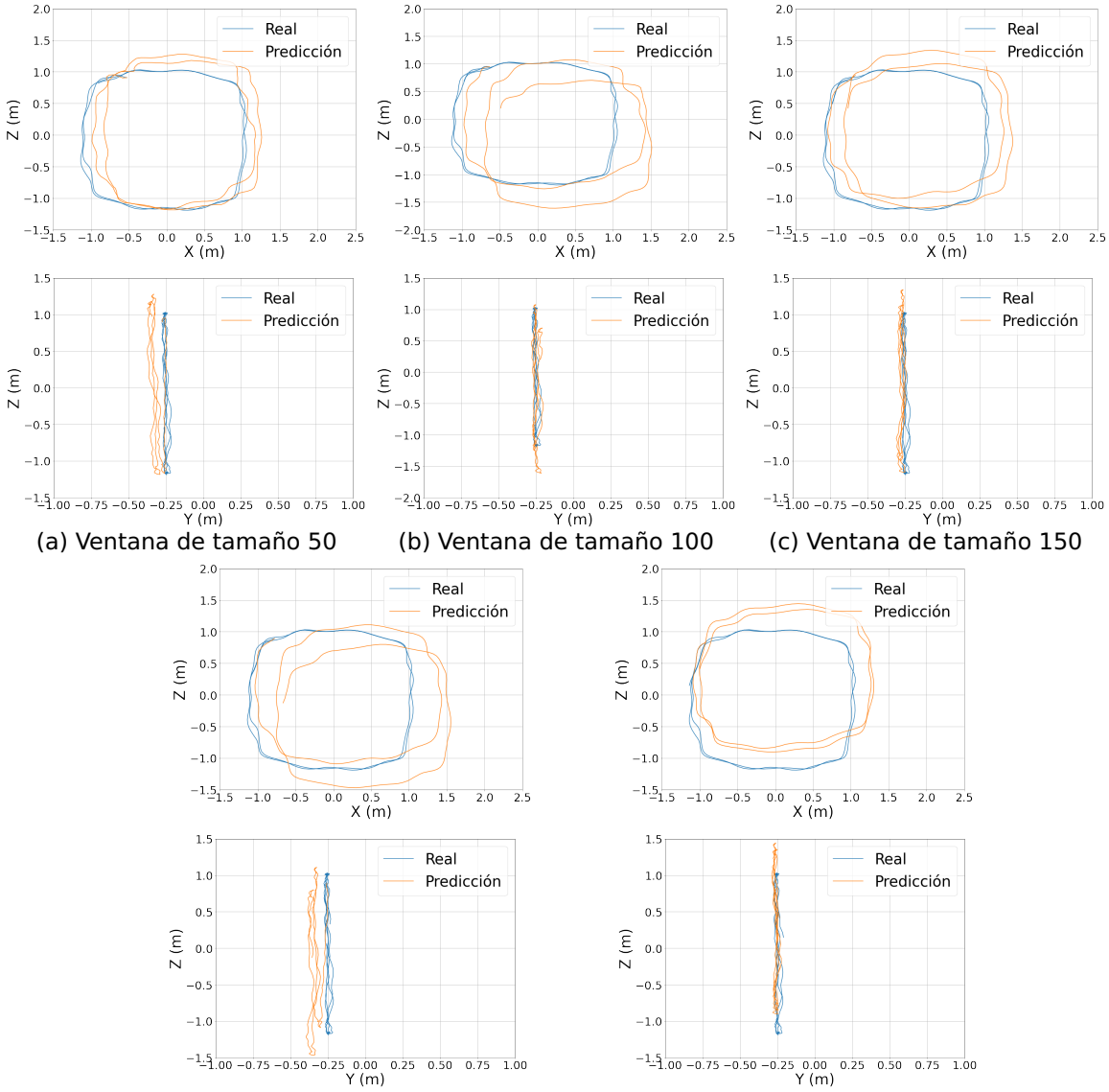


(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 27. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #2 entrenada con 10 archivos de la aplicación #1.

Al utilizar 10 archivos para el entrenamiento, los resultados presentaron una pequeña mejoría en el plano (X, Z), en comparación con los de la figura 26. Al igual que la red entrenada con seis archivos, se observó que el error en el plano (Y, Z) disminuyó. Finalmente, se entrenó la red #2 con 14 archivos, los resultados se muestran en la figura 28.



(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 28. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #2 entrenada con 14 archivos de la aplicación #1.

Al comparar los resultados de la red entrenada con 6 y 14 archivos, figuras 26 y 28, observamos que la predicción parece mejorar al aumentar el número de datos. En algunos casos, la forma de la trayectoria predicha parece conservarse salvo un desplazamiento en comparación con la trayectoria real. Por ejemplo, para una ventana de 250 con 14 archivos podemos ver que la trayectoria predicha por la red se encuentra desplazada con respecto a la real pero presenta la forma de un cuadrado. En la figura 29 se muestra un resumen de los resultados.

Comparando el error promedio como función de la cantidad de archivos, se puede

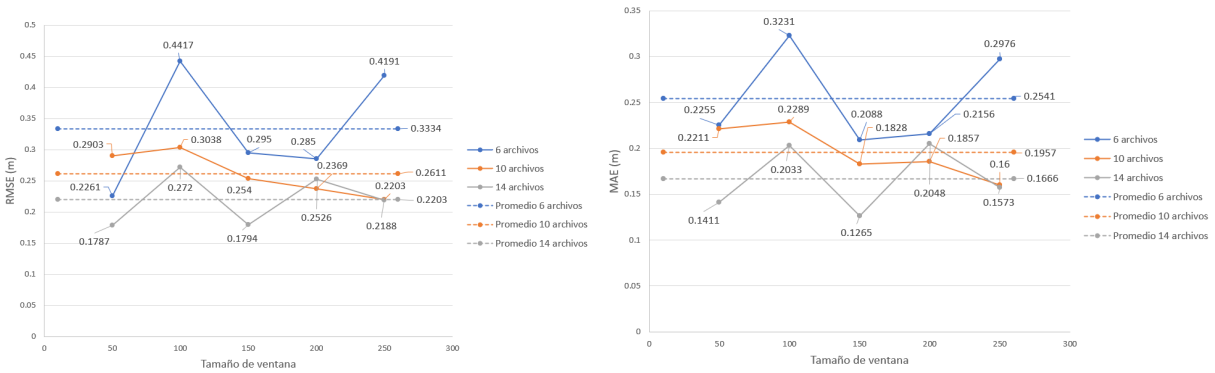
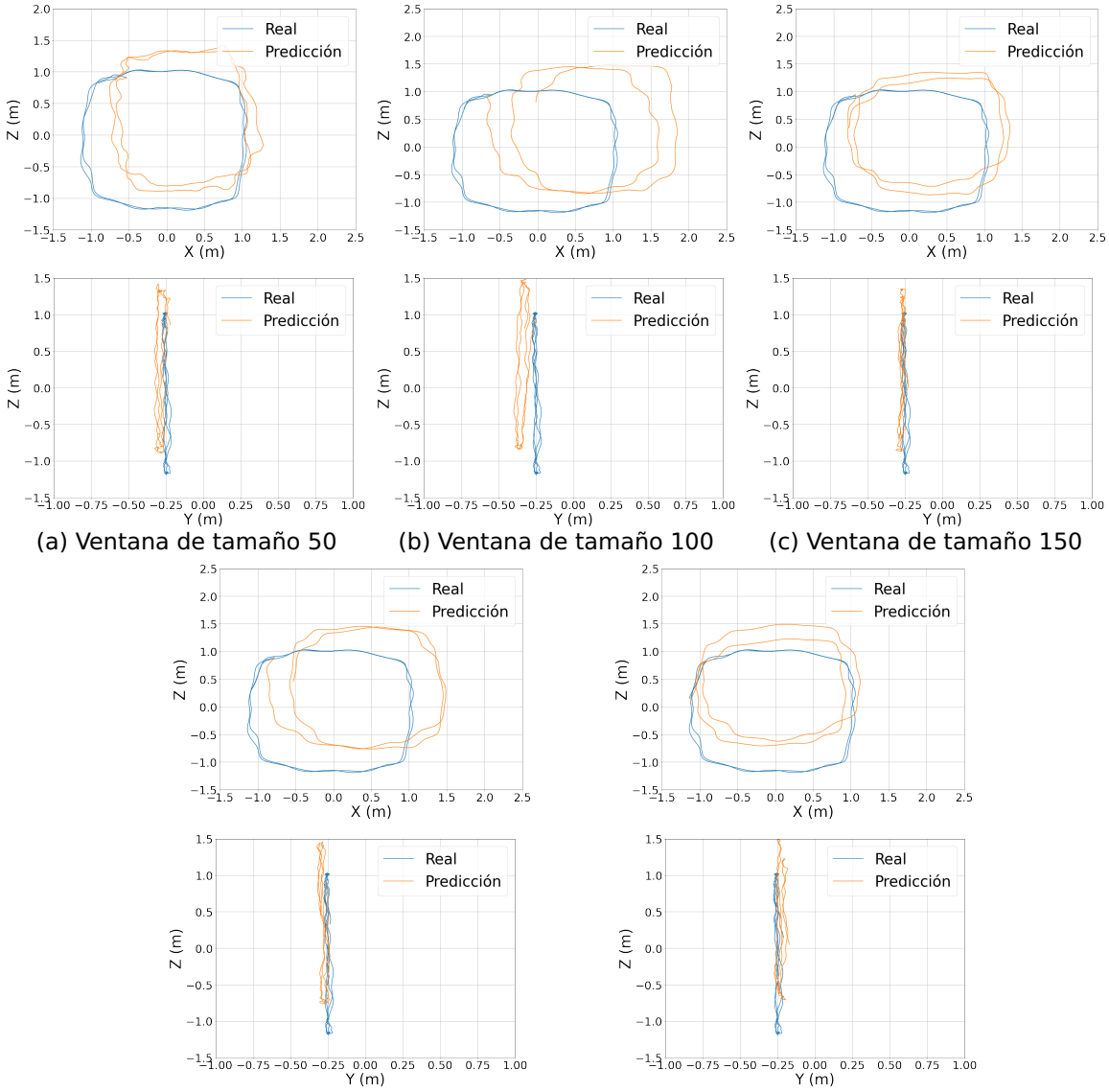


Figura 29. RMSE y MAE de los resultados de la red #2 entrenada con la aplicación #1.

apreciar que conforme aumenta la cantidad de datos, el error promedio disminuye. En cuanto al tamaño de la ventana, para 6 y 14 archivos, una ventana de tamaño 150 presenta un menor error. Para 10 archivos la ventana con menor error fue de tamaño 250. A pesar de que la red #2 reduce el error en el plano (Y, Z) con respecto a la red original, la predicción en el plano (X, Z) es más precisa con la red original que con la red #2. Después de analizar estos resultados, surgió la duda sobre si las lecturas relacionadas con la orientación (giroscopio) eran necesarias, ya que como no utilizamos cuaterniones, probablemente incluir datos de la orientación podría añadir ruido a la predicción de la red. Por lo tanto, se diseñó una nueva red que considera únicamente los datos del acelerómetro como entrada.

6.1.3. Red #3: (a) → ($\Delta\hat{\mathbf{P}}_{\mathbf{G}}$)

En esta versión, el vector de entrada a la red consiste únicamente en un vector en tres dimensiones compuesto por la aceleración lineal registrada por el IMU, por lo que no se utilizan cuaterniones y los vectores de posición $\Delta\mathbf{P}$ quedan representados desde un marco de referencia global. Como en este caso no se utiliza ningún valor relacionado con la orientación del IMU, se empleó solamente la función de pérdida que corresponde al MAE entre la posición predicha y la real. Al igual que las versiones anteriores, la red se entrenó con 6, 10 y 14 archivos, variando el tamaño de las ventanas. La primera versión fue entrenada con seis archivos y las ventanas de tamaño 50, 100, 150, 200 y 250. Estos resultados se muestran en la figura 30.

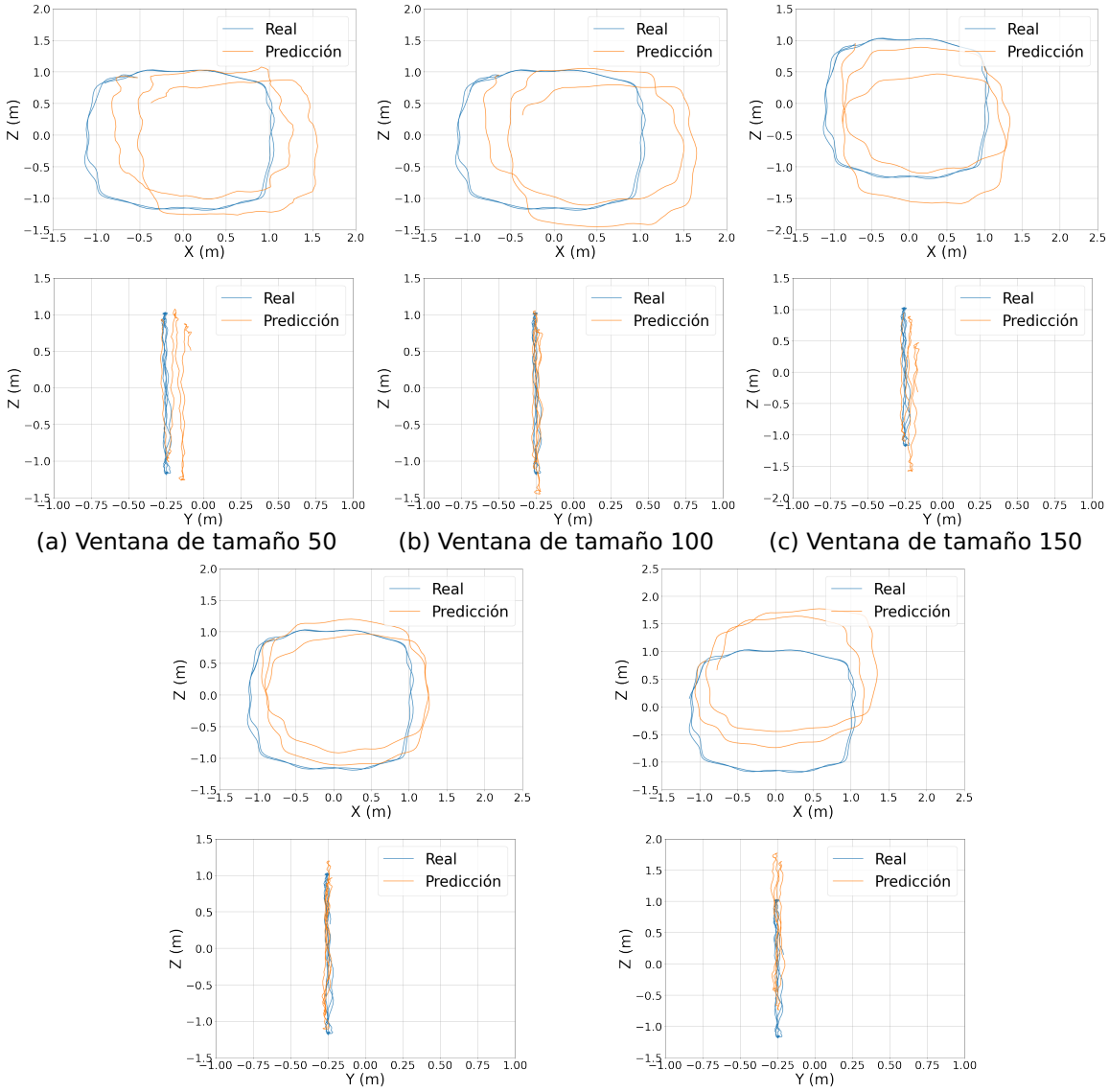


(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 30. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #3 entrenada con seis archivos de la aplicación #1.

Podemos notar que aún sin utilizar lecturas del giroscopio el error en el plano (Y, Z) disminuye con respecto a la red original. Sin embargo, el error en el plano (X, Z) aumenta en comparación con la red original y la red #2. Posteriormente, entrenamos la red con 10 archivos, como se observa en la figura 31, para comprobar si es posible disminuir el error añadiendo más datos.



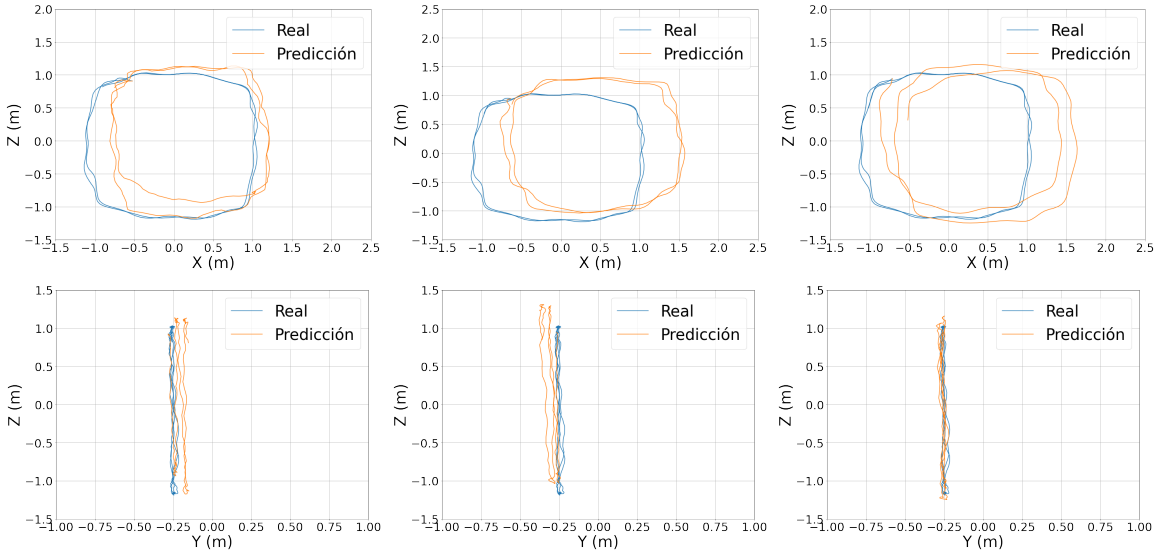
(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

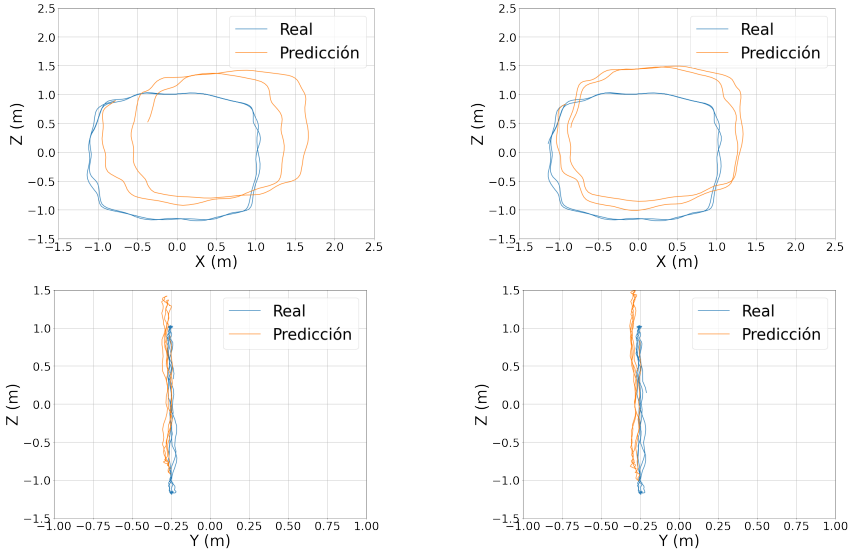
Figura 31. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #3 entrenada con 10 archivos de la aplicación #1.

Para 10 archivos los resultados no parecen mejorar, aunque el error en el plano (Y, Z) es similar. El error en el plano (X, Z) sigue siendo notorio. Finalmente se entrenó una versión con 14 archivos, como se observa en la figura 32.

La red entrenada con 14 archivos parece mostrar una mejoría en la predicción en comparación con las anteriores que consideran menos archivos. Sin embargo, los resultados no son tan buenos como los de la red original y la #2. En la figura 33 se muestra un resumen de los resultados.



(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150



(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 32. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #3 entrenada con 14 archivos de la aplicación #1.

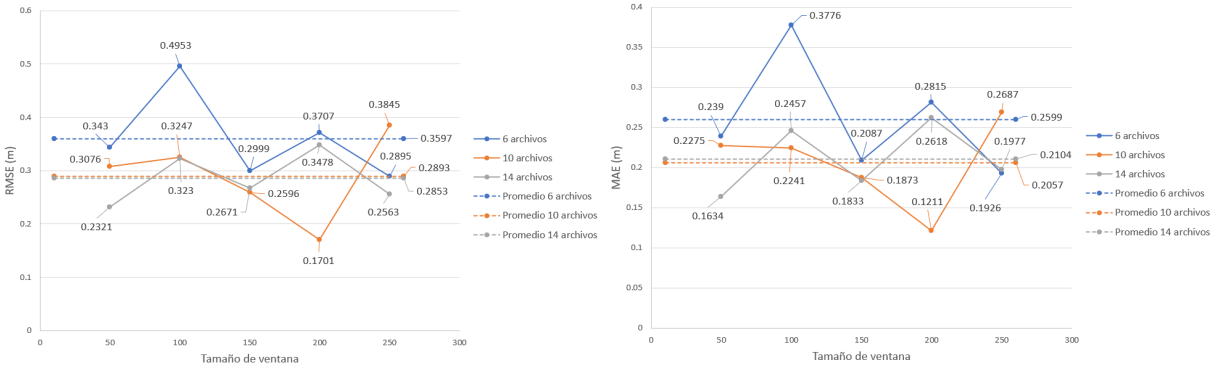


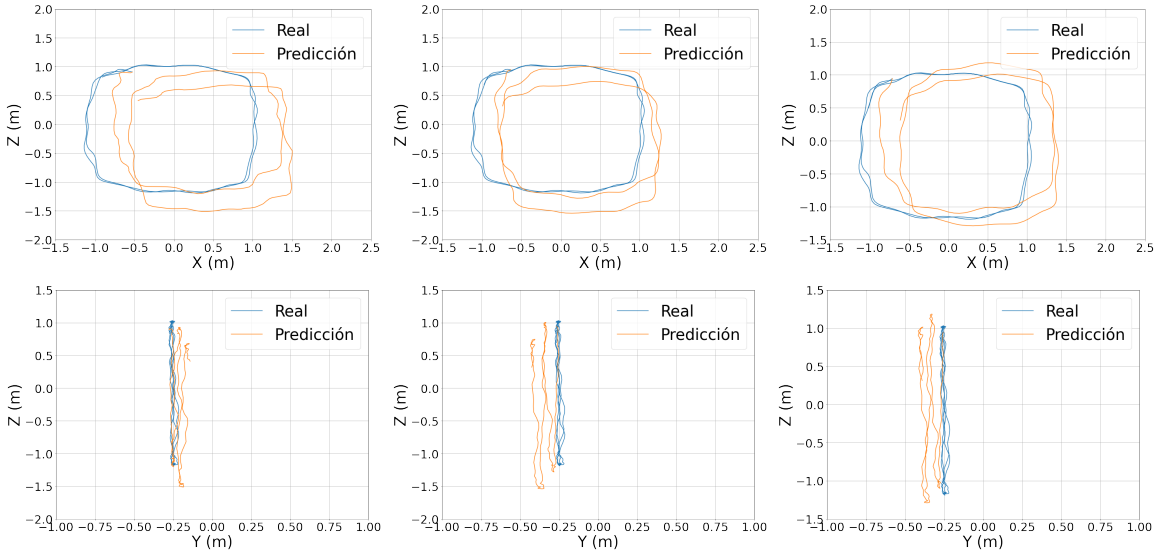
Figura 33. RMSE y MAE de los resultados de la red #3 entrenada con la aplicación #1.

Se puede notar en la figura 33 que el promedio para 10 archivos es menor que el de seis archivos. Sin embargo, el promedio para 14 archivos no es menor que el de 10, lo que sugiere que la red ha llegado a un límite y a pesar de incrementar el número de datos, no mejorará la predicción. Comparando la figura 29 con la figura 33, parece ser que utilizar las lecturas del giroscopio como parte de la entrada de la red beneficia a la predicción de la trayectoria.

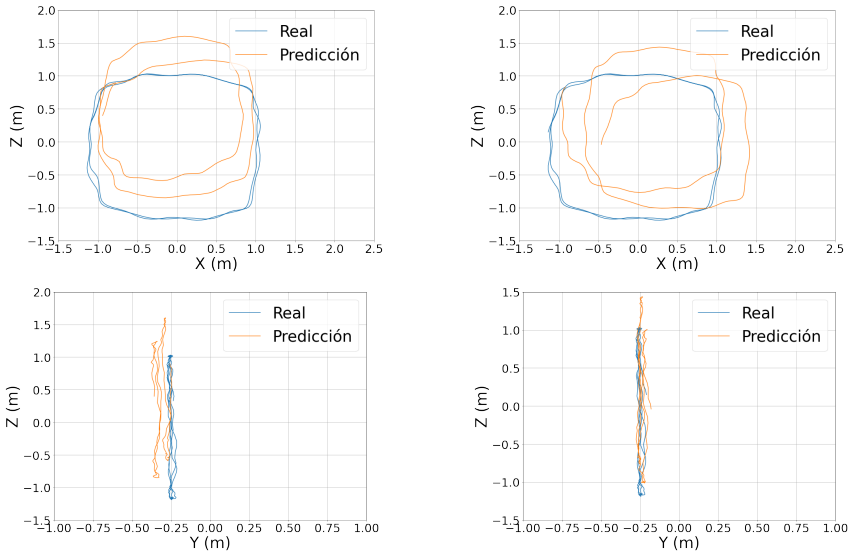
6.1.4. Red #4: $\Delta\mathbf{P}$ globales: $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}}, \Delta\hat{\mathbf{Q}})$

De los resultados obtenidos por la red #2 se pudo observar que el uso de cuaterniones contribuye al incremento del error de deriva en el plano (Y, Z). Sin embargo, falta aclarar si los cuaterniones incrementan error al momento de rotar los vectores $\Delta\mathbf{P}$ o cuando se utilizan para graficar los resultados. Para solucionar esta duda se entrenó una nueva versión donde, a diferencia de la red #2 y #3, sí se realiza la predicción de los cuaterniones, pero en este caso los vectores $\Delta\mathbf{P}$ no son rotados a un nuevo marco de referencia. Los cuaterniones solo son utilizados para representar la orientación durante el entrenamiento; en otras palabras, la trayectoria se puede reconstruir empleando únicamente los vectores $\Delta\mathbf{P}_{\mathbf{G}}$ predichos. El vector de entrada para la red es de seis dimensiones, tres correspondientes a la aceleración y tres a la velocidad angular, mientras que el vector de salida es de dimensión siete, tres componentes asociadas al desplazamiento $(\Delta x, \Delta y, \Delta z)$ y cuatro al cuaternión predicho $(\Delta q_w, \Delta q_x, \Delta q_y, \Delta q_z)$. Como en las modificaciones pasadas, la red se entrenó con un número diferente de archivos y tamaño de ventanas. En la figura 34 se muestran los resultados de la red #4 entrenada con los primeros seis archivos de la aplicación #1.

Como se puede apreciar en la figura 34, el error en el plano (Y, Z) disminuye notablemente en comparación a la red original, lo que sugiere que los cuaterniones incrementan el error cuando se realiza la rotación de los vectores de posición $\Delta\mathbf{P}$. Para nuestra base de datos, esta rotación no es necesaria, ya que los datos capturados provienen de un marco de referencia global. Para tratar de obtener una mejor precisión se procedió a entrenar la red #4 con 10 archivos. Los resultados se presentan en la figura 35.

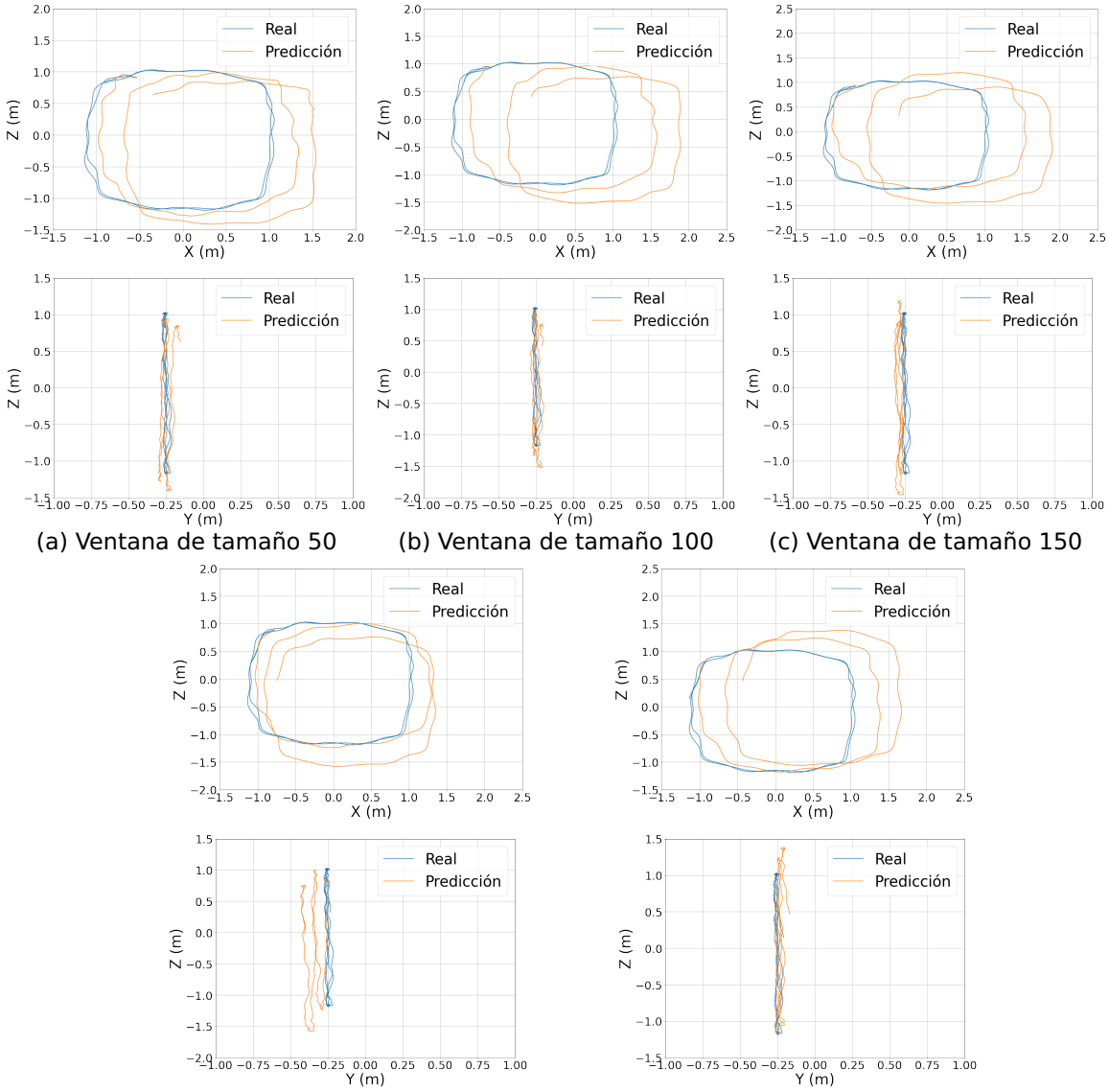


(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150



(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 34. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #4 entrenada con seis archivos de la aplicación #1.

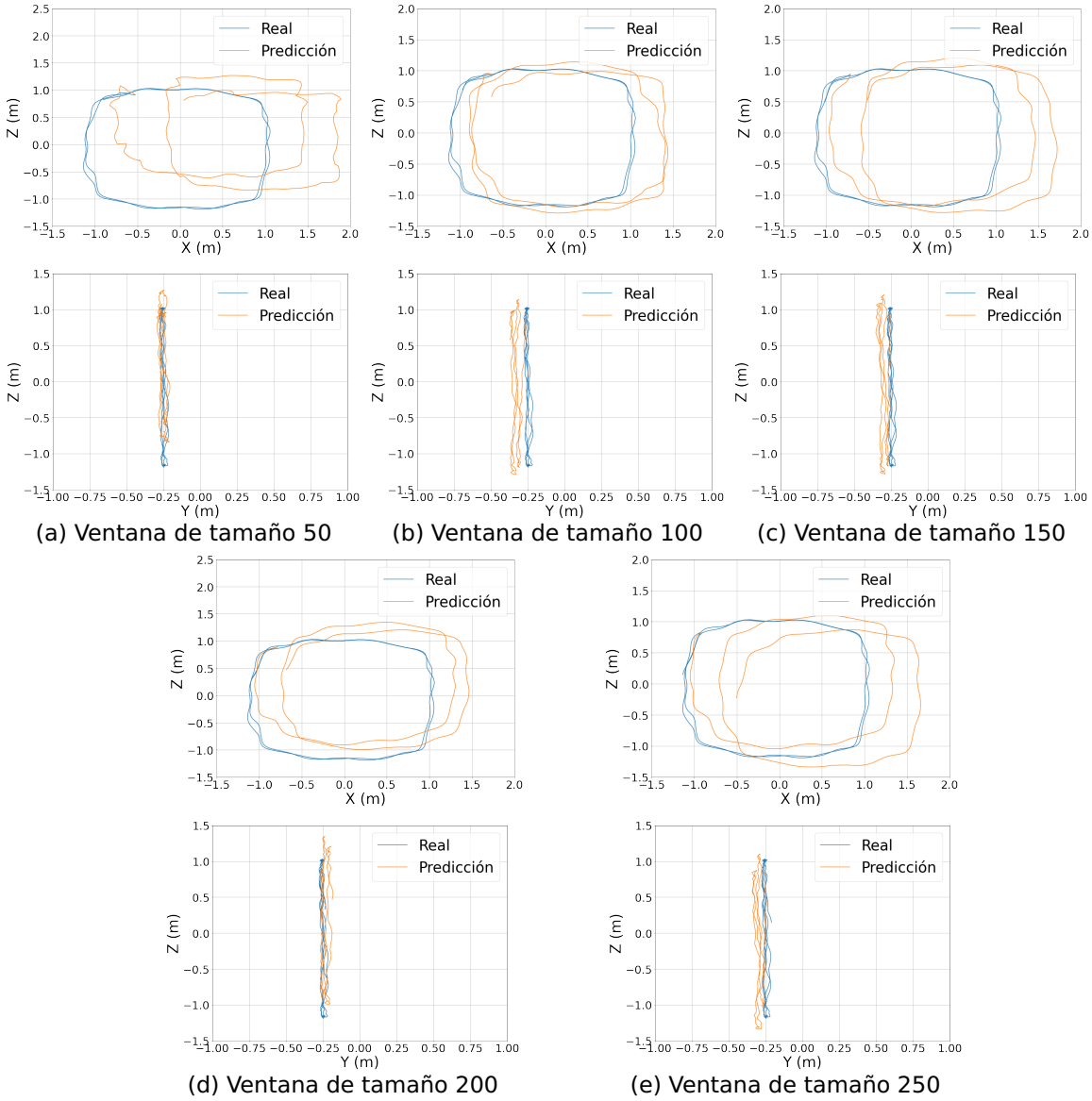


(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 35. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #4 entrenada con 10 archivos de la aplicación #1.

Como se puede observar, el error en el plano (X, Z) no muestra mejoría a pensar de tener una mayor cantidad de datos, mientras que el error en el plano (Y, Z) disminuye apreciablemente en la mayoría de ventanas. Finalmente, se entrenó la red con 14 archivos, como se muestra en la figura 36.



(a) Ventana de tamaño 50 (b) Ventana de tamaño 100 (c) Ventana de tamaño 150

(d) Ventana de tamaño 200 (e) Ventana de tamaño 250

Figura 36. Resultados después de graficar 30 segundos de la trayectoria predicha por la red #4 entrenada con 14 archivos de la aplicación #1.

En la mayoría de los casos, la predicción con 14 archivos muestra mejoría en comparación con los resultados para 6 y 10 archivos, excepto en la ventana de tamaño de 50, donde el error aumenta en el plano (X, Z). En la figura 37 se presentan los errores para los diferentes parámetros que fueron usados en esta versión. Como se puede apreciar en la figura 37, el error promedio no parece disminuir al incrementar la cantidad de datos.

En las tablas 2 y 3 se muestran el error RMSE y MAE obtenido en todas las versiones entrenadas.

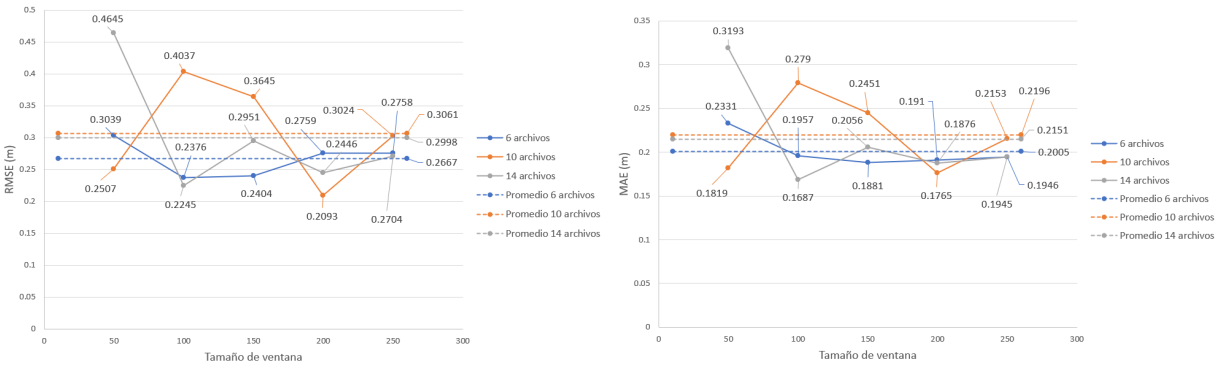


Figura 37. RMSE y MAE de los resultados de la red #4 entrenada con la aplicación #1.

Tabla 2. RMSE (m) para las diferentes redes utilizadas.

Archivos	**Ventana**	**Red original**	**Red #2**	**Red #3**	**Red #4**
		$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{L}}, \Delta\hat{\mathbf{Q}})$	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$	$(\mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}}, \Delta\hat{\mathbf{Q}})$
6	50	0.1839	0.2261	0.3430	0.3039
6	100	0.2809	0.4417	0.4953	0.2376
6	150	0.2855	0.2950	0.2999	0.2404
6	200	0.3032	0.2850	0.3707	0.2759
6	250	0.2915	0.4191	0.2895	0.2758
Promedio (sd)		**0.2690 (0.048)**	**0.3334 (0.093)**	**0.3597 (0.083)**	**0.2667 (0.028)**
10	50	0.2401	0.2903	0.3076	0.2507
10	100	0.1671	0.3038	0.3247	0.4037
10	150	0.2561	0.2540	0.2596	0.3645
10	200	0.2131	0.2369	0.1701	0.2093
10	250	0.1924	0.2203	0.3845	0.3024
Promedio (sd)		**0.2224 (0.036)**	**0.2611 (0.035)**	**0.2893 (0.080)**	**0.3061 (0.080)**
14	50	0.1866	0.1787	0.2321	0.4645
14	100	0.1409	0.2720	0.3230	0.2245
14	150	0.1732	0.1794	0.2671	0.2951
14	200	0.2071	0.2526	0.3478	0.2446
14	250	0.3071	0.2188	0.2563	0.2704
Promedio (sd)		**0.2030 (0.063)**	**0.2203 (0.042)**	**0.2853 (0.048)**	**0.2998 (0.096)**

Tabla 3. MAE (m) para las diferentes redes utilizadas.

Archivos	**Ventana**	**Red original**	**Red #2**	**Red #3**	**Red #4**
		$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{L}}, \Delta\hat{\mathbf{Q}})$	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$	$(\mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}}, \Delta\hat{\mathbf{Q}})$
6	50	0.1572	0.2255	0.239	0.2331
6	100	0.2086	0.3231	0.3776	0.1957
6	150	0.2427	0.2088	0.2087	0.1881
6	200	0.2433	0.2156	0.2815	0.1910
6	250	0.2186	0.2976	0.1926	0.1946
Promedio (sd)		**0.2142 (0.035)**	**0.2541 (0.052)**	**0.2600 (0.074)**	**0.2005 (0.018)**
10	50	0.1978	0.2211	0.2275	0.1819
10	100	0.1323	0.2289	0.2241	0.2790
10	150	0.2104	0.1828	0.1873	0.2451
10	200	0.1699	0.1857	0.1211	0.1765
10	250	0.1582	0.1600	0.2687	0.2153
Promedio (sd)		**0.1737 (0.031)**	**0.1957 (0.029)**	**0.2057 (0.055)**	**0.2196 (0.043)**
14	50	0.1425	0.1411	0.1634	0.3193
14	100	0.1142	0.2033	0.2457	0.1687
14	150	0.1355	0.1265	0.1833	0.2056
14	200	0.1772	0.2048	0.2618	0.1876
14	250	0.2433	0.1573	0.1977	0.1945
Promedio (sd)		**0.1625 (0.051)**	**0.1666 (0.036)**	**0.2103 (0.042)**	**0.2151 (0.060)**

En las tablas 4 y 5 se presentan promedios generales del RMSE y MAE, respectivamente, dependiendo la cantidad de archivos utilizados, es decir, se promediaron los resultados de todas las redes entrenadas utilizando únicamente 6 archivos, después las de 10 archivos y finalmente las de 14 archivos, sin considerar el tamaño de ventana ni la arquitectura de la red.

De las tablas 4 y 5 podemos observar que el promedio por número de archivos es similar. En las figuras 38 y 39 se muestran diagramas de caja donde se puede apreciar con más detalle los resultados obtenidos.

Con base a la figuras 38 y 39, se observa una tendencia de disminución del error cuando se utilizan 14 archivos. Esto se nota en el respectivo diagrama de caja en el cual se presentan la mediana, media aritmética, el mínimo y máximo no atípicos más pequeños con respecto a los elementos equivalentes para los diagramas de 6 y 10 archivos.

Tabla 4. Promedio general del RMSE (m) por cantidad de archivos.

Archivos	6	10	14
Promedio (sd)	0.3096 (0.075)	0.2697 (0.067)	0.2521 (0.074)

Tabla 5. Promedio general del MAE (m) por cantidad de archivos.

Archivos	6	10	14
Promedio (sd)	0.2322 (0.052)	0.1987 (0.041)	0.1886 (0.050)

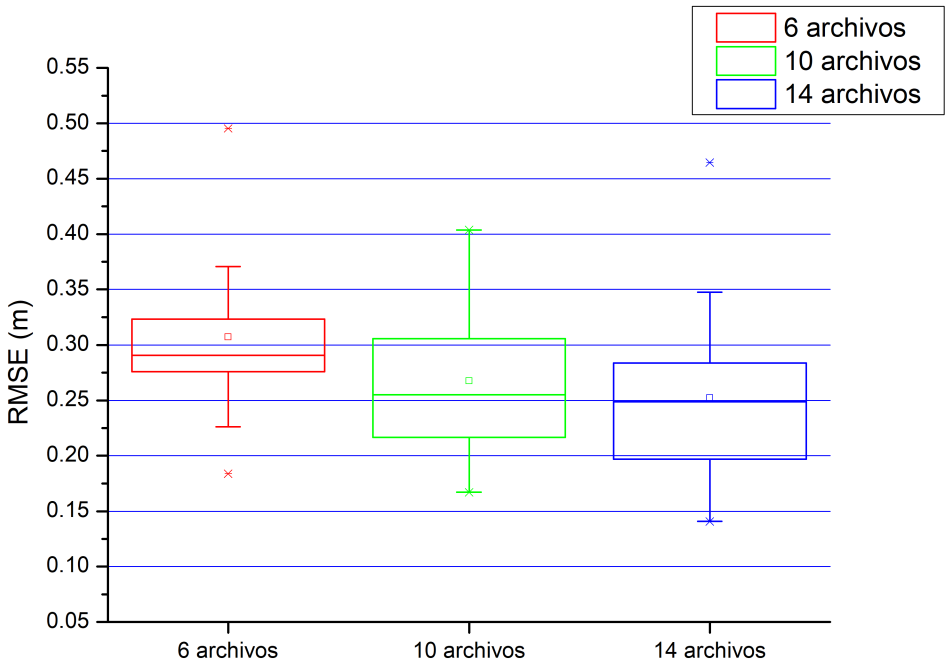


Figura 38. Diagrama de caja para el RMSE por archivos.

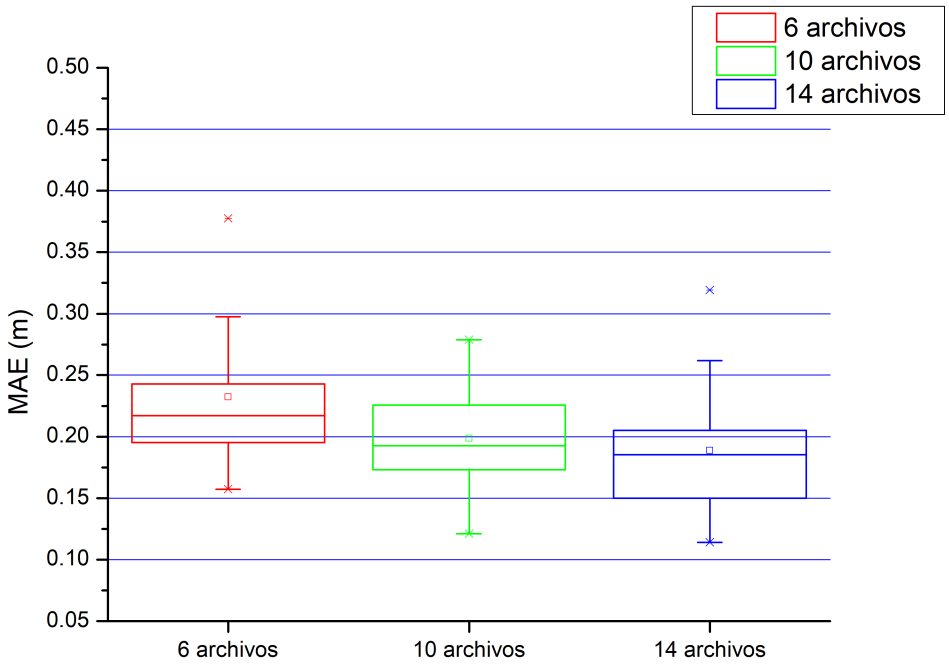


Figura 39. Diagrama de caja para el MAE por archivos.

Se realizo una análisis similar pero esta vez considerando el tipo de arquitectura. En las tablas 6 y 7 se puede observar que la red con un menor promedio de error, independientemente del número de archivos, es la versión original, seguida por la red #2. La red #3 presenta el error promedio más grande, sin embargo, no es muy diferente al de la red #4, por lo que para verificar de manera más precisa estos resultados, se procedió a realizar su correspondiente diagrama de cajas, como se aprecia en las figuras 40 y 41.

Podemos ver de las figuras 40 y 41 que la red original cuenta con la mediana y media aritmética más pequeñas. Además, los valores máximo y mínimo no atípicos tienden a estar por debajo de los asociados a las otras redes. Lo anterior sugiere que la red original tiende a presentar el mejor desempeño. Sin embargo, es importante notar que en un sistema de rastreo de posición para un visor de VR el error en la coordenada de la altura sería altamente perceptible por un usuario, por lo que los resultados obtenidos por la red #2 (figura 28) pudieran ser preferibles sobre los obtenidos por la red original (figura 24).

Tabla 6. Promedio general del RMSE (m) por arquitectura.

Versión	Red original	Red #2	Red #3	Red #4
Promedio (sd)	0.2346 (0.055)	0.2716 (0.075)	0.3114 (0.076)	0.2909 (0.071)

Tabla 7. Promedio general del MAE (m) por arquitectura.

Versión	Red original	Red #2	Red #3	Red #4
Promedio (sd)	0.1835 (0.043)	0.2055 (0.053)	0.2253 (0.060)	0.2117 (0.042)

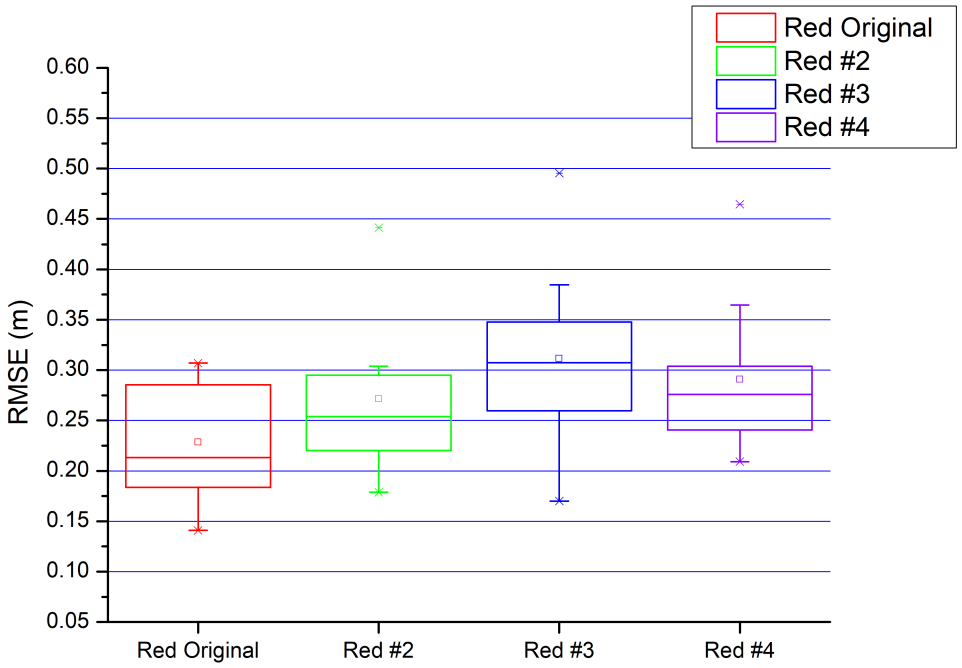


Figura 40. Diagrama de caja para el RMSE por red.

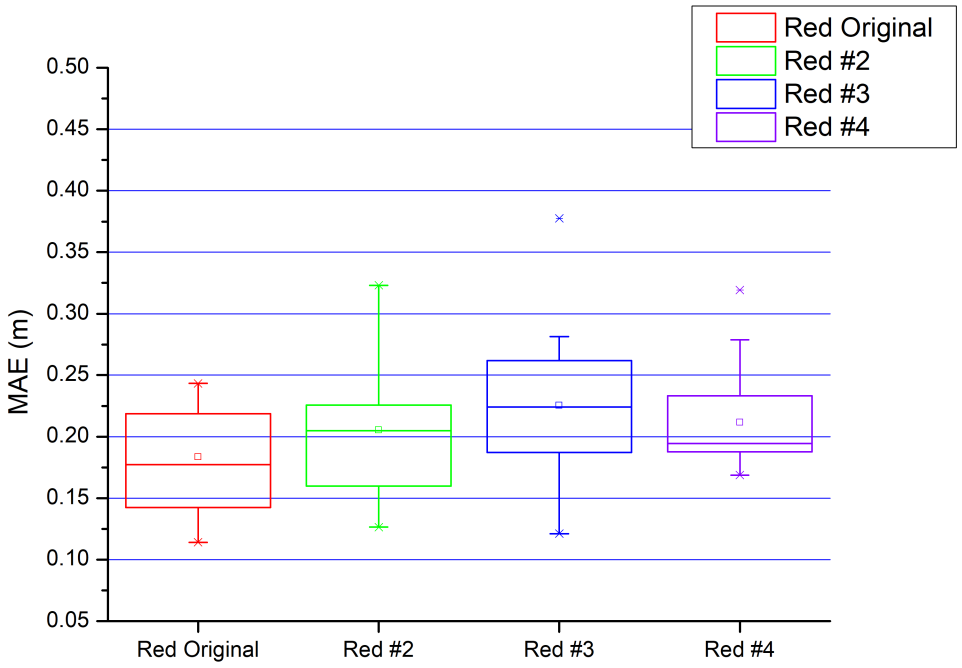


Figura 41. Diagrama de caja para el MAE por red.

Finalmente, se obtuvo el promedio de los resultados en función del tamaño de ventana, sin considerar el número de archivos ni la arquitectura de la red. Observando las tablas 8 y 9, no es fácil determinar cuál de los resultados es mejor, ya que los valores son muy cercanos entre si, por lo que para tener una conclusión más clara, en las figuras 42 y 43 se muestran los diagramas de caja correspondientes a estos resultados.

En el caso de una ventana de tamaño 200 la mediana esta por debajo del resto, sin embargo, el rango intercuartil es menor en el caso de la ventana de tamaño 150, además de que la media aritmética y el máximo no atípico son los más bajos, por lo que podríamos decir que la ventana de tamaño 150 presenta uno de los mejores resultados.

Tabla 8. Promedio general del RMSE (m) por tamaño de ventana.

Ventana	50	100	150	200	250
Promedio (sd)	0.2673 (0.082)	0.3013 (0.106)	0.2642 (0.052)	0.2597 (0.060)	0.2857 (0.065)

Tabla 9. Promedio general del MAE (m) por tamaño de ventana.

Ventana	50	100	150	200	250
Promedio (sd)	0.2041 (0.051)	0.2251 (0.075)	0.1937 (0.036)	0.2013 (0.044)	0.2082 (0.044)

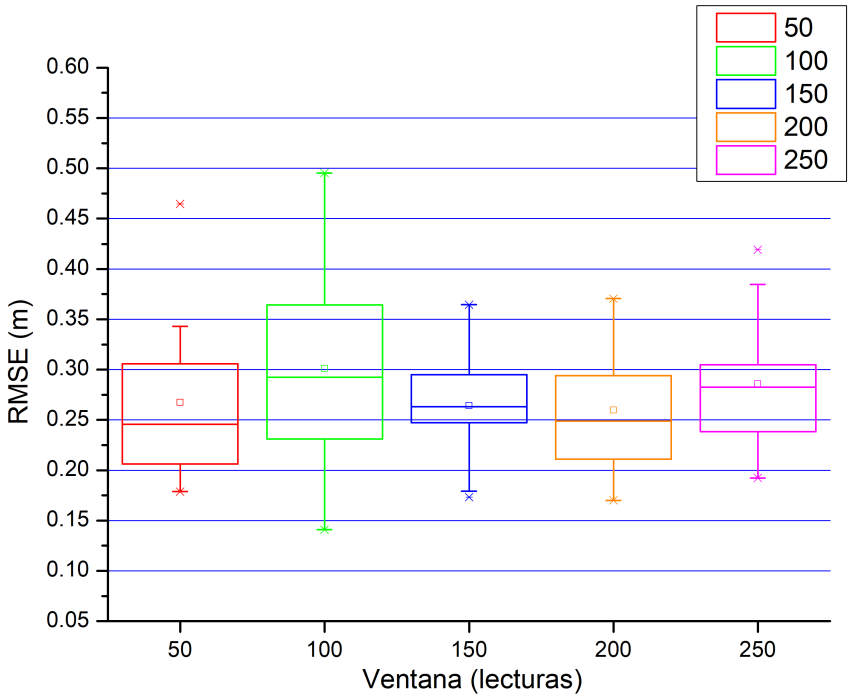


Figura 42. Diagrama de caja para el RMSE por ventana.

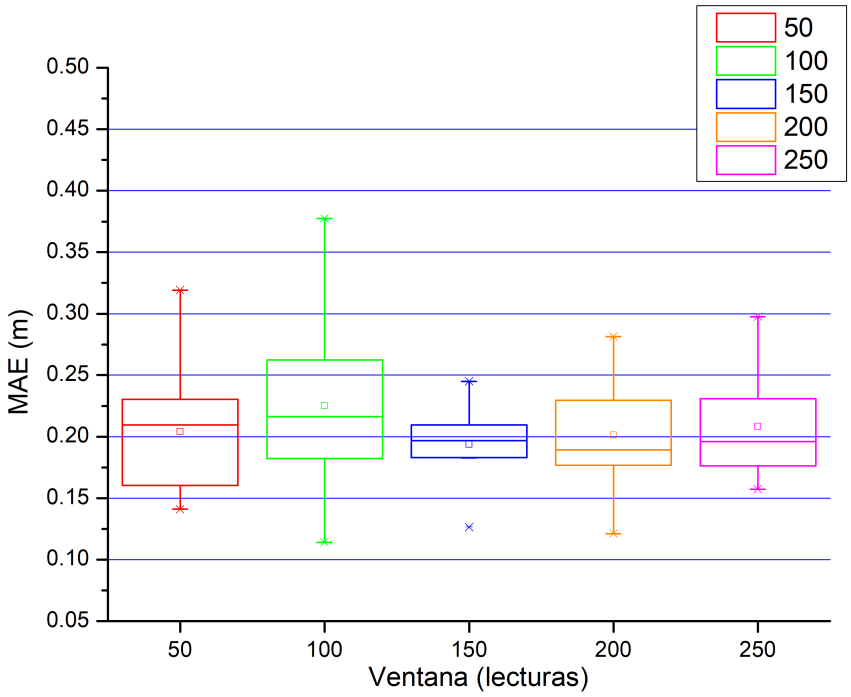


Figura 43. Diagrama de caja para el MAE por ventana.

Debido a que existe una correlación entre los valores del RMSE y el MAE, a partir de este momento se dejará de utilizar el RMSE como métrica para analizar los resultados presentados en este trabajo, por lo que únicamente se calculará el error MAE entre los valores predichos por la red y la trayectoria real. Además, en este caso, como los resultados a los que mide el error son diferencias geométricas, el MAE da una visión más clara sobre el promedio del mismo.

6.2. Implementación de una nueva aplicación

Hasta ahora, todos los resultados han sido obtenidos con redes entrenadas utilizando los datos inerciales de la aplicación #1, en la cual los usuarios tenían que caminar sobre un cuadrado. No obstante, esta aplicación solo es una de las seis que fueron diseñadas, por lo que se contaba con datos inerciales que no han sido utilizados. Por lo tanto, se decidió entrenar redes neuronales adicionales utilizando las aplicaciones restantes. Se entrenaron las redes neuronales mencionadas anteriormente a excepción de la red #3, que no utiliza las mediciones del giroscopio como entrada, esto debido

a que la red presentó un desempeño similar a la red #4. En este caso, a diferencia de las redes presentadas anteriormente, ahora se utilizaron los datos de la aplicación #2, donde los participantes tenían que caminar y pasar por debajo de tres obstáculos posicionados a diferentes alturas, como se puede apreciar en la figura 13(b) de la sección 4.3.2. Para evitar entrenar nuevamente una gran cantidad de redes neuronales, como en la aplicación pasada, se decidió tomar los parámetros de las redes que tuvieron un mejor desempeño. Para esta nueva aplicación se entrenaron únicamente 2 versiones por cada modificación hecha a la red, una con 10 y otra 14 archivos, ambas con una ventana de 150 lecturas inerciales, ya que en la mayoría de los casos el promedio del error se aproximaba al error mostrado en la ventana de tamaño 150. Debido a que en esta aplicación no se entrenaría con redes cubriendo la totalidad de los parámetros, se decidió entrenar tres veces cada una las redes con las configuraciones elegidas, esto para comprobar si la aleatoriedad intrínseca de las redes neuronales afectaba en gran medida a la predicción.

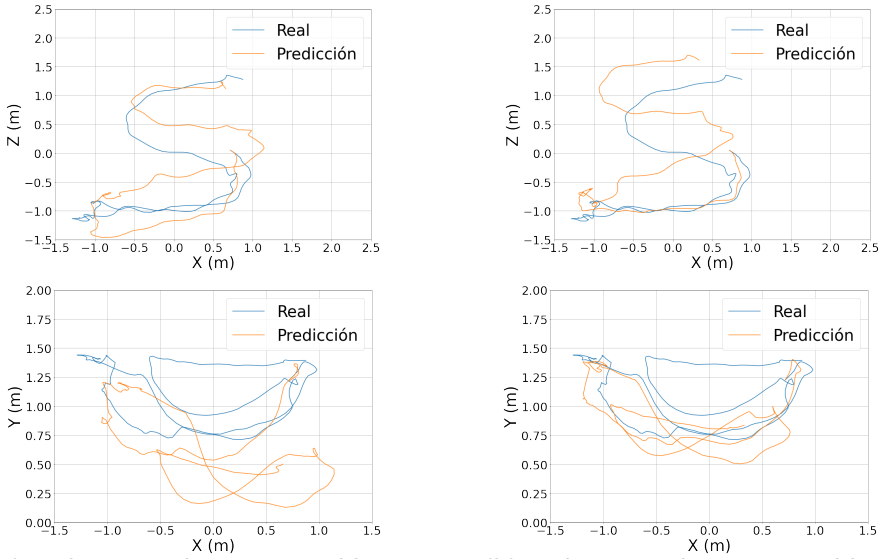
6.2.1. Red original $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{L}}, \Delta\hat{\mathbf{Q}})$: aplicación #2

La primera red neuronal entrenada fue la red original 6DOF IO, con 10 y 14 archivos utilizando una ventana de tamaño 150. Como se mencionó anteriormente, cada red se entrenó un total de tres veces y se eligió la que mejor resultado daba. En la figura 44 se muestran los resultados obtenidos por la red original entrenada con los datos de la aplicación #2.

Como se observa en la figura 44, los movimientos en esta aplicación son más complejos que en la aplicación #1, por lo que a la red se le dificulta realizar una predicción acertada. A simple vista se puede apreciar que el error es mayor que el de los resultados obtenidos por la red original entrenada con datos de la aplicación #1. Sin embargo, para la versión con 14 archivos, el error en el plano (Y, Z) disminuye en comparación con la red entrenada con 10 archivos.

6.2.2. Red #2 $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$: aplicación #2

Como se comentó anteriormente, también se entrenaron dos modificaciones hechas a la red. La red #2, que no utiliza cuaterniones, se entrenó tres veces utilizando



(a) Red entrenada con 10 archivos (b) Red entrenada con 14 archivos

Figura 44. 30 segundos de la trayectoria predicha por la red original entrenada con la aplicación #2.

10 y 14 archivos, con una ventana de tamaño 150.

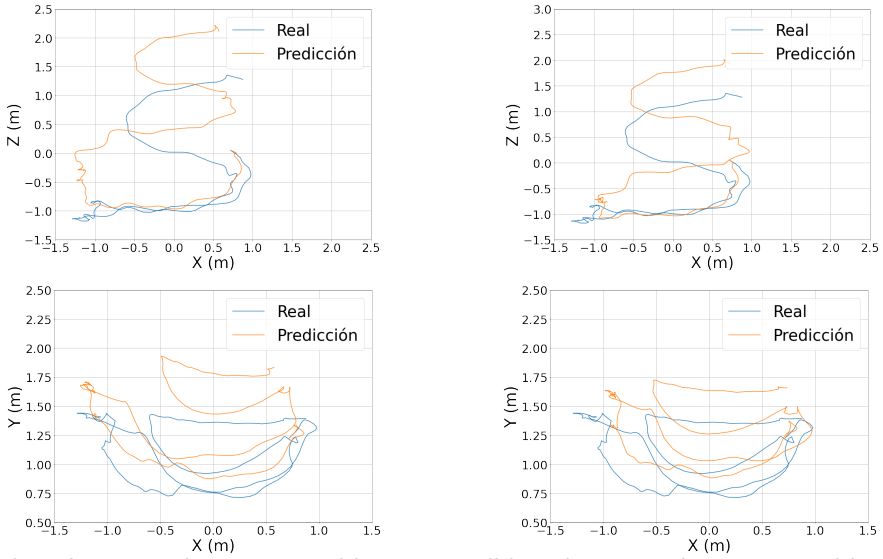
Se puede apreciar en la figura 45 que no hay una diferencia notable entre las dos versiones de la red, por lo que el añadir más datos a la red no parece mejorar la predicción.

6.2.3. Red #4 $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}}, \Delta\hat{\mathbf{Q}})$: aplicación #2

Posteriormente, se entrenaron tres versiones de la red #4, donde los vectores de posición $\Delta\mathbf{P}$ no eran rotados a un nuevo marco de referencia pero se seguían utilizando los cuaterniones para predecir la orientación.

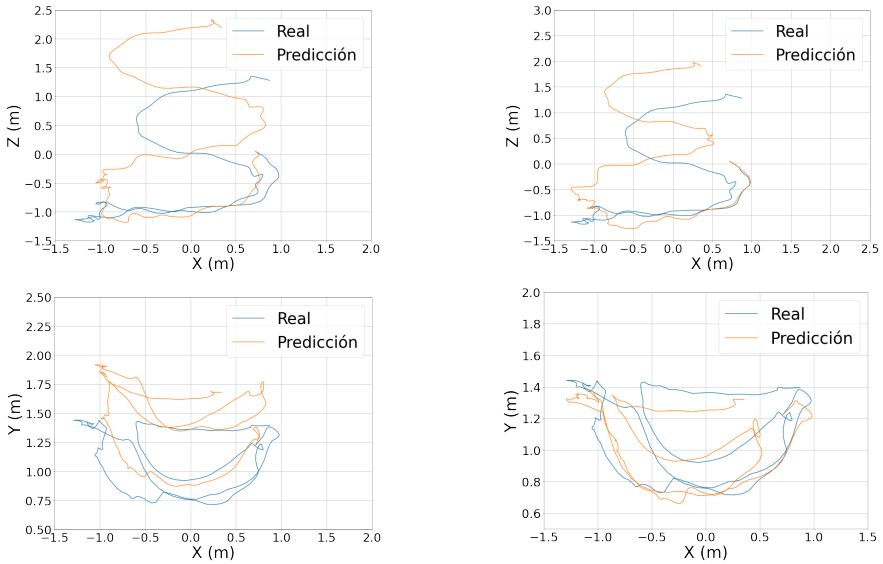
Al igual que en el caso anterior, se puede observar en la figura 46 que añadir datos inerciales no parece mejorar el error. El error en el plano (Y, Z) parece ser menor para el caso de la red entrenada con 14 archivos, pero para el plano (X, Z) el error no mejora. Los resultados cuantitativos del error se muestran en la tabla 10, incluyendo los resultados de las versiones repetidas de cada red. En la figura 47 se presentan los diagramas de caja para los resultados por número de archivos.

Como se muestra en la tabla 10, se puede notar que cuando se entrena la misma red utilizando el mismo número de archivos y el mismo tamaño de ventana, los resul-



(a) Red entrenada con 10 archivos (b) Red entrenada con 14 archivos

Figura 45. 30 segundos de la trayectoria predicha por la red #2 entrenada con la aplicación #2.



(a) Red entrenada con 10 archivos (b) Red entrenada con 14 archivos

Figura 46. 30 segundos de la trayectoria predicha por la red #4 entrenada con la aplicación #2.

tados varían notablemente. Por ejemplo, observando los resultados de la red original entrenada con 14 archivos, se aprecia que el error en un caso llega hasta los 0.3705 metros, mientras que en otro caso llega a los 0.2430 metros. Un comportamiento similar está presente para el resto de las redes. En la figura 47 podemos observar que para 14 archivos la media aritmética, la mediana y el rango intercuartil disminuyen notablemente en comparación con los 10 archivos.

Tabla 10. MAE (m) de las diferentes redes entrenadas utilizando la aplicación #2 con 10 y 14 archivos, ambas con una ventana de tamaño 150.

Archivos	**Entrenamiento**	**Red original**	**Red #2**	**Red #4**
		$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{L}}, \Delta\hat{\mathbf{Q}})$	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}}, \Delta\hat{\mathbf{Q}})$
10	1	0.3281	0.4127	0.3969
10	2	0.3585	0.4389	0.4338
10	3	0.4803	0.3813	0.3832
Promedio (sd)		**0.3890 (0.081)**	**0.4110 (0.029)**	**0.4046 (0.027)**
14	1	0.2430	0.3762	0.2531
14	2	0.3705	0.2665	0.4156
14	3	0.3246	0.3405	0.3931
Promedio (sd)		**0.3127 (0.065)**	**0.3277 (0.056)**	**0.3539 (0.088)**

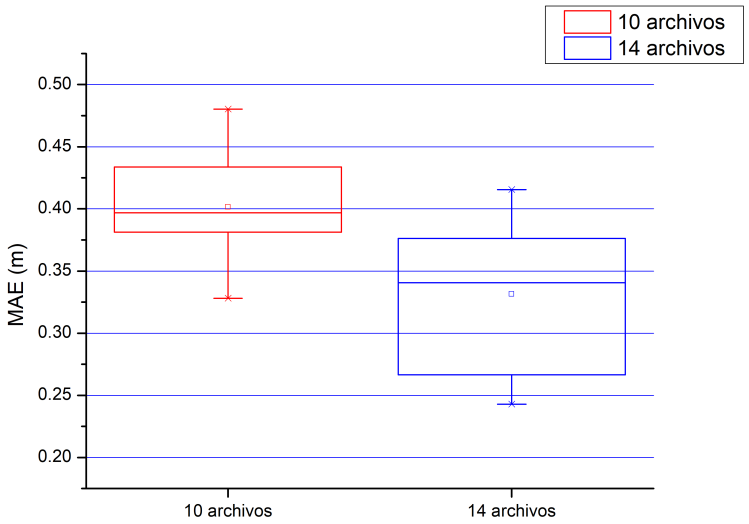


Figura 47. Diagrama de caja para el MAE por archivos de la aplicación #2.

En los resultados anteriores se observó que las redes entrenadas tienen dificultad para predecir los movimientos de la aplicación #2, la causa de esto puede ser que los datos contienen movimientos más complejos de representar para la red, ya que a diferencia de la aplicación #1, donde la altura de los usuarios no variaba, en estos archivos existen movimientos en los tres ejes cardinales. Observando la tabla 10, se puede afirmar que aumentar el número de archivos reduce el error de la predicción, y dado que se concluyó algo similar para los resultados de la aplicación #1, se decidió realizar un entrenamiento adicional añadiendo archivos de las seis aplicaciones desarrolladas. Lo anterior permitiría que la red tenga acceso a una mayor variedad de datos y podría mejorar la predicción de la trayectoria. Se eligió la red original para realizar el entrenamiento con los archivos de las seis aplicaciones, ya que es la red que predecía

con mayor precisión la trayectoria.

6.3. Entrenamiento utilizando las seis aplicaciones

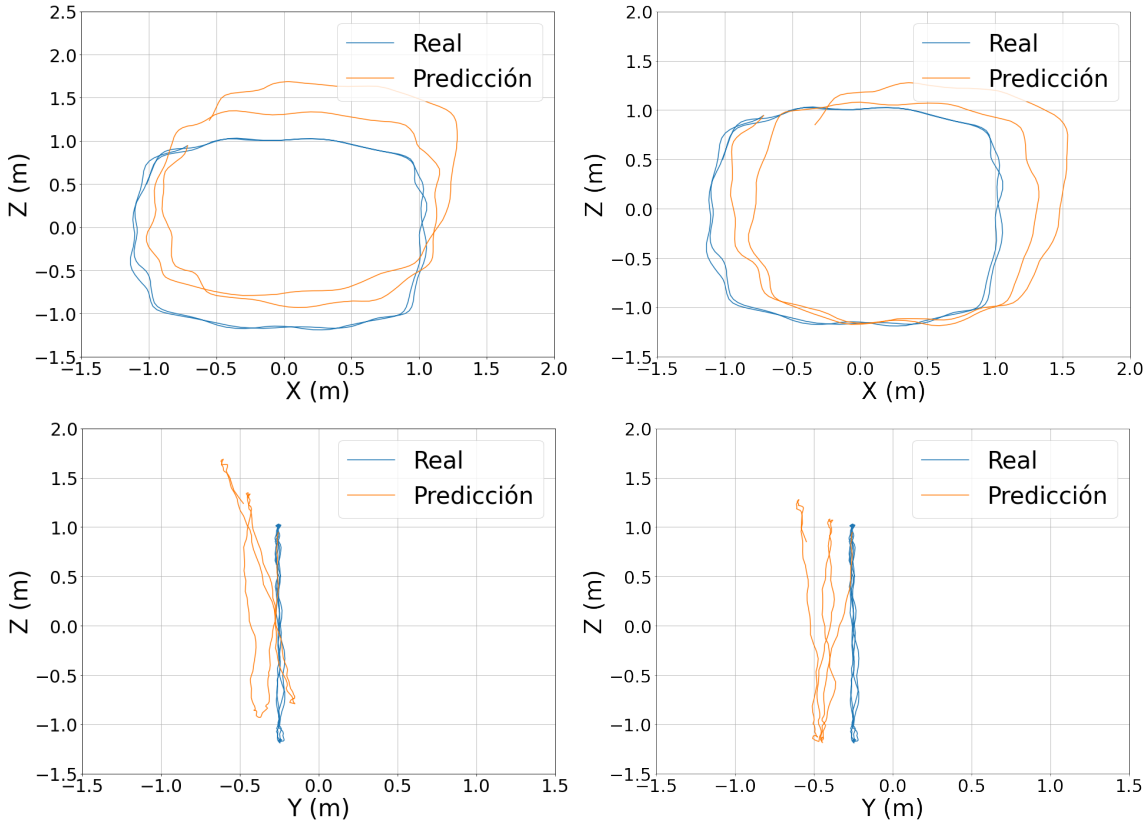
Analizando los resultados de las figuras 39 y 47, se puede notar que conforme aumenta el número de archivos, el error se reduce. Basados en esta observación, se procedió a entrenar una red neuronal utilizando los datos de las seis aplicaciones, mencionadas en el capítulo 4. La red seleccionada para ser entrenada fue la red original, ya que, a pesar del error que muestra en el plano (Y, Z), fue la que menor promedio de error reportó. En la primera prueba de entrenamiento, se utilizaron los primeros seis archivos de cada aplicación, para un total de 36 archivos, en la segunda prueba se utilizaron los primeros 10 archivos de cada aplicación, para un total de 60 archivos, y finalmente, de los 15 archivos disponibles para cada aplicación, se tomaron 14 archivos para entrenar, haciendo un total de 84 archivos, y dejando uno para validación. Todos los entrenamientos fueron hechos con una ventana de tamaño 150.

Como se aprecia en la figura 48 y en la tabla 11, el error no siempre disminuye aunque se aumente el número de archivos. Una explicación de esto puede ser que la red ha alcanzado un límite en cuanto a la capacidad de predicción que tiene, y aunque se aumente el número de datos, la red no podrá mejorar su resultado. Para tratar de solucionar esto, se propuso hacer una modificación a la red para aumentar su capacidad de predicción, lo cual se describe en la siguiente sección.

6.4. Añadir capas LSTM bidireccionales

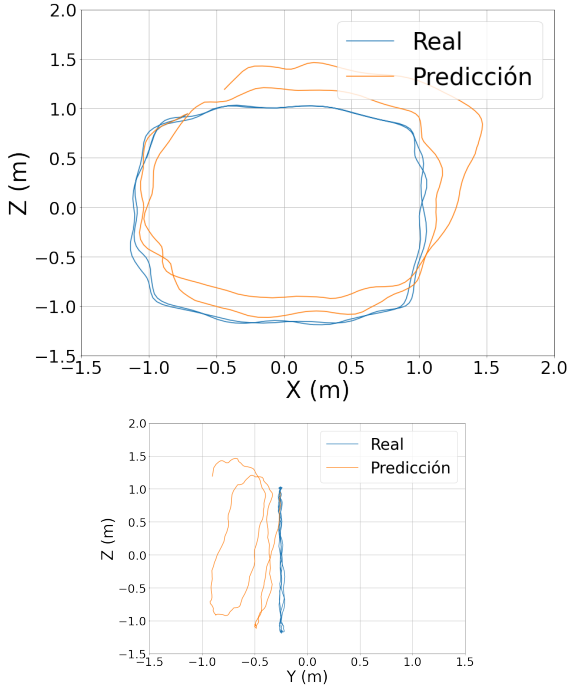
Debido a que la red neuronal utilizada posiblemente llegó a un punto donde a pesar de incrementar el número de datos para el entrenamiento no es posible mejorar su capacidad de predicción, se decidió implementar una nueva arquitectura para mejorar el desempeño. Esto mediante el uso de dos capas LSTM bidireccionales adicionales, para un total de cuatro capas LSTM, que dan a la red la posibilidad de generalizar una mayor cantidad de información.

De los resultados en la tabla 11 se puede observar que el error en el plano (Y, Z) no se reduce al aumentar el número de archivos. Por lo tanto, se decidió entrenar



(a) 36 archivos de entrenamiento.

(b) 60 archivos de entrenamiento.



(c) 84 archivos de entrenamiento

Figura 48. Resultados después de graficar 30 segundos de la trayectoria predicha por la red original entrenada con 6, 10 y 14 archivos de cada una de las seis aplicaciones creadas.

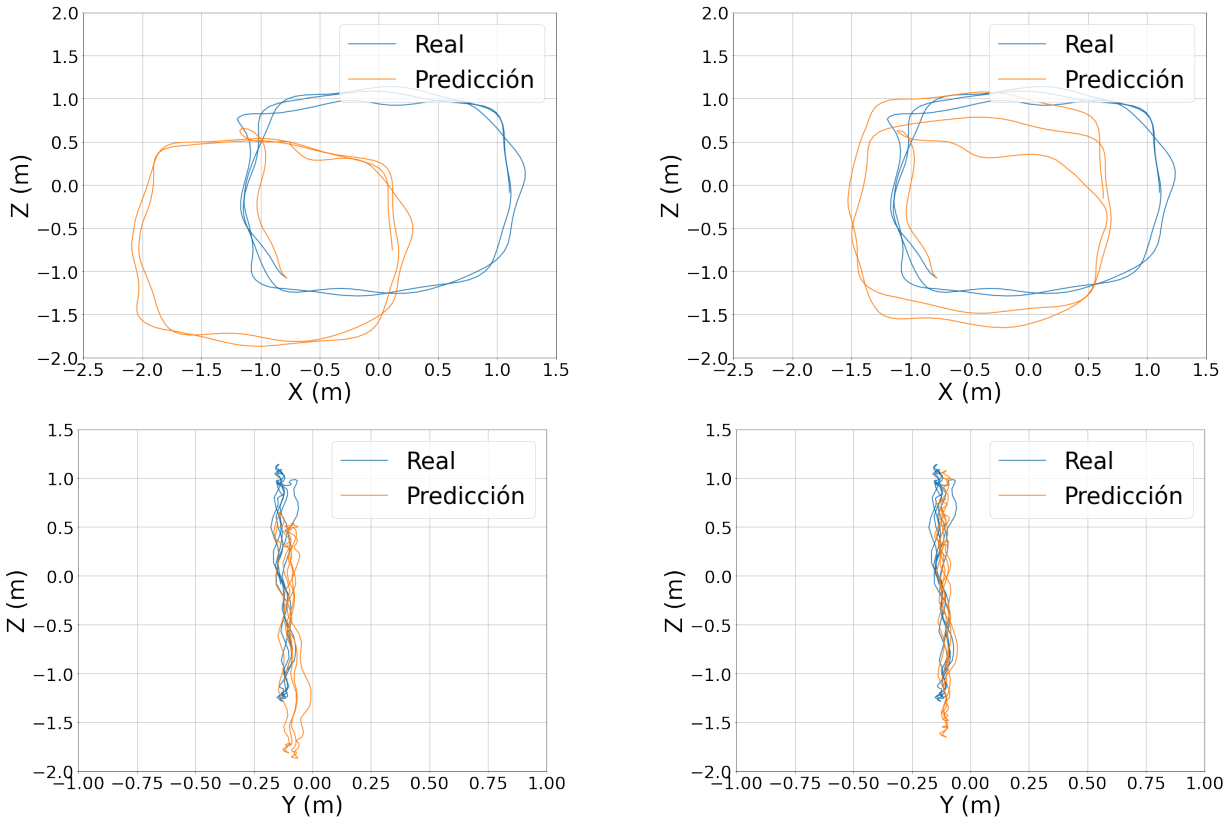
Tabla 11. MAE (m) de la red original entrenada con archivos de todas las aplicaciones.

Archivos	**Red Original**
	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{L}}, \Delta\hat{\mathbf{Q}})$
36	0.2370
60	0.2044
84	0.2324

una nueva red a partir de la versión #2, definida en la sección 5.3.2, que no utiliza cuaterniones y sus vectores de posición están descritos en un marco de referencia global $(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$. Se eligió esta arquitectura porque con esta versión de la red se redujo considerablemente el error en el plano (Y, Z) en comparación con la red original y su predicción en el plano (X, Z) mejora al aumentar el número de archivos, como se observa en la figura 29. Por esta razón, se espera que al tener acceso a más datos de entrenamiento, la red con capas adicionales será capaz de aumentar su capacidad de predicción en comparación con la versión anterior con 2 capas LSTM bidireccionales.

De manera paralela, se entrenó una red adicional con la misma arquitectura pero que no utiliza las lecturas del giroscopio como entrada $(\mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$, lo cual se puede considerar como una versión con menos parámetros y que recibe la información mínima para predecir los vectores $\Delta\hat{\mathbf{P}}_{\mathbf{G}}$. Después de comparar ambas redes se continuará utilizando la que mejores resultados presente. Para ambas redes se utilizó una ventana de tamaño 100, ya que como en este caso se busca tener la mayor cantidad de datos, mientras más pequeña es la ventana, mayor es la cantidad de ventanas creadas. Como el entrenamiento para dichas redes podría tomar mucho más tiempo, se eligió una ventana más pequeña, pero no lo suficiente como para comprometer el tiempo de entrenamiento. A continuación se muestran los resultados de la comparación entre las dos redes entrenadas utilizando 14 archivos de cada una de las seis aplicaciones creadas, para un total de 84 archivos.

Como se aprecia en la figura 49, la trayectoria calculada con la red entrenada con datos del giroscopio presenta un desfase mayor con respecto a la trayectoria real. Mientras que la trayectoria generada con la red entrenada sin lecturas del giroscopio, a pesar de también mostrar un desfase, este es menor que en el otro caso. Esta trayectoria fue tomada de la aplicación #1, pero como las redes han sido entrenadas con datos de todas las aplicaciones, se analizaron trayectorias de otras aplicaciones. En la siguiente figura se muestra una trayectoria de la aplicación #2, donde los usuarios



(a) Red entrenada con giroscopio.

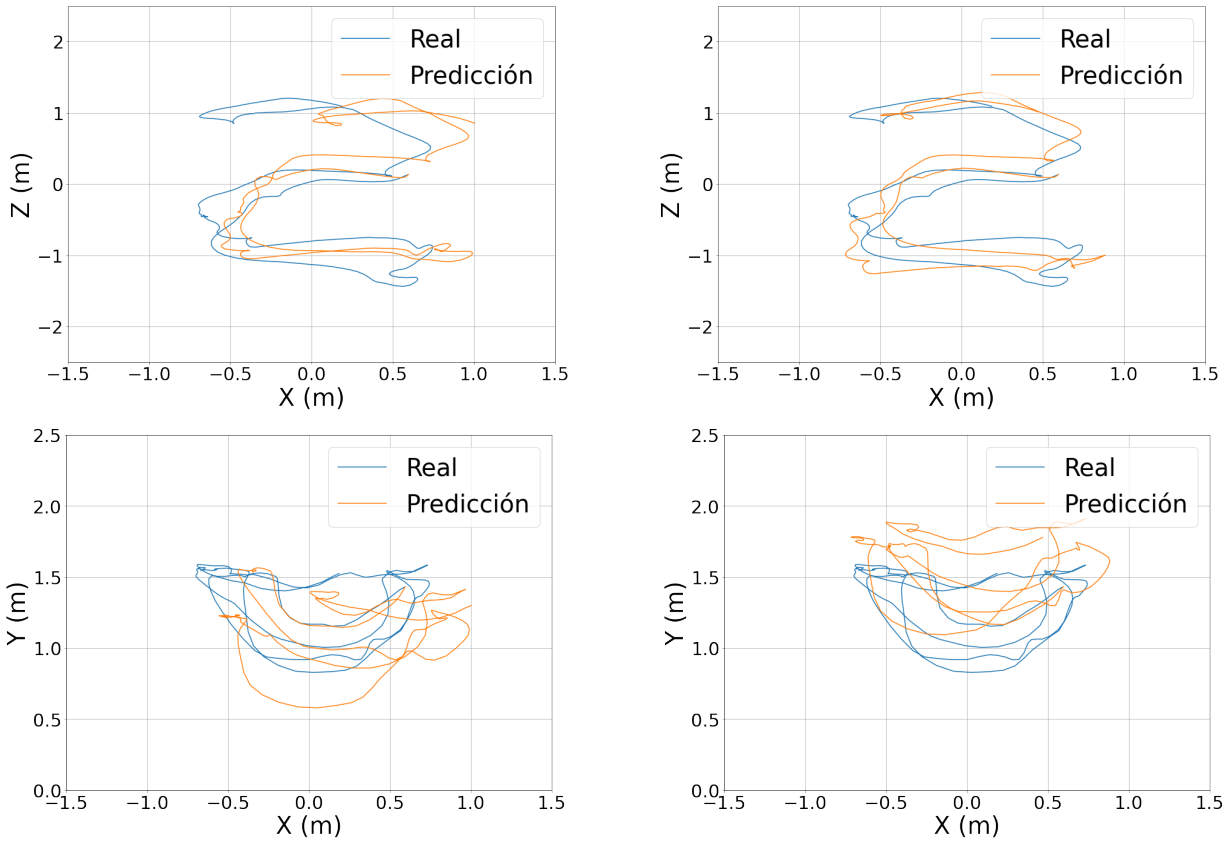
(b) Red entrenada sin giroscopio.

Figura 49. Comparación entre los resultados de graficar una trayectoria de la aplicación #1 utilizando una red entrenada con datos del giroscopio y otra sin ellos.

debían agacharse para pasar por debajo de obstáculos.

En la figura 50 se puede observar que la trayectoria tiene movimientos significativos en los tres ejes cardinales, lo cual complica la predicción de la red. Se observa que la versión que utiliza las lecturas del giroscopio presenta un pequeño desfase en los ejes X y Z en comparación con la red que no las utiliza. Posteriormente, se realizó la predicción de una trayectoria tomada de la aplicación #4, detallada en las sección 4.3.4. Aquí los usuarios realizaban rotaciones de la cabeza para observar los alrededores y se podían desplazar de su punto inicial.

En la figura 51 se puede apreciar que las redes no son capaces de predecir correctamente la trayectoria en este caso. Esto puede deberse a la complejidad que tiene esta aplicación, ya que a diferencia de las anteriores, no existe un patrón de movimiento definido, por lo que el resto de las trayectorias de esta misma aplicación pueden variar bastante. Finalmente, se realizó una predicción de la trayectoria de la aplicación #5,



(a) Red entrenada con giroscopio.

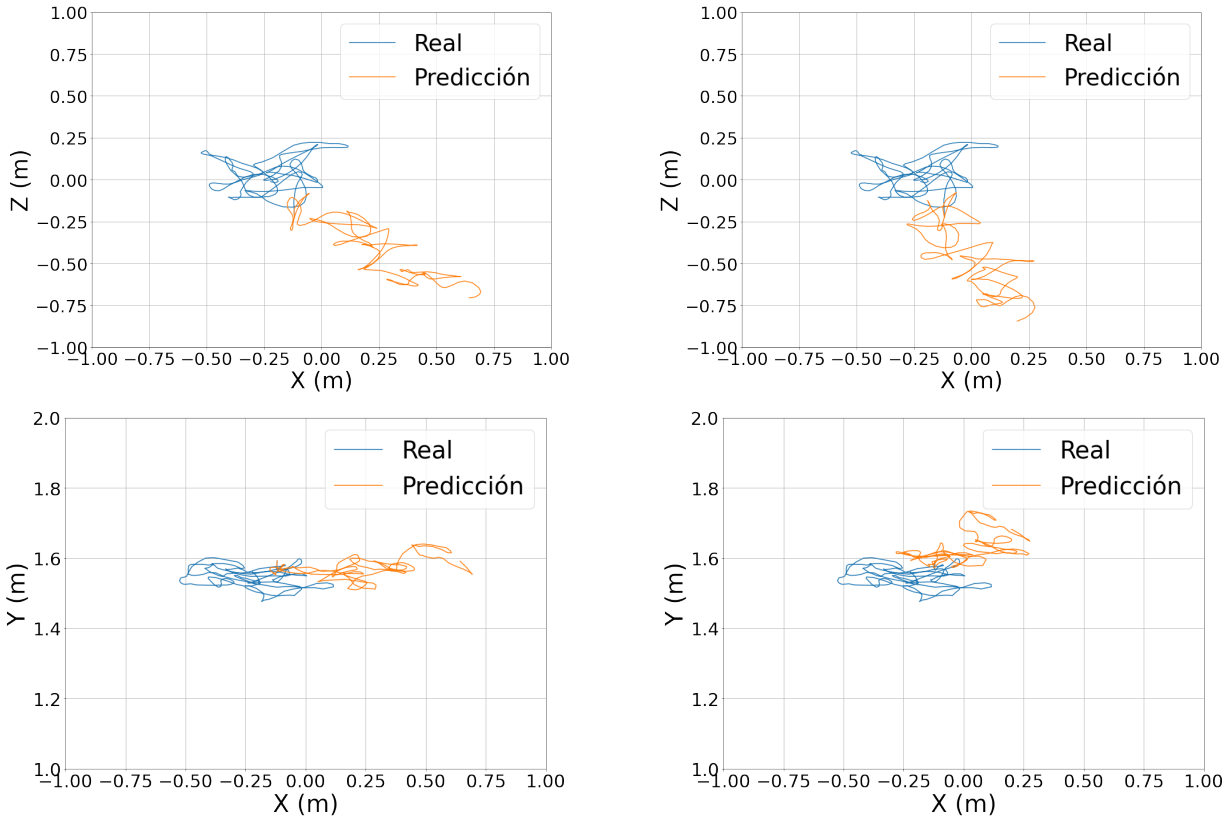
(b) Red entrenada sin giroscopio.

Figura 50. Comparación entre los resultados de graficar una trayectoria de la aplicación #2.

que es donde los usuarios deben esquivar proyectiles arrojados hacia ellos, por lo que tenían que moverse rápidamente de izquierda a derecha.

Se puede ver en la figura 52 que a pesar de que la trayectoria cambia rápidamente, la predicción se realiza de manera correcta para ambas redes. Una posible explicación es que los archivos de la aplicación #5 comparten movimientos repetitivos, por lo que la red es capaz de aprenderlos y predecirlos de manera correcta, a diferencia de los movimientos de la aplicación #4, donde existe más libertad para el usuario, lo cual hace que no compartan mucha información y se dificulte su aprendizaje por la red. En la tabla 12 se muestra el error MAE calculado para cada una de las redes entrenadas.

De los resultados mostrados en la tabla 12, podemos concluir que el uso del giroscopio como entrada a la red causa que el error aumente, esto puede deberse a que al añadir esta información adicional, el ruido en las lecturas aumenta. Para comprobar que realmente el uso de capas LSTM adicionales beneficia en la predicción de la red, se entrenó una versión de la red neuronal que no utiliza las lecturas del giroscopio, red



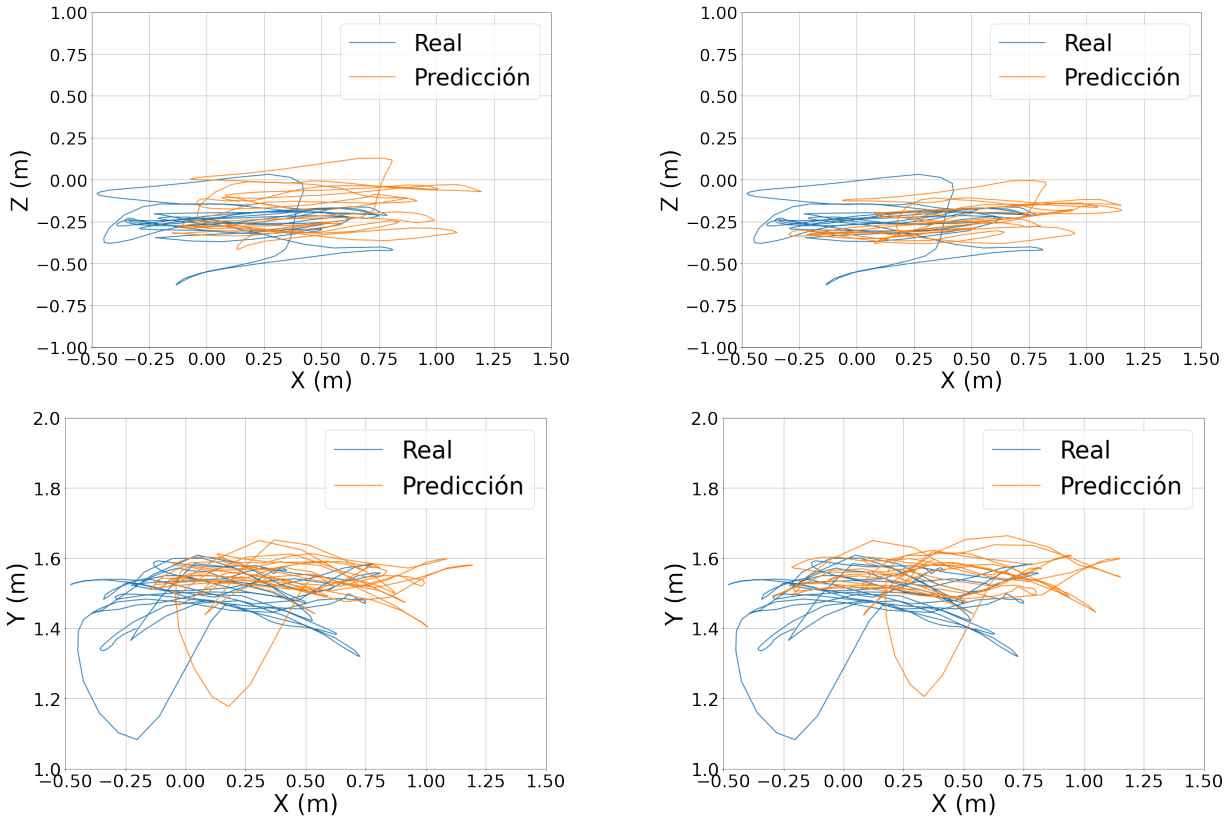
(a) Red entrenada con giroscopio.

(b) Red entrenada sin giroscopio.

Figura 51. Comparación entre las predicciones de una trayectoria de la aplicación #4.

#3 mostrada en la sección 6.1.3, con cuatro capas LSTM bidireccionales. Al igual que en la sección 6.1.3, para el entrenamiento se usaron 14 archivos de la aplicación #1 y se comparó con su versión entrenada con dos capas LSTM bidireccionales.

Observando los resultados de la tabla 13 y la figura 53 se puede afirmar que utilizar cuatro capas LSTM ayudan a la red a reducir el promedio del error. Por lo tanto, se procedió a hacer un nuevo entrenamiento empleando esta arquitectura para cada una de las aplicaciones desarrolladas, es decir, se entrenó una red con datos únicamente de la aplicación #2, otra red utilizando datos de la aplicación #3, y así sucesivamente con el resto de las aplicaciones.



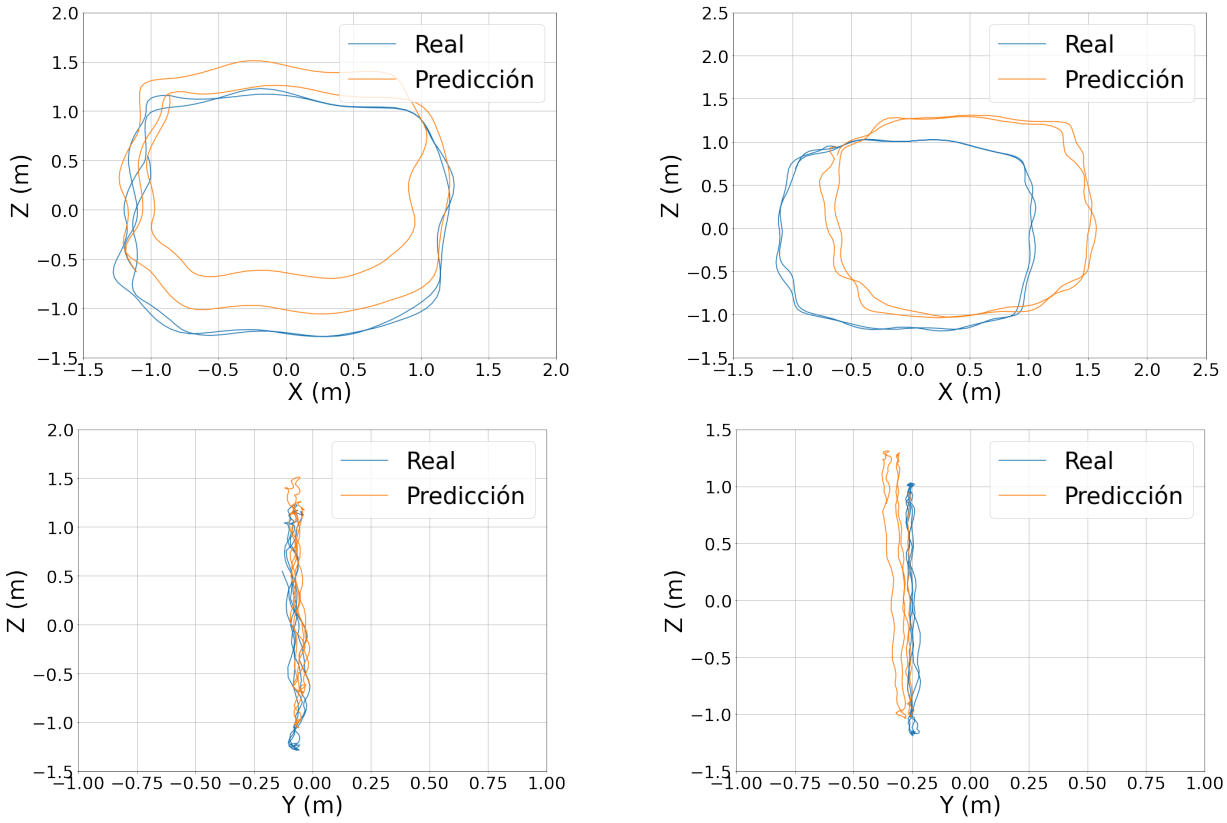
(a) Red entrenada con giroscopio.

(b) Red entrenada sin giroscopio.

Figura 52. Comparación entre predicciones de una trayectoria de la aplicación #5.

Tabla 12. Error MAE (m) de las dos redes entrenadas.

Aplicación	**MAE**	
	Red #2 - 4LSTM	**Red #3 - 4LSTM**
	$(\boldsymbol{\omega}, \mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$	$(\mathbf{a}) \rightarrow (\Delta\hat{\mathbf{P}}_{\mathbf{G}})$
1	0.4549	0.2155
2	0.1927	0.1850
4	0.3022	0.2640
5	0.1447	0.1353



(a) Red con cuatro capas LSTM bidireccionales

(b) Red con dos capas LSTM bidireccionales

Figura 53. (a) Red entrenada con 14 archivos y cuatro capas LSTM bidireccionales. (b) Red entrenada con 14 archivos y dos capas LSTM bidireccionales.

Tabla 13. MAE (m) de dos redes entrenadas con diferente número de capas.

Archivos	**Red (a) → ($\Delta\hat{\mathbf{P}}_{\mathbf{G}}$)**	
	4 LSTM	2 LSTM
14	0.1294	0.2457

6.5. Comparativa: red general y red específica

La red utilizada en la sección 6.4, que utiliza 4 capas LSTM bidireccionales, fue entrenada utilizando la mayor cantidad de archivos disponibles para cada aplicación en la base de datos, y como contiene información de todas las aplicaciones, se espera que la red sea capaz de predecir trayectorias con diversos movimientos. En este caso, surge la pregunta sobre si una red con esta arquitectura, entrenada con un movimiento en especifico, es mejor al momento de predecir una trayectoria de la misma naturaleza con la que fue entrenada, que una red entrenada con todos los archivos los cuales consideran una mayor clase de movimientos. Para esto, se tomó la red entrenada en la sección 6.4 que no utiliza las lecturas del giroscopio, ya que fue la que mejores re-

sultados obtuvo, y se entrenó una nueva red con la misma arquitectura pero utilizando únicamente los datos de la aplicación #1, donde los usuarios tenían que caminar sobre un cuadrado. En la figura 54 se muestra una comparación entre la predicción de una trayectoria con la red general, entrenada con todos los archivos posibles, y una red que solo utiliza datos de la aplicación #1.

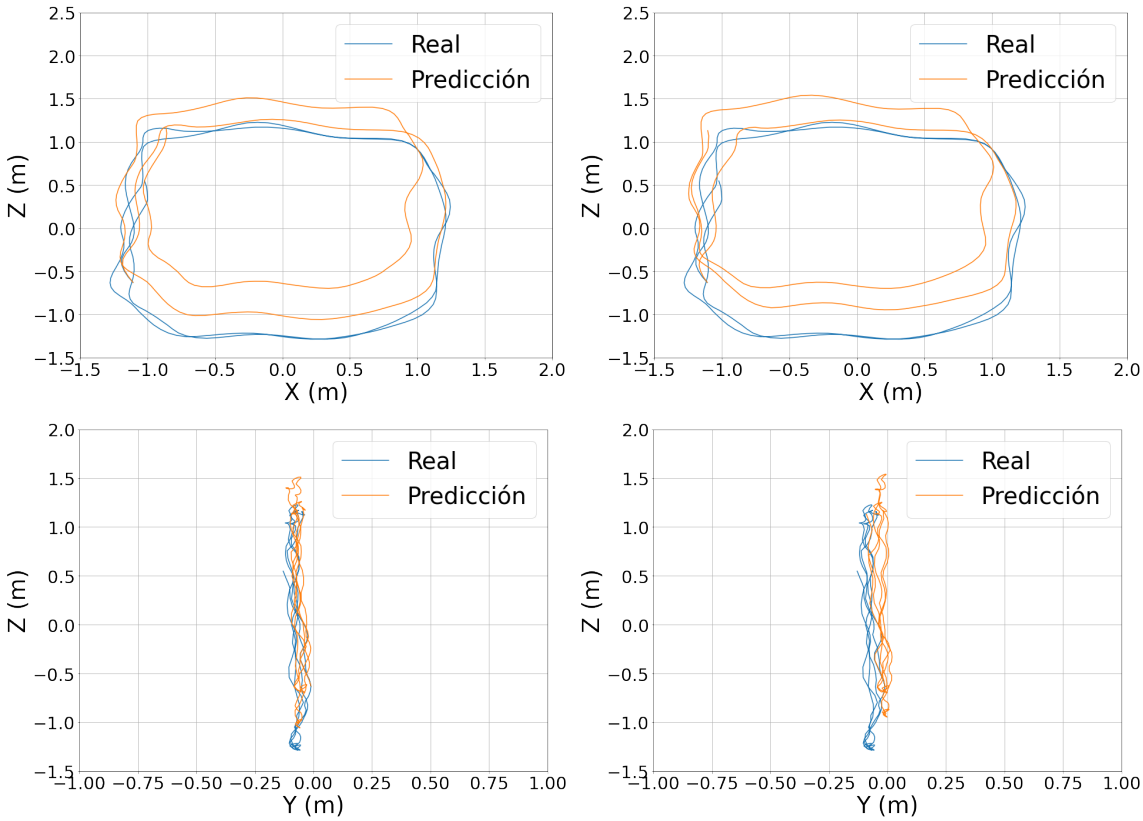
Como se aprecia en la figura 54, ambas redes parecieran funcionar correctamente cuando se trata de predecir una trayectoria de la aplicación #1, pero observando la tabla 14 podemos ver que la red entrenada solo con datos de la aplicación #1 tiene un error menor. Después, se procedió a utilizar una red entrenada utilizando únicamente datos de la aplicación #2, para compararla con la red entrenada con todos los archivos.

En la figura 55 y en la tabla 15 se muestran los resultados de la red general entrenada con datos de todas las aplicaciones presentó un error menor que la red entrenada únicamente con datos de la aplicación #2.

Similarmente, se entrenó una red entrenada únicamente con datos de la aplicación #3. En este caso, el mejor resultado se presenta para la red entrenada particularmente con datos de la aplicación #3, como se observa en la tabla 16. Para la red general se puede apreciar en la figura 56 que la predicción tiene un desplazamiento hacia la derecha en relación con la trayectoria real. También se analizó una red entrenada exclusivamente con datos de la aplicación #4.

Para la aplicación #4 se observa en la tabla 17 y la figura 57 que la red general tiene un mejor desempeño. Posteriormente, se entrenó otra red considerando solo datos de la aplicación #5.

Como se observa en la figura 58 y en la tabla 18, la red general tiene un mejor desempeño. Finalmente, se analizó una red entrenada solo con archivos de la aplicación #6 y sus resultados se compararon con los de la red general.

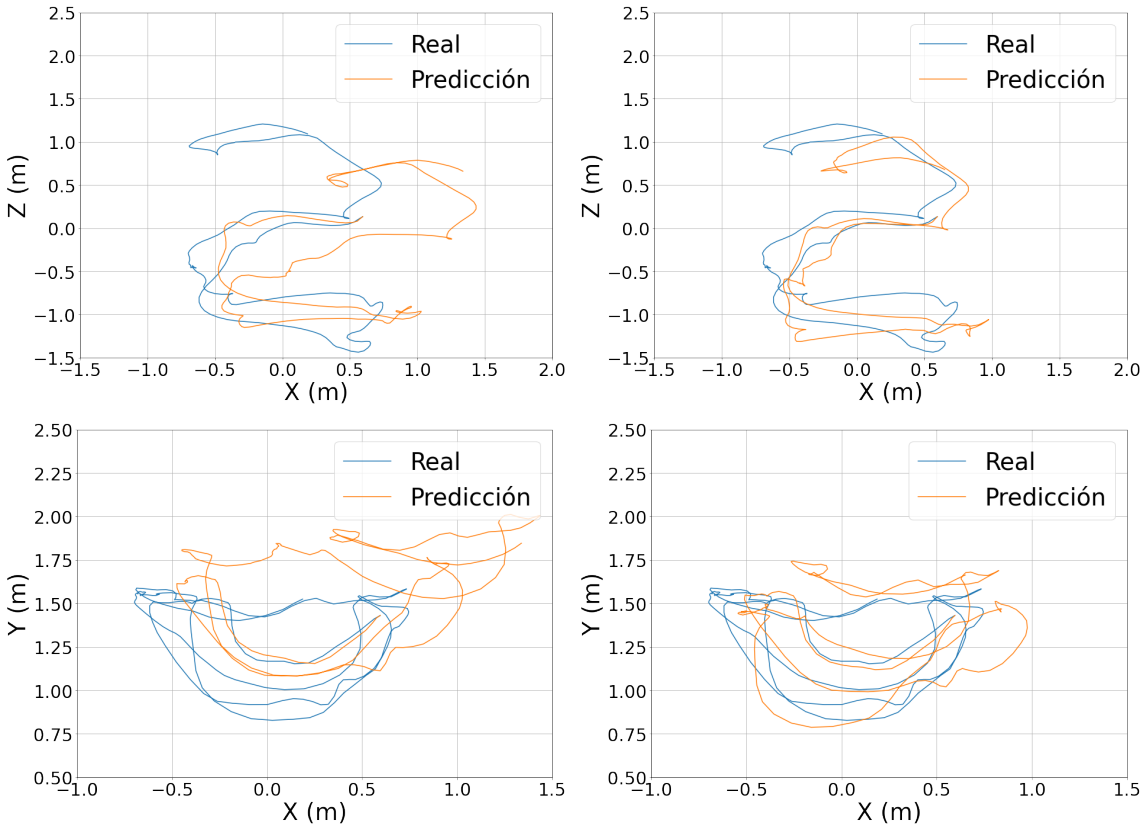


(a) Red entrenada con 13 archivos. (b) Red entrenada con todos los archivos.

Figura 54. (a) Red entrenada únicamente con datos de la aplicación #1. (b) Red entrenada utilizando datos de las seis aplicaciones. La trayectoria predicha por ambas redes forma parte de los datos de la aplicación #1.

Tabla 14. MAE (m) de las dos redes utilizadas para estimar la trayectoria de un archivo de la aplicación #1.

Red	**MAE**
Particular	0.1294
General	0.1550

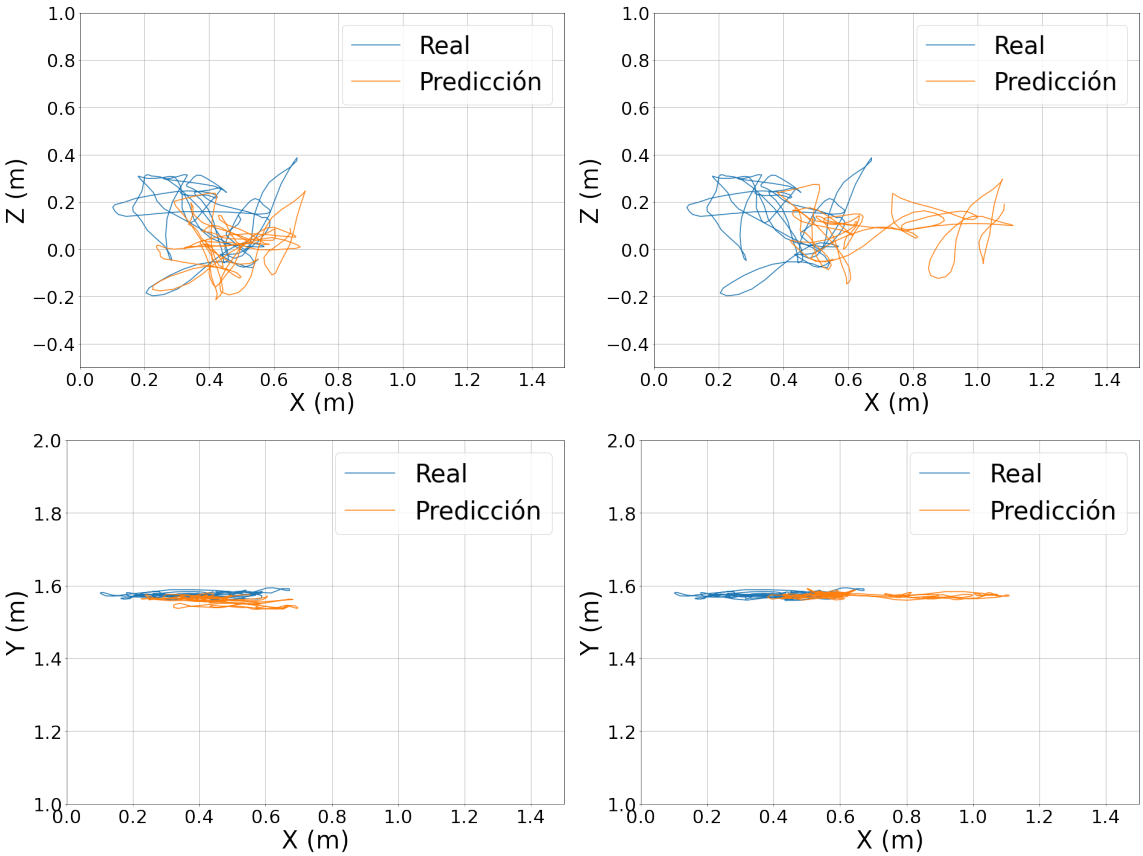


(a) Red entrenada con 13 archivos. (b) Red entrenada con todos los archivos.

Figura 55. (a) Red entrenada únicamente con datos de la aplicación #2. (b) Red entrenada utilizando datos de las seis aplicaciones. Ambas redes predicen un archivo tomado de la aplicación #2.

Tabla 15. MAE (m) de las dos redes utilizadas para estimar la trayectoria de un archivo de la aplicación #2.

Red	MAE
Particular	0.3273
General	0.1571

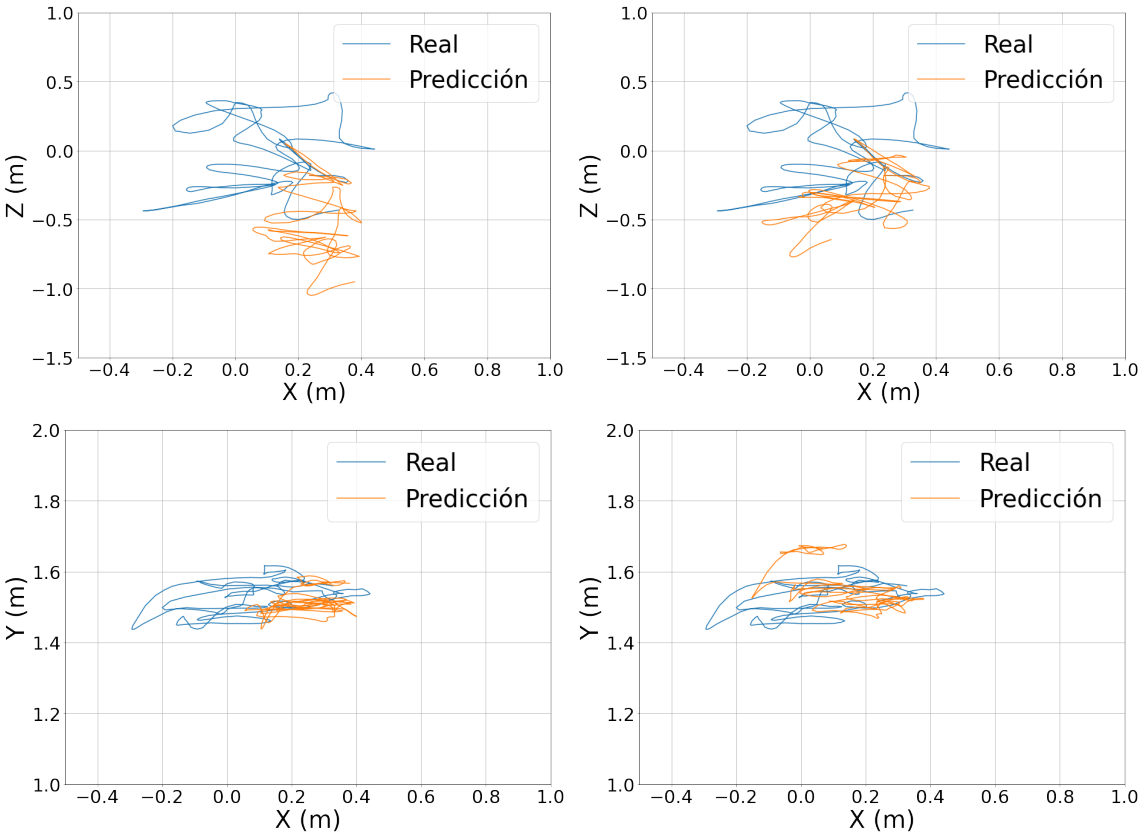


(a) Red entrenada con 13 archivos. (b) Red entrenada con todos los archivos.

Figura 56. (a) Red entrenada únicamente con datos de la aplicación #3. (b) Red entrenada utilizando datos de las seis aplicaciones. Ambas redes predicen un archivo de la aplicación #3.

Tabla 16. MAE (m) de las dos redes utilizadas para estimar la trayectoria de un archivo de la aplicación #3.

Red	MAE
Particular	0.0737
General	0.1315

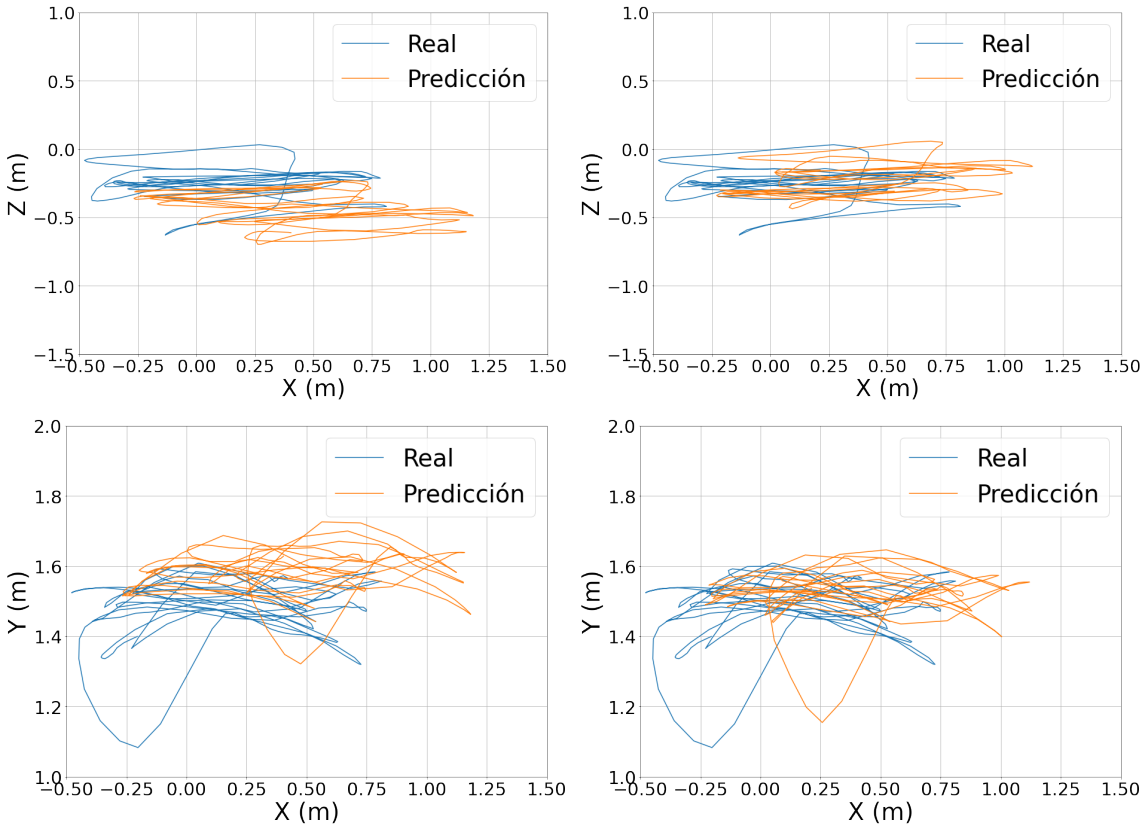


(a) Red entrenada con 13 archivos. (b) Red entrenada con todos los archivos.

Figura 57. (a) Red entrenada únicamente con datos de la aplicación #4. (b) Red entrenada utilizando datos de las seis aplicaciones. Ambas redes predicen un archivo de la aplicación #4.

Tabla 17. MAE (m) de las dos redes utilizadas para estimar la trayectoria de un archivo de la aplicación #4.

Red	MAE
Particular	0.2199
General	0.1423



(a) Red entrenada con 13 archivos. (b) Red entrenada con todos los archivos.

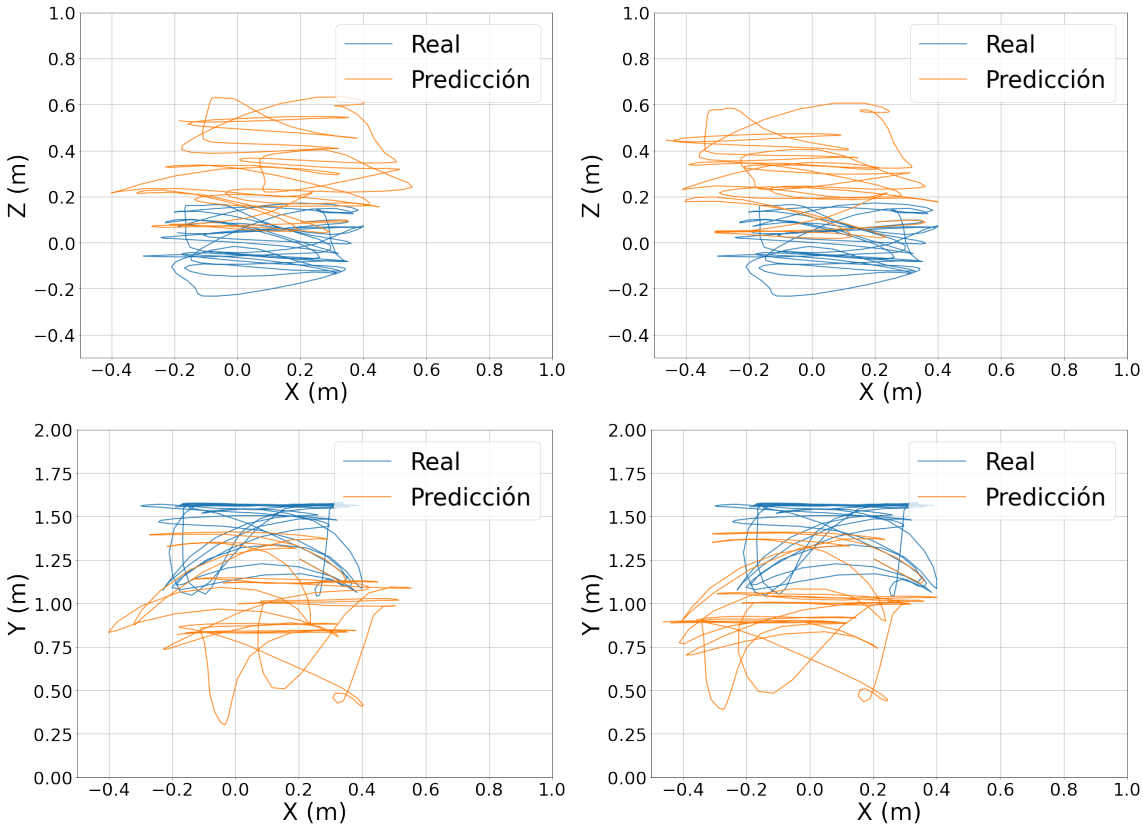
Figura 58. (a) Red entrenada únicamente con datos de la aplicación #5. (b) Red entrenada utilizando datos de las seis distintas aplicaciones. Ambas redes predicen un archivo de la aplicación #5.

Tabla 18. MAE (m) de las dos redes utilizadas para estimar la trayectoria de un archivo de la aplicación #5.

Red	MAE
Particular	0.2018
General	0.1246

En este caso, ambas redes presentaron resultados similares, como se observa en la tabla 19 y en la figura 59. En la tabla 20 se presenta un resumen de los resultados cuantitativos de las 6 aplicaciones.

De los resultados mostrados en la tabla 20 y en la figura 60 podemos concluir que la red general tiene un mejor comportamiento al tratar de predecir todos los movimientos y que su error no difiere mucho entre las diferentes aplicaciones. Se puede afirmar entonces que es una red que predice de mejor manera una gran variedad de movimientos. En cambio, una red que solo fue entrenada para un tipo de movimiento en específico, probablemente no tendrá un buen desempeño cuando trate de predecir



(a) Red entrenada con 13 archivos. (b) Red entrenada con todos los archivos.

Figura 59. (a) Red entrenada únicamente con datos de la aplicación #6. (b) Red entrenada utilizando datos de las seis distintas aplicaciones. Ambas redes predicen un archivo de la aplicación #6.

Tabla 19. MAE (m) de las dos redes utilizadas para estimar la trayectoria de un archivo de la aplicación #6.

Red	MAE
Particular	0.2871
General	0.2923

movimientos que no vio durante su fase de entrenamiento. Para comprobar esto, en la siguiente sección se realizan pruebas utilizando una red entrenada con un movimiento en específico, por ejemplo datos de la aplicación #1, haciendo predicción de movimientos no vistos anteriormente, por ejemplo de la aplicación #2.

6.6. Inferencia de una clase no vista.

Para comprobar la precisión de una red entrenada con un cierto tipo de movimiento al tratar de predecir tipos de movimientos no antes vistos, se utilizaron dos redes de la sección anterior, la red entrenada con datos de la aplicación #3, donde los usua-

Tabla 20. MAE (m) de los resultados obtenidos de una red general entrenada con datos de todas las aplicaciones y una red entrenada con una aplicación en particular.

Aplicación	Red	
	General	**Particular**
1	0.1550	0.1294
2	0.1571	0.3273
3	0.1315	0.0737
4	0.1423	0.2199
5	0.1246	0.2018
6	0.2923	0.2871
Promedio (sd)	0.1671 (0.063)	0.2065 (0.095)

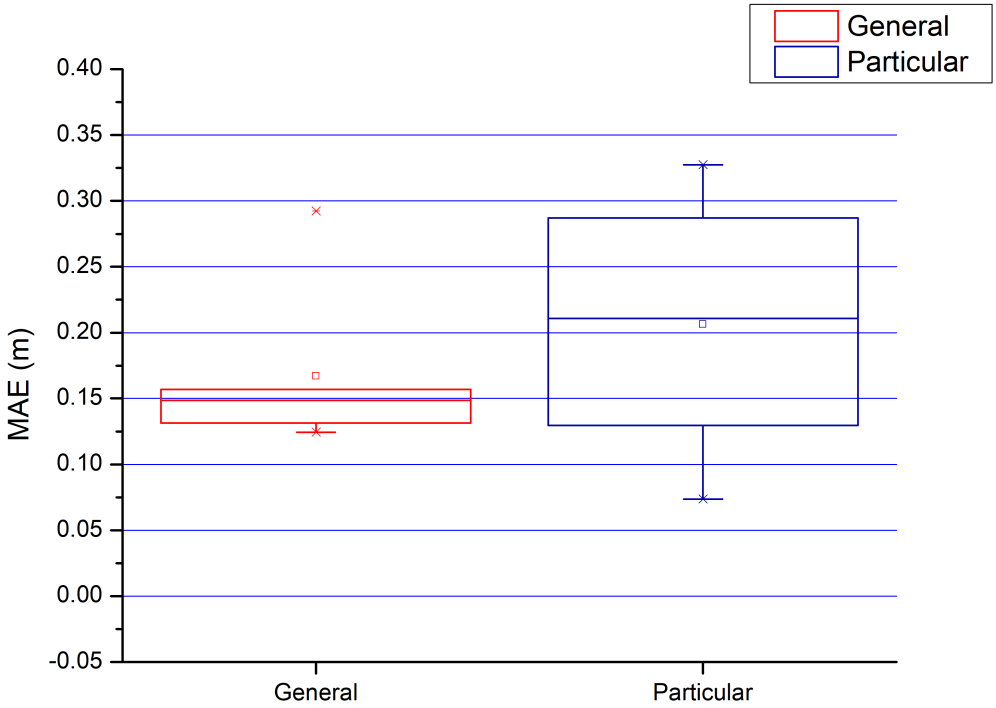


Figura 60. Diagrama de caja para el MAE de una red general y una particular.

rios tenían que girar a su alrededor y la traslación era opcional, y la entrenada con la aplicación #2, donde los usuarios debían trasladarse y pasar por debajo de tres obstáculos. Ambas redes fueron empleadas para tratar de predecir movimientos de la aplicación #1, donde se debía caminar sobre un cuadrado, y de la aplicación #5, donde los usuarios debían esquivar los proyectiles que se dirigían hacia ellos. A continuación se muestran los resultados obtenidos al utilizar la red entrenada con datos de la aplicación #3 para predecir movimientos de la aplicación #1 y #5.

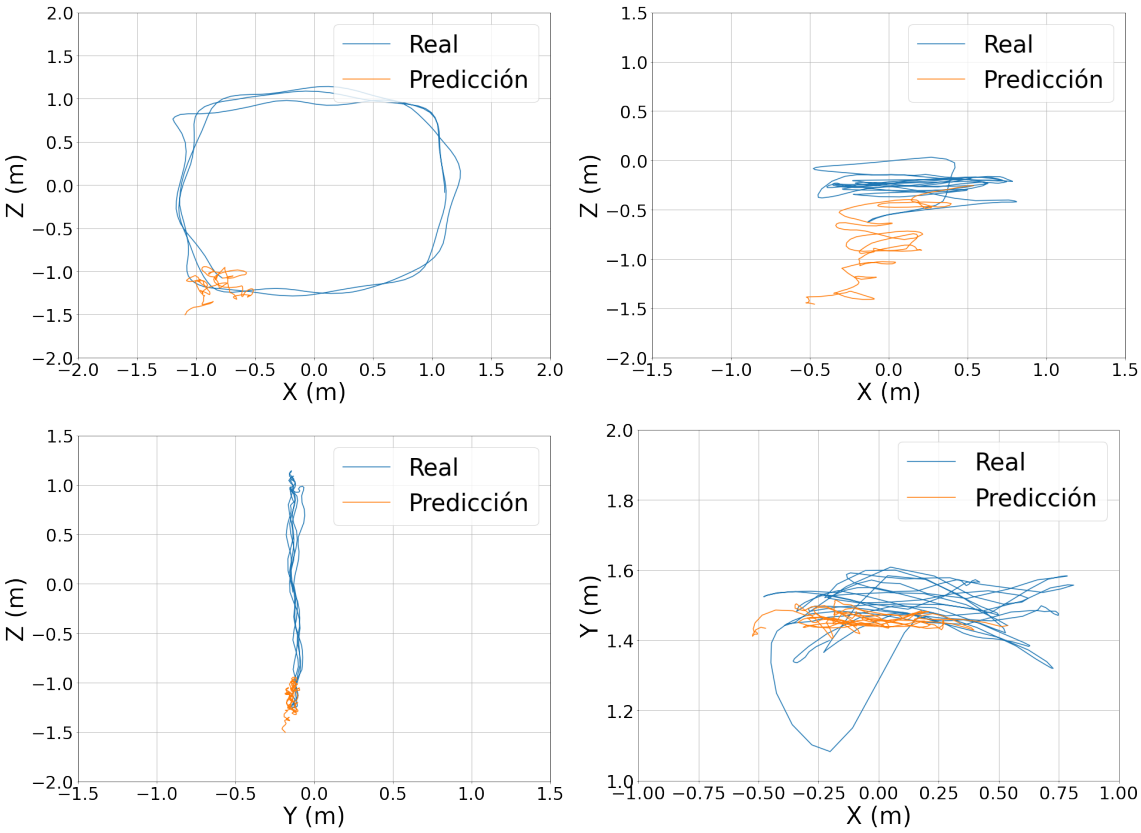
Como se aprecia en la figura 61, la red no es capaz de predecir correctamente una trayectoria que no fue observada durante su entrenamiento. En la tabla 21 se observa que al predecir una trayectoria de la aplicación #1 se alcanza un error, en promedio, de 0.7 metros aproximadamente, mientras que para la aplicación #5, el error alcanza los 0.33 metros. Debido a que esta red fue entrenada con una aplicación donde los usuarios no se trasladaban, a la red se le dificulta predecir movimientos translacionales como los que se observan en la aplicación #1. La segunda red fue entrenada con datos de la aplicación #2, donde los usuarios tenían que desplazarse constantemente, por lo que se espera que la red sea capaz de predecir de mejor manera los movimientos de traslación.

A partir de la imagen 62 y la tabla 22, podemos observar que la predicción de la aplicación #1 mejoró en comparación con la red anterior, esto debido a que esta red contiene datos de movimientos traslacionales. Sin embargo, la predicción del movimiento de la aplicación #5 empeoró en comparación con la anterior. De los resultados mostrados en esta sección, se puede concluir que la red general entrenada con todos los datos disponibles tiene un buen desempeño cuando se emplea para predecir diferentes tipo de movimientos.

Debido a que este trabajo es uno de los primeros atacando el problema de deriva en sensores inerciales para cascos VR, no hay un punto de comparación para los resultados obtenidos. Es por esto que se decidió comparar nuestros resultados con la trayectoria que se obtendría utilizando el método de doble integración, con el fin de comprobar la exactitud de las redes neuronales para corregir el error de deriva causado por los sensores inerciales.

6.7. Comparativa: doble integración y redes neuronales

Para tener un punto de referencia sobre la corrección del error de deriva utilizando las redes neuronales, se compararon las trayectorias predichas por redes con las que se producirían utilizando el método de la doble integración. Las redes neuronales utilizadas para comparar fueron las entrenadas con cuatro capas LSTM bidireccionales y sin considerar la velocidad angular como entrada, ya que como se vio en la sección 6.4, se mostró un mejor resultado utilizando esta arquitectura.



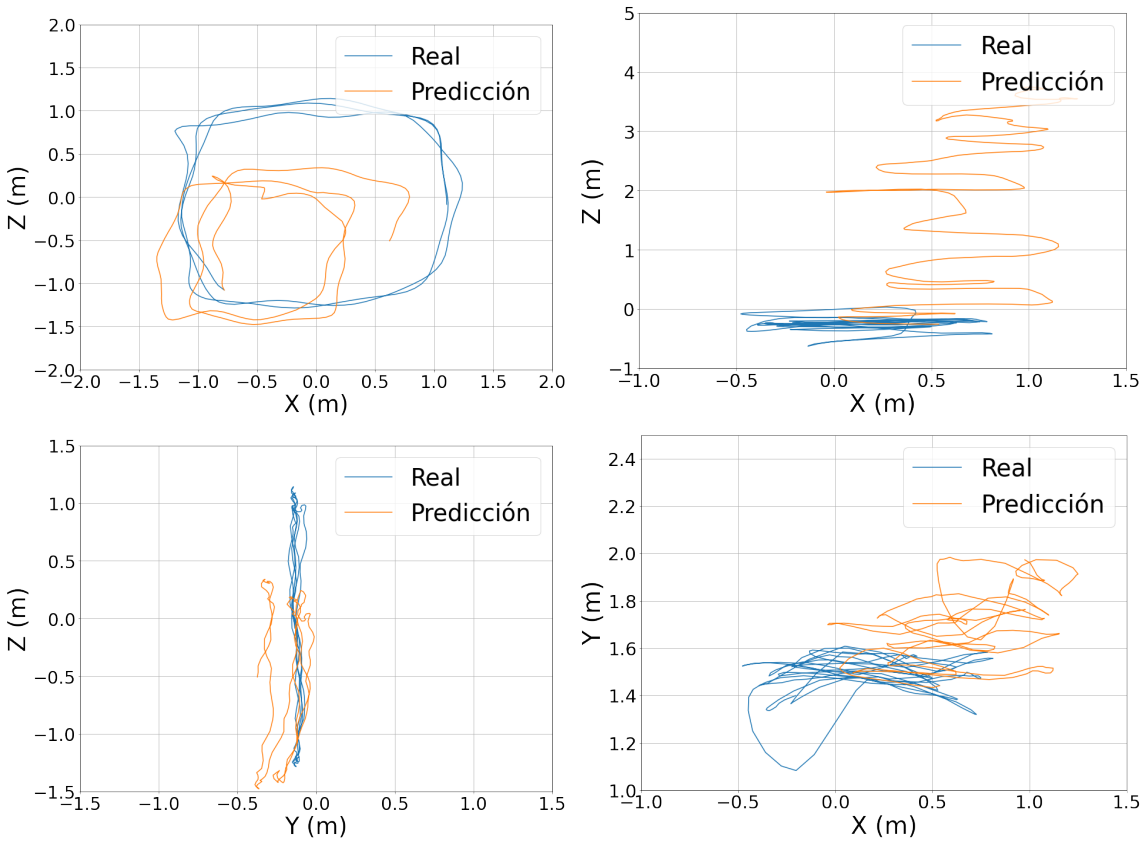
(a) Predicción de la aplicación #1.

(b) Predicción de la aplicación #5.

Figura 61. Resultados al realizar predicciones de movimientos no vistos por la red neuronal entrenada con datos de la aplicación #3.

Tabla 21. MAE (m) de la predicción de trayectorias no vistas.

Aplicación	MAE
1	0.7002
5	0.3305



(a) Predicción de la aplicación #1. (b) Predicción de la aplicación #5.

Figura 62. Resultados al realizar predicciones de movimientos no vistos por la red neuronal entrenada con datos de la aplicación #2.

Tabla 22. MAE (m) de la predicción de trayectorias no vistas.

Aplicación	MAE
1	0.3404
5	0.9196

La comparación entre los resultados de las redes y el método de doble integración fue dividido en tres partes:

- Calcular velocidad a partir de aceleración.
- Calcular posición a partir de velocidad.
- Calcular posición a partir de aceleración.

La razón por la cuál se decidió dividir el trabajo en estas tres partes fue para observar en cual paso de la doble integración es que se genera más rápidamente el

error de deriva. Para cada uno de estos puntos fue necesario entrenar una red, es decir, se entrenaron tres redes, una que convierte aceleración en velocidad, otra que predice posición a partir de velocidad, y finalmente una red que transforma aceleraciones en posiciones, como se han utilizado normalmente a lo largo de este capítulo. La primera comparación muestra los resultados de obtener una velocidad a partir de aceleraciones, utilizando tanto una red neuronal como la integración. Por simplicidad, se utilizaron únicamente los datos de la aplicación #1 para entrenar dicha red.

Como podemos ver en la tabla 23 y en la figura 63, cuando se trata de predecir una velocidad a partir de datos inerciales del acelerómetro, el mejor resultado se obtiene al integrar los datos, mientras que la predicción de la red presenta aproximadamente el doble de error.

En una segunda etapa, se entrenó otra red neuronal que predice una posición a partir de velocidades lineales. Estos resultados fueron comparados con la trayectoria obtenida al integrar la velocidad. Como se aprecia en la figura 64 y en la tabla 24, para el caso donde se predice una trayectoria a partir de velocidades, ambos métodos muestran un comportamiento similar después de 30 segundos. En la figura 65 se presenta la trayectoria obtenida al utilizar ambos métodos.

Finalmente, se entrenó una red donde se predice la posición a partir de las lecturas inerciales de la aplicación #1 y se compararon los resultados con la técnica de la doble integración. Observando las figuras 66 y 67, y la tabla 25, se puede concluir que la doble integración genera un error considerablemente mayor en comparación con los resultados obtenidos por la red. Como se había mencionado en el capítulo 2, el error de deriva crece cuadráticamente cuando se trata de obtener una trayectoria a partir de la aceleración, por lo que en este caso es mejor utilizar un método alternativo, y como se muestra en la imagen 67, utilizar la red neuronal mostró un mejor desempeño con estos datos.

Tabla 23. MAE (m/s) de los dos métodos utilizados para estimar velocidad.

Método	MAE
Integración	0.0528
Red neuronal	0.1057

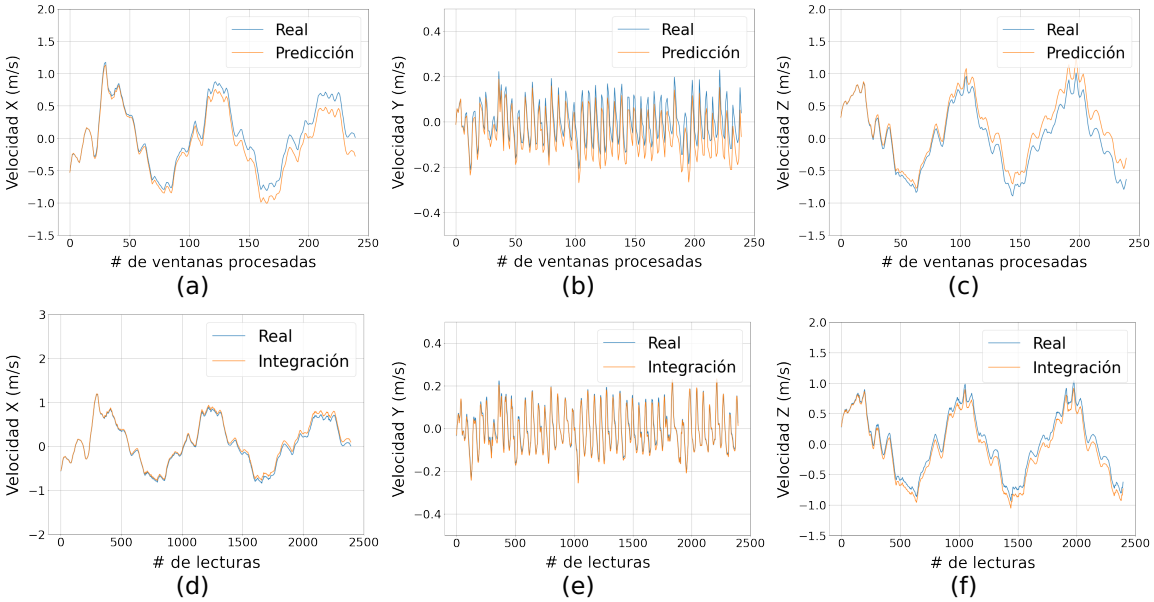


Figura 63. Las figuras (a), (b) y (c) presentan las velocidades predichas a partir de aceleraciones para las coordenadas X, Y y Z, respectivamente, haciendo uso de una red neuronal. Las figuras (d), (e) y (f) presentan las velocidades obtenidas a partir de aceleraciones para las coordenadas X, Y y Z, respectivamente, utilizando el método de la integración. En todos los casos se presentan 30 segundos de datos inerciales.

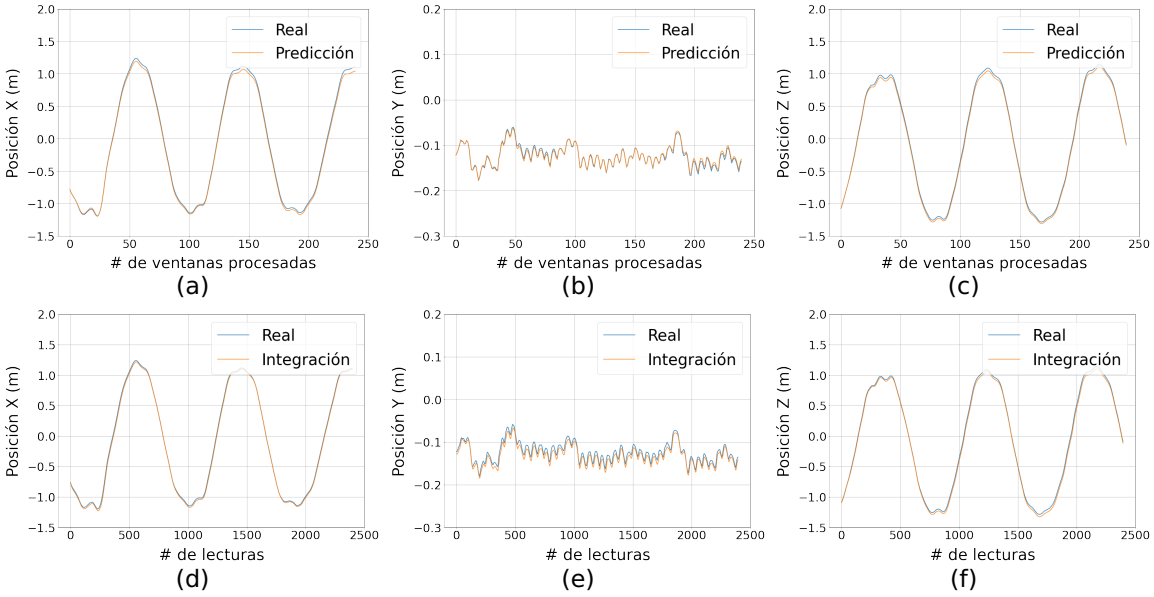
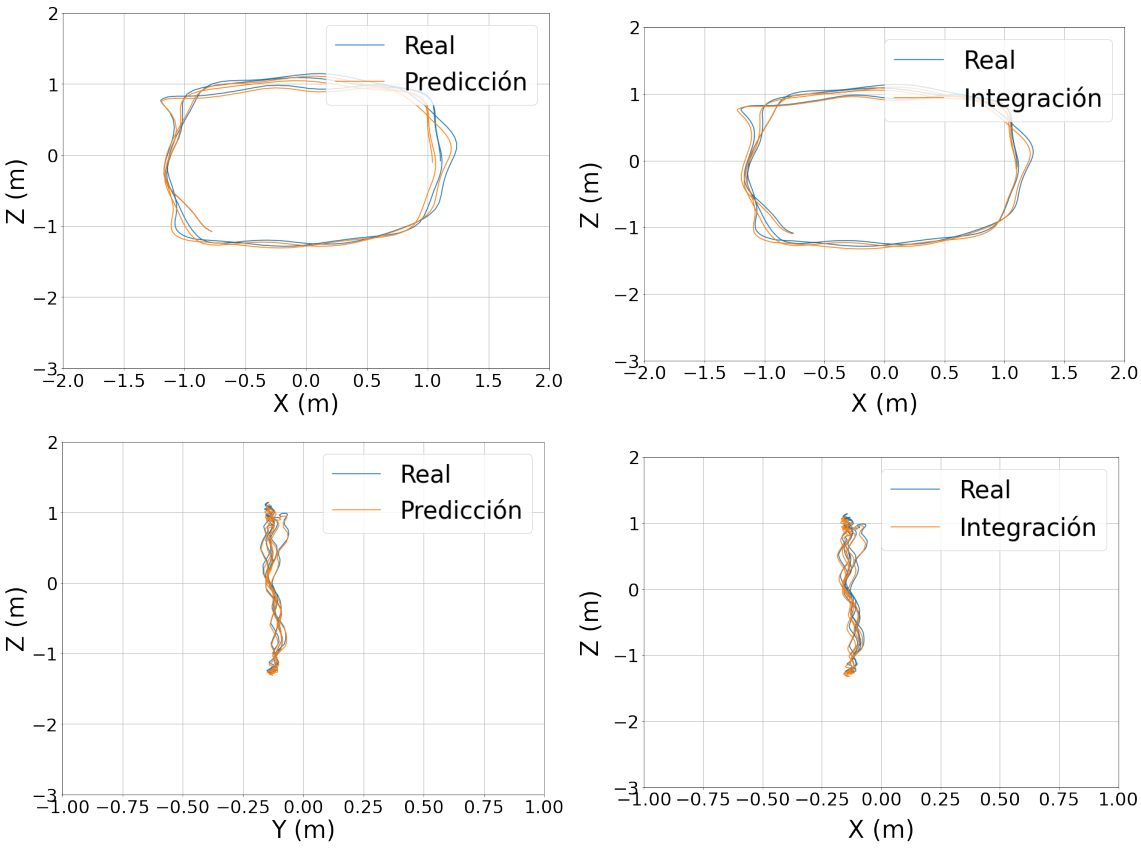


Figura 64. Las figuras (a), (b) y (c) presentan la posición predichas a partir de velocidades para las coordenadas X, Y y Z, respectivamente, haciendo uso de una red neuronal. Las figuras (d), (e) y (f) presentan la posición obtenida a partir de velocidades para las coordenadas X, Y y Z, respectivamente, utilizando el método de la integración. En todos los casos se presentan 30 segundos de datos inerciales.

Tabla 24. MAE (m) de los dos métodos utilizados para estimar posición.

Método	**MAE**
Integración	0.0208
Red neuronal	0.0223



(a) Resultados utilizando la red neuronal. (b) Resultados utilizando doble integración.

Figura 65. Trayectoria predicha por la red neuronal y el método de la integración a partir de velocidades.

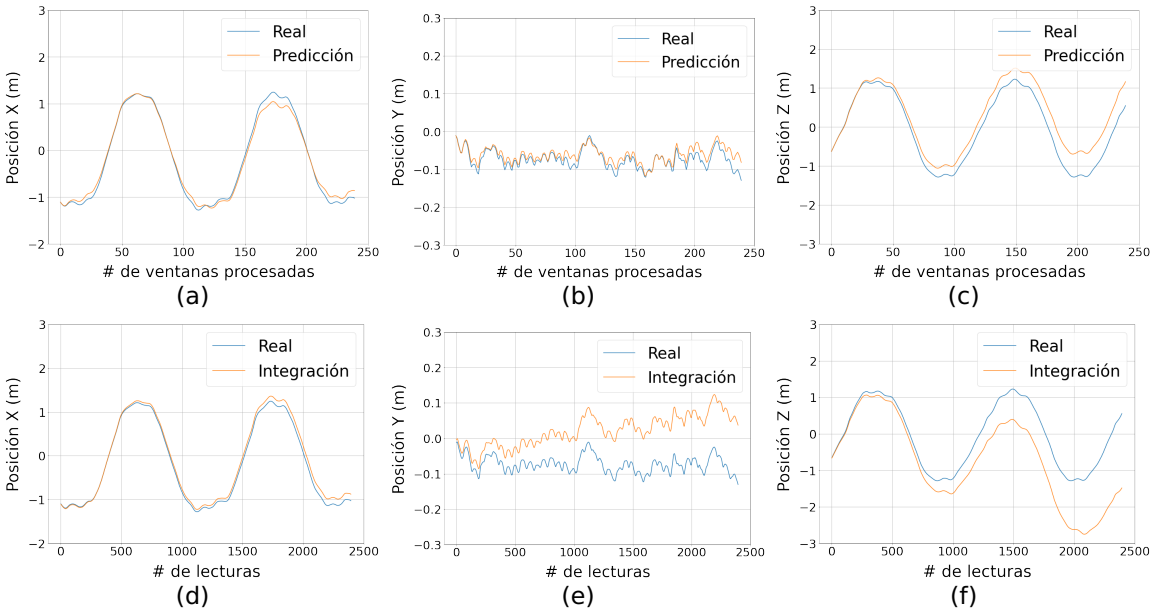
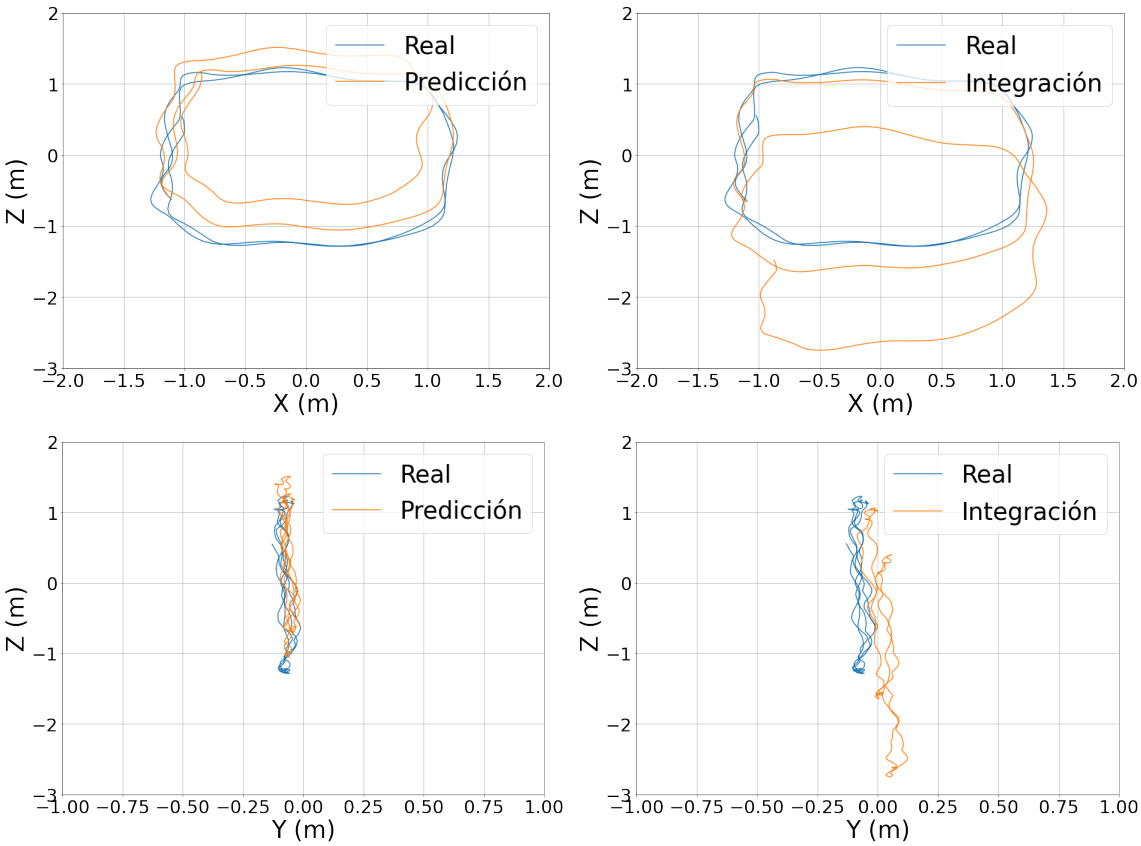


Figura 66. Las figuras (a), (b) y (c) presentan la posición predichas a partir de aceleraciones para las coordenadas X, Y y Z, respectivamente, haciendo uso de una red neuronal. Las figuras (d), (e) y (f) presentan la posición obtenida a partir de aceleraciones para las coordenadas X, Y y Z, respectivamente, utilizando el método de la doble integración. En todos los casos se presentan 30 segundos de datos inerciales.



(a) Resultados utilizando la red neuronal. (b) Resultados utilizando doble integración.

Figura 67. Trayectoria predicha por la red neuronal y el método de la doble integración.

Tabla 25. MAE (m) de los dos métodos utilizados para estimar posición a partir de la aceleración.

Método	**MAE**
Integración	0.7785
Red neuronal	0.2155

En este capítulo se abordaron diferentes metodologías para tratar de solucionar el error que se presenta cuando se predice una posición a partir de lecturas inerciales utilizando una red neuronal. De los diferentes resultados mostrados podemos decir que algo importante al momento de predecir posiciones es que a mayor cantidad de archivos, mejor es la precisión de la red. Aunque en ciertas arquitecturas probadas pareciera que aumentar el número de archivos no mejoraba la predicción, en general, los diagramas de caja mostraron que existe menor error para las redes entrenadas con 14 archivos.

También, fue posible determinar que el tamaño de ventanas no influye de manera considerable para la predicción, aún así, las mejores ventanas fueron 150 y 200 lecturas inerciales.

De los resultados obtenidos al comparar una red entrenada con diferentes aplicaciones y una entrenada con una aplicación en particular, pudimos notar que utilizar un conjunto de entrenamiento con mayor diversidad de movimientos produce, en general, un mejor desempeño al momento de tratar de predecir diferentes tipos de movimientos.

Finalmente, se pudo observar que el error obtenido al inferir una posición a partir de lecturas inerciales de la aceleración es mayor que el error resultante al hacer la predicción a partir de lecturas inerciales de la velocidad.

Capítulo 7. Conclusiones

7.1. Discusión

Como parte del estado del arte para el problema de corrección de deriva se han propuesto soluciones basadas en el uso de sensores exteroceptivos, propioceptivos y algoritmos de aprendizaje profundo. En cuanto a la predicción de trayectorias en realidad virtual, no se encontró ningún estudio relacionado, por lo que este trabajo se considera uno de los primeros en tratar de estimar la posición en ambientes virtuales con algoritmos de aprendizaje profundo. A consecuencia de esto, tampoco se ha registrado una base de datos que pueda ser utilizada para el entrenamiento de las redes neuronales. Por lo que una de las principales contribuciones de este trabajo fue la creación de una base de datos de lecturas inerciales recolectadas a partir de un IMU localizado en un HMD.

Al no existir trabajo previo relacionado que aborde la predicción de trayectorias en ambientes virtuales, no hay un punto de comparación claro para los resultados obtenidos. Sin embargo, los resultados de este trabajo, en específico los calculados por la red con cuatro capas LSTM bidireccionales entrenada con archivos de todas las aplicaciones, son superiores en comparación con los basados en el método de la doble integración y mejoran la estimación de la coordenada correspondiente a la altura con respecto a métodos propuestos en la literatura.

De los resultados obtenidos se pudo observar que cuando la altura a la que se encuentra el casco no varía demasiado, como sucede en la aplicación #1, predecir vectores de posición globales $\Delta\mathbf{P_G}$ arroja mejores resultados en la inferencia de la altura. A partir de lo anterior, se logró identificar que predecir cuaterniones genera deriva en la predicción de las redes, lo cual explica el desfase en la altura observado en el trabajo de Lima *et al.* (2019). Corregir el problema en la predicción de la altura es un paso importante para este trabajo, ya que el eje que modela la altura del usuario es una coordenada crítica.

Relacionado a lo anterior, LaValle (2016) detalla que utilizando una técnica llamada “redirected walking” es posible hacer que un usuario efectúe movimientos en un

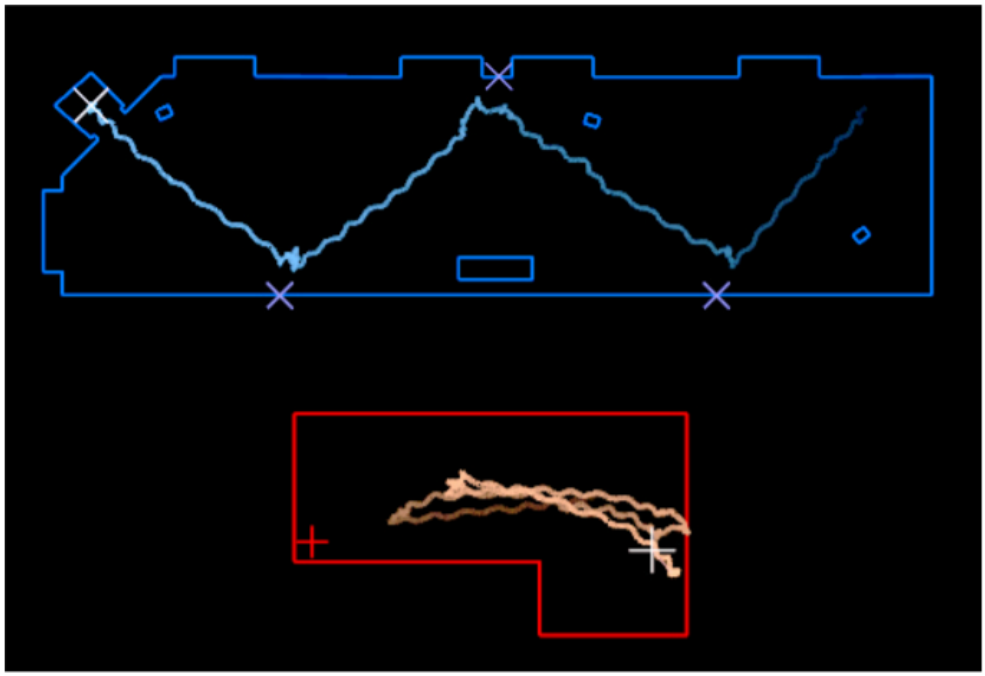


Figura 68. En color azul se presenta la trayectoria de un usuario en un ambiente virtual. En rojo la trayectoria del mismo usuario en el mundo real (Razzaque *et al.*, 2001).

mundo virtual realizando patrones diferentes en el mundo real. Un ejemplo de esto se presenta en el trabajo de Razzaque *et al.* (2001), como se observa en la figura 68.

Podemos notar en la figura 68 como en un mundo virtual un usuario simula caminar con un patrón de zigzag cuando en realidad únicamente está caminando hacia la izquierda y la derecha. Sin embargo, no existe un método para hacer lo mismo con la altura del usuario. Lo anterior implica que puede haber discrepancias entre el movimiento percibido a través del casco y el movimiento físico del usuario en el plano perpendicular a la gravedad, y que el usuario del casco no es capaz de percibirlas. No obstante, esto no se presenta para el eje de la altura, por lo que la corrección en el error de este eje es un punto clave en la predicción de trayectorias en ambientes virtuales.

Los mejores resultados obtenidos en este trabajo, obtenidos por la red general con cuatro capas LSTM bidireccionales, son de aproximadamente 15 centímetros de error después de predecir la trayectoria por 30 segundos. Sin embargo, se requiere mejorar estos resultados para poder ser utilizados en un sistema de realidad virtual. Para obtener una experiencia inmersiva en ambientes virtuales es indispensable realizar un seguimiento preciso de los movimientos del usuario, por lo que sería deseable poder

obtener resultados con un error de un par de centímetros por minuto de ejecución de una aplicación de realidad virtual.

7.2. Trabajo futuro

A partir de los resultados de este trabajo, se observó que hay casos en los que al aumentar el número de archivos se presenta una mejoría en la predicción de las redes. Por lo que a futuro, generar más archivos para cada una de las seis aplicaciones creadas podría favorecer el entrenamiento y reducir el error generado. También, el expandir la base de datos con más aplicaciones representando nuevos movimientos no considerados en este trabajo puede ser de utilidad para nuevos desarrollos.

Se exploró modificar la arquitectura de la red 6DOF IO, alterando sus entradas, sus salidas, el número de capas LSTM bidireccionales y la cantidad de datos para su entrenamiento. Sin embargo, existe la posibilidad de utilizar otras arquitecturas, por ejemplo, las redes transformers, con las cuales se han obtenido resultados muy prometedores en el estado del arte para otros problemas con información secuencial.

Durante el desarrollo de la sección 6.7, se observó que existe menor error de predicción al obtener la posición a partir de la velocidad, que en la predicción de la velocidad a partir de la aceleración. Esto puede ser debido a que la señal de aceleración tiene componentes de mayor frecuencia que la señal de velocidad, y en la literatura (Rahaman *et al.*, 2019) se ha reportado que las redes tienen una tendencia a predecir de manera más precisa componentes de baja frecuencia. Un análisis más detallado en esta dirección puede ser considerado como un trabajo futuro interesante.

Finalmente, siguiendo la línea marcada por el trabajo de Lima *et al.* (2019), se optó por implementar el sistema de rastreo de posición como una inferencia de un vector de traslación, tal que sumando una secuencia de estos vectores a una posición inicial, se genera una estimación de la trayectoria seguida por el casco. Sin embargo, un enfoque alternativo sería estimar a través de una red neuronal el error de deriva para una pequeña ventana de tiempo, y corregir el resultado de la doble integración de la aceleración utilizando un filtro complementario y la deriva estimada (LaValle, 2016).

Literatura citada

Asraf, O., Shama, F., y Klein, I. (2021). Pdrnet: A deep-learning pedestrian dead reckoning framework. *IEEE Sensors Journal*, pp. 1–8.

Basu, A. (2019). A brief chronology of virtual reality. *ArXiv*, **abs/1911.09605**. Consultado el día 12 de junio del 2021 de: https://www.ArXiv.

Borrego, A., Latorre, J., Alcañiz, M., y Lloréns, R. (2018). Comparison of oculus rift and htc vive: Feasibility for virtual reality-based exploration, navigation, exergaming, and rehabilitation. *Games for health journal*, **7 (3)**: 151–156.

Brooks Jr, F., Burbeck, C., Durlach, N., Ellis, M. S., Lackner, J., Robinett, W., Srinivasan, M., Sutherland, M. I., Urban, D., y Wenzel, D. E. (1992). Research directions in virtual environments. *Computer Graphics*, **26**: 153–173.

Burdea, G. (2006). Virtual reality technology - an introduction. *IEEE Virtual Reality Conference (VR 2006)*, pp. 307–307.

Chen, C., Lu, X., Markham, A., y Trigoni, N. (2018a). Ionet: Learning to cure the curse of drift in inertial odometry. *Proceedings of the AAAI Conference on Artificial Intelligence*, **32**: 6468–6476.

Chen, C., Zhao, P., Lu, C. X., Wang, W., Markham, A., y Trigoni, A. (2018b). Oxiod: The dataset for deep inertial odometry. *ArXiv*, **abs/1809.07491**. Consultado el día 30 de octubre del 2021 de: https://www.ArXiv.

Chen, C., Zhao, P., Lu, C. X., Wang, W., Markham, A., y Trigoni, A. (2020). Deep-learning-based pedestrian inertial navigation: Methods, data set, and on-device inference. *IEEE Internet of Things Journal*, **7**: 4431–4441.

Cipresso, P., Giglioli, I., Raya, M. A., y Riva, G. (2018). The past, present, and future of virtual and augmented reality research: A network and cluster analysis of the literature. *Frontiers in Psychology*, **9**: 1–20.

Clark, R., Wang, S., Wen, H., Markham, A., y Trigoni, N. (2017). Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, **31**: 1–7.

Cruz-Neira, C., Sandin, D. J., y DeFanti, T. A. (1993). Surround-screen projection-based virtual reality: the design and implementation of the cave. *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pp. 135–142.

Desai, P., Ajmera, K., y Mehta, K. (2014). A review paper on oculus rift-a virtual reality headset. *ArXiv*, **abs/1408.1173**. Consultado el día 26 de junio del 2021 de: https://www.ArXiv.

Earnshaw, R. A. (2014). Virtual reality systems. Academic press.

Esfahani, M. A., Wang, H., Wu, K., y Yuan, S. (2020). Aboldeepio: A novel deep inertial odometry network for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, **21**: 1941–1950.

Geiger, A., Lenz, P., y Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. En: *2012 IEEE conference on computer vision and pattern recognition*. IEEE, pp. 3354–3361.

Hesch, J., Kozminski, A., y Linde, O. (2019). Powered by ai: Oculus insight. https://ai.facebook.com/blog/powered-by-ai-oculus-insight/. Consultado el día 16 de agosto del 2021.

Jost, T. A., Nelson, B., y Rylander, J. H. (2019). Quantitative analysis of the oculus rift s in controlled movement. *Disability and Rehabilitation: Assistive Technology*, **16**: 632 – 636.

Kovalenko, D., Migukin, A., Ryabkova, S., y Chernov, V. (2021). Pluto: Motion detection for navigation in a vr headset. *ArXiv*, **abs/2107.12030**. Consultado el día 30 de septiembre del 2021 de: https://www.ArXiv.

LaValle, S. (2016). Virtual reality. National Programme on Technology Enhanced Learning (NPTEL).

LaValle, S. M., Yershova, A., Katsev, M., y Antonov, M. (2014). Head tracking for the oculus rift. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 187–194.

Lim, H., Park, C., y Myung, H. (2019). Ronet: Real-time range-only indoor localization via stacked bidirectional lstm with residual attention. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3241–3247.

Lima, J. P., Uchiyama, H., y ichiro Taniguchi, R. (2019). End-to-end learning framework for imu-based 6-dof odometry. *Sensors (Basel, Switzerland)*, **19**: 1–16.

Liu, W., Caruso, D., Ilg, E., Dong, J., Mourikis, A. I., Daniilidis, K., Kumar, V. R., y Engel, J. (2020). Tlio: Tight learned inertial odometry. *IEEE Robotics and Automation Letters*, **5**: 5653–5660.

Maddern, W. P., Pascoe, G., Linegar, C., y Newman, P. (2017). 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, **36**: 15 – 3.

Magnusson, T. (2019). Echoes of other worlds: sound in virtual reality. *International Journal of Performance Arts and Digital Media*, **15**: 122 – 123.

Mazuryk, T. y Gervautz, M. (1996). Virtual reality-history, applications, technology and future. Citeseer.

Nafees, A. (2016). Oculus rift : A rift in reality. Consultado el día 20 de septiembre del 2021 de: https://www.researchgate.net/.

Ng, A. K. T., Chan, L. K. Y., y Lau, H. Y. K. (2017). A low-cost lighthouse-based virtual reality head tracking system. *2017 International Conference on 3D Immersion (IC3D)*, pp. 1–5.

Niehorster, D. C., Li, L., y Lappe, M. (2017). The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research. *i-Perception*, **8**(3): 1–23.

Oculus (2019). Oculus rift s: Análisis. https://www.realovirtual.com/articulos/5346/oculus-rift-s-analisis. Consultado el día 29 de octubre del 2021.

Poznyak, T., Chairez, I., y Poznyak, A. (2018). *Ozonation and biodegradation in environmental engineering: Dynamic neural network approach*. Elsevier.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., y Courville, A. (2019). On the spectral bias of neural networks. En: *International Conference on Machine Learning*. PMLR, pp. 5301–5310.

Razzaque, S., Kohn, Z., y Whitton, M. C. (2001). Redirected walking. *Proceedings of Eurographics*, pp. 1–6.

Reina, S. C., Solin, A., Rahtu, E., y Kannala, J. (2018). Advio: An authentic dataset for visual-inertial odometry. *ArXiv*, **abs/1807.09828**. Consultado el día 15 de septiembre del 2021 de: https://www.ArXiv.

Sensoryx (2019). Positional traking. `https://www.sensoryx.com/position-tracking/`. Consultado el día 29 de octubre del 2021.

Shahid, U. (2017). Design and implementation of robotic arm that copies the human arm. `https://www.researchgate.net/figure/Explanation-of-roll-pitch-and-yaw-movements-of-a-gyroscope_fig10_319619198`. Consultado el día 23 de enero del 2022.

Sorensen, B. R., Donath, M., Yang, G., y Starr, R. C. (1989). The minnesota scanner: a prototype sensor for three-dimensional tracking of moving body segments. *IEEE Trans. Robotics Autom.*, **5**: 499–509.

Sun, S., Melamed, D., y Kitani, K. (2021). Idol: Inertial deep orientation-estimation and localization. Consultado el día 29 de octubre del 2021 de: https://www.ArXiv.

Vázquez, F. (2018). Skejul meetings with deep learning. `https://towardsdatascience.com/skejul-meetings-with-deep-learning-5efab285b111`. Consultado el día 28 de octubre del 2021.

Wu, J. y Li, G. (2020). Drift calibration using constrained extreme learning machine and kalman filter in clustered wireless sensor networks. *IEEE Access*, **8**: 13078–13085.

Yan, H., Herath, S., y Furukawa, Y. (2020). Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3146–3152.

Yan, K. y Zhang, D. (2016). Correcting instrumental variation and time-varying drift: A transfer learning approach with autoencoders. *IEEE Transactions on Instrumentation and Measurement*, **65**: 2012–2022.

Yao, S., Hu, S., Zhao, Y., Zhang, A., y Abdelzaher, T. (2017). Deepsense: A unified deep learning framework for time-series mobile sensing data processing. *Proceedings of the 26th International Conference on World Wide Web*, pp. 351–360.